Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Investigating the Application of Transfer Learning in Microgrid Energy Management

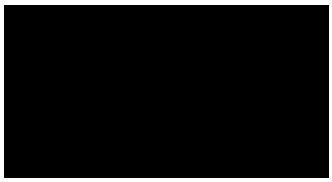
Patrick Lee
Supervisor: Ivana Dusparic

April 30, 2020

A Dissertation submitted in partial fulfilment
of the requirements for the degree of
Master in Computer Science (MCS)

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Signed:  Date: 30/04/2020

Abstract

Climate change is a legitimate concern in the world today and, through the increased occurrence and intensity of extreme weather and a string of record high temperatures, is a concern that we are already feeling the effects of. The biggest culprit of this phenomenon is the vast amounts of CO2 that are being pumped into the air primarily due to the carbon-intensive energy generation ubiquitous in the world today. Renewable energy supply (RES) is a key component to achieve sustainable clean energy production by shifting away from energy generation with harmful by-products and towards solar, wind, or hydro energy. However, currently, the high implementation and maintenance cost, unpredictability in energy generation, and the lack of an established system to manage the generation effectively when it occurs are proving to be the barriers to fully adopting RES. To overcome these obstacles it is important to have efficient and effective energy management systems in place. Solutions are being developed to optimise devices such as PV panels and wind turbines to curtail their awkward generation schedule and efficiently store any generated energy for later use. The solutions are also being adopted on a large scale, across a group of RES generation devices and energy stores and loads called microgrids. Currently a machine learning model known as Reinforcement Learning (RL) is proving successful in optimising the management system for a microgrid. In RL, an agent (the microgrid) learns how to perform a task (energy management) by interacting with its environment (consumption/production profiles and devices connected to the grid). The issue, however, is that these RL models need a lot of time for training to find the optimal policy as each agent is being trained from scratch, (i.e., with no prior knowledge). Transfer learning (TL) is an approach towards reducing this training delay.

TL is a relatively new phenomenon where a pre-trained model, or a partially pre-trained model, is used to speed up the training of another model that is learning a separate task. The pre-trained model is referred to as the source model and the model to be trained is referred to as the target model. In this paper, I investigate the application of a TL approach on a Deep RL model (1) that looks to optimise the management of energy in a microgrid. The model contains production and consumption profiles from a residential customer in Belgium which are used as the training samples. The goal of the energy system is to find the best policy to handle the demand and supply of energy in the microgrid by managing a short-term and long-term storage connected to the grid. The TL approach used in this paper is known as Weight Initialisation and is implemented by taking the weights produced by the source model, after it has been trained, and using those values to initialise the weights for the target model. The scenarios tested were source models generated from different configurations of the microgrid (Small, Medium, Large) and used to initialise the weights of a different configuration. This took the approach a step further to analyse the value of the knowledge obtained by one configuration of

microgrid compared to another. This paper looks to determine the effect of TL on a Deep RL model for energy management and to also investigate whether the knowledge learned from a smaller or larger microgrid is more useful to a different microgrid.

Acknowledgements

I would like to thank everyone who has helped and supported me throughout the last five years at this university, from family, friends, and professors to gym assistants and baristas the whole lot. The brightest stars amongst that bunch are my family, whose support and dedication to my well being and education has pushed me to the brink of this degree and set me up for a lifetime of striving to repay them. I would also like to thank my girlfriend, Indre, for her love and support and for putting up with the many hours I spent sitting behind my laptop. Last but not least, I would also like to thank my supervisor, Prof. Ivana Dusparic, for keeping me on track and heading in the right direction throughout this dissertation.

Muchísimas gracias a todos!

Despite a global pandemic, impending and irreversible environmental damage, and an intellectual pandemic threatening the sanctity of fact, this paper was produced for your perusal. *The lone and level sands stretch far away.*

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

Renewable energy is a popular topic these days with the damage current non-renewable energy production methods have inflicted on the environment. A sustainable application of RES on a global scale would go a long way to reducing the rate in which our climate is changing, a figure that feels increasingly insurmountable each day. The ultimate goal is to have all energy production sourced from RES but there are a lot of challenges that we must tackle before then. RES faces difficulties, such as its unpredictable generation schedule and location dependency (for optimum energy generation and then in comparison to the main grid), which leads to the disproportion of its cost with the energy it generates. This makes RES problematic to adopt globally and at the scale required. However, solutions are being developed and more and more countries have been accelerating their energy production towards renewables. Ireland (2) is a good example of this, RES made up 11% of the gross final energy consumption in the country as of 2019, with a target of 16% to be met by 2020. RES generated electricity is up 3.1% to 33.2% from 2018, with a 40% target in 2020, and overall fossil fuel demand is down to 15% from 2005. The large investment in improving RES technology has become a major catalyst to the trends we are seeing. As the implementation becomes more cost-effective and accessible, it will lead to further adoption that will reduce the cost even more through economies of scale. Different areas can be looked at in the RES production cycle to improve its efficiency but one area that my field of computer science can contribute to is how the device, or the grid of devices, can optimise its energy management. The area of computer science that is showing good results is Reinforcement Learning (RL), a learning technique where an agent interacts with an environment to maximise the rewards it can obtain. It is a commonly used technique to solve a Markov Decision Process (MDP), which is a mathematical framework for modelling decision making that involves an element of randomness. There are many examples of successful applications of RL, and its variations, in energy management. These vary in scope, some look at building energy management systems to save energy

when occupancy is low (3) or to generate a 24 prediction based on previous consumption (4). The scope can increase to energy management for microgrids (1) or be built just for regulating consumption at home (5). The area of energy management this paper looks at is microgrids and at a particular deep RL model from *Francois-Lavet* et al. (1). The microgrid has four components: {Production profiles, Consumption profiles, Short term battery storage (STS), Long term hydrogen storage (LTS)}. The production and consumption profiles for the model were collected from a residential customer located in Belgium. The production is generated from a solar PV panel and the consumption is standard residential consumption from the customer. The data collected is over two years with various values for summer and winter production and consumption. The two different storage types are used to handle the demand in the system. The desired behaviour of the system is to charge the LTS when demand is low and can be handled by the STS and current generation of that particular time step and then to discharge the LTS when demand is high and cannot be handled by the STS or current production. In the case of a surplus of energy produced the STS is charged.

The RL model that implements this behaviour, is an area of RL known as Deep Reinforcement Learning (DRL). DRL models operate in the same way as RL models except they incorporate Neural Networks (NN) to be able to solve more complex tasks. A NN is a network of nodes that are organised in layers and each node behaves like a neuron in the human brain, so a collection attempts to mimic the way a brain processes information. The system is formalised as a partially observable MDP where the microgrid is the agent and the environment is a production and consumption tuple of values where the agent must act correctly in the manner described above. If the microgrid takes the correct action it will receive a reward. The NN is used to approximate the values of the actions the microgrid can take (ie to either charge or discharge the LTS) so the microgrid can take the action that yields the highest reward. The weights are then updated depending on the consequences of the action, for example, they will be altered in such a way that if the microgrid took the incorrect action for a particular input the weights will then process the same input differently to give the correct action a higher value. At the end of the training process the weights are tuned to the input profile and also the characteristics of the microgrid to enable the microgrid to take the actions that yield the highest reward for any input.

RL models, such as the one described, normally require prolonged testing and extensive computational power to be effective for the specific task, which only increases as more scenarios are introduced to create a more robust system. A way to reduce this training period is to reuse pre-trained models and adapt them to the different tasks. This is the bases of Transfer Learning (TL), a machine learning technique that looks to transfer knowledge from one model (source model) to the model in question (target model) for

more efficient learning. TL has been very effective in tasks such as image classification where it is common for models that have been trained on large databases of millions of images, such as ImageNet (6), to be fine-tuned for a separate task. It has also been very useful in situations where there are limited or no historical training samples for the target task and other models trained for a similar task can be used. An example of this is zero-shot learning which is where there are no samples to work on and the task is to translate between two languages where no sample translations exist and the model has to try and draw similarities from other translations and the machine translation models that have been trained for them.

This thesis uses a particular branch of TL known as Weight Initialisation, which takes the weights of a trained network and uses them to initialise the network of the target model. The weights are first tuned to the source task by being regularly updated during the training of the source model and by the end of training can process inputs to the model in the desired way. The approach is then to use the values of these weights as the initial values of the weights for the target model. The weights will then be adjusted to the target model as it trains but this will involve smaller updates compared to the scenario where the target models weights have been initialised. It is then extremely beneficial if the tasks of the source and target models are similar, as the optimal weights should then be close in value.

1.2 Thesis Objective

The general aim of this thesis is to investigate the effectiveness of TL for the energy management system of a microgrid and promote its potential use for systems of other devices. Current research in this space either focuses on the learning model for a single microgrid and does not take the step towards using TL between different models or, when TL is used, is limited in scope to residential or office buildings and the research solves a subsection of the large problem of optimising the management of energy. This paper uses a single agent learning model as a foundation for applying the TL technique weight initialisation. This approach is also uncommon in energy management and was chosen due to the similarity of the tasks of the source and target models. The evaluation of this approach will be how the models that underwent initialisation compare to the models that begin training with no initialisation of weights. The comparison will be made using validation and test scores that the model receives during the validation stage of the training process.

A secondary aim of this thesis is to investigate the value of the knowledge obtained by different microgrid configurations. As the transfer will be performed between microgrids with varying resources and capacity, it introduces a separate analysis of which

pre-trained weights will yield a better performance when initialising a model of a different configuration. The comparison will be between up-scaling and down-scaling the transfer, in other words determining whether a microgrid with a larger capacity obtains more valuable knowledge than a microgrid with limited resources. The evaluation of this aim will also be conducted using the validation and test scores of the initialised model and the baseline model with the corresponding characteristics.

1.3 Thesis Contribution

This thesis provides an initial investigation into the use of TL for energy management systems of microgrids, motivating its use as a way to speed up the training process of these learning models to enable them to reach a certain level of performance more quickly. It looks to combine the areas of TL and energy management for further research to build upon and advance in different ways: by varying the TL approach, microgrid architecture and the environment the microgrid operates in. The approach could even be tested using different energy generation and control devices. This approach uses weight initialisation, a TL technique that can be extremely effective in this context due to the similarities between the domains and tasks of this particular TL problem and the results show promise that a more flexible and robust solution can be developed from its concepts. The research also provides some insight into the value of the knowledge obtained by learning models of different configurations. The idea of up-scaling or down-scaling the weight initialisation from smaller to larger microgrids and vice versa is investigated and the results motivate the need for consideration in these scenarios.

1.4 Thesis Layout

The subsequent sections of this paper are organised as follows: **Section 2**, *Background and Related work*, will first look at RL and TL and their different concepts and then look at the cutting-edge research. This section will then look at microgrids and RES and discuss the current climate around energy management. I will conclude this chapter by dissecting the research that combines RL/TL and energy management to explain the motivation for this paper. **Section 3**, *Design*, will explain the learning model this paper is based on and also give an outline of the transfer method I propose. **Section 4**, *Implementation*, will discuss how the model was constructed and run, beginning with the DRL model (1) and ending with the TL implementation. **Section 5**, *Evaluation*, present the objectives, metrics, and scenarios in the evaluation and then analyse the results from each scenario. **Section 6**, *Conclusion*, will give a summary of this research and outline its contribution and any future work that could build on this.

2 Background and Related Work

In this section, I will focus on all areas that make up this research. I will begin with RL and TL and introduce some of their concepts, referencing existing research to give an idea of how they evolved into their current state. I will conclude the section on TL with a look at Weight Initialisation, the approach used in this paper. I will then give a background into RES and microgrids, giving a brief explanation of each as a prelude to the final section that looks at the current research of learning models for energy management. I will summarise research that has merged these two disciplines and explain where my research fits into that landscape.

2.1 Reinforcement Learning (RL)

RL is an area of machine learning that provides the foundations for TL, as TL is a way to speed up the training process in an RL problem. RL is a learning model that contains an agent and the environment it exists in. The system can be modelled in different ways with different characteristics but in general they all follow the same basic structure. The agent inhabits states in the environment and learns about the environment through experimentation, it takes an action that within the environment which can potentially bring the agent to a new state, and then is rewarded if that action was favourable towards the aim of the task. The agent behaves in a way to maximise the reward function of the system.

A Markov Decision Process (MDP) is an example of a problem that can be solved using RL techniques. It is also how the microgrid system is formalised in *Francois-Lavet* et al. (1), the model used in this paper. A basic MDP is constructed with 5 elements: {S,A,T,R,γ}, *State space* S, *Action space* A, *Transition function* T, *Reward function* R and *Discount factor* γ. The state space and actions contain all possible states the agent can inherit and all the possible actions the agent can take. The transition function determines the next state the agent will enter depending on the action it takes and the reward function determines the reward the agent receives, which could also be a penalty for a wrong action. The discount factor essentially dictates whether the agent seeks the

action that gives instant reward or considers future actions. The Markovian aspect means that the agent is only concerned with current observations and no historical experience, i.e., no memory. An MDP also carries the concept of observability, where an observation in a fully observable MDP is the same as the state of the environment.

There are several approaches to using RL for an MDP and one of the main decisions to be made when formulating the model is how the agent gains experience of the environment. *offline* learning refers to a model where the experience of the world is learned a priori and in *online* learning the experience is collected in real-time by the agent. An online approach introduces the concept of *Exploitation* vs *Exploration*, as the agent must balance maximising reward through what is known already and potentially finding greater reward through taking different actions. The sharing of focus is normally implemented by assigning a probability to the choice between the two options. This probability is usually denoted by epsilon (ϵ) and decreases during the training process so the agent focuses on the best policy and takes less random actions. RL can also be sub-categorised by the instructions or policy the agent follows in the environment. This can be simplified as Model-based vs Model-free RL. In model-based approaches the agent forms an estimate of the world it lives in so it can make informed choices. Once the estimate has been formed the agent does not need to explore much more of the environment, making this approach, potentially, extremely sample efficient. In model-free techniques, there are value-based approaches where the agent is only concerned with the best state and the best action to take in that state to maximise the reward. Most algorithms are model-free as model-based becomes severely impractical as the state and action space becomes larger.

Q-learning is an example of an off-policy or model-free approach and is one of the most popular RL techniques used to solve an MDP. Its goal is to learn a policy that maximises the total reward and must balance between exploration and exploitation. As the agent learns, a Q-table is created that contains state-action pairs with a Q-value that represents the value of taking that action when in that agent. It has some properties such as the *learning rate* and the *discount factor*, similar to the MDP, where the learning rate is usually denoted by alpha (α) and represents how much weight you give to new information and the discount factor, as mentioned, is recognised as gamma (γ) and is used to balance future reward from immediate. Q-learning has some variations that give different properties, such as ϵ-greedy that favours a lot of exploration at the beginning of the simulation and then moves steadily towards exploitation overtime. In an on-policy model the agent learns a stochastic policy function that maps state to action. **SARSA** is an example of on-policy which has a similar structure to Q-learning but differs in the behaviour of the agent. In SARSA, the agent will choose each action according to a specific policy, whereas Q-learning just

seeks to maximise reward but may deviate to explore.

2.1.1 Deep Reinforcement Learning (DRL)

DRL (7) (8) is an area of RL which incorporates Neural Networks (NN). A NN (9) is an adaptive system that mimics how the human brain works. It is a network of sometimes millions of nodes which process the input given to them and adapt depending on the goal of the system. NNs for deep learning contains multiple layers of nodes that process a different part of the input. The first layer is the input layer, which is succeeded by x number of 'hidden' layers that process the inputs and then there is the final output layer that determines the result. A simple NN (10) can work using gradient descent and utilising back-propagation to tweak its internal weights (denoted as θ) at each iteration to move closer to the function goal. A common example of a task could be the use of a NN to classify digits in the MNIST dataset. The first layers make more specific determinations, the top horizontal bar in the digit number '5' for example. The layers nearer the end of the network, however, will hold representations closer to the final classification, i.e., the variations of the number '5' written in its entirety.

