Seminar

Machine Learning for Wireless Communications, Winter Term 2018/2019

Regularization and Overfitting

Institute for Digital Communications (IDC)

Shrishti Saha Shetu

February 1, 2019

Abstract

In wireless communication scenario, which involves huge mobile traffic volumes, real-time extraction of fine-grained analytics, and efficient management of network resources; machine learning techniques can be a potential tool to help manage the rise in data volumes and algorithm-driven applications to maximize user experience. The precondition for a machine learning algorithm to be successful is to perform well on testing data. But this condition doesn't get fulfilled because of overfitting, which is a phenomenon where a machine learning model fits the training data too well but fails to perform well on the testing data. Here regularization helps, which is the process of regularizing the machine learning parameters by constraining, regularizing, or shrinking the coefficient estimates towards zero.

In this work, we address both the theoretical and practical aspects of overfitting and regularization with applications to wireless resource management. We first discover the theoretical reasoning of overfitting and how regularization helps and then on implementation side we show the effects of regularization on a practical DNN-based approximation to a popular power allocation algorithm named WMMSE.

1 Introduction

The central challenge in machine learning is that we must perform well on new, previously unseen inputs, not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called generalization [Ian16].

Typically, at the time of training, we can compute some error, measured on the training set called the training error, and we reduce this training error. Reducing training error with some specified constraint is an optimization problem. But machine learning is different from optimization as we want the generalization error, also called the test error, to be minimized.

The factors determining how well a machine learning algorithm will perform are its ability to make the training error small and to make the gap between training and test error small. These factors correspond to the two central challenges in machine learning namely "underfitting" and "overfitting". Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set, while overfitting occurs when the gap between the training error and test error is too large. By altering model's capacity, we can control whether a model is more likely to overfit or underfit. One way to control the capacity of a learning algorithm is to choose its hypothesis space, the set of functions that the learning algorithm is allowed to select from. Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with.

The no free lunch theorem implies that we must design our machine learning algorithms to perform well on a specific task. We do so by including a set of preferences into the learning algorithm. The behavior of our algorithm is strongly affected not just by how large we make the set of functions allowed in its hypothesis space, but by the specific identity of those functions. We can also give a learning algorithm a preference for one solution in its hypothesis space to another. More generally, we can regularize a model that learns a function by adding a penalty called a regularizer to the cost function. Expressing preferences for one function over another is a more general way of controlling a model's capacity than including or excluding members from the hypothesis space. We can think of excluding a function from a hypothesis space as expressing an infinitely strong preference against that function.

The philosophy of deep learning in general is that a very wide range of tasks may all be solved effectively using very general-purpose forms of regularization [Ian16]. And obviously this philosophy does include the solution to our problem of reducing overfitting.

In this work we mainly focus on the theoretical aspects of different kinds of regularization to reduce overfitting within the model and their practical applications related to different field of studies. Later at the end, we will discuss the effect of practical implementation of two common types of regularization techniques namely "L2 Norm regularization" and "Dropout regularization" on a DNN-based approximation to a popular power allocation algorithm named WMMSE.

2 Theory

2.1 Capacity, Overfitting and Underfitting

As we discussed earlier, the ability to perform well on previously unobserved inputs is called generalization. The generalization error is defined as the expected value of the error on a new input. The concept of generalization error leads to the theory of capacity of a machine learning algorithm. Capacity and overfitting are closely related. Overfitting occurs when the learned function $f(x;\theta)$ becomes sensitive to the noise in the dataset. As a result, the function will perform well on the training set but not perform well on other data from the joint probability distribution of input x and output y. Thus, the more overfitting occurs, the larger the generalization error is.

The amount of overfitting can be tested using cross-validation methods, by splitting the data into simulated training data and validation data. The model is then trained on training data and evaluated on the validation data. The testing data is previously unseen by the algorithm and so represents a random sample from the joint probability distribution of input x and output y. This validation data allows us to approximate the expected error and the generalization error.

Informally, model's capacity is its ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set. As already discussed, capacity of a learning algorithm can be controlled by choosing its hypothesis space. For example, the linear regression algorithm has the set of all linear functions of its input as its hypothesis space. We can generalize linear regression to include polynomials, rather than just linear functions, in its hypothesis space. Doing so increases the model's capacity.

In figure 1, we can see a linear, quadratic and degree-9 predictor attempting to fit a problem where the true underlying function is quadratic. The linear function is unable to capture the curvature in the true underlying problem, so it underfits. The degree-9 predictor is capable of representing the correct function, but it also includes infinitely many other functions that pass exactly through the training points. We have little chance of choosing a solution that generalizes well when so many wildly different solutions exist. In this example, the quadratic model is perfectly matched to the true structure of the task so it generalizes well to new data.

Capacity is not determined only by the choice of model. The model specifies which family of functions the learning algorithm can choose from when varying the parameters in order to reduce a training objective. This is called the "representational capacity" of the model. In many cases, finding the best function within this family is a very difficult optimization problem. In practice, the learning algorithm does not actually find the best function, but merely one that significantly reduces the training error. This is called "effective capacity".

Depending on model's capacity, training and test error behave differently. In figure 2, we can observe at the left end of the graph, training error and generalization error are both high. This is the underfitting regime. As the capacity increases, training error decreases, but the gap

