CS 189 Project T - Hyperparameters

Daniel Abraham, Ganesh Vurimi,
Matthew Bronars, Warren Deng, Rasika Iyer

November 2020

1 Introduction

For any machine learning model, we can classify the aspects of the model into parameters and hyperparameters. Parameters simply refer to the coefficients or weights of the model that are choose by the model itself by optimizing them to minimize the error or loss function (or perhaps to maximize a cost function). A simple example is if model is trying to fit a parabola of the form $y = ax^2 + bx + c$, to a set of points, then a, b and c are the parameters that the model will try to learn. Hyperparameters on the other hand are various parameters set by the model architect to dictate how the model goes about the learning process. These can affect the accuracy of the model, the training time and the types of data that is learned. You have already seen some hyperparameters before such as the regularization constant in L1 and L2 regularization, the activation function used by MLP's and neural nets in general as well as the learning rate for neural networks. This note will describe some of the common hyperparameters, the effect they can have on models, and the random search method of selecting hyperparameters vs the grid search technique.

2 Some Common Hyperparameters

2.1 The Regularization Term of Ridge Regression

Perhaps the hyperparameter that you have you seen the most in your coursework thus far is the regularization term λ in ridge regression. Recall the ridge regression solution: $(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{y}$. λ is chosen to be a non-negative value with the value 0 simplifying the above expression into the traditional least squares formula and a weight of 1 may drive the weights down to 0 due to the fact that the penalty term penalizes the norm of w which grows larger as the weights increase in size. This also effects the error of the model. As seen in figure 1, when the regularization term is 0, the training error is at its minimum. This is because as mentioned, $\lambda = 0$ is the same as using OLS, which has the sole purpose of finding the best fit. However, introducing and increasing the regularization term will penalize the weights so the model will put more emphasis on the magnitude of the weights selected, which can increase the training error. The important takeaway is that λ is a hyperparameter since there is no exact formula to choose it or to learn it, but is rather set by the designer of the model based on the context of the problem and possibly through some trial and error.

2.2 Degree/Type of Approximating Function

Another set of hyperparameters that can be customized is the degree and type of approximating function. You may recall from the feature engineering unit that the features used should belong to a family of universal approximators, such as polynomials or the Fourier series. Both of these can be used to approximate different types of underlying models. The feature space can also be elevated as well to allow for better approximations. For example if we are trying to approximate a circle with only x and y coordinates, we may not get the

[1]Taken from fa20 CS 189 note 2

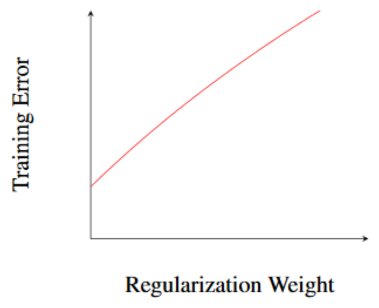
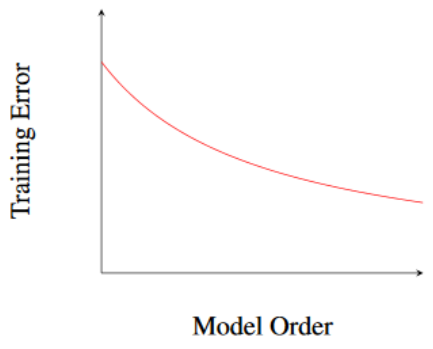
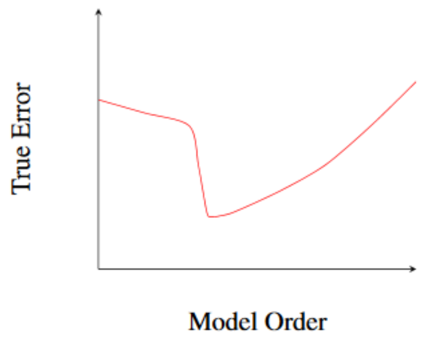


Figure 1: Training Error vs. Regularization Weight [1]



(a)



(b)

Figure 2: (a) Training error will always decrease as the model becomes more complex. (b) However, a complex model is prone to overfitting and cause the true error to increase [2]

best results but if we elevate these to x^2 and y^2, we have a much better approximation. One key point to remember is that as the order of the model increases, the training error will keep decreasing due to the fact that a more complex model can approximate the data better and better. However, this will result in *overfitting* because the model will be over-precisely fit on the training data that it will not be able to account for variations within the actual data. See figure 2.

2.3 PCA Projection

Recall that PCA is a method of reducing the dimensionality of the data matrix. Let us formally define the PCA projection and then highlight the relevant hyper parameter.

Say we have a data matrix $X \in \mathbb{R}^{n \times d}$. Our goal is to reduce the dimension of X to have k columns such that $X_k \in \mathbb{R}^{n \times k}$ and $k < d$. PCA will result in the **best** matrix X_k. What do we mean by best? Best means the matrix X_k that is the most similar to the original matrix X by having the smallest difference defined as $\min_{X_k} ||X - X_k||^2 = \min_{X_k} \sum_{i,j} (X_{i,j} - X_{k\ i,j})^2$. Or in other words, X_k is the $n \times k$ matrix that captures the most variance in the data in X.