The DRL algorithm (1) used in this paper is a Deep Q-Network (DQN). DQNs enhance regular RL Q-networks to be able to handle complex tasks such as those that have images as input (i.e., millions of pixels). *Mnih* et al. 2015 (11) was one of the first to use a DQN and used it to learn to play seven Atari 2600 games, where the input is raw pixels and the output is value functions. They found that the model was able to learn the complex policies required to complete the games. The difference between Q-learning and Deep Q-learning is not as simple as applying a NN to Q-learning as a NN is not very effective at learning exact values as it can diverge easily. A good definition is that Deep Q-learning refers to using an NN as a function approximator for Q-learning and includes the additional techniques that take care of the instability of NNs (12). One of these techniques is known as 'Double Q-learning' that creates separate values for the selection of an action and the evaluation of an action, whereas in regular Deep Q-learning these values are the same. *Van Hasselt* et al. 2015 (13) from Google's deepMind proposed the Double Q-learning approach for the same seven Atari games used by *Mnih* (11) and yielded better performance.

Experience Replay

The classic example of a DQN (14) is the use of a deep NN to approximate the value for the state-action pair in Q-learning. This can become unstable but the inherent temporal dependency of an MDP, that is the reliance on past experiences to predict future ones, can be exploited through a concept known as experience replay. This functionality enables the agent to reuse past experiences. At the end of every training

step in a DQN, where a small set of states, actions, rewards and next states $\{S, A, R, S'\}$ are used to train the network before it approximates the state-action function. It can be said to store the trajectory of the MDP as it offers a history of correct actions to guide the agent (15). There are different approaches to experience replay, *Novati* et al. (16) address the situation where the agent should deviate from its past behaviours and therefore the experience replay is unhelpful. They propose an approach they coined 'Remember and Forget Experience Replay' (ReF-ER) which regulates the experiences and removes those that have little chance of occurring under the current policy. *Andrychowicz* et al. (17) also looks for a more efficient process by proposing a replay for rewards that are sparse and binary. *Schaul* et al. (18) look to give significance to an experience so it is prioritised over other transitions that were less relevant. The DRL model in this paper uses experience replay in a DQN by containing a list of previous observations that are sampled at the end of each training step to update the state-action values.

2.2 Transfer Learning (TL)

TL is a process where the knowledge gained from completing one task is used to speed up the process of learning how to complete another task. The concept was based on the action a human takes when faced with a task they have not yet completed, they will draw on knowledge they learned from solving other tasks, generally utilising knowledge from tasks that are more closely related. Within a computer science context, TL can be applied to RL models to speed up their learning on a particular task by using information from a trained model that learned a separate, but usually similar task.

2.2.1 Formal definition and notation

TL can be defined as per *Pan* and *Yang* (19) and *Weiss* et al. (20):

A *domain* D is defined by two parts, a feature space X and a marginal probability distribution P(X), where X = $\{x_1, ..., x_n\} \epsilon X$. In the case of a machine learning application for facial recognition and every detail of a face is taken as a feature, then X is the space of all possible feature vectors and X is a particular learning sample, where x_i is the *i-th* feature vector corresponding to the *i-th* component of the input, a section of the image for example.

For a given domain D, a task T is defined by two parts, a label space Y, and a predictive function f($\cdot$), which is learned from the feature vector and label pairs $\{x_i, y_i\}$ where $x_i \epsilon$ X and $y_i \epsilon$ Y. If we consider the facial recognition application, Y is the set of labels which would include all facial images and their corresponding information (i.e.,

labels) currently stored in the database, y_i takes on that label value, and f(x) is the learner that predicts the label value of the input image x.

From the definitions above, a domain $D = \{X, P(X)\}$ and a task $T = \{Y, f(\cdot)\}$. Now, D_S is defined as the source domain data where $D_S = \{(x_{S1}, y_{S1})..., (x_{Sn}, y_{Sn})\}$, where $x_{Si} \epsilon X_S$ is the *i-th* data instance of D_S and $y_{Si} \epsilon Y_S$ is the corresponding class label for x_{Si}. In the same way, D_T is defined as the target domain data where $D_T = \{(x_{T1}, y_{T1})..., (x_{Tn}, y_{Tn})\}$, where $x_{Ti} \epsilon X_T$ is the *i-th* data instance of D_T and $y_{Ti} \epsilon Y_T$ is the corresponding class label for $x_T i$. Further, the source task is notated as T_S, the target task as T_T, the source predictive function as $f_S(\cdot)$, and the target predictive function as $f_T(\cdot)$.

TL can then be defined as the process of improving the target predictive function $f_T(\cdot)$ by using the related information from source domain D_S and target domain T_S, when given D_S and its corresponding task T_S and D_T and its corresponding task T_T and $D_S \neq D_T$ or $T_S \neq T_T$. (20)

2.2.2 Classifying problems & common issues

Labeled data availability

Machine learning problems are commonly classified as unsupervised or supervised learning problems; with the former referring to a problem with only unlabeled data available and the latter referring to a problem with some labeled data available. In unsupervised learning the correct classification of the data is unknown and the model is then used to find patterns within the dataset. In supervised learning the correct classification is known by the designer of the model and the task is then to process the large amount of data into those classifications. A problem that combines labeled and unlabeled data can be described as semi-supervised.

TL problems have also adopted these terms to describe the amount of labeled data contained in the source and target domains, however, their definitions are not unanimously agreed on yet. Supervised TL is less disputed, *Daume* et al. 2007 (21) and *Chattopadhyay* et al. 2011 (22), among others, consider it as a problem with abundant labeled source data and limited but some labeled target data. An example is a feature-based TL approach from *Pan* et al. 2010 (23) that looks to find new feature representation for source and target domain to resolve the marginal distribution differences. They also define semi-supervised TL as containing abundant labeled source data and no labeled target data where others, *Gong* et al. 2012 (24) and *Blitzer* et al. 2006 (25), would classify some limited labeled target data as semi-supervised and no labeled target data as unsupervised.

Domain and task similarity

TL problems are normally classified as either *Inductive* or *Transductive*. A scenario where the source and target domains match, $D_S = D_T$, but the tasks do not, $T_S \neq T_T$ is referred to as *Inductive TL*. This can occur when the labels spaces or the conditional probability distributions do not match. Depending on whether the source domain contains labeled data or not this can also be considered as multitask learning or self-taught learning, two approaches that are discussed later in this paper. *Transductive TL* is the opposite, the domains differ, $D_S \neq D_T$. but the tasks are the same, $T_S = T_T$. This type of TL problem can also be sub-categorised depending on the reason for the difference, either in the feature space or marginal probability distributions.

The sub-categories can be defined as follows: A *Homogeneous* problem is one where $X_S = X_T$, the source and target domain both contain the same feature space. The TL problem this paper solves is an example of a homogeneous problem. A *Heterogeneous* problem is where $X_S \neq X_T$, the domains do not have the same feature space. In the case of heterogeneous tasks the approach is generally to adapt both domains to a common latent space and then proceeding to solve the problem as if it were a homogeneous problem. In terms of tasks, it is uncommon but still possible to come across the situation where $Y_S \neq Y_T$, i.e., the label space doesn't match. This could happen, for example, if the source domain has a binary classification whereas the target domain contains multiple labels. It is more common, however, to find the situation where the conditional probability distributions are different between the source and target ($P(Y_S|X_S) \neq P(Y_T|X_T)$). This situation occurs when a particular X is used as input for the source and target models and yields a different Y for each and is quite common in practice. There are many ways to address this unbalance in domains. Under-sampling is one example, where the important cases are the minority class in a sample set and the size is too small to be learned appropriately by the model. For example, in an ML application for classifying software defects, if the samples that contain faults were disproportionately smaller (the minority class) than samples that contain correct behaviour then to under-sample the majority class would then allow the model generalise the minority class behaviour.

Common issues

There are a few issues that can arise in TL problems. The most notable is the concept of negative transfer (26), where the information being transferred from the source domain is detrimental to the target learner. In other words, a target learner trained on only the target domain performs better than a target learner that is trained from both the source and target domains.

Another complication is frequency feature bias which causes the marginal distribution to be different between the source and target domain. An example would be between if the feature space of both was made of words from text documents and some words are more common in one document than the other. Context feature bias is a similar issue but is where the conditional distributions are different between the source and target domains. Using the example of the text document again, this represents the situation where a word has a different meaning in the source domain compared to the target domain. A word could have a more positive connotation within a different context.

2.2.3 Approaches

There are a few ways to approach a TL problem and the choice of approach is dependent on the similarity of the source and target domains. This section will give a brief description of each and some examples of current research.

In **Instance-based** TL (27) (28), training samples from the source domain that are most similar to the training samples of the target model are selected. They are then used in conjunction with the target training samples during the models training process. *Chattopadhyay* et al. (22) used this approach when they proposed the model CP-MDA that looks to reduce the conditional probability difference between the source and target domain and the model 2SW-MDA that looks to reduce both the conditional and marginal probability differences. The aim was to build a classifier that predicts labels of unlabeled target data using training samples from multiple source domains combined with a few labeled target samples. They achieve this by building separate classifiers per each source domain and combine the hypotheses generated by each based on similarity measures between the source and target domain. This approach is also known as ensemble learning and is common throughout all variations of statistical modelling. It is the process of building multiple models that are combined with the idea that the average of all perspectives will give the best solution. *Zhong* et al. 2009 (29) map the marginal distribution of target domain and source domain into a common kernel space and uses a sample selection strategy to bring the conditional probabilities of the two domains closer. *Duan* et al. 2009 (30) propose a multiple-source domain adaptation method (Domain Adaptation Machine (DAM)) that learns a target classifier for label prediction of patterns from the target domain by leveraging a set of pre-computed classifiers independently learned with the labeled patterns from multiple source domains. *Gao* et al. 2008 (31) take a similar approach to utilise multiple source domains by creating a locally weighted ensemble framework. The local weighting is determined on the performance of each model for predicting a certain test sample. This framework achieved a 97% accuracy rating, a 24% increase on the base model.

Feature-based TL (32) (24) (28) (33) is achieved by finding the feature representations that translate well from the source to target domain and it is possible to implement supervised or unsupervised feature-based methods depending on the availability of labeled data. Feature-based approaches can be generalised into two different categories where both look to bring the source and target domains closer together and reduce the marginal probability distributions between the two domains. Asymmetric feature transformation transforms one domain to the other and symmetric feature transformation transforms both domains to a common latent feature space. For symmetric feature transformation, an example would be the Spectral Feature Alignment (SFA) algorithm proposed by *Pan* et al. 2010 (23), that looks to construct a common latent space that can bridge the gap between the domain-specific words of the source and target domains. The latent space includes words that are domain-specific and domain-independent and these clusters are used to train sentiment classifiers in the target domain. Some new approaches are looking at the idea of creating an instance/feature-based hybrid approach to isolate the important training samples but also bring the domains closer and therefore increase the set of samples that can be used for the model.

2.2.4 Deep Transfer Learning (DTL)

DTL refers to the application of TL to deep learning algorithms, such as DRL, that utilise NNs. The transfer is usually conducted by tuning the layers of a pre-trained neural network to adapt to the target task. The layers can either be discarded, re-used(known as feature extraction), or fine-tuned to suit the target task without losing the knowledge accumulated by the network (i.e., the value of the weights). Other approaches, such as one/zero-shot learning and multitask learning, have been investigated to handle problems with limited labeled data availability.

Approaches

Off-the-shelf (OTS) is an approach where the entire pre-trained neural network from the source learner is used for the target learner. It is uncommon for a model that is trained for one task to work perfectly for a separate task, so the OTS models are normally used for general related tasks, such as image processing. The OTS models available are normally massive, containing millions of nodes, and have been trained using robust machines on huge datasets. Examples: AlexNet (34), VGGNet (35), GoogLeNet (36), ResNet (37) and DenseNet (38).

In **Feature extraction**, every layer of the pre-trained model is reused except the final fully-connected output layer. The output layer can then be customised to the target task after the reused layers have processed the input target samples. A further

advantage is that this allows the use of a less computationally heavy linear model to be applied to the output meaning it is usually well equipped for target tasks with a small dataset. There are a few popular examples of pre-trained models available to be used for research and experiments. Google's Inception models are some that are used frequently (39). *Razavian* et al. 2014 (40) used a CNN representation called OverFeat that had been trained on the recognition dataset ILSVRC13. They performed feature extraction on the CNN and applied it to different recognition tasks with varying degrees of similarity to the task OverFeat was initially trained for. The results were compared to the state of the art models for these recognition tasks and found the performance to be comparable.

Fine-tuning looks to adjust different layers within a pre-trained network, not just the final layer. The idea is to fine-tune an existing model so it is better suited to the target task. The options available for adjusting the layers are to 'freeze' a pre-trained layer (i.e., to leave it unaltered) or re-train the layer and 'tune' it to the task, but, it is also possible to add extra layers to the network as part of this approach. It is normally prudent to fine-tune these layers but try and maintain the meaning they acquired from their previous training by decreasing the learning rate. An example of fine-tuning in use is from *Kornblith* et al. 2019 (41), who investigated the performance of the ImageNet database as a means for training effective models. More specifically, they determine that if a model performs better on ImageNet then it will perform better on other vision tasks. They compared 16 classification networks on 12 different image classification datasets and discovered that, when the networks were used for fine-tuning, then a good performance on ImageNet generally leads to a good performance across other tasks. As a caveat, they also discovered that on more specific, fine-grained tasks the use of ImageNet for pre-training had little benefit.

There are a few other approaches that have been subject to an increasing amount of research recently. **Domain Adaptation (DA)** (42) (43) (44) (45) (46) is an example that looks to adapt the source domain so it is closer to the target domain. TL and DA often get confused with one another and there are conflicting definitions across the literature, but, this paper will follow the definition that, as per (47), DA represents a change in the input domain but no change in the task, whereas TL can apply to changes in the task as well. DA is normally associated with situations where the marginal probability distributions are different between the source and target domains, $P(X_s) \neq P(X_t)$. It looks to rectify a change in the input domain (i.e., in an image processing application, the input images are from different devices between the source and target tasks). A simplistic example would be a sentiment classifier that determines whether a movi.e., review is negative or positive. The target task could be to classify sentiment again, but for reviews of a separate product.

Among the other approaches is **Domain confusion** (48) (49) that looks to bring the domains closer by 'confusing' the source domain. This is done by introducing another goal to the source model and the effect of this extra training can make the model more suited to the target task. **Multitask Learning** tries to blur the lines between the source and target domain by training the model on several tasks simultaneously. This goes against conventional TL where the target task is not introduced until after the source learner has completed the source task. **One-shot Learning** and **Zero-shot Learning** are approaches that are effective when labeled training samples are scarce in the case of one-shot or non-existent in the case of zero-shot. *Fei-Fei* et al. 2006 (50) investigate one-shot learning by applying a Bayesian implementation to the idea that knowledge obtained from learning completely different task to the target task can be useful in the event of very few training samples for the target task. Zero-shot learning is a more extreme approach, where there are no labeled examples to learn a task. An example of this is in machine translation, where the task is to be able to translate text from one language to another, and there are currently no translation samples that map one language to the other (i.e., there is no labeled target data). It can then be useful to take a machine translation model between other languages for transfer to infer meaning and syntax to the target model.

2.2.5 Weight Initialisation

textbfWeight initialisation is an approach that is common for many deep learning models and can normally be achieved without transfer. It assumes that the models for the source and target tasks share some parameters or prior distribution of hyperparameters. Weights exist in the nodes of a NN and are used to determine the effect that the node has in the network to process the input. In forward-propagation, matrix multiplication occurs between the input value and the weights of one layer, and that value is passed on to the next layer, through an activation function, where the same process occurs again. The weights are updated through back-propagation, which tweaks the values so that future input will yield an output that is closer to the desired result.

Activation Functions

The most common activation functions are *Sigmoid*, *Softmax*, *Tanh* and *ReLU*. Before discussing these functions I will first introduce the concepts of exploding and vanishing gradients. Exploding gradients occur when the error gradient, used in updating the weights of a network, becomes very large and the value of weight overflows to Not a Number (NaN) values. A vanishing gradient is the opposite, the update value is so small that the convergence is very slow and in extreme cases the network fails to

converge at all.

Activation functions, simply, introduce a non-linear property to the network so more complex behaviour can be modelled, otherwise, only linear patterns could be found in the data. The *Sigmoid* activation function maps the inputs to the range [0,1] and the *Tanh* function maps to [-1,1]. For both functions the values saturate at the extremes 0,1 and -1,1 but Tanh is usually preferred due to being zero-centered, which allows the values to be tweak more precisely.