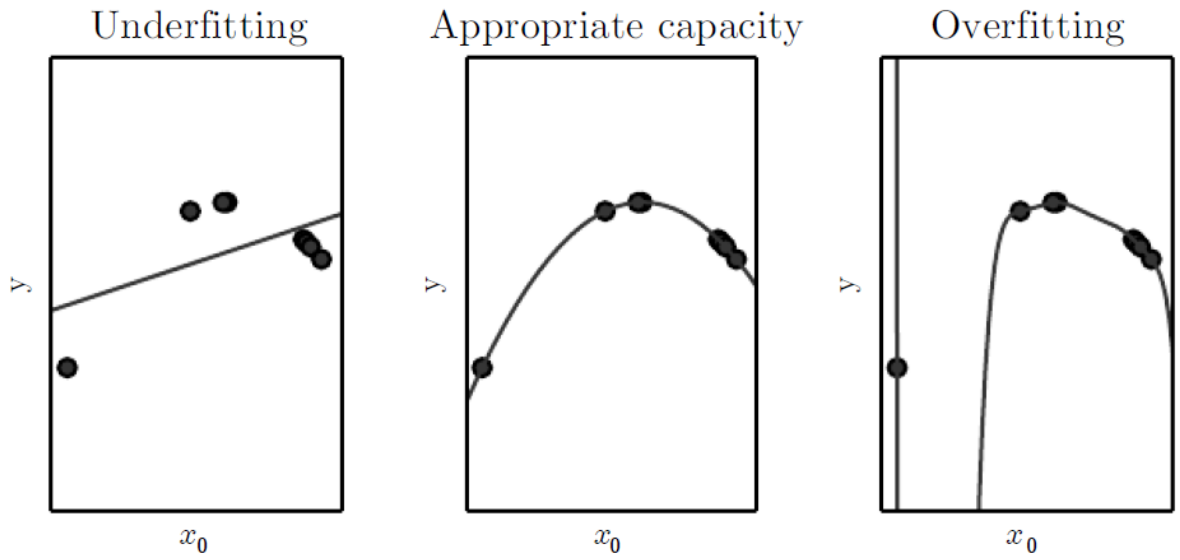


Figure 1: Example for underfitting , overfitting and appropriate capacity [Ian16].

between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and the model enter into the overfitting regime, where capacity is too large, above the optimal capacity.

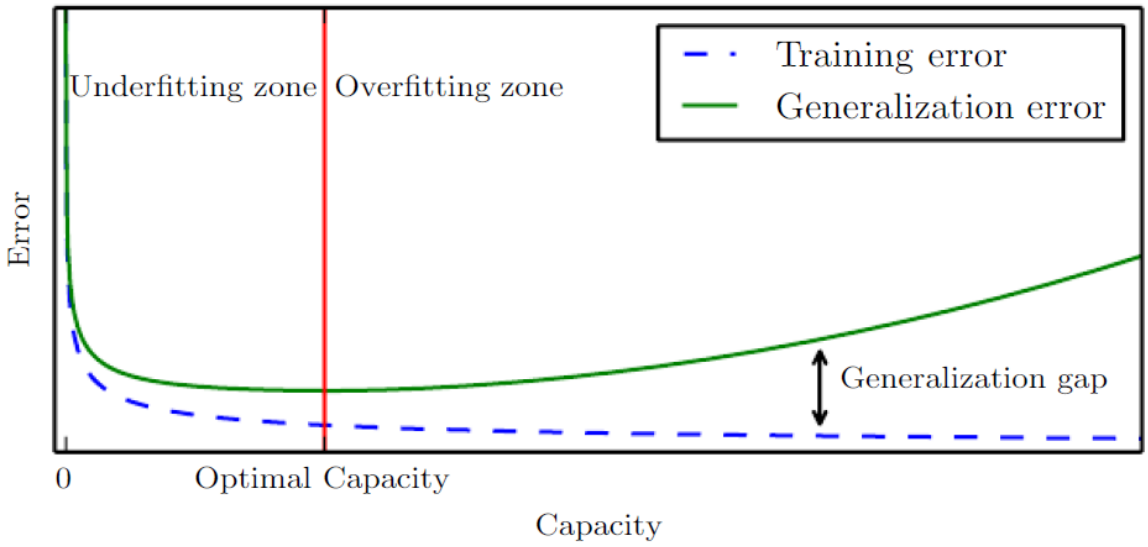


Figure 2: Relationship between model's capacity and error [Ian16].

2.2 Regularization

Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as Regularization [Ian16].

There are many regularization strategies. Some put extra constraints on a machine learning model, such as adding restrictions on the parameter values. Some add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values. If chosen carefully, these extra constraints and penalties can lead to improved performance on the test set.

In the context of deep learning, most regularization strategies are based on regularizing estimators. Regularization of an estimator works by trading increased bias for reduced variance (shown in figure 3). An effective regularizer is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias. The overfitting regime is where variance rather than bias dominates the estimation error. The goal of regularization is to take a model from the overfitting regime to a optimal capacity by minimizing generalization error.

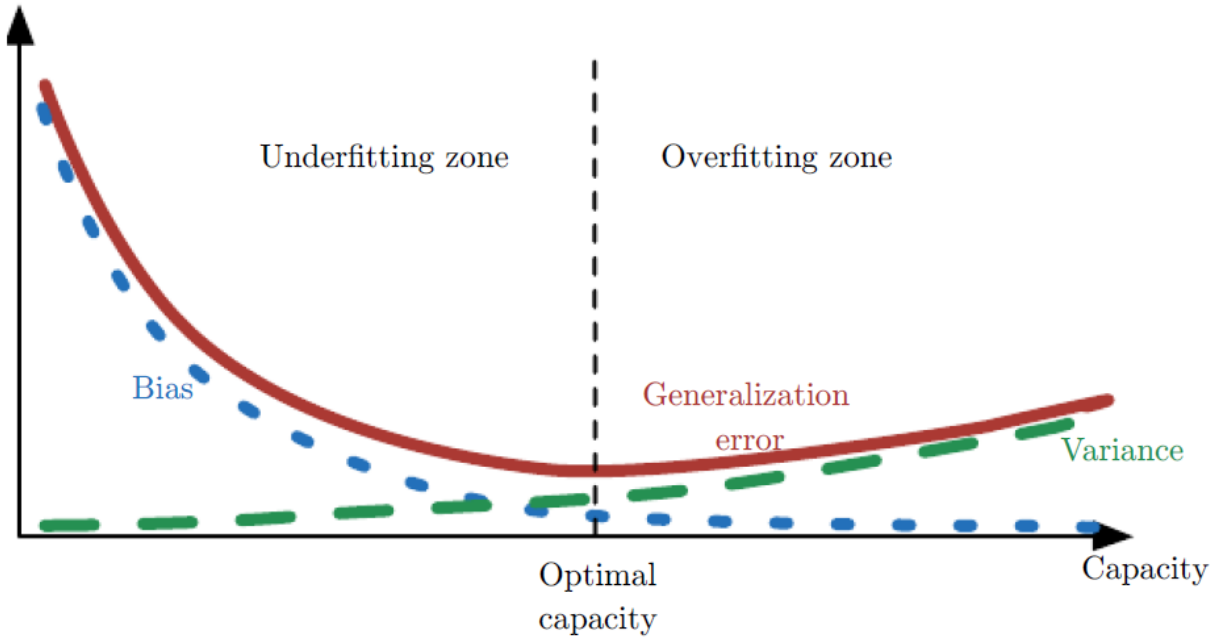


Figure 3: Trading off bias and variance to minimize mean squared error [Ian16].

There are different kinds of regularization techniques. But here we will discuss only four general types of regularization techniques namely Dataset Augmentation, Parameter Norm Penalties, Dropout and Early stopping techniques.

2.2.1 Parameter Norm Penalties

The most traditional form of regularization applicable to deep learning is the concept of parameter norm penalties. This approach limits the capacity of the model by adding the penalty $\Omega(\theta)$ to the objective function resulting in equation (1):

$$\tilde{J}(\omega; X, Y) = J(\omega; X, Y) + \alpha\Omega(\theta) \tag{1}$$

Here $\alpha \in \Big[0, \infty\Big)$ is a hyperparameter that weights the relative contribution of the norm penalty term, $\Omega(\theta)$, relative to the standard objective function J.

When the optimization procedure tries to minimize the objective function, it will also decrease some measure of size of the parameters θ. An important point to be noted is that the bias terms in the affine transformations of deep models usually require less data to be fit and are usually left unregularized. Without loss of generality, we will assume that the goal of regularization is to regularize only the weights ω.

The L2 parameter norm penalty drives ω closer to the origin by adding the regularization term $\frac{\alpha}{2} \|\omega\|_2^2$. The objective cost function for L2 regularization then can be referenced by equation (2) :

$$\tilde{J}(\omega; X, Y) = J(\omega; X, Y) + \frac{\alpha}{2} \|\omega\|_2^2 ; \tag{2}$$

The update rule of gradient decent for all the weights in the neural network using L2 norm penalty can be shown by equation (3):

$$\omega \leftarrow (1 - \epsilon\alpha)\omega - \epsilon\nabla_\omega J(\omega; X, Y) \tag{3}$$

This means the weights multiplicatively shrink by a constant factor at each step. An illustration of the effect of L2 regularization on the value of the optimal weights ω has been shown in figure 4. The blue ellipses represent contours of equal value of the unregularized objective function. The red circles represent contours of equal value of the L2 regularizer. At the point $\tilde{\omega}$, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from ω^*. Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls $\omega 1$ close to zero. In the second dimension, the objective function is very sensitive to movements away from ω^*. The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of $\omega 2$ relatively little.

While L2 weight decay is the most common form of weight decay, there are other ways to penalize the size of the model parameters. Another option is to use L1 regularization.

The L1 parameter norm penalty can be mathematically formulated by equation (4) :

$$\tilde{J}(\omega; X, Y) = J(\omega; X, Y) + \alpha \|\omega\|_1 \tag{4}$$

The corresponding gradient for equation (4) is formulated as followed,

$$\nabla_\omega \tilde{J}(\omega; X, Y) = \nabla_\omega J(\omega; X, Y) + \alpha sign(\omega) \tag{5}$$

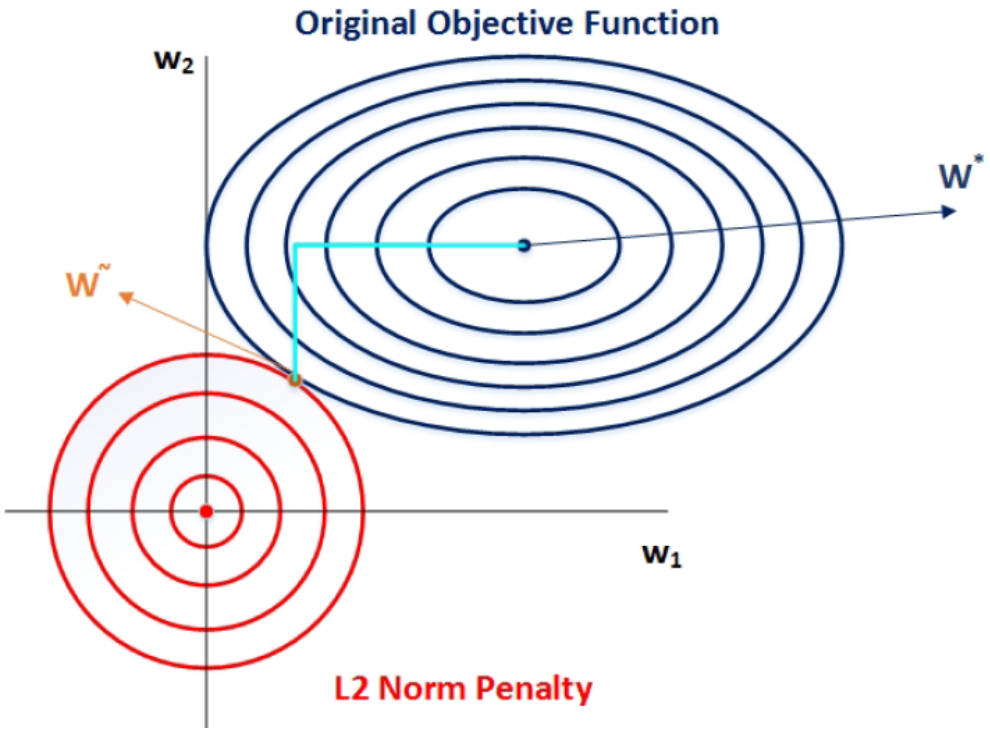


Figure 4: Effect of L2 regularization [Ali11].

By inspecting equation (5), we can see immediately that the effect of L1 regularization is quite different from that of L2 regularization. Specifically, we can see that the regularization contribution to the gradient no longer scales linearly with each ω_i; instead it is a constant factor with a sign equal to $sign(\omega_i)$. One consequence of this form of the gradient is that we will not necessarily see clean algebraic solutions to quadratic approximations of $J(\omega; X, Y)$ as we do for L2 regularization.

In comparison to L2 regularization, L1 regularization results in a solution that is more sparse. Sparsity in this context refers to the fact that some parameters have an optimal value of zero. The sparsity of L1 regularization is a qualitatively different behavior than arises with L2 regularization. This effect can be shown by an illustration in figure 5.

The sparsity property induced by L1 regularization has been used extensively as a feature selection mechanism. Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used. In particular, the well known LASSO (Tibshirani, 1995) model integrates an L1 penalty with a linear model and a least squares cost function. The L1 penalty causes a subset of the weights to become zero, suggesting that the corresponding features may safely be discarded.