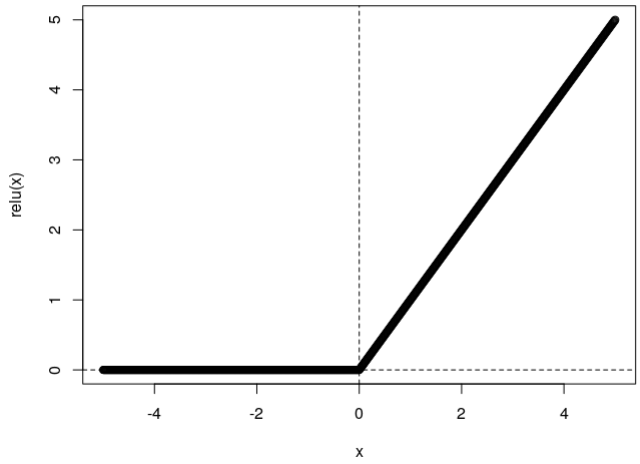


Figure 2.1: ReLU activation function

Rectified Linear Unit or ReLU is a popular function that is less computationally expensive than Tanh and Sigmoid and varies any input above zero effectively. Positive values are mapped in the range [0, infinity] according to the line shown in Fig. 2.1 and any values below zero are set to zero. This is the negative effect of ReLU, as these neurons that become zero can 'die', remaining zero and not affecting the input. It is not uncommon for networks using ReLU to have a large percentage of the network filled with nodes that will not contribute to the network.

Non-transfer approaches

The next step is to choose the weight initialisation method and this can be done without transfer which leads to a more arbitrary approach. Setting the weights too high or low can lead to the same problems of vanishing and exploding gradients. AlexNet (34) took the approach of using Gaussian noise that follows a normal distribution ($\sim$ N(μ=0, σ^2=0.01)) for their initialisation, but this is not effective to very deep NNs with lots of nodes. This problem is compounded with the use of ReLU which leads to cases of vanishing and exploding gradients. The next approach, coined the *Xavier Initialisation* by authors *Glorot and Bengio* (51), provides a solution by taking the size

of the NN into account before initialisation, the weights are inversely proportional to the number of nodes in the network. The next progression from Xavier Initialisation is *Kaiming Initialisation* from *He* et al.(52), which takes the activation function used into account, which is extremely beneficial with the ubiquitous nature of ReLU and its variations.

Transfer approaches

While the non-transfer methods above are useful when the model is being trained from scratch with no access to pre-trained weights, it is a big advantage if a close estimate is available. While weight initialisation can be effective in cases of supervised TL, it is extremely effective when dealing with semi-supervised or unsupervised problems where there are very few labeled target samples. Some algorithms that can be utilised for this approach are sparse coding (a dictionary as parameters), multiple kernel learning and deep learning where the weights of the kernels and neural network respectively are used as parameters *Kumagai* 2019 (53). The aforementioned paper looks at self-taught learning, a concept first introduced by *Raina* et al. 2007 (54), check the performance of sparse coding using that approach. Self-taught learning is defined as TL with unlabeled data for predictive tasks and can be considered as a form of semi-supervised learning. It uses the unlabeled data to build a new representation and then expresses labeled data in this new representation. Normal classification approaches are then used in this space. Ensembled learning, similar to its use in instance-based transfer, is an approach where multiple source learner models are trained on samples specific to their task before they are combined to initialise the target model. The idea is to different 'perspectives' of the knowledge and finding an average to use for the target model that has a similar task to multiple source tasks. Once the weights have been initialised there are different approaches to updating them. This introduces the concepts of hard weight sharing and soft weight sharing. In hard weight sharing, the exact weights are shared and the learning rate is decreased to not unlearn previous knowledge and in soft weight sharing the model is penalised if the weights deviate from the given weights. Regardless of the approach chosen it is generally agreed that it is prudent to start with some related preset weights and fine-tune them instead of training the model from zero. For this paper the model uses the ReLU activation function and the approach is similar to hard weight sharing by using a low learning rate but there is still scope for the microgrid to fine-tune its weights.

2.3 Energy Management

Efficient energy management can offer lots of benefits whether that be within an office building, a home energy system or, on a wider scale across a large grid of connected devices. In all cases, the users benefit from becoming less wasteful with their energy and then less wasteful with their spending. This can then be reallocated to improve productivity and efficiency for a business or government or can be used for savings or to increase the standard of living for a household. There are also unquantifiable benefits, automated temperature settings for example, that improve comfort. In terms of RES, energy management is vital to curb the intermittent nature of its energy production and allow it to become much more efficient and cost-effective.

2.3.1 Renewable Energy Supply (RES)

(55) Renewable energy can be split into two categories: technologies that supply energy (RES) and technologies that make efficient use of the generated energy. The latter includes systems such as smart meters, virtual power plants (VPP), and combined heat and power (CHP). Supply technologies include wind, solar, and hydro energy and each comes with their benefits and drawbacks. Hydropower is very restrictive due to the significant initial investment cost, environmental considerations, and the sunk cost of relocating large populations to construct dams. Wind power (56) (57) and solar power (58) have become more feasible with the recent advancement in their efficiency and subsequent reduction in cost. Where they struggle is the irregularity of their production, they are both dependent on weather conditions which are particularly problematic in more volatile regions. This characteristic introduces the need to combine multiple RES devices of equally varying types and applying techniques to optimise their generation and storage so supply is there to meet demand. This introduces the relatively new concept of a microgrid.

2.3.2 Microgrid

A microgrid is a small scale power system with a cluster of loads and distributed energy resources operating together. Among its contents are energy management, control and protection devices, and associated software and the grid operates in conjunction with the wider utility grid (59). One of the main advantages of a microgrid is that it decentralises the power network (i.e., it can work autonomously and maintain supply to customers if the main grid fails). This is an operation known as 'Islanding'. A microgrid can contain a variety of different renewable energy generators, such as wind turbines, PV panel arrays, and wave generators, and can also include diesel generators. The fluctuating nature of the RES from PV panels and turbines leads to the need to include

some storage capability. There are several challenges in microgrid research from finding the optimal sizing and dispatch strategy to minimising the Levelised Energy Cost (LEC), but these generalise to the encompassing problem of finding an optimal management/scheduling system, possibly with the help of historical data of RES and other microgrids. The microgrid model (1) used in this paper applies DRL to a microgrid to find this optimal management strategy. The model contains PV panels for energy generation, a short term storage (battery), and a long term storage (hydrogen cell). The environment is partly described with a deterministic simulator (where, as much data as necessary is generated) and partly with a limited batch of real stochastic time series (load and production).

2.4 Learning Models in Energy Management

The current research that brings together the topics of machine learning/AI and energy management is varied with a range of domains and techniques used. The space that focuses on the transfer between RES devices specifically is small and that which incorporates microgrids is smaller still. Within that space, there is a mix of RL (60) (61), DRL (62) (63) (1) (64) and Multi-Agent RL (MARL) MAS (65) (66) (67).

Table 2.1: Machine learning research for energy management

Authors	**TL approach**	**Scope**	**Algorithm(s)**
Mosaico (3)	Fine-tuning	Building (occupancy prediction)	AlexNet + SVR
Pardamean (68)	Fine-tuning	Building (occupancy prediction)	Multiple CNN + RHC
Fan (4)	Fine-tuning + Weight init.	Building (consumption forecast)	CNN + LTSM
Banda (69)	Fine-tuning	Building (energy prediction)	Multiple ML models
Mocanu (70)	OTS (no labeled data)	Building (energy prediction)	Q-learning/SARSA + DBN
Hooshmand (71)	Fine-tuning	Energy assets (energy prediction)	CNN (limited data)
Schreiber (72)	-	Wind turbine (production forecast)	CNN (limited data)
Essayeh (5)	-	Microgrid	Q-learning

Both *Mosaico* et al. 2019 (3) and *Fan* et al. 2020 (4) look at building energy management. The former (3) fine-tunes an existing CNN, *AlexNet*, for image processing to predict the occupancy of a building based on thermal camera images. The model learns through a historical database taking a K-Nearest Neighbour (KNN) approach to match the input image with similar historical images. AlexNet (34) is a deep learning NN that was trained on the massive ImageNet database of millions of images and it outperformed all other candidates at the ImageNet Large Scale Visual Recognition Challenge in 2012 (6), also beating Microsoft's deep CNN (73) that won in 2015. The general features of the first few layers are extracted and then the last few layers are configured to the occupancy prediction task. More specifically, the neuron activations of one of the last layers are used as inputs to a Support Vector Regressor (SVR). The last three hidden layers of AlexNet have their features computed, using images with known

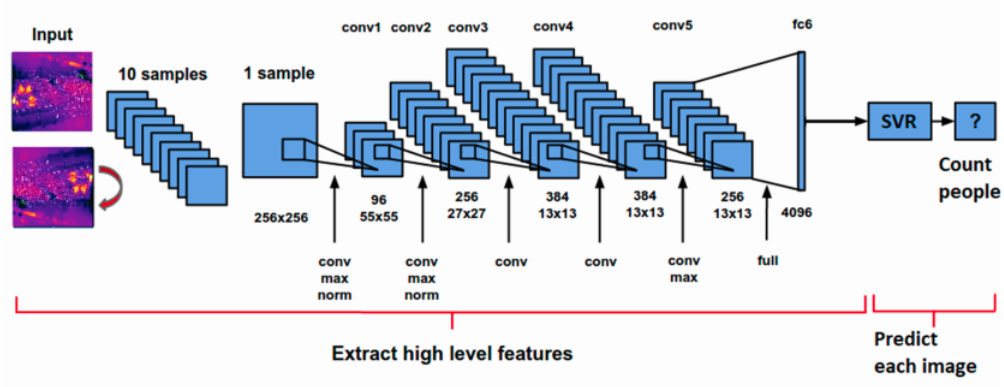


Figure 2.2: AlexNet fine-tuning architecture

values (i.e., known number of occupants), and then these features are used to train the linear SVR. The performance of each is determined by a validation set and the best performing layer is used for the model, *fc6* in the scheme of the proposed algorithm (see Fig. 2.2).

Pardamean et al. 2019 (68) conducted almost identical research to *Mosaico*, but using CCTV footage instead of thermal camera images to detect the occupancy of the building. They used CNNs pre-trained on ImageNet and then fine-tuned them on a dataset crafted for human counting tasks (Room Human Counting (RHC)) by adding fully-connected layers instead of the existing prediction layer from the pre-trained CNNs. The models were then ready to process the CCTV footage. They experimented with multiple pre-trained CNNs (AlexNet, VGGNet, GoogLeNet, ResNet, DenseNet) and found, akin to *Mosaico*, that AlexNet performed the best.

Fan et al. 2020 (4) is a time series prediction of the energy demand of a building for the next 24 hours and the knowledge can be transferred from building-to-building. The model uses historical power consumption and outdoor condition data, weather forecasts, the day/month, and building type and gives a consumption forecast for each of the next 24 hours. For evaluation there are three models used for the target task and two involve transfer. The first model follows the same architecture as the source model and is used as the baseline for comparison. The first transfer approach is feature extraction with each layer of the pre-trained source model used with fixed weights, except for the final output layer which is trained and fine-tuned on the available training samples of the target model. The second transfer approach is weight initialisation with the weights from the pre-trained model set as the weights for the target model and then they are fine-tuned using the same training samples. *Mocanu* et al. 2016 (70) investigated using both Q-learning and SARSA to predict building energy consumption with no labeled source data. Both techniques are built for finite states and performance is naturally hindered as the number of states grows and the Q-tables get larger. A Deep Belief

Network (DBN) is then incorporated into both models to make them suitable for continuous states. A DBN is composed of several Restricted Boltzmann Machines which consist of a visible layer and a hidden layer in its simplest form and defines the joint probability over the hidden and visible layer p($\mathbf{v}$,$\mathbf{h}$):

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}$$

The model was tested on multiple different building types with varying characteristics, which allowed the model to improve while training on each configuration and become increasingly better at adapting to new profiles. The model was initially trained on Commercial, Industrial, and Lighting data and then used to predict four types of residential building consumption. The results also showed a significant improvement in energy prediction accuracy through the addition of the DBN. *Banda* et al. 2019 (69) also investigate the use of TL for building energy prediction with two leisure centres and an office building used as the test environments. First they investigated the performance of five different prediction algorithms: decision trees, random forest (RF), lightGBM, k-nearest neighbour (k-NN), and ensemble extra trees (EET). The best performing models were then used for transfer between the three building profiles and in different cases of available data. The training set size was set from 1% up to 80%. The evaluation metrics were $RMSE$, MAE, MSE, and R^2 and was shown that the models built from transfer performed better than those built from scratch.

The current RL/TL research focusing on RES devices normally uses PV panels, wind turbines, or microgrids with varying goals. Some look to solve the problem of energy consumption forecasting with limited historical data (71) (72) where others try and find the optimal strategy (5), which is the focus of this paper. Others take the approach of MARL, or a multi-agent system, with each agent taking a different role in the microgrid environment. *Hooshmand* et al. 2019 (71) proposed predictive models for energy assets like electricity loads and PV power generations where the amount of historical data is not sufficient to effectively train the predictive model. The model structure was based on VGGNet (35) (a CNN that also won the ImageNet challenge but in 2014) with five convolutional layers, one dense layer with ReLU activation and one linear output layer. They first trained the model on large public datasets that they pre-processed to extract the common features amongst the energy assets. The model was then fine-tuned using the limited training data for the target task. The convolutional layers were frozen, to conserve the features learned from the large dataset, and only the last fully-connected layers were trained. The results showed lower error in the evaluation when compared to three other models. These models were SARIMA (a standard time-series prediction technique that does not require a lot of data), a fresh CNN only trained on the target data with no pre-training and then with the best pre-trained model used directly in an OTS approach. *Schreiber.* 2019 (72) develops a research proposal to address the

problem of limited or no historical data to predict day-ahead forecasts of the expected power generation of a wind turbine after investigating similar work into TL with wind turbines. He investigates how to model 3 scenarios: without historical data, with limited historical data, and then with increasing historical data. He uses DRL methods using NNs to process the non-linear relationship of the numerical weather prediction (NWP) input and historically measured wind power generation that can be used to learn about the turbine's location.

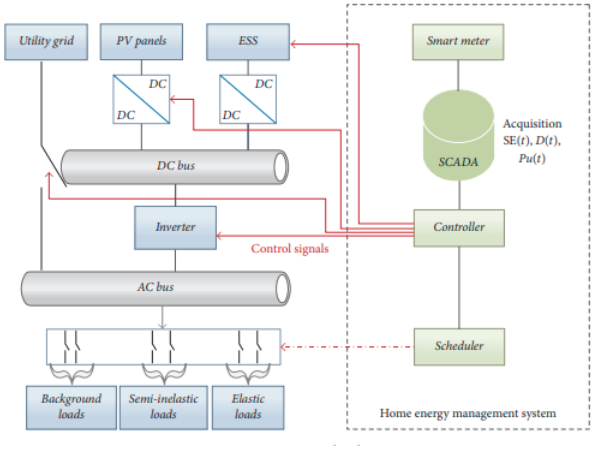


Figure 2.3: Smart microgrid architecture for home energy management system

Essayeh et al. 2018 (5) used a stochastic approach based on a model-free MDP and implemented it using Q-learning to find an optimal strategy for cost-effective energy usage in a home energy management system. The final model is adaptable to be scaled for different profiles of microgrids but this requires testing. The home microgrid architecture (see Fig. 2.3) contains two sources of energy: utility energy and renewable energy resource (RER). The RER is the primary supply and the utility energy behaves as a backup. Any locally generated energy is directly consumed and not sent through the main grid, this allows the microgrid to be flexible and to operate in connected mode or islanding mode. An Energy Storage System (ESS) is implemented to store excess generated energy (which is treated as a non-deterministic value due to the erratic nature of RES) and act as a reserve in the case of insufficient supply for the demand. It also acts as the backup when islanding mode is in operation. The grid also has other additional infrastructure to enable more features. A HEMS infrastructure introduces a smart meter that enables two-way communication to the control centre: sending detailed measurements of electricity consumption and receiving information on utility prices. Also, real data of the energy consumption from a household is collected at the smart meter and transferred to the Supervisory Control and Data Acquisition (SCADA) Centre to be stored and processed. This information can be used to tweak the home energy management system at the centre and SCADA also has an extensive database to offer forecasting.

The MDP used contains three properties $\{S, A, R\}$ as the P representing the transition function i.e., the probability that action a in state s will lead to state s' is not used. This value is dictated on human whim on the energy they use, which fluctuates depending on the time (i.e., summer vs winter), which is very difficult to represent statistically. So a model-free version is used.