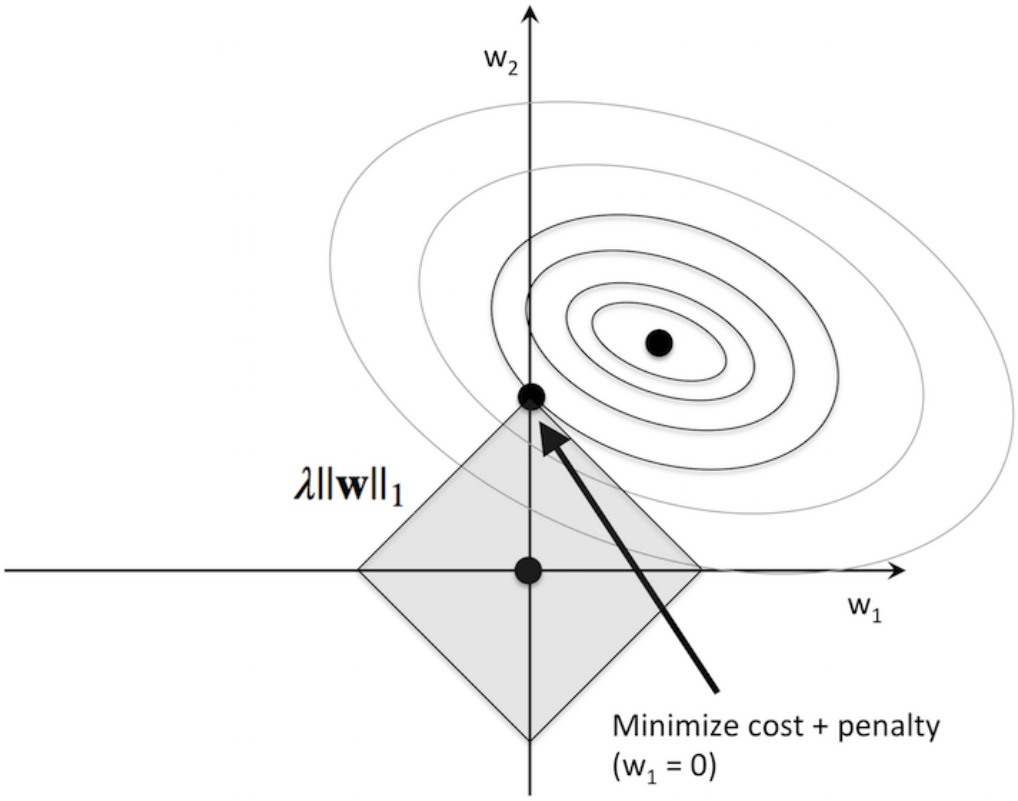


Figure 5: Effect of L1 regularization [Ali11].

2.2.2 Dataset Augmentation

The best way to make a machine learning model generalize better is to train it on more data [Ian16]. Of course, in practice, the amount of data we have is limited. Furthermore, labelling is an extremely tedious task. One way to get around this problem is to create fake data and add it to the training set. For some machine learning tasks, it is reasonably straightforward to create new fake data. Certain tasks such as steering angle regression require dataset augmentation to perform well.

But this approach is not as readily applicable to many other tasks. For example, it is difficult to generate new fake data for a density estimation task unless we have already solved the density estimation problem.

In machine learning tasks like "Image Classifier", color jitter is a very effective method to augment datasets. It is also extremely easy to apply. Fancy PCA was proposed by Krizhevsky in the famous Alex net paper [Alex12]. It is a way to perform color jitter on images.

Horizontal Flipping is another popular way of dataset augmentation, which is applied on data that exhibit horizontal asymmetry. Care must be taken to propagate the labels through this transformation. Horizontal flipping can be applied to natural images and point clouds. Essentially, one can double the amount of data through horizontal flipping. In figure 6, an example for horizontal flipping has been shown. Other basic methods of dataset augmentation

includes rotating, zooming, scaling, cropping, translating(moving along the x or y axis)and adding Gaussian noise operations.

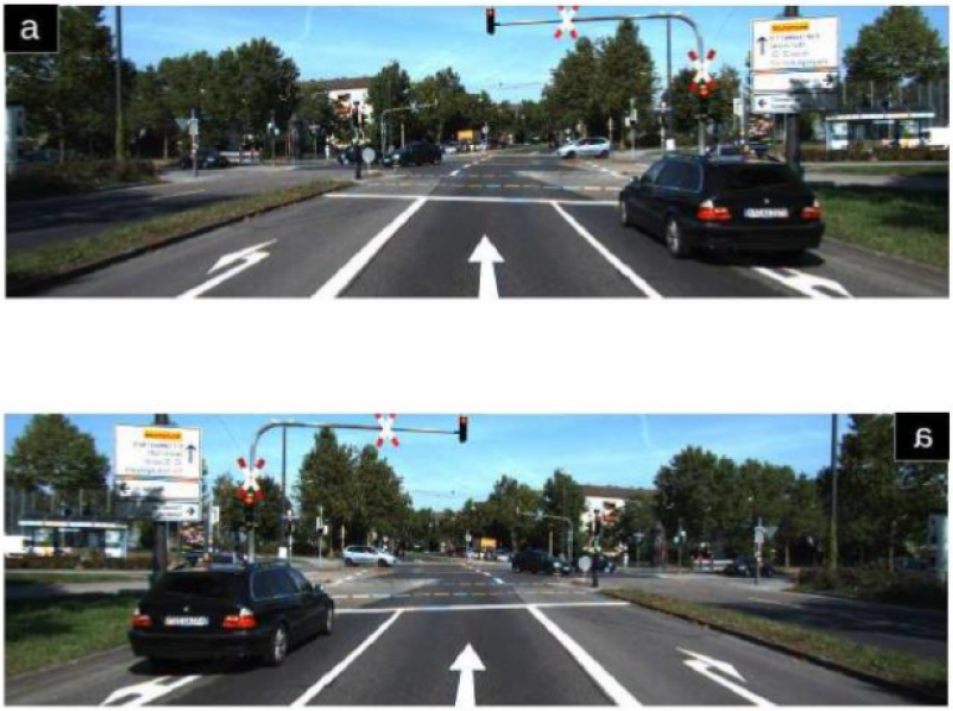


Figure 6: Illustration for horizontal flipping [Ali11].