$\forall(s,a)\ Q(s,a) = 0.$
$iter \leftarrow 1$
Set ζ and γ ▷ The convergence condition and the learning parameter
Initialize $s(iter = 1)$
repeat
  if $D(iter) < 0$ **then**
    Charge the battery Eq. (2)
  else
    define $\alpha(iter)$, $P_{iter}(s,a)$
    generate random number ϵ
    if $\epsilon \leq P_{iter}(s)$ **then**
      $\pi(s) \longleftarrow \arg\max_a Q(s,a)$
    else $\pi(s) \leftarrow$ random action else than $\arg\max_a Q(s,a)$
    end if
    Execute $\pi(s)$
    Update the state of the battery Eq. (2)
  end if
  Observe $Reward(s, \pi(s), s(iter + 1))$ and Update Q; Equation (5)
  $iter \leftarrow iter + 1$
until $\forall(s,a) : |Q_{updated}(s,a) - Q(s,a)| < \zeta$

Figure 2.4: Q-learning algorithm for microgrid MDP

The state S is defined by three values: Net demand of the prosumer, the amount of energy stored in the battery, the price of the utility energy. The actions depend on the state, the natural decision when the supply is more than the demand is to use the generated energy for demand and to charge the battery (a_0). The next choice is either to (a_1) feed the net demand D with only the utility energy or (a_2) use the battery to feed the net demand (complete with utility is stored energy is not enough). The reward R takes into account the utility minus the cost and the chosen policy (π^*) is one that maximises the reward.

$$\pi* = arg_\pi max E[\sum_t R_t(S(t), a(t), S(t+1))|\pi]$$

The algorithm used is the same as in regular Q-learning, the state-action with the highest Q-value is selected in each iteration and then the reward R is used to update the Q-value (see Fig. 2.4).

The papers discussed in this section give an idea of the current climate of the research that combines learning with energy management. The TL approaches are generally limited to a smaller scope or to solve sub-problems, such as building occupancy, which does not address the main problem of an overall energy management system. The research done by *Essayeh* et al. (5) represents a very similar approach to the paper (1)

that this paper is using as a foundation for the TL approach, as they both look at a microgrid structure albeit with different architectures. The research from *Fan* et al. (4) also provides some inspiration for this paper as they apply weight initialisation as part of their TL approach, despite being in the context of building consumption forecasting. For this research. Therefore, by seeing there was a gap in the literature for a conventional TL approach applied to a microgrid architecture I decided to bring those two areas together in the hope of spurring on more investigation and clarify its effectiveness.

3 Design

In this section, I will first describe the DRL model that provides the basis for the weight initialisation approach. I will look at its architecture and describe the main components before looking at the parameters of the DQN and the steps involved in the training process. I will then lay out the approach to weight initialisation I took and discuss certain design decisions.

3.1 DRL Model

The DTL approach is applied to a DRL model for energy management in a microgrid. This problem can be formalised as a partially observable MDP, where the microgrid takes the role of the agent that interacts with the environment. The microgrid agent contains a short-term storage (STS) battery and a long-term storage (LTS) Hydrogen tank, production profiles from a residential PV panel, and consumption profiles from the same resident. The STS and LTS are represented by a battery energy storage and Hydrogen fuel cells respectively. The use of Hydrogen fuel cells as energy storage is becoming more widespread and has the potential to power many portable devices from cars to smartphones. It is currently integrated in power plants (74) to provide electricity to homes and businesses and also as a backup energy supply for telecommunications companies. Along with its large capacity for long term storage, Hydrogen fuel cells (75) also produce zero emissions and harmless by-products such as electricity, heat, and potable water from the Hydrogen and oxygen. It also has an energy-to-weight ratio ten times better than lithium-ion batteries, allowing a better range while being lighter and able to occupy smaller volumes. However, Hydrogen is expensive and normally requires almost half the energy it contains to isolate it and prevent it from bonding with other elements. Hydrogen also requires water electrolysis to obtain it initially which uses a lot of energy which, can be sourced from renewables, but is usually generated with coal. The characteristics of Hydrogen are defined in the model by the efficiency with which it releases energy and the instantaneous power it can store. The STS battery is also defined by its efficiency and then its maximum capacity.

The agent must choose whether to charge the LTS, discharge it. The agent will be rewarded by filling the LTS or discharging it when needed to help with demand; and will be penalised if the energy in STS and the current production cannot satisfy demand and the LTS has not been used. The LTS is discharged if the agent finds the demand too much for the battery and the battery gets loaded in the event of a surplus of supply. The full DRL solution is defined in the paper (1) as so:

'*In order to approach the Markov property, the state of the system* $s_t \epsilon S$ *is made up of a history of features of observations* $O_t^i, i \epsilon \{1, ..., N_f\}$*, where* $N_f \epsilon N$ *is the total number of features. Each* O_t^i *is represented by a sequence of punctual observations over a chosen history of length* $h^i : O_t^i = [o_{t-h^i+1}^i, ..., o_t^i]$ *(the history length may depend on the feature). At each time step, the agent observes a state variable* s_t *, takes an action* $a_t \epsilon A$ *and moves into a state* $s_{t+1} \sim P(\cdot|s_t, a_t)$*. A reward signal* $rt = \rho(s_t, a_t, s_{t+1})$ *is associated to the transition* (s_t, a_t, s_{t+1})*, where* $\rho : S \times A \times S \rightarrow R$ *is the reward function. We then define the* γ*-discounted optimal Q-value function:*

$$Q^*(s, a) = max_\pi E_{s_{t+1}, s_{t+2}, ...}[\sum_{k=t}^{\infty} \gamma^{k-T} r_k | s_t = s, a_t = a, \pi]$$

We propose to approximate Q^* *using a NN. We denote by* $Q(\cdot, \cdot; \theta_k)$ *the so-called Q-network. NNs offer generalization properties that are adapted to high-dimensional sensory inputs such as temporal series. The NN parameters* θ_k *may be updated using stochastic gradient descent (or other related techniques) by sampling batches of transitions* (s, a, r, s') *in a replay memory, updating the current value* $Q(s, a; \theta_k)$ *towards a target value* $Y_k^Q = r + \gamma$ *arg* $max_{a' \epsilon} Q(s', a'; \theta_k^-)$ *where* θ_k^- *refers to parameters from some previous Q-network called the target Q-network as introduced in Mnih et al. (2015) (11). When using the squared-loss, a Q-learning update is obtained as follows:*

$$\theta_{k+1} = \theta_k + \alpha(Y_k^Q - Q(s, a; \theta_k)) \nabla_{\theta_k} Q(s, a; \theta_k)$$

$\alpha =$ scalar step size called the *learning rate.*

The DRL algorithm is designed so that it is defined by both a deterministic simulator (to generate data) and a limited batch of real stochastic time series (load and production).

The model uses an NN architecture with an output for each discretised action:

- a_0 - discharge LTS
- a_1 - charge LTS

The NN architecture for the Q-Net is built on a large continuous non-handcrafted feature space that uses convolutional layers to extract meaningful features from the time series (see Figure. 3.1). The convolutional layers are 2 convolutions of 16 filters of

2x1 with stride 1 succeeded by a convolution with 16 filters of 2x2 with stride 1. The fully-connected layers that follow contain 50 and 20 neurons respectively and the activation function used is the Rectified Linear Unit (ReLU).

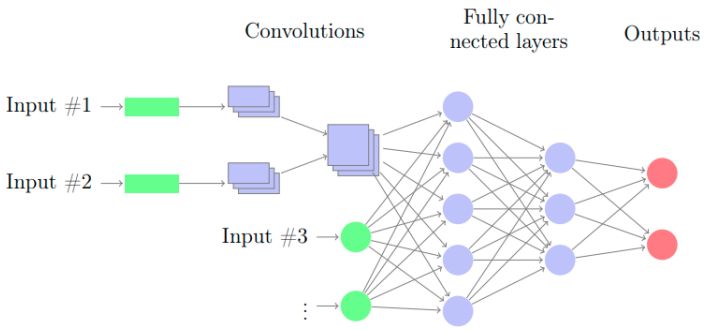


Figure 3.1: The NN architecture.

The output represents the q-value for each action the agent can take (discharge or charge the LTS). The time-series data is split into training sets and also validation and testing sets and the validation strategy periodically evaluates how well the policy performs on the unseen data to prevent overfitting in the agent on the limited training data. The model also looks to minimise errors, such as positive bias when learning imitation of optimal solutions and also associated errors with scenario aggregation in stochastic programming (76). The result of training the model is the selection of the best policy (ie. the best NN), which is determined by the validation and test scores generated at the end of each epoch.

3.2 TL Model

The TL method I used is weight initialisation. Each node in the DRL network has a weight that is used to process the input. The values are updated based on the training samples and are updated at the end of each epoch. It is expected that the more epochs the model is trained for the more accurate the weights become to the task and the more diverse the training samples the more robust the model is to the different states that exist in the environment. The reason weight initialisation was chosen for this research is the fact that the source and target models share the same task and differ in configurations. The assumption behind the approach is that the source and target tasks share parameters. Other approaches of fine-tuning or feature extraction will be better utilised between different microgrid models where the process of determining the similarities between the source and target task is more difficult. Weight initialisation presents itself as the best choice and also introduces the secondary objective of this research which is to determine whether a microgrid with fewer resources can accumulate

more valuable information during training than a microgrid with more resources.

3.2.1 Microgrid Configurations

The weights to be transferred were obtained by training learning models of microgrids with three different configurations of STS and LTS. The values altered were the maximum capacity of the STS, the instantaneous power of the LTS, and the efficiency of both the LTS and STS. The efficiencies are represented by a percentage that dictates the portion of the total energy stored in either the LTS or STS that will be released as it is unlikely the entire amount of energy stored in a battery will be transferred out without loss. The three different configurations were denoted Small, Medium, and Large concerning their respective capacities. The medium configuration is the configuration used in the original paper and is therefore assumed to be more optimised to the environment and also a strong reflection of a standard-sized and resourced microgrid. The values of the parameters for the small and large microgrid configurations were chosen to contain less and more capacity than the medium configuration respectively. The large values were chosen so that the STS capacity was more than the average consumption per day from the data, 18.3 kWh, and more than the maximum power on a given day, 1.7 kW. The small configuration was then selected to have an equal 'distance' from the medium in terms of capacity as the large configuration, to have a significant difference in the results.

Small Microgrid	
Battery Size	8 kWh
Battery Efficiency	90%
Hydrogen Instantaneous Power	0.7 kW
Hydrogen Efficiency	70%

Medium Microgrid	
Battery Size	15 kWh
Battery Efficiency	90%
Hydrogen Instantaneous Power	1.1 kW
Hydrogen Efficiency	65%

Large Microgrid	
Battery Size	22 kWh
Battery Efficiency	85%
Hydrogen Instantaneous Power	1.7 kW
Hydrogen Efficiency	75%

Table 3.1: Properties of the three different microgrid configurations

3.2.2 Training Time

The models were then trained for 30 epochs. This number was chosen due to the time constraint with training for the intended 200 epochs. With the resources at my disposal, 60 epochs would take fifteen hours to complete. Upon further investigation, after 30 epochs the model would show significant improvement from early epochs and a pattern to suggest the effect further training would have and would be complete in

three hours. 30 epochs reduced to 20 epochs for training the models with initialised weights due to the number of models and the fact that 20 epochs showed similar results and trends to 30 epochs and in half the time, 90 minutes. The model runs a certain amount of steps per epoch and for this model the number of steps is equal to the number of hours in a year or around 8759 hours and randomly samples the forecast for a particular time t from that year. The volume of the experience replay enables the agent to call on very diverse samples including different seasons where consumption/production is significantly different (i.e., summer vs winter).

3.2.3 Network Parameters

DQN Parameters	
Learning Rate	0.0002
Learning Rate Decay	0.99
Epsilon Start (min)	1 (0.3)
Epsilon Decay	0.005
Replay Memory Size	1000000
Batch Size	32

Table 3.2: Values of parameters in the Deep Q-network

Table 3.2 displays the values of some of the key parameters of the network used when training the initialised models. The learning rate α is normally set low after initialisation to preserve the information the weights learned from the source model. However, I wanted to make sure there was enough scope for new information to be learned during the training of the target model due to the difference in configuration. To choose the correct value I experimented using a small configuration to medium configuration weight initialisation scenario with the learning rate set to $\alpha = 0.01$, $\alpha = 0.2$ and α=0.0002 and trained the models for 20 epochs. Table 3.3 shows the results of these tests and it is clear that α=0.2 shows markedly worse performance than the lower values of 0.01 or 0.0002. The higher value was causing an effect known as overshooting, this is where the model does not converge as the high learning rate is causing the updates to the weights to be too large. $\alpha = 0.01$ and 0.0002 both show high scores and good improvement, however, I decided to leave the value at 0.0002 due to its better results towards the end of the training.

The learning rate decay value dictates by how much the learning rate will decrease by throughout the training, in this case 99%. Epsilon ϵ determines the probability the agent will opt for the best action for its current state or for a random action to explore the environment and potentially find a better action. The probability the agent chooses the best action is $1 - \epsilon$ and a random action is selected at probability ϵ. The current set up was for the agent to follow an ϵ-greedy policy, which makes exploration a near

	Learning rate @ 0.01	Learning rate @ 0.2	Learning rate @ 0.0002
FIRST epoch id0	-155.55	-963	-141.299
FIRST epoch id1	-197.50	-963	-199.25
LAST epoch id0	-116.83	-963	-63.96
LAST epoch id1	-168.83	-963	-142.74
BEST epoch id0	-96.3	-407	-63.96
BEST epoch id1	-139.54	-474	-142.74

Table 3.3: Varying the learning rate α

certainty at the start of training. ϵ also decays during training to focus on the best policy and take less random actions. The replay memory in this model allows the agent to predict future production and consumption better by replaying previous experiences. In (1) different length sets of historical observations are tested but for this research the value remained with the default amount. The batch size was also tested to determine the value that resulted in a reasonable training time with the computational resources available.

3.2.4 Evaluation Process

The score of each model was calculated in the validation period of training where the model would be tested on unseen training samples and an average would be calculated of all the NNs for a validation score. The best NN would then be selected and tested again to give a test score and the results were saved as a tuple (validation score, test score). The value of the scores was calculated based on the reward the agent earned during that testing period. This occurred at the end of each epoch and then at the end of the full training the epoch with the NN with the best tuple of scores was selected and the weights were saved.

4 Implementation

In this section, I will first explain the programming environment I used to run the model and perform the weight initialisation. This will include the programming language used, the important frameworks and libraries, and then the operating system and processing hardware that was used to run the training process. I will then outline the system architecture of the DRL learning model used from *Francois-Lavet* et al. (1), explaining the different modules and how they interact. I will finish by illustrating the weight initialisation implementation between the different configurations and highlight where in the DRL model process the transfer occurs.

4.1 Programming Environment

The program was run using the programming language Python with the machine learning tools Keras and Tensorflow. Tensorflow is a framework that runs high-level and low-level APIs for a range of machine learning tools and Keras is built on top of Tensorflow and provides high-level APIs for NNs and was used to create the DQN architecture and behaviour. The code for the DRL model from *Francois-Lavet* et al. (1) can be found on Github (77). Running the experiments I used the Google Cloud Platform running a Ubuntu 18.04 LTS instance with 2 n1-standard CPUs of 7.5 GB memory and initially an NVIDIA Tesla P100 GPU. However, after extensive testing altering batch sizes, device assignment, and total epochs, the GPU proved ineffective. The final optimised instance contained just 2 standard CPUs of 5.25 GB memory and would train a model for 30 epochs in three hours and 20 epochs in 90 minutes.

4.2 DRL Model

This section will look at the different components of the DRL model developed by *Francois-Lavet* et al. (1), giving an overview of how they function together in the whole architecture. I will look at each component in some detail and discuss the values and functionality associated with them in this implementation. The main components of

this model are the **Agent**, **Environment**, **Controller** and the **Network architecture** with additional components **Policy** and **Replay Memory** that provide additional functionality. To put it simply the **Agent** is operating within the **Environment** and taking actions following an ϵ-greedy **Policy** and considering previous observations in the **Replay Memory**. The **Controllers** monitor and alter certain values throughout the training and the **Network** dictates how the values of the state-action pairs are calculated to determine the reward or penalty for the agent.