When comparing machine learning benchmark results, it is important to take the effect of dataset augmentation into account. Often, hand-designed dataset augmentation schemes can dramatically reduce the generalization error of a machine learning technique. To compare the performance of one machine learning algorithm to another, it is necessary to perform controlled experiments.

2.2.3 Early Stopping

While training large models with sufficient representational capacity to fit the task, we often observe that training error decreases steadily over time, but validation error begins to rise again [Ian16]. The occurrence of this behaviour in the scope of our applications is almost certain. Then the challenge is to train the network long enough that it is capable of learning the mapping from inputs to outputs, but not training the model so long that it overfits the training data.

One strategy might be practical in application, to store a copy of the model parameters, while in training phase every time the error on the validation set improves and when the training algorithm terminates, we can return these parameters instead of the latest parameters. The algorithm should terminate when no parameters have improved over the best recorded validation

error for some pre-specified number of iterations. This strategy is termed as early stopping. One visual illustration for early stopping has been shown in figure 6.

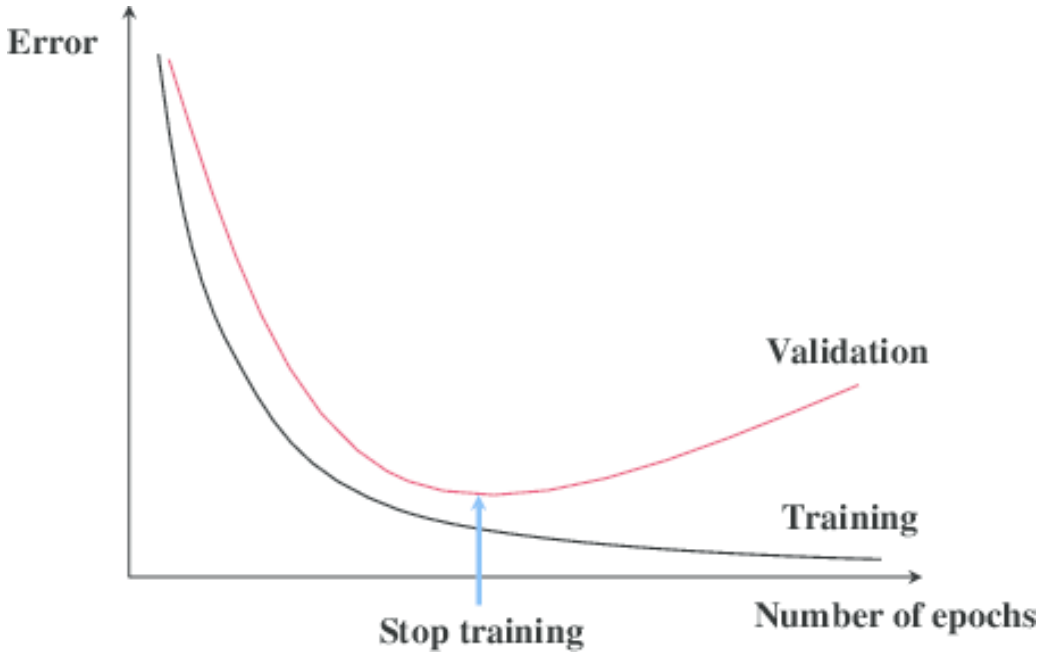


Figure 7: Illustration for early stopping [Ali11].

There are three elements in order to use early stopping; they are: monitoring model performance, trigger to stop training and the choice of model to use [Ali11].

Monitoring the performance of the model is essential. This requires the choice of a dataset that is used to evaluate the model. It is also common to split the training dataset and to use a subset as a validation dataset. Afterwards this validation dataset can be used to monitor the performance of the model during validation. Performance of the model is evaluated on the validation set at the end of each epoch, which adds an additional computational cost during training. This can be reduced by evaluating the model less frequently, such as every 2, 5, or 10 training epochs.

Trigger for stopping the training must be chosen then. The trigger will use a monitored performance metric to decide when to stop training. This is often the performance of the model on the validation dataset, such as the error or loss. In the simplest case, training is stopped as soon as the performance on the validation dataset decreases as compared to the performance on the validation dataset at the prior training epoch (e.g. an increase in error).

More elaborate triggers may be required in practice. This is because the training of a neural network is stochastic and can be noisy. Plotted on a graph, the performance of a model on a validation dataset may go up and down many times. This means that the first sign of overfitting may not be a good place to stop training [Plt98].

Model choice is the last step to implement early stopping regularization. This will depend on the trigger chosen to stop the training process. For example, if the trigger is a simple decrease in

performance from one epoch to the next, then the weights for the model at the prior epoch will be preferred. If the trigger is required to observe a decrease in performance over a fixed number of epochs, then the model at the beginning of the trigger period will be preferred. Perhaps a simple approach is to always save the model weights, if the performance of the model on a holdout dataset is better than at the previous epoch. That way, we will always have the model with the best performance on the holdout dataset.

2.2.4 Dropout

A motivation for dropout comes from a theory of the role of sex in evolution. Sexual reproduction involves taking half the genes of one parent and half of the other, adding a very small amount of random mutation, and combining them to produce an offspring. The asexual alternative is to create an offspring with a slightly mutated copy of the parent's genes [Nit11].

In mathematical approximation, dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks. Bagging involves training multiple models, and evaluating multiple models on each test example. This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.

But instead of bagging method, dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network, as illustrated in figure 8. Here, the base network has two visible units and two hidden units. There can be sixteen possible subsets of these four units. All sixteen subnetworks that may be formed by dropping out different subsets of units from the original network have been shown. In this small example, a large proportion of the resulting networks have no input units or no path connecting the input to the output. This problem becomes insignificant for networks with wider layers, where the probability of dropping all possible paths from inputs to outputs becomes smaller.

To train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent. Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network. The mask for each unit is sampled independently from all of the others. Dropout allows us to represent an exponential number of models with a tractable amount of memory. Furthermore, dropout removes the need to accumulate model votes at the inference stage. It can intuitively be explained as forcing the model to learn with missing input and hidden units.

Computationally dropout is very cheap, using dropout during training requires only $O(n)$ computation per example per update, to generate n random binary numbers and multiply them by the state. Dropout does not significantly limit the type of model or training procedure that can be used. It works well with nearly any model that uses a distributed representation and can be trained with stochastic gradient descent.