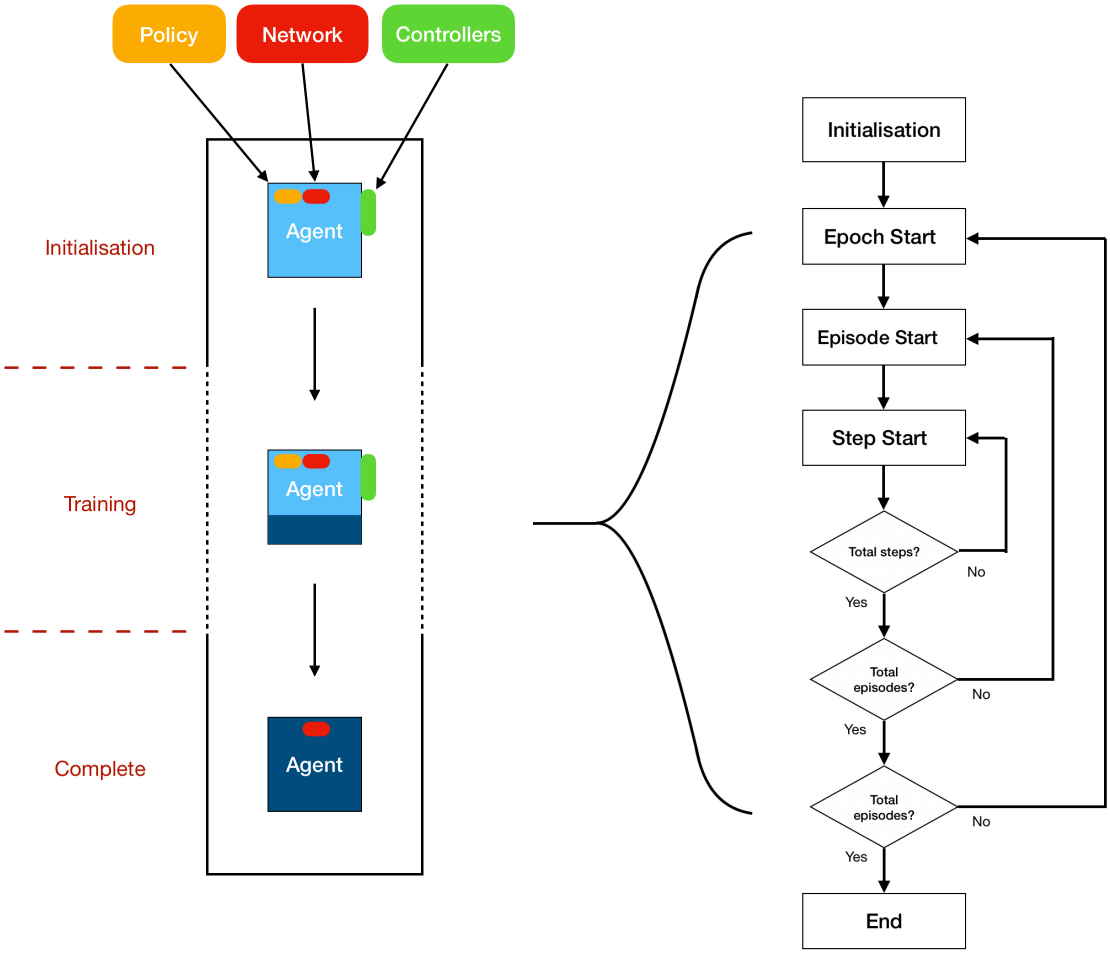


Figure 4.1: Model training process

4.2.1 Agent

The agent component interacts with the given environment and takes actions that yield either a positive reward or a penalty to shape its behaviour. Within the module are classes that define the behaviour of the agent but it also requires the input of other components to define how it behaves in the environment. These include controllers to schedule when the agent trains and when it is tested and also a network component to

approximate the Q-values of each action it can take. The module also contains the functionality to export the weights of the network and also set them at any stage. The parameters the agent module contains are the environment, the learning algorithm (DQN) used, the replay memory size, batch size, and the policy it follows for training and testing. In this module, the instructions to run the entire training process, which is illustrated in Fig. 4.1, are defined. The Policy, Network are initialised with the agent and in the case of the Controllers, attached. The layout of the training process of the agent is represented by the flow chart (right) where each epoch, episode, and step is looped through. A step refers to the agent being trained on one particular set of training samples that include a consumption and production value that the agent has to act on. In this model an episode contains 365*24 steps, therefore each step represents the consumption/production for one hour of the historical data collected, and each episode then cycles through a year of samples. A total of three episodes are run per epoch, therefore a total of three years of samples are processed, including repetitions. The entire training process of the agent is laid out in Algorithm. 1.

Algorithm 1: Agent training process

```
Start with a random Q-Net;
for epochs in totalEpochs do
    for episodes in totalEpisodes do
        Fill up replay memory with all observations, actions and rewards using an
         agent using an ϵ-greedy policy;
        for steps in totalSteps do
            if rand(0:1)<ϵ then
                Random action (with a uniform probability over all actions);
            else
                Best policy π(s)= max_{a∈A} Q(s, a; θ_k) ;
            end
            Train agent and perform update of θ_{k+1};
            Decrease (ϵ);
        end
        Train agent and perform update of θ_{k+1};
        Decrease (ϵ);
    end
    Train agent and perform update of θ_{k+1};
    Decrease the learning rate (α), (ϵ) and increase the discount factor (γ);
    Perform validation sequence and update best NN;
end
```

At the validation and test phases the best policy is always chosen (i.e., $\epsilon = 0$).

Replay Memory

The replay memory (RM) is used in conjunction with the agent and their classes are defined together. The RM module organises all the historical actions, states, and rewards from previous experiences and, depending on the sample size, supplies a random batch of historical observations to be used to train the agent before it takes an action. If the agent is in test mode then the agent does not use the RM and just acts according to the policy.

Policy

The policy module is much shorter than the other components and simply defines the policy the agent takes at each time step when deciding which action to perform. The policy used is ϵ-greedy which selects the best action the majority of the time but takes a random action occasionally. In this model the best action is taken with probability 1-ϵ and a random action taken at ϵ.

4.2.2 Environment

The environment module defines the interaction of the agent for one-time step. The environment is initialised with the production and consumption profile dataset, with a sample pair used for each step the agent takes. The microgrid storage characteristics are also defined here with the STS capacity, the LTS instantaneous power, and both their efficiencies initialised. The dynamics defined in each time step is to process the agents action and calculate the repercussions of that action depending on the consumption and production sample on that step.

4.2.3 Controller

The controllers are attached to the agent and define when particular stages of training occur. This module also processes changes at the end of each episode and epoch, on the action taken and then at the end of the whole training process. There is a controller each for the learning rate (α), exploration probability (ϵ), and the discount factor. The learning rate and exploration probability decrease over time to reduce the value of new knowledge later in the training and to shift the actions of the agent to the best policy respectively. There are a few other controllers that manage various modules in the system. The interleave_test_epoch controller is used to add a test epoch in between training epochs, the trainer controller ensures the agent trains periodically on the dataset and the find_best controller keeps a record of the current best performing NN of the training process.

4.2.4 Network

The network module describes the architecture of the DQN model used to calculate the transition scores for the agent. It uses the Neural Network class from Keras to initialise the replay dimensions, batch size, and the number of actions before the layers of the network, as described in Section 3, are stacked on top of each other. Within the module the states and actions are processed along with their corresponding reward. These come as a batch of training samples, which the network trains on and then updates the q-values. The average loss of each q-value is calculated along with the root-mean-squared-error (RMSE) which are both returned. The RMSE calculates the difference between the prediction of a model and the actual value that should have been predicted. The training processes of the network are called from the agent, which is instructed to begin training at the end of every action it takes.

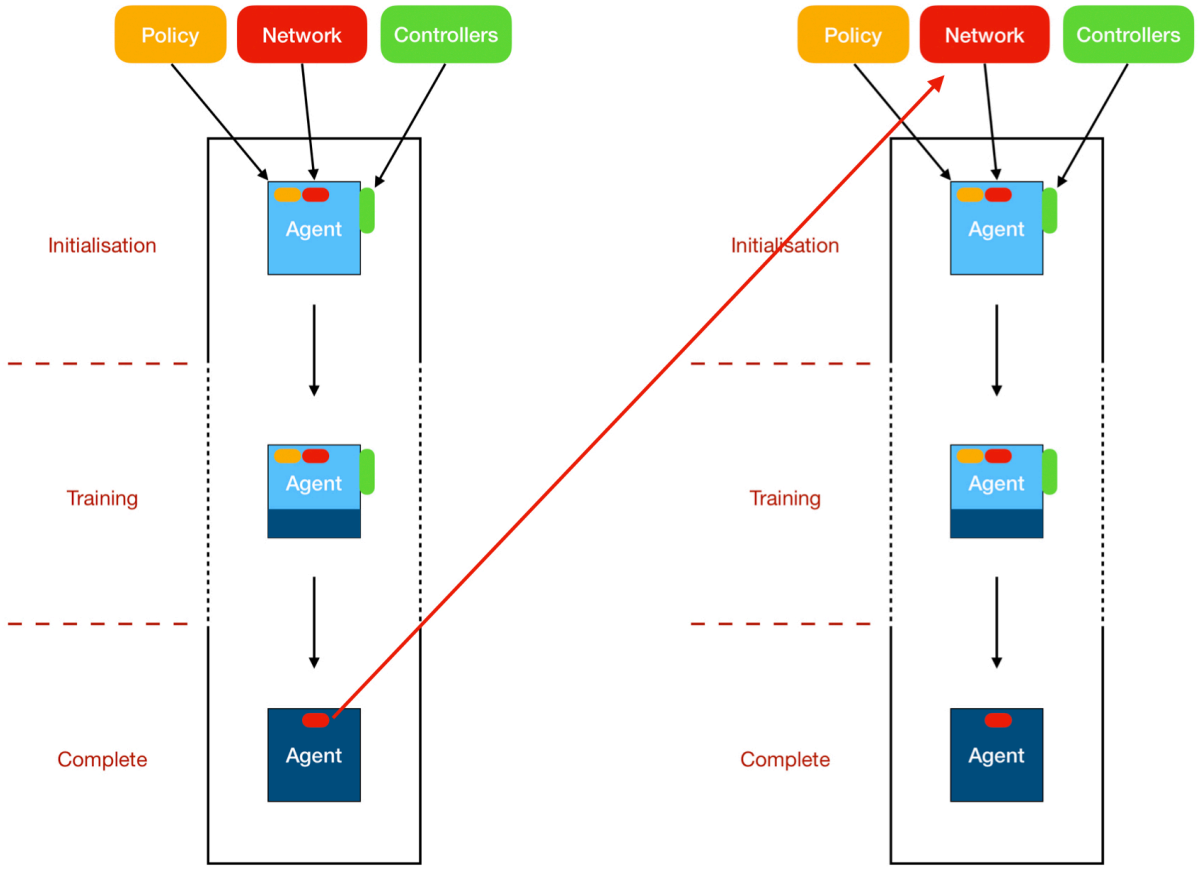


Figure 4.2: Transfer of weights between source and target models

4.3 TL Model

I designed the weight initialisation process to work in conjunction with the DRL model as it has been implemented. The DRL model completes a full round of training and the

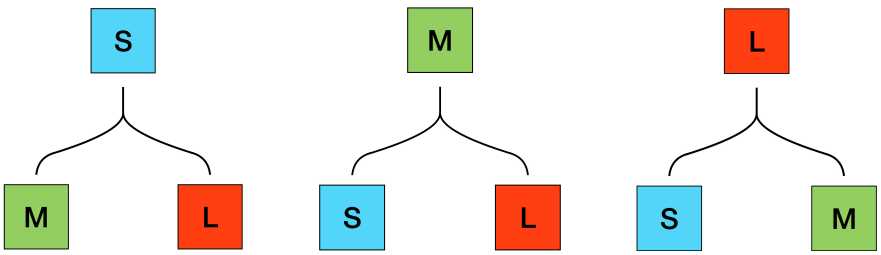
weights are saved for the baseline models and a new model completes a full round of training with initialised weights. The three baseline models (Small, Medium, Large) were trained fully first to produce the weights for transfer and then the target models were trained using those weights. Fig. 4.2 shows the baseline model (left) complete training and then the weights are set into the network at the beginning of training for the target models (right). The entire testing process took 18 hours of training time.

4.3.1 Baseline Models

(a) Small (b) Medium (c) Large

The baseline models were trained first and before any weight initialisation took place. The weights were all initialised to zero to not introduce any bias. The baseline models were differentiated by modifying the STS/LTS values in the environment module where they are initialised, following the values defined in Section 3.2. The models were trained for 30 epochs with 365*24 steps at each episode which led to a training time of 3 hours. Once training concluded the weights of the best performing NN were saved to an external file to be loaded into the target model. The best performing NN is the NN that achieves the highest validation and test score across the whole training process.

4.3.2 Transfer Models



(a) Small (b) Medium (c) Large

Figure 4.4: Weight transfer scenarios

Once the layers of the network had been initialised for the target agent the weights of one of the source baseline models were loaded and set for that network. Fig.4.4 shows the different transfer scenarios that were tested, the weights of the small baseline model was used to initialise a medium and large target model, the weights of the medium

baseline model were used for a small and large target model and finally the weights of the large baseline were used for a small and medium target model. Once the weights were initialised the model was allowed to train as normal and the weights could be altered and tuned regularly throughout the process. These models ran for 20 epochs to reduce the training time by half and still collect substantial results for evaluation and comparison. This part of the implementation took around 9 hours, at around 90 minutes per model.

5 Results and Evaluation

In this section, I will list the objectives of the evaluation and give a brief explanation of each before looking at the metrics I will use for evaluation. I will then outline the different scenarios that will be tested, linking them to the objectives, and discussing the potential outcomes and their significance. Finally, I will present an analysis of the results that will conclude with a summary of the results that looks at the effect of up-scaling vs down-scaling the weights between smaller and large configurations of microgrid.

5.1 Evaluation Objectives

There are two objectives to this research:

1. Show that weight initialisation can be effective to speed up the training of the energy management system of a microgrid.
2. Compare the value of the knowledge obtained from a microgrid with sufficient resources to a microgrid with fewer resources.

The first objective converts the motivation of this paper into an evaluation objective based on the approach and scenarios investigated. The purpose of the paper is to investigate the use of TL in the context of a microgrid energy management system with weight initialisation as the chosen approach. The drawback from this initial aim is the use of only one TL approach and also only one microgrid grid architecture, a more thorough approach spanning multiple microgrids with multiple real-life data collections would provide a conclusive answer and solution but is unfortunately beyond the scope and resources of this research. The second objective stems from the chosen approach also, as a consequential aim but one that will provide interesting insight. Through the use of different configurations of the microgrid the possibility of investigating the behaviour of each becomes available.

1. Effect of Weight Initialisation

The aim is to determine the effect of weight initialisation between different configurations of the microgrid. The process of evaluating this aim is to use the weights of the baseline models to initialise the weights of the target models. Whether or not the effect is a positive one is not critical as either results will give insight into the behaviour of the model and the potential behind this TL approach. However, the desired result is that the scores of the target models with initialised weights will begin with a better performance (i.e., have higher scores than the baseline from the first epoch) and/or achieve a better score after training.

2. Value of Knowledge

A secondary objective, due to the design of multiple configurations of microgrid, from smaller to larger capacity and resources, is to investigate the effect of transferring knowledge from one size of the microgrid to another. The microgrids vary in storage capacity and efficiency but they all have to manage the same consumption and production profiles, therefore the test will be to determine if a microgrid with fewer resources will obtain more useful information about the environment than a microgrid that should have enough reserve to handle demand in most cases. The metrics for this comparison will be if the target models begin with a better score using weights from a smaller or larger microgrid and then also the extent to which they improve during the training. Lots of improvement will suggest that the knowledge obtained was not sufficient, whereas a plateau in improvement suggests that performance has peaked.

5.2 Evaluation Metrics

5.2.1 Validation and Test Scores

The primary metric for comparing each target model and then judging any improvement in the performance of the model over the baseline models is the validation and test scores of each model. The time-series data from the original paper, a production and consumption profile from a residential customer from Belgium, is split into training, validation, and test data, and its scores are determined through its performances when processing the latter two sets. Each is determined in the same way, the difference between the two is that the NN used for the testing stage is determined during the validation stage with the best performing NN chosen. The scores are calculated by taking the average sum of rewards per episode. The reward is calculated proportionally to the difference in the LTS, positively if it is discharged in full and

negatively if it is charged in full, and then readjusted proportionally to the excess or deficit of the demand less the current energy in STS/LTS plus the production for that time step. This metric will be used to judge both objectives, to determine if weight initialisation can lead to a better performance, and to analyse the effect of weights from a particular microgrid configuration.