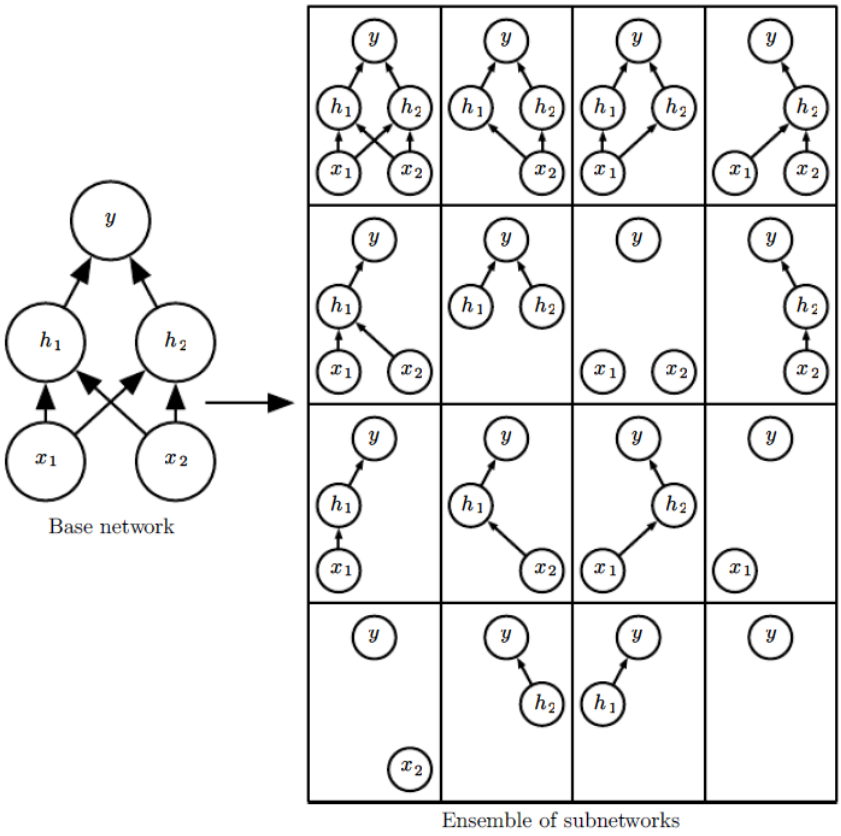


Figure 8: Illustration for dropout [Ian16].

2.3 Implementation Model

For our implementation we used the code from the paper [Hao18] titled "Learning to Optimize: Training Deep Neural Networks for Wireless Resource Management". The main idea was to treat a given resource optimization algorithm as a "black box", and try to learn its input/output relation by using a deep neural network (DNN). The goal was the power allocation for each transmitter so that the weighted system throughput is maximized. The system mathematical problem can be shown by equation (6) :

$$\max_{p_1,\ldots,p_k} \sum_{k=1}^{K} \alpha_k \log(1 + \frac{|h_{kk}|^2 p_k}{\sum_{j\neq k} |h_{kj}|^2 p_j + \sigma_k^2}) \tag{6}$$

$$s.t. \;\; 0 \leq p_k \leq P_{max} \quad \forall k = 1, 2.., K$$

Here P_{max} denotes the max power of each transmitter and $\alpha_k > 0$ are the weights.

Problem (6) is known to be NP-hard. Various power control algorithms have been proposed, among which the WMMSE algorithm has been very popular. The WMMSE algorithm converts the weighted sum-rate (WSR) maximization problem to a higher dimensional space where it is easily solvable, using the well-known MMSE-SINR equality. By following the steps in [Qshi11],

Theorem 1 , it can be shown that problem (6) is equivalent to the modified weighted MSE minimization problem referenced by equation (7) :

$$\min_{(w_k,u_k,v_k)_{k=1}^K} \sum_{k=1}^{K} \alpha_k(w_k e_k - \log(w_k)) \tag{7}$$

$$s.t.\ \ 0 \leq v_k \leq \sqrt{P_k} \quad \forall k = 1; 2..; K;$$

Here, e_k is defined as,$e_k = (u_k|h_{kk}|v_k - 1)^2 + \sum_{j \neq k}(u_k|h_{kj}|v_j)^2 + \sigma_k^2 u_k^2$

Theoretical results from [Hao18] indicate that a large class of algorithms, including WMMSE, can be approximated very well by a neural network. The proposed approach of that work uses a fully connected neural network with one input layer, multiple hidden layers, and one output layer as shown in figure 9. The input of the network is the magnitude of the channel coefficients $|h_{kj}|$, and the output of the network is the power allocation P_k . Further, they use ReLU as the activation function for the hidden layers activation. Additionally, to enforce the power constraint in equation (6) at the output of DNN, they also choose a special activation function for the output layer, given by, $y = \min(\max(x; 0); P_{max})$.

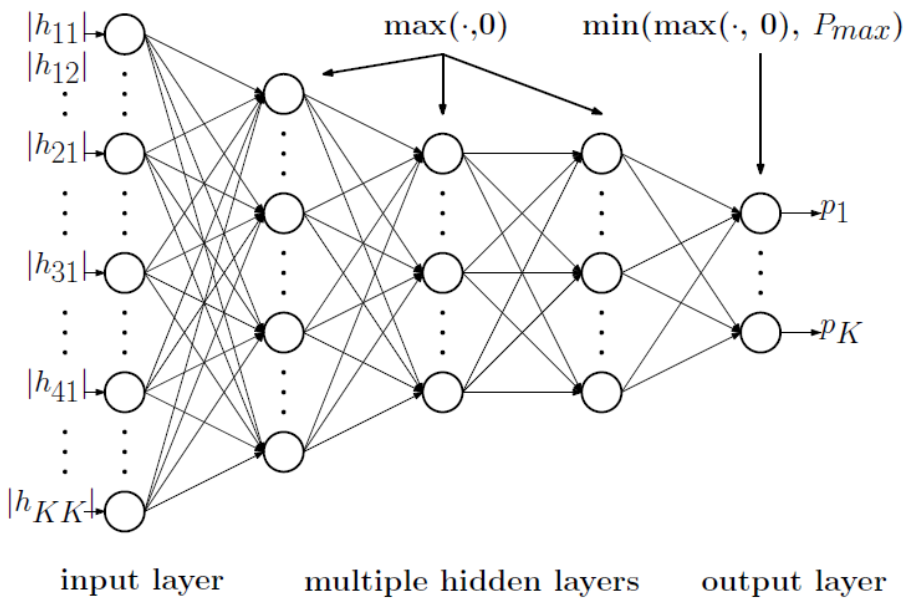


Figure 9: Schematic diagram for DNN implementation model [Hao18].