Training Time

A component to consider for evaluating the effectiveness of the weight initialisation is the time saved by transferring the weights versus training the model from scratch with no initialisation. A big factor of the popularity of TL is its potential to speed up the training of a learner model by reusing knowledge from pre-trained models. In short, if the knowledge from the source model can get the target model to have a better performance more quickly then the approach is effective. For the evaluation scenarios investigated in this paper and explained in the next section the comparison of the transfer and baseline models will be using their scores obtained after 20 epochs and therefore the time taken for the training of each will be the same and comparison will be unnecessary. In general, reusing knowledge from a pre-trained model will always lead to less training time to reach a particular performance level if the transfer is effective, thus the calculation of this metric is dependent on the scores for each scenario which will determine whether or not the approach should be considered for future implementation.

5.3 Evaluation Scenarios

The scenarios for the valuation can be defined in three different categories:

1. Small Weight Transfer
2. Medium Weight Transfer
3. Large Weight Transfer

In each scenario, the size description (Small, Medium, Large) refers to the baseline microgrid configuration used and the scenario involves the weights of that configuration being used to initialise the weights of the target model to be trained. Each scenario contains two such processes. The baseline model will initialise the weights of models with a different configuration, as shown in Fig. 4.4.

5.3.1 Small Weight Transfer

In small weight transfer the small baseline model is used to initialise the weights of a medium and a large configuration model. The small microgrid has an STS capacity of 8 kWh and an efficiency of 90% and an LTS of 0.7 kW instantaneous power and efficiency of 70%. Due to its restricted capacity, the small microgrid will struggle to keep up with the consumption of the energy system and therefore will have to become very efficient through its actions. The average consumption per day is 18kWh and maximum power per day required is 1.7 kW, with the STS capacity at 8kWh and LTS power at 0.7kW for the microgrid, it shows that it will have to have enough in STS or be accurate when discharging LTS to satisfy consumption If the agent takes a wrong decision the penalty is compounded by the volume of energy it cannot handle and thus has a bigger impact.

The scores for this configuration are expected to be low due to its limitations, however, the knowledge it accrues could potentially be more valuable than the rest of the configurations due to the fact it is heavily penalised for any mistake. This scenario will be evaluated on both objectives where objective 1 will be determined by the improvement of the medium and large models with respect to their baselines and also if a higher score early on in the training is higher. This scenario will be particularly interesting for objective 2 to see what knowledge is collected under pressure. It is the scenario where the biggest example of up-scaling weight transfer will take place, from small to large, and this can be compared with a smaller up-scale of small to medium transfer.

5.3.2 Medium Weight Transfer

In medium weight transfer the medium baseline model is used to initialise the weights of a small and a large configuration model. The medium microgrid has an STS capacity of 15 kWh and an efficiency of 90% and an LTS of 1.1 kW instantaneous power and efficiency of 65%. As it is the medium configuration used for weight transfer in this scenario it will be a test of how effective transfer will be from the model from the original paper and thus one that has been optimised to the model and the data. It has the configurations to handle the demand on a given day but still have the capability to sustain a high loss if the wrong action is taken.

This scenario will also be evaluated on both objectives. In the case of objective 2, this scenario will be the only model to perform up-scaling and down-scaling. For initialising the weights of the small microgrid the effect of scaling down will be investigated and vice versa for initialising the large microgrid. This will give a direct analysis as both transfer will be with the same weights. Also, in both instances the scaling up or down

will be a shorter 'jump' between configurations than transfer between small and large. For objective 1, the evaluation will also be between the baseline models for large and small and then their target, initialised model scores. It is expected that this configuration will not have as low scores as the small microgrid and not as high as the large microgrid. It is also expected that the improvement in the score over the epochs will be smoother and potentially larger than the other configurations due to its architecture defined as the authors of the DRL model intended.

5.3.3 Large Weight Transfer

In large weight transfer the large baseline model is used to initialise the weights of a small and a medium configuration model. The large microgrid has an STS capacity of 22 kWh and an efficiency of 85% and an LTS of 1.7 kW instantaneous power and efficiency of 75%. This configuration has more than sufficient capacity to cover the average consumption, 18 kWh, with 22 kWh STS capacity and the maximum power per day, 1.7 kW, with an equal 1.7 kW instantaneous power from the LTS. Therefore, in the event of an incorrect action, for example, if the LTS is not discharged and the consumption is not matched by production and the STS is required, then it will most likely have enough in reserve to cope and thus the agent will not be penalised as often as the small and medium-sized microgrids and not to the same degree if it is.

Both objectives again will be used to evaluate this scenario and objective 1 will be evaluated in the same way to the other two scenarios in that the baseline models of the small and medium configuration will be compared to their respective target models. It is expected that the scores will be higher for this model but not necessarily for transfer, for objective 2, the value of the knowledge is expected to be worse than the two other configurations as the agent will not receive the appropriate severity of check for a poorly chosen action

5.4 Results

In this section, I will first analyse the results of the baseline models before looking at the results of the target, initialised models described in the evaluation scenarios. For each model there is a graph of two values: validation score and test score. They are both calculated in the validation environment with the validation scores used to determine the best NN for that epoch and the test scores selected as the best NN. When discussing the results of each model I will refer to the two scores as a tuple (validation score, test score). To compare the models I will present the differences between the initialised model and its corresponding baseline model after 20 epochs with a column to show the difference where a positive value denotes a higher score achieved by the transfer and a negative value denotes a worse score achieved by the transfer

5.4.1 Baseline Models

The baseline models were trained before any weight initialisation took place. They were each trained for 30 epochs, taking between 3 hours and 15 minutes and 3 hours and 45 minutes for each model to be trained. The following graphs represent the validation and test scores of the model at the end of each epoch. I will analyse the values of the scores for each and explain their disparity and the behaviour that the graphs show.

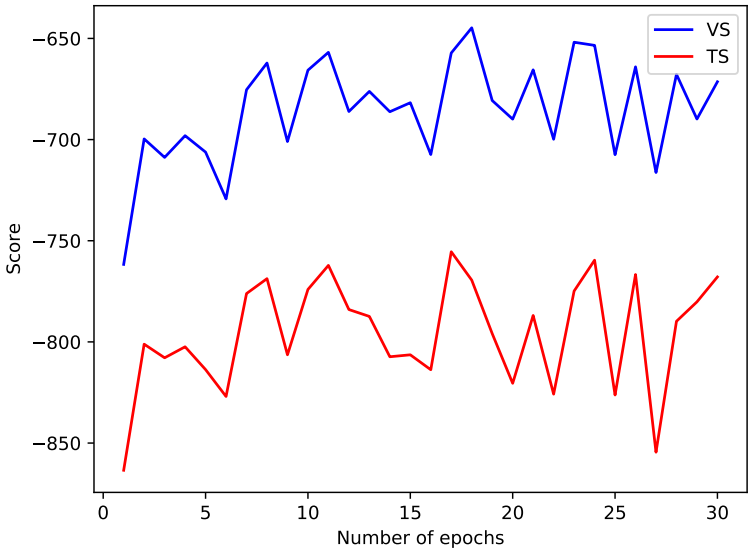


Figure 5.1: Small grid baseline

The validation and test scores for the small configuration begin at (-761, -863) and finish on (-671, -767) with the best score at epoch 18 (-644, -769) and the total time taken for training was 3 hours 15 minutes. The improvement is erratic but better than

the medium configuration improvement with increases of (+90, +96) to the end and (+117,+94) to the best score and the trend suggests that the model could continue to improve. Overall the values are much lower than either the medium (∼-140) or the large (∼ 200) configurations. This is because this configuration has much less resource capacity to cope with the consumption at each time step and in some cases may not be able to handle the demand at a particular step even when taking the correct action is taken. This keeps the score low throughout the training as the reward will be negative the majority of the time. This also explains the extreme fluctuation in the scores as each decision has a larger impact on this model than the other models.

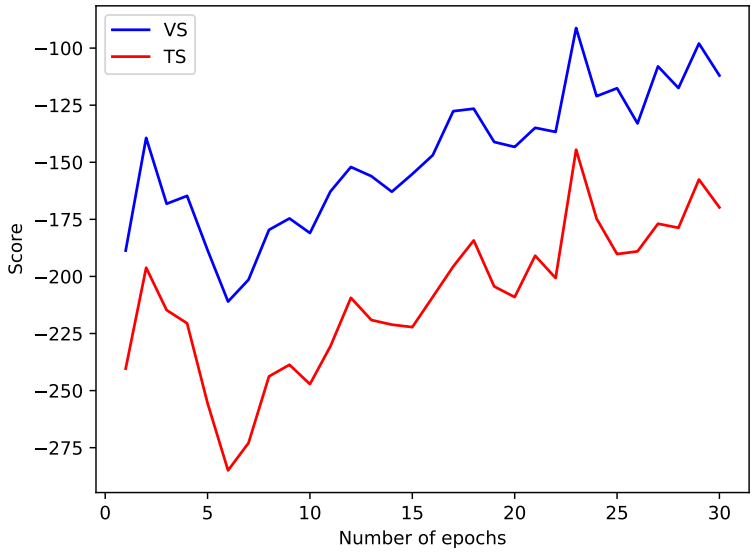


Figure 5.2: Medium grid baseline

The baseline model for the medium configuration trained for a total of 3 hours and 35 minutes and began with the scores (-188, -240) and finished on (-112, -169) with the best score at epoch 23 (-91, -144). This shows an improvement of (+76, +71) to the final epoch and (+97, +96) to the best score. It shows a high initial fluctuation as it encounters different scenarios before settling and steadily rising to a high score relative to its starting point. The results show a consistent improvement as it is trained and a trend to suggest it will keep on improving for more epochs. The score is much better than the small microgrid as the medium microgrid has more capacity to deal with the demand but still experiences drops and negative values because it can lack for some samples.

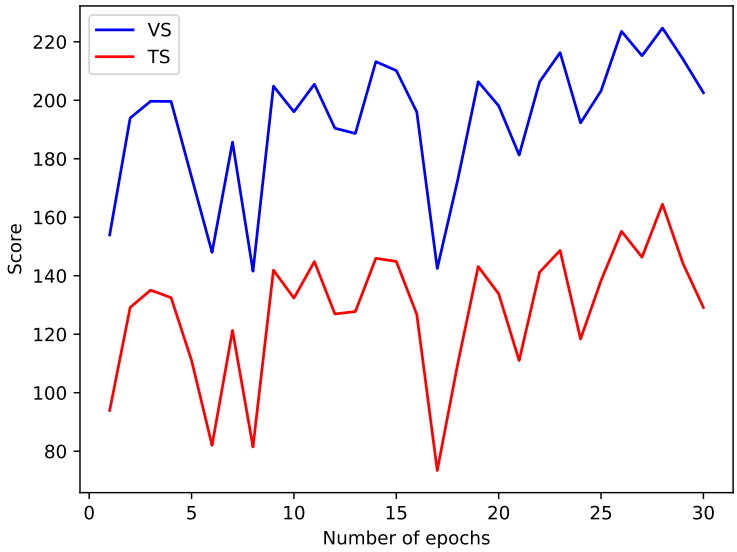


Figure 5.3: Large grid baseline

The large baseline shows much higher scores than either the small or medium microgrid, as the only model with positive scores. The model begins with the scores (154, 94) and finishes on (202, 129) with the best score at epoch 28 (225, 164) with the training time at 3 hours and 45 minutes. This shows an increase of (+48, +35) between the first and last epoch and an increase of (+71, +70) between the first and best epoch, which is very close to the last epoch. The improvement is less than the other configurations and the trend is also a reasonable rise from its early scores but the fluctuation throughout the training is extreme. The erratic nature can be attributed to the 'shock' to the model when it encounters unfamiliar samples and is not equipped to choose the best policy as it has not been pushed to learn the correct approach in other samples. This is because it has more than enough resources to cope with the demand in the day and therefore will not make as many mistakes as other models and will not learn as quickly. The reasonable improvement suggests that it learns through the training but not to the degree of the medium configuration.

Baseline Models Summary

Each baseline model has its unique characteristics from analysis of the results which has the promise to offer refreshing insight for each scenario. Overall the models for each configuration of microgrid show that the validation score does increase as the model is trained, but with a more steady and larger improvement for the medium configuration that the model is optimised for. The small and large configurations show more volatile results but stark differences in values. The volatility of the small configuration is in line with the results obtained from *Francois-Lavet* (1), who found that smaller microgrids

are more challenging due to the larger impact that small decisions have on the microgrid. (i.e., a decision to not discharge the hydrogen when demand is high could put a huge demand on the battery that cannot be met). The large microgrid has high, positive scores due to the increased energy it can store which is more equipped to handle the demand than the medium and small configurations.

5.4.2 Initialised Models

In this section, the scores of the initialised transfer models will be compared to the baseline of those models for the same number of epochs, 20, for a total training time of between 1 hour and 15 minutes and 1 hour and 40 minutes. Each section will contain a graph representing the scores of the transfer model over the training process and a table that will contain scores of the transfer and baseline models for comparison, these will be in the form explain of a tuple (validation score, test score). This table will contain the scores of the first and last epochs of the transfer and baseline model and the individual improvement of the model over the training process. There will be another column where the scores of the first and last models are compared and the difference calculated and represented as a tuple as well (transfer_vs - baseline_vs, transfer_ts - baseline_ts).

Small weight transfer (Small to Medium)

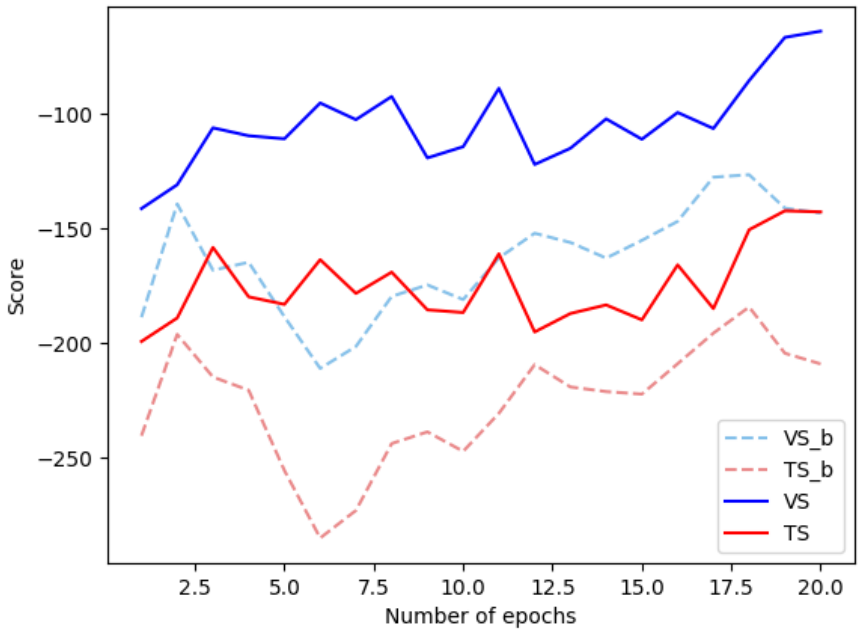


Figure 5.4: Small to medium

Epoch	Transfer - 20 Epochs	Baseline - 20 Epochs	Comparison
First	(-141, -199)	(-188, -240)	**(+47, +41)**
Last	(-64, -143)	(-143, -209)	**(+79, +66)**
Improvement	**(+77, +56)**	**(+45, +31)**	

Both the transfer and the baseline models improve over the training process, however, the transfer model shows a 43% greater individual improvement. This is reflected in a comparison of the individual improvements and also the comparison between the scores of the last epoch for both models. The transfer model also begins at a much higher score than the baseline, a difference of (+47, +41) suggesting that the knowledge contained in the weights of the small is valuable to the medium model. The greater improvement suggests that the further training the model receives, the more diverse knowledge it accumulates and it is then more flexible to achieve a high score with different samples.

The medium model showed consistent improvement for the baseline model so it is expected that a model with initialised weights should follow a similar pattern. The fact, however, that the model has a greater improvement shows the effectiveness of weight transfer and also suggests that the value of a pre-trained smaller microgrid has information that is useful to a larger one. This scenario gives significant credence to the value of TL through weight initialisation as the transfer has immediately set the model on a different level.

Small weight transfer (Small to Large)

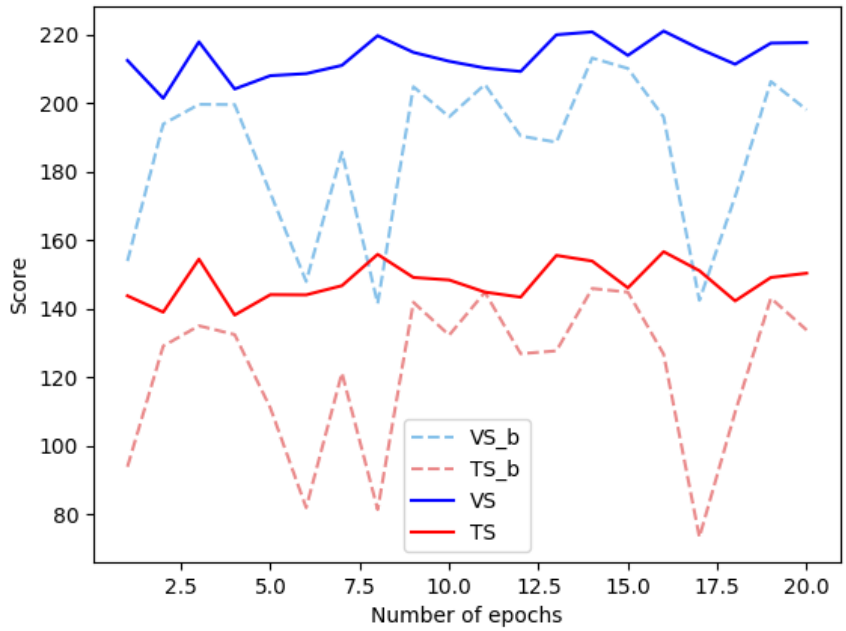


Figure 5.5: Small to large

Epoch	Transfer - 20 Epochs	Baseline - 20 Epochs	Comparison
First	(212,144)	(154, 94)	**(+58, +50)**
Last	(218,150)	(198, 134)	**(+20, +16)**
Improvement	**(+6, +6)**	**(+45, +40)**	

The small to large transfer model and baseline model for the large configuration both have very different individual improvements. The transfer model improves very little during the 20 epochs of training where the baseline model undergoes quite a substantial improvement, similar to the baseline model for the medium configuration. The comparison between the transfer and the baseline model shows that the transfer model has much higher starting scores but the lack of improvement means the baseline model closes that gap by the end of the training. However, the transfer model still displays a better performance from the beginning that the baseline does not reach at all.

The behaviour of the transfer model by not showing improvement throughout training could suggest that the knowledge acquired from the weights of the small microgrid enables the large configuration to take the correct decision for each sample and has now maximised the potential reward it can achieve. This is backed up by the fact that the baseline trained for a further 10 epochs and was only able to match (225, 164) the score of the transfer model in that time before it plateaued in a similar fashion.

Medium weight transfer (Medium to Small)

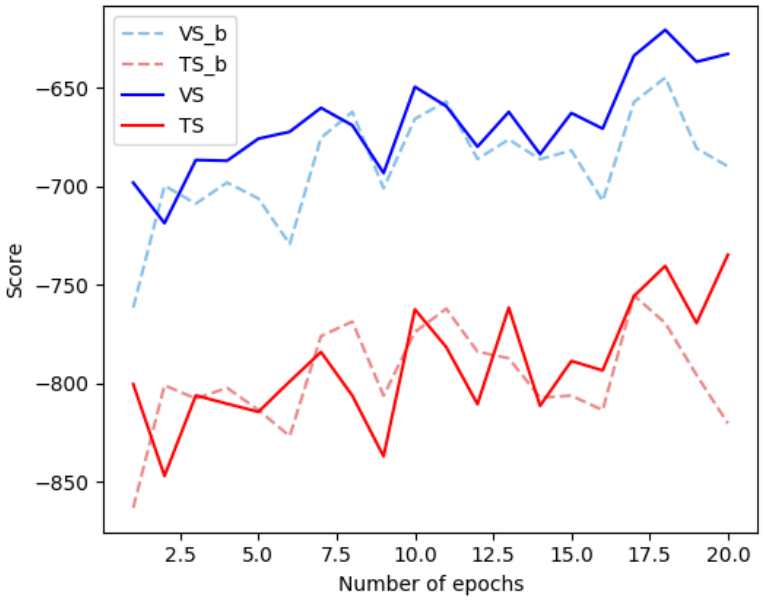


Figure 5.6: Medium to small

Epoch	Transfer - 20 Epochs	Baseline - 20 Epochs	Comparison
First	(-696, -801)	(-761, -863)	**(+65, +62)**
Last	(-633, -735)	(-690, -820)	**(+57, +85)**
Improvement	**(+63, +66)**	**(+71, +43)**	

The medium to small transfer model and small baseline model both show the same levels of improvement over the training period. The transfer model has an initial increase in score through the initialisation and due to the similar levels of improvement over the training that difference is maintained throughout. This is interesting as it shows that down-scaling transfer is just as effective as up-scaling, although the improvement through training is not enhanced. This suggests that the knowledge from the medium microgrid is valuable but the knowledge the small microgrid obtains through training is still novel. Another explanation is that the relative distances in there are configuration is different from small to large for example and the knowledge does not have such a dramatic effect on performance.

Medium weight transfer (Medium to Large)

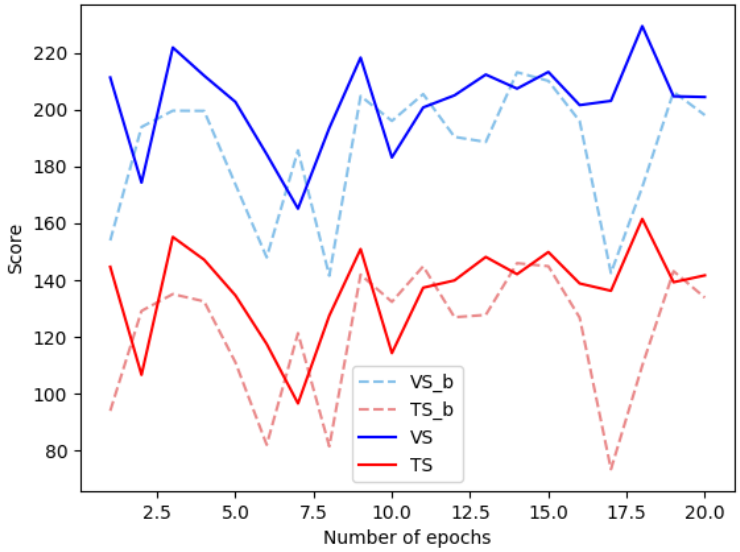


Figure 5.7: Medium to large

Epoch	Transfer - 20 Epochs	Baseline - 20 Epochs	Comparison
First	(211,144)	(154, 94)	**(+57, +50)**
Last	(204,142)	(198, 134)	**(+6, +8)**
Improvement	**(-7, -2)**	**(+45, +40)**	

The medium to large transfer model and large baseline model both show markedly different levels of improvement throughout training. The baseline model shows standard improvement as has been explained earlier but the transfer model regresses and leads to a negative change in the scores being obtained. The scores decrease significantly between epochs 2 and 10 with a high fluctuation. This suggests that the model had to go through some readjustment to update the weights to be more tuned to the correct behaviour of its configuration. The final negative score is only slightly less (-7, -2) as the model has trained back up to the original score through each epoch.

The story changes slightly when we look at the comparison as the initialisation does give the model a higher initial score and even after regressing through training the scores are still higher than the scores achieved by the baseline model. This shows that the transfer was beneficial and had a positive effect but required some alteration that led to unnecessary training time spent. However, the behaviour is similar to the small to large transfers where it appeared that the knowledge passed on all the important information and there was not much more reward to maximise.

Large weight transfer (Large to Small)

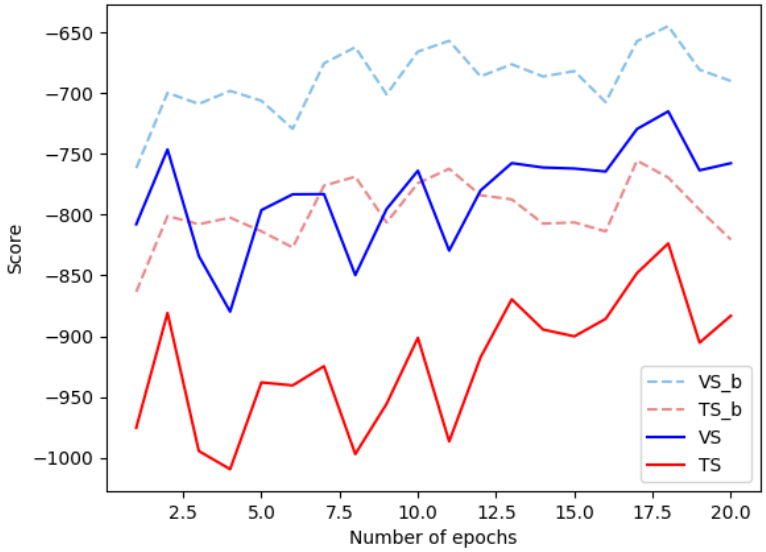


Figure 5.8: Large to small

Epoch	Transfer - 20 Epochs	Baseline - 20 Epochs	Comparison
First	(-808, -975)	(-761, -863)	**(-47, -112)**
Last	(-758, -883)	(-690, -820)	**(-68, -63)**
Improvement	**(+50, +92)**	**(+71, +43)**	

This is the scenario with the biggest down-scale of weight transfer from the largest configuration of the microgrids to the smallest configuration of the microgrids. The individual improvement is comparable between the transfer model and large baseline model, showing that the performance is not affected, and the fluctuation of the scores throughout the training is similar to the behaviour of the baseline model scores. The big difference, however, is the value of the scores specifically. The transfer model begins at (-808, -975) at epoch one for a difference of (-47, -112) and this gap persists throughout training. This effect suggests that the knowledge learned from the larger microgrid is detrimental to the small microgrid and has resulted in negative transfer, where the transfer of knowledge from the source domain has had a more negative effect on the model than if it did not receive any transfer. The model improves gradually but the transfer has only increased the training time needed to get to an equivalent score and has not offered any benefit.

Large weight transfer (Large to Medium)

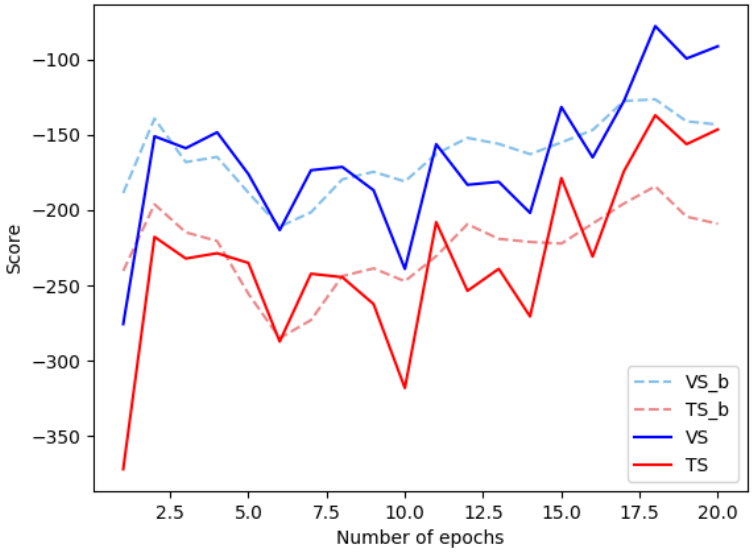


Figure 5.9: Large to medium

Epoch	Transfer - 20 Epochs	Baseline - 20 Epochs	Comparison
First	(-276, -372)	(-188, -240)	**(-88, -132)**
Last	(-91, -146)	(-143, -209)	**(+52, +63)**
Improvement	**(+185, +226)**	**(+45, +31)**	

This scenario of large to medium transfer and comparison with the medium baseline model shows remarkable results compared to previous scenarios. Similar to the large to small transfer scenario the initialisation of weights from the large model has led to a huge initial drop in the scores obtained from the model. However, the individual improvement from the transfer model is far greater than the improvement seen from the baseline to the point where the transfer model surpasses the score of the baseline by the end of the training process, The scores are even comparable, though not quite as strong as the small to medium up-scaling transfer (-64, -143). The scores seem to suggest that by simply initialising the model with weights that bear some relation to the task, data, or model then the model achieves a higher score more quickly.

5.4.3 Summary

These results show that the effect of weight initialisation has been very positive throughout all the scenarios. For each model the initial scores have been much higher than the initial score of the baseline and in the cases of small and medium weight transfer they were higher or on par with the final scores of the baseline even after it had trained for the same period. These are promising results that can be a basis for further research within weight initialisation or another TL technique.

Up-scaling versus Down-scaling

The general pattern is that the models of a smaller configuration transfer more useful information when initialising models of a larger configuration (S to M, S to L, M to L). These tests yield scores that begin at a much higher level and they improve during training to even better scores than the baseline models. This is understandable when considering the sensitivity of a small microgrid to the actions it takes, a wrong decision will have a larger impact so the policy is more strict and the model is forced to learn the optimum policy quickly so the information the small baseline model has obtained is more valuable. The models that use a larger configuration for its initial weight values (L to M, L to S, M to S) do gain some information, the improvement is comparable to other scenarios and even more so in the case of L to M, but the starting scores are much worse compared to the baseline model. This suggests that the knowledge learned by a large microgrid is counter-productive to a smaller resourced microgrid as the penalties it receives for a wrong action in its baseline training are not as large. This is due to the penalties being proportional to the total energy consumption it cannot satisfy which is normally very low. However, the score of the L to M transfer shows final scores after 20 epochs that are better (-91, -146) than the final scores from the baseline medium model after 20 epochs (-143, -209), despite a worse starting point and including a remarkably large improvement during the training. This would infer that there is some useful knowledge from a larger configuration. These outcomes also follow a similar theme that was found in *Francois-Lavet* (1) that showed any additional, pertinent information was useful to the microgrid's performance and the smaller resourced microgrid behaves erratically due to its far smaller capacity to match demand.

6 Conclusion

6.1 Overview

This paper has shown that the TL approach of reusing the weights of a pre-trained model for a new model can be used effectively in the context of different microgrid configurations. The scores show a better performance in the target model when the weights of a model with a smaller configuration is used. When the weights of a larger model are used for initialisation the target model is immediately set back but, particularly in the case of large to medium weight transfer, the improvement is remarkably double the improvement of any other model during their training process. The results have provided a base for future investigation to be performed combining the areas of microgrid energy management and Tl.

6.2 Contribution

This research has contributed further investigation into the use of transfer learning for energy management and optimisation. It has shown that the knowledge learned by an agent whose task is to find the best energy management strategy within a microgrid, can be transferred between different configurations and sizes of microgrids. The benefit of this outcome is that new microgrids can look to have reduced pre-processing time for their management system to be working efficiently. The model processes one year's worth of hourly consumption/production rates per epoch and can show results within two hours, however, to train a model for two hundred epochs like the original research it will require more time. There is still more investigation needed to determine the benefit of transfer between different microgrid architecture or consumption profiles and experimentation with different TL approaches to assess the most effective transfer would be beneficial. The promising results, although, suggest that as microgrids become more ubiquitous in varying sizes and facing different conditions that there is scope to share knowledge effectively between them and can do so in a reasonable amount of time.

Another significant result of this research is the consideration of the value of knowledge

learned from a source model. The differences between the source and target learner models of a microgrid is important to take into account. It was shown that a microgrid with sufficient resources does not learn a sufficient policy for smaller microgrids as it is less sensitive to wrong decisions. This may require a more fine-tuned approach to transferring some knowledge that could potentially be useful.

6.3 Future Work

There are many areas in which this research could be furthered and these can be split into different categories that relate to the specific component that can be expanded.

Non-Transfer Weight Initialisation

A useful comparison with this research would be to investigate the effectiveness of a non-transfer weight initialisation approach compared to the transfer based. The investigation could include techniques such as Xavier and Kaiming initialisation and determine the need for a pre-trained model if the results are comparable.

TL Approach

One variable in this research that could be altered is the TL approach used. A pre-trained model could be fine-tuned to the new microgrid, with the last few layers free for training while the previous layers are frozen. Another approach could be to organise the microgrids to train simultaneously and be exchanging information continuously using an approach known as Parallel TL. *Taylor* et al. (78) conducted successful research in this area focusing on charging electric vehicles. A different multi-agent approach could also be tested for these models, where agents have different roles in a single microgrid (67) (**?**). This could yield a more flexible model that could be scaled for different features and the number of devices.