3 Our approach

The results observed from the original work illustrated in figure 10, shows that the model start to overfit after some certain number of epochs (More specifically the validation error start to

increase since around 100 epochs). As from our discussion to theory till last section, we already know that overfitting causes the problem in case of better evaluation of a machine learning algorithm. For a model to be successful in practical wireless communication field, it must not be overfitted. So we modified the original implementation model to apply L2 norm penalty, Dropout, and L2+Dropout regularization technique in the model.

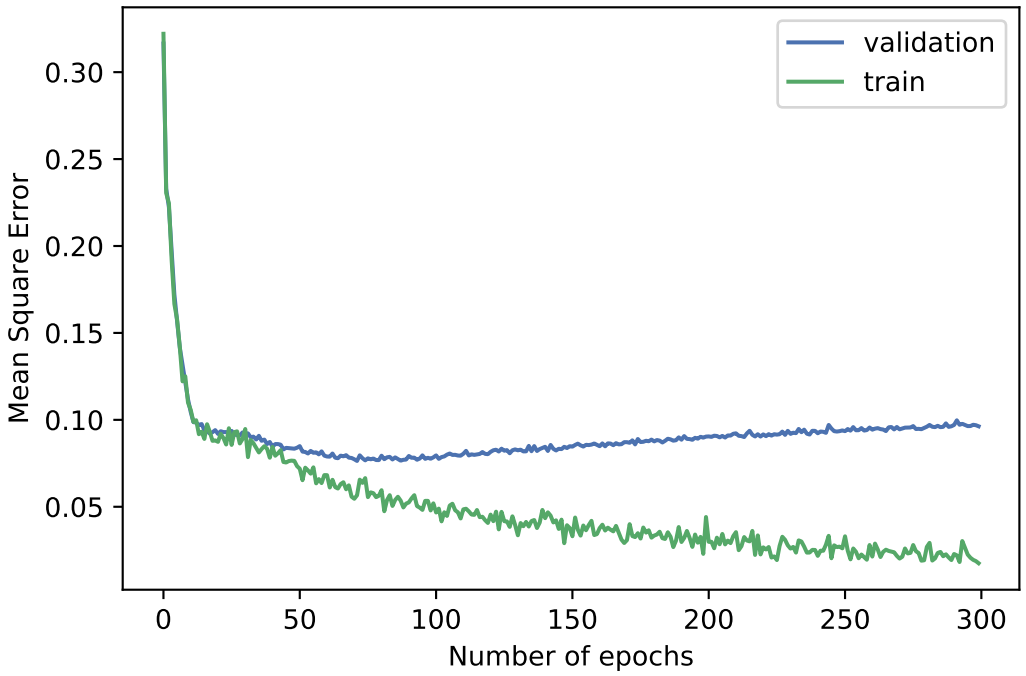


Figure 10: Illustration of overfitting in the original model.

3.1 Results and Discussion

At the starting point we modified the model cost function to include the L2 regularization techniques. The simulated results for L2 regularization techniques are shown in figure 11. We can see from the figure that for regularization constant 10^{-3}, model underfits and results in higher mean square error. We can also observe that for regularization constant 10^{-6}, model overfits because this constant implies less regularization. And finally, for regularization constant 10^{-4}, model shows comparatively better results; where it neither overfits or underfits even for 300 epochs.

In the second step, we modified the model to include dropout regularization technique. We implemented the model with two different keep probabilities (specifically, keep probability equal to 0.5 and keep probability equal to 0.75). We see the effect of dropout technique in the figure 12. With keep probability equal to 0.5(sub-figure a), the model results in higher mean square

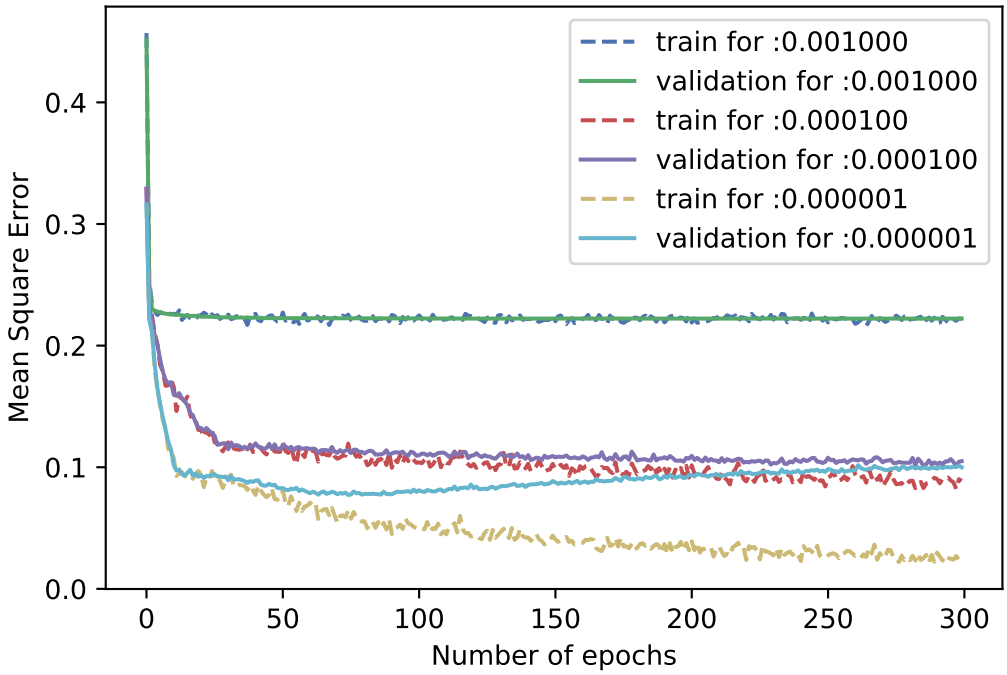
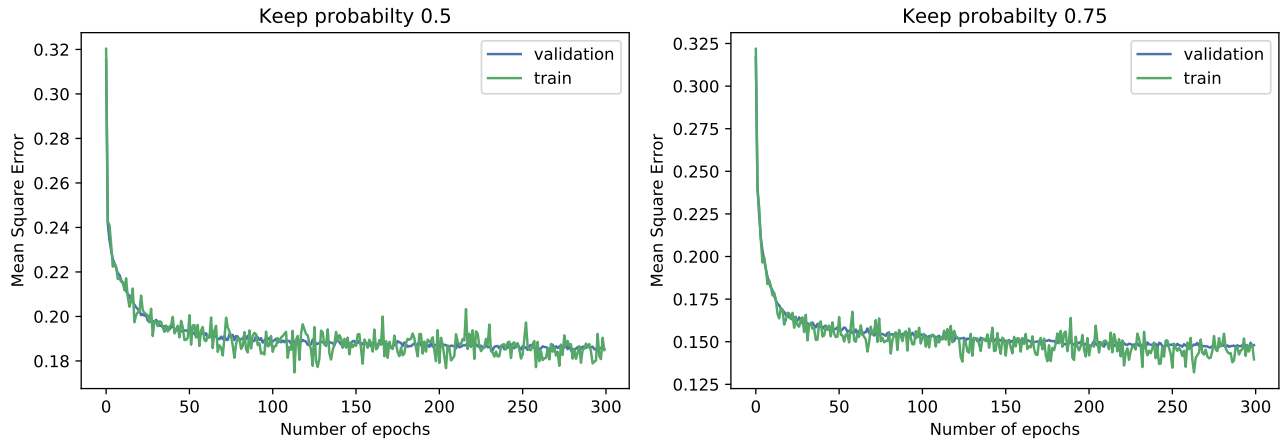