Consumption/Production Profiles

Future work could also look at varying the production and consumption profiles of the RES on the microgrid. The model used had profiles from Belgium for both winter and summer, but further research could include regions that have different weather characteristics (i.e., less sun exposure, more frequent winds, more dry or humid climate). This would allow analysis into the effectiveness of TL between microgrids that face different environments and therefore different optimal policies.

Microgrid Architecture

Further investigation could also try and transfer the information across domains to different microgrid architectures or even lone RES devices for smaller-scale energy management. This would involve a more nuanced approach to TL to try and understand where the similarities lie and if the transfer is possible to achieve an effective result.

Bibliography

[1] Vincent François-Lavet, David Taralla, Damien Ernst, and Raphael Fonteneau. Deep reinforcement learning solutions for energy microgrids management. page 7.

[2] Denis Dineen. Energy in ireland 2019 report. page 96.

[3] Gabriele Mosaico, Matteo Saviozzi, Federico Silvestro, Andrea Bagnasco, and Andrea Vinci. Simplified state space building energy model and transfer learning based occupancy estimation for HVAC optimal control. In *2019 IEEE 5th International forum on Research and Technology for Society and Industry (RTSI)*, pages 353–358. doi: 10.1109/RTSI.2019.8895544. ISSN: 2687-6809.

[4] Cheng Fan, Yongjun Sun, Fu Xiao, Jie Ma, Dasheng Lee, Jiayuan Wang, and Yen Chieh Tseng. Statistical investigations of transfer learning-based methodology for short-term building energy predictions. 262:114499, . ISSN 0306-2619. doi: 10.1016/j.apenergy.2020.114499. URL `http://www.sciencedirect.com/science/article/pii/S0306261920300118`.

[5] Chaimaa Essayeh, Mohammed Raiss El-Fenni, and Hamza Dahmouni. Cost-effective energy usage in a microgrid using a learning algorithm. URL `https://www.hindawi.com/journals/wcmc/2018/9106430/`.

[6] ImageNet large scale visual recognition competition 2012 (ILSVRC2012), . URL `http://image-net.org/challenges/LSVRC/2012/results.html#abstract`.

[7] Sergey Ivanov and Alexander D'yakonov. Modern deep reinforcement learning algorithms. URL `http://arxiv.org/abs/1906.10025`.

[8] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. A survey and critique of multiagent deep reinforcement learning. 33(6):750–797. ISSN 1573-7454. doi: 10.1007/s10458-019-09421-1. URL `https://doi.org/10.1007/s10458-019-09421-1`.

[9] Enzo Grossi and Massimo Buscema. Introduction to artificial neural networks. 19: 1046–54. doi: 10.1097/MEG.0b013e3282f198a0.

[10] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. 11(3):219–354. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000071. URL http://arxiv.org/abs/1811.12560.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. 518(7540):529–533. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://www.nature.com/articles/nature14236.

[12] Ray Heberer. Why going from implementing q-learning to deep q-learning can be difficult. URL https://towardsdatascience.com/why-going-from-implementing-q-learning-to-deep-q-learning-can-be-difficult-36e7e Library Catalog: towardsdatascience.com.

[13] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. URL http://arxiv.org/abs/1509.06461. version: 3.

[14] Christopher Watkins and Peter Dayan. Technical note: Q-learning. 8:279–292. doi: 10.1007/BF00992698.

[15] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. . URL http://arxiv.org/abs/1901.00137.

[16] Guido Novati and Petros Koumoutsakos. Remember and forget for experience replay. URL http://arxiv.org/abs/1807.05827.

[17] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. URL http://arxiv.org/abs/1707.01495.

[18] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. URL http://arxiv.org/abs/1511.05952.

[19] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. 22(10): 1345–1359. ISSN 1041-4347. doi: 10.1109/TKDE.2009.191. URL http://ieeexplore.ieee.org/document/5288526/.

[20] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. 3(1):9. ISSN 2196-1115. doi: 10.1186/s40537-016-0043-6. URL https://doi.org/10.1186/s40537-016-0043-6.

[21] Hal Daumé III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P07-1033.

[22] Rita Chattopadhyay, Jieping Ye, Sethuraman Panchanathan, Wei Fan, and Ian Davidson. Multisource domain adaptation and its application to early detection of fatigue. volume 6, pages 717–725. doi: 10.1145/2020408.2020520.

[23] Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th international conference on World wide web - WWW '10*, page 751. ACM Press, . ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772767. URL http://portal.acm.org/citation.cfm?doid=1772690.1772767.

[24] Boqing Gong, Yuan Shi, Fei Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2066–2073. IEEE. ISBN 978-1-4673-1228-8 978-1-4673-1226-4 978-1-4673-1227-1. doi: 10.1109/CVPR.2012.6247911. URL http://ieeexplore.ieee.org/document/6247911/.

[25] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, Sydney, Australia, July 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W06-1615.

[26] Zirui Wang, Zihang Dai, Barnabas Poczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11285–11294. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.01155. URL https://ieeexplore.ieee.org/document/8953565/.

[27] Jiayuan Huang, Alexander J Smola, Arthur Gretton, Karsten M Borgwardt, and Bernhard Scholkopf. Correcting sample selection bias by unlabeled data. page 21, .

[28] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. 22(2):199–210, . ISSN 1045-9227, 1941-0093. doi: 10.1109/TNN.2010.2091281. URL http://ieeexplore.ieee.org/document/5640675/.

[29] Erheng Zhong, Wei Fan, Jing Peng, Kun Zhang, Jiangtao Ren, Deepak Turaga, and Olivier Verscheure. Cross domain distribution adaptation via kernel mapping.

In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 1027. ACM Press. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557130. URL http://portal.acm.org/citation.cfm?doid=1557019.1557130.

[30] Lixin Duan, Ivor W. Tsang, Dong Xu, and Tat-Seng Chua. Domain adaptation from multiple sources via auxiliary classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8. ACM Press. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553411. URL http://portal.acm.org/citation.cfm?doid=1553374.1553411.

[31] Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge transfer via multiple model local structure mapping. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, page 283. ACM Press. ISBN 978-1-60558-193-4. doi: 10.1145/1401890.1401928. URL http://dl.acm.org/citation.cfm?doid=1401890.1401928.

[32] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. page 8.

[33] Yuan Shi and Fei Sha. Information-theoretical learning of discriminative clusters for unsupervised domain adaptation. URL http://arxiv.org/abs/1206.6438.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. 60(6):84–90. ISSN 00010782. doi: 10.1145/3065386. URL http://dl.acm.org/citation.cfm?doid=3098997.3065386.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. URL http://arxiv.org/abs/1409.1556.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. URL http://arxiv.org/abs/1409.4842.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. . URL http://arxiv.org/abs/1512.03385.

[38] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. . URL http://arxiv.org/abs/1608.06993.

[39] google/inception, . URL https://github.com/google/inception. original-date: 2015-09-13T03:23:51Z.

[40] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519. IEEE. ISBN 978-1-4799-4308-1. doi: 10.1109/CVPRW.2014.131. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6910029.

[41] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better ImageNet models transfer better? In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2656–2666. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00277. URL https://ieeexplore.ieee.org/document/8954384/.

[42] Judy Hoffman, Trevor Darrell, and Kate Saenko. Continuous manifold based adaptation for evolving visual domains. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 867–874. IEEE. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.116. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909511.

[43] Jun Yang, Rong Yan, and Alexander G. Hauptmann. Adapting SVM classifiers to data with shifted distributions. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, pages 69–76. doi: 10.1109/ICDMW.2007.37. ISSN: 2375-9259.

[44] Yusuf Aytar and Andrew Zisserman. Tabula rasa: Model transfer for object category detection. In *2011 International Conference on Computer Vision*, pages 2252–2259. IEEE. ISBN 978-1-4577-1102-2 978-1-4577-1101-5 978-1-4577-1100-8. doi: 10.1109/ICCV.2011.6126504. URL http://ieeexplore.ieee.org/document/6126504/.

[45] David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, volume 6314, pages 213–226. Springer Berlin Heidelberg. ISBN 978-3-642-15560-4 978-3-642-15561-1. doi: 10.1007/978-3-642-15561-1_16. URL http://link.springer.com/10.1007/978-3-642-15561-1_16.

[46] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *CVPR 2011*, pages

1785–1792. IEEE. ISBN 978-1-4577-0394-2. doi: 10.1109/CVPR.2011.5995702. URL http://ieeexplore.ieee.org/document/5995702/.

[47] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. page 11.

[48] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. URL http://arxiv.org/abs/1511.05547.

[49] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. URL http://arxiv.org/abs/1505.07818.

[50] Li Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. 28(4): 594–611. ISSN 0162-8828. doi: 10.1109/TPAMI.2006.79. URL http://ieeexplore.ieee.org/document/1597116/.

[51] Xavier Glorot and Yoshua Bengio. Understanding the difculty of training deep feedforward neural networks. page 8.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. . URL http://arxiv.org/abs/1502.01852.

[53] Wataru Kumagai and Takafumi Kanamori. Risk bound of transfer learning using parametric feature mapping and its application to sparse coding. 108(11): 1975–2008. ISSN 1573-0565. doi: 10.1007/s10994-019-05805-2. URL https://doi.org/10.1007/s10994-019-05805-2.

[54] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 759–766. ACM Press. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273592. URL http://portal.acm.org/citation.cfm?doid=1273496.1273592.

[55] Shahrouz Abolhosseini, Almas Heshmati, and Jörn Altmann. A review of renewable energy supply and energy efficiency technologies. page 36.

[56] Sandhya Sundararagavan and Erin Baker. Evaluating energy storage technologies for wind power integration. 86(9):2707–2717. ISSN 0038-092X. doi: 10.1016/j.solener.2012.06.013. URL http://www.sciencedirect.com/science/article/pii/S0038092X12002253.

[57] Ida Kubiszewski, Cutler J. Cleveland, and Peter K. Endres. Meta-analysis of net energy return for wind power systems. 35(1):218–225. ISSN 09601481. doi: 10.1016/j.renene.2009.01.012. URL https://linkinghub.elsevier.com/retrieve/pii/S096014810900055X.

[58] Jun-Ki Choi and Vasilis Fthenakis. Economic feasibility of recycling photovoltaic modules. 14(6):947–964. ISSN 1530-9290. doi: 10.1111/j.1530-9290.2010.00289.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1530-9290.2010.00289.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1530-9290.2010.00289.x.

[59] Taha Selim Ustun, Cagil Ozansoy, and Aladin Zayegh. Recent developments in microgrids and example cases around the world—a review. 15(8):4030–4041. ISSN 1364-0321. doi: 10.1016/j.rser.2011.07.033. URL http://www.sciencedirect.com/science/article/pii/S1364032111002735.

[60] Fabiano Pallonetto, Mattia De Rosa, Federico Milano, and Donal P. Finn. Demand response algorithms for smart-grid ready residential buildings using machine learning models. 239:1265–1282. ISSN 0306-2619. doi: 10.1016/j.apenergy.2019.02.020. URL http://www.sciencedirect.com/science/article/pii/S0306261919303101.

[61] José R. Vázquez-Canteli and Zoltán Nagy. Reinforcement learning for demand response: A review of algorithms and modeling techniques. 235:1072–1089. ISSN 0306-2619. doi: 10.1016/j.apenergy.2018.11.002. URL http://www.sciencedirect.com/science/article/pii/S0306261918317082.

[62] Ying Ji, Jianhui Wang, Jiacan Xu, Xiaoke Fang, and Huaguang Zhang. Real-time energy management of a microgrid using deep reinforcement learning. 12(12):2291. doi: 10.3390/en12122291. URL https://www.mdpi.com/1996-1073/12/12/2291.

[63] Liang Yu, Weiwei Xie, Di Xie, Yulong Zou, Dengyin Zhang, Zhixin Sun, Linghua Zhang, Yue Zhang, and Tao Jiang. Deep reinforcement learning for smart home energy management. pages 1–1. ISSN 2327-4662, 2372-2541. doi: 10.1109/JIOT.2019.2957289. URL http://arxiv.org/abs/1909.10165.

[64] Kumar, Hareesh. Explainable AI: Deep reinforcement learning agents for residential demand side cost savings in smart grids. URL https://www.groundai.com/project/explainable-ai-deep-reinforcement-learning-agents-for-residential-demand-side-cos
1.

[65] Soukaina Boudoudouh and Mohamed Maâroufi. Multi agent system solution to microgrid implementation. 39:252–261. ISSN 2210-6707. doi: 10.1016/j.scs.2018.02.020. URL http://www.sciencedirect.com/science/article/pii/S221067071730464X.

[66] C. M. Colson, M. H. Nehrir, and R. W. Gunderson. Multi-agent microgrid power management. 44(1):3678–3683. ISSN 1474-6670. doi: 10.3182/20110828-6-IT-1002.01188. URL http://www.sciencedirect.com/science/article/pii/S1474667016441820.

[67] Roberto Netto, Guilherme Ramalho, Benedito Bonatto, Otavio Carpinteiro, Antonio Zambroni de Souza, Denisson Oliveira, and Rodrigo Braga. Real-time framework for energy management system of a smart microgrid using multiagent systems. 11:656. doi: 10.3390/en11030656.

[68] Bens Pardamean, Hery Harjono Muljo, Tjeng Wawan Cenggoro, Bloomest Jansen Chandra, and Reza Rahutomo. Using transfer learning for smart building management system. 6(1):110. ISSN 2196-1115. doi: 10.1186/s40537-019-0272-6. URL https://doi.org/10.1186/s40537-019-0272-6.

[69] Paul Banda, Muhammed A. Bhuiyan, Kevin Zhang, and Andy Song. Transfer learning for leisure centre energy consumption prediction. In João M. F. Rodrigues, Pedro J. S. Cardoso, Jânio Monteiro, Roberto Lam, Valeria V. Krzhizhanovskaya, Michael H. Lees, Jack J. Dongarra, and Peter M.A. Sloot, editors, *Computational Science – ICCS 2019*, Lecture Notes in Computer Science, pages 112–123. Springer International Publishing. ISBN 978-3-030-22734-0. doi: 10.1007/978-3-030-22734-0_9.

[70] Elena Mocanu, Phuong H. Nguyen, Wil L. Kling, and Madeleine Gibescu. Unsupervised energy prediction in a smart grid context using reinforcement cross-building transfer learning. 116:646–655. ISSN 0378-7788. doi: 10.1016/j.enbuild.2016.01.030. URL http://www.sciencedirect.com/science/article/pii/S0378778816300305.

[71] Ali Hooshmand and Ratnesh Sharma. Energy predictive models with limited data using transfer learning. URL http://arxiv.org/abs/1906.02646.

[72] Jens Schreiber. Transfer learning in the field of renewable energies – a transfer learning framework providing power forecasts throughout the lifecycle of wind farms after initial connection to the electrical grid. URL http://arxiv.org/abs/1906.01168.

[73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, . doi: 10.1109/CVPR.2016.90. ISSN: 1063-6919.

[74] Fuel cells power up: Three surprising places where hydrogen energy is working, . URL `https://www.nationalgeographic.com/news/energy/2014/04/140403-fuel-cells-hydrogen-wal-mart-stationary/`.

[75] Matthew A. Pellow, Christopher J. M. Emmott, Charles J. Barnhart, and Sally M. Benson. Hydrogen or batteries for grid storage? a net energy analysis. 8(7): 1938–1952. doi: 10.1039/C4EE04041D. URL `https://pubs.rsc.org/en/content/articlelanding/2015/ee/c4ee04041d`.

[76] Sirus Mohammadi, Soodabeh Soleymani, and Babak Mozafari. Scenario-based stochastic operation management of MicroGrid including wind, photovoltaic, micro-turbine, fuel cell and energy storage devices. 54:525–535. ISSN 0142-0615. doi: 10.1016/j.ijepes.2013.08.004. URL `http://www.sciencedirect.com/science/article/pii/S0142061513003359`.

[77] VinF. VinF/deer. URL `https://github.com/VinF/deer`. original-date: 2016-01-21T10:29:30Z.

[78] Adam Taylor, Ivana Dusparic, Maxime Guériau, and Siobhán Clarke. Parallel transfer learning in multi-agent systems: What, when and how to transfer? In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. doi: 10.1109/IJCNN.2019.8851784. ISSN: 2161-4407.