Figure 11: Mean square error for L2 regularization with different regularization constants.

error but is highly regularized and for keep probability equal to 0.75(sub-figure b), the model error decreases further and it is still regularized. By tuning the model's keep probability, we can reduce the model error further without increasing mean square error or risking to converge in overfitting regime.



(a) (b)

Figure 12: Mean square error for dropout regularization with different keep probabilities.

In the last step, we apply L2 norm penalty and Dropout regularization technique together to observe the effect of regularization in our implementation model, shown by figure 13. We see that in this graph even with lower value of regularization constant the model overfits less than what we have seen in figure 11. For higher value of regularization constant, model gets regularized more strongly which proves the theoretical perspective discussed in the earlier section.

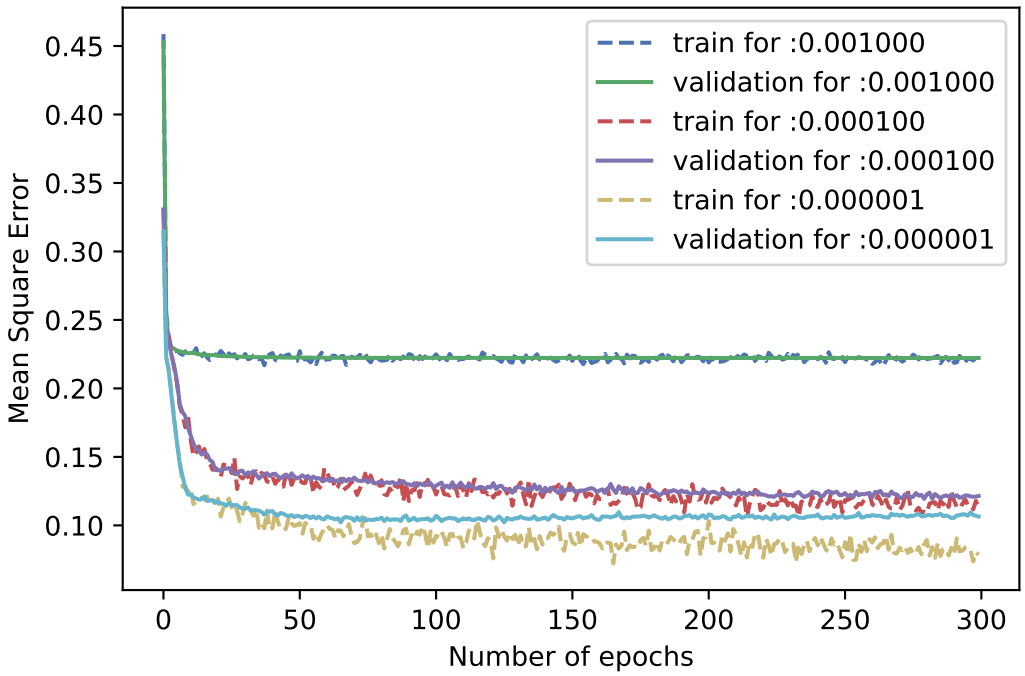


Figure 13: Mean square error for L2+Dropout regularization with different regularization constants.

From the observed results; it is evident that with proper trading of number of epochs for training, regularization constants and keep probability for dropout technique, a model with proper capacity is achievable.

4 Conclusions

We observed the effects of regularization on the implementation model. The goal to achieve a model, which will have low mean square error in training stage and will not overfit, requires a careful trading among different hyperparameters. With the limited scope of this work, we couldn't apply early stopping and dataset augmentation techniques. But these two general regularization techniques also can be better alternatives to continue further this work.

References

[Ian16] I. Goodfellow, Y. Bengio, A. Courville: *Deep Learning*, MIT Press, 2016.

[Hao18] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, N. D. Sidiropoulos:*Learning to Optimize: Training Deep Neural Networks for Wireless Resource Management*, IEEE Transactions on Signal Processing, vol. 66, no. 20, pp. 5438-5453, 15 Oct.15, 2018.

[Qshi11] Q. Shi, M. Razaviyayn, Z. Q. Luo, and C. He:*Learning to Optimize: Training Deep Neural Networks for Wireless Resource Management*, IEEE Transactions on Signal Processing, vol. 59, no. 9, pp. 4331–4340, 2011.

[Plt98] Prechelt L. :*Early Stopping - But When?*,Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 1524. Springer, Berlin, Heidelberg, 1998.

[Nit11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov:*Dropout: A Simple Way to Prevent Neural Networks from Overfitting*,Journal of Machine Learning Research 15 (2014) 1929-1958, 2011.

[Ali11] Ali Harakeh:*Lecture 3: Regularization For Deep Models*,University of Waterloo, May 23, 2017.

[Alex12] A. Krizhevsky, I. Sutskever,G. E. Hinton: *ImageNet Classification with Deep Convolutional Neural Networks*,NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, 2012.