When using the data matrix in a classification or a regression problem, we can make the choice of using the raw data matrix X or a low rank approximation X_k. What is interesting is that often times, reducing the dimensionality of the data matrix from d down to k will often times improve our results. So, it is often a good idea to perform hyper-parameter optimization to find the most optimal value of k for our given ML tool. These ML tools can be least squares, logistic regression, ridge regression, SVMs, and the list goes on.

[2]Taken From fa20 CS 189 Note 2

[3]https://medium.com/@PABTennnis/how-to-create-a-neural-network-from-scratch-in-python-math-code-fd874168e955

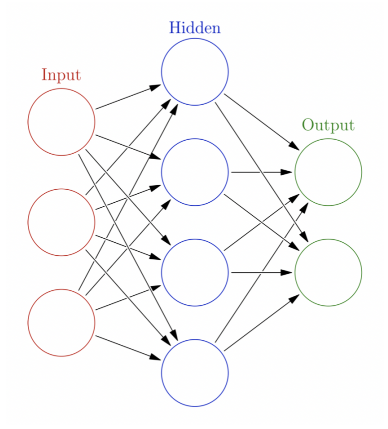


Figure 3: Example drawing of a multilayer perceptron [3]

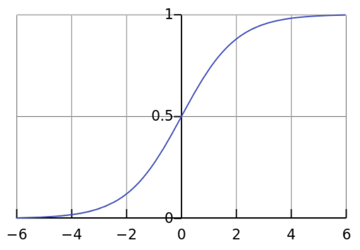


Figure 4: Sigmoid activation function [4]

2.4 Architecture of Neural Network

Recall that multilayer perceptrons have three different types of layers: input, hidden, and output as depicted in figure 3. However, please note there can be multiple hidden layers. Each of the circles in figure 3 represents a neuron. In neural networks, there are several hyperparameters including the number of neurons in each layer, the number of hidden layers, the learning rate, and the activation function.

Using more neurons reduces training error, but can also lead to overfitting of the data. Therefore, the number of neurons will need to be balanced, so the generality is not lost.

Multilayer perceptrons with just one hidden layer are universal approximators. In other words, given a continuous function, there exists a network with one hidden layer such that the output of the circuit can be made arbitrarily close to the output of the given function for all inputs. Adding a layer increases the dimensional complexity of the training data. Multiple layers can be used to represent convex regions, so it is not limited to data that is linearly separable.

The learning rate is used to training a neural network through gradient descent. It scales the magnitude of the weight updates to minimize the loss function. If the learning rate is too low, it will take a very long time to train the model. If the learning rate is too high, the gradient descent algorithm can diverge.

Lastly, the activation function is a threshold function. It takes in the weighted sum of the input neuron, adds the bias, and decides whether the neuron gets activated or not.

Examples of activation functions include Sigmoid, Tanh, and ReLu. Sigmoid activation functions tend to work well with classification problems, which is clear from the graph in figure 4. There are more widely used activation functions and in fact, you can use an activation function of your choice.

[4]https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

2.5 Learning Rate/Step Size in Gradient Descent

Recall the update step in the gradient descent algorithm:

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

The step size term, also known as a learning rate dictates how big of a 'step' the algorithm takes in the direction of the steepest gradient. The size of this step size can be crucial as one that is too large will continually overshoot the minimum and not converge while one that is too small can take extremely long to converge or may become trapped in a local minima. A helpful visualization is shown in figure 5.

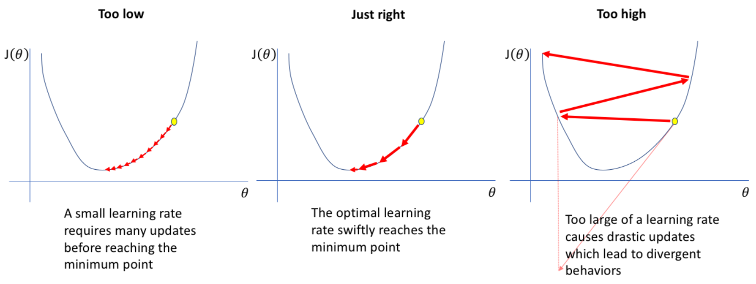

Too low
Just right
Too high
J(θ)
J(θ)
J(θ)
θ
θ
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 5: The consequences of different learning rates [5]

3 Hyperparameter Tuning

There are several ways to pick a combination of hyperparameters that will work best for the model that is being trained. Random search and grid search will be discussed below.

3.1 Random Search vs. Grid Search

Neither of these methodologies are very complicated and their names give away their secrets. Random search simply involves selecting combinations of hyperparameters randomly, often from predefined distributions/ranges. Grid search on the other hand involves methodically checking every combination of hyperparameters from user defined ranges. In cases such as ridge regression, it may seem trivial as to which method is used. After all, the regularization term is bounded between 0 and 1, so a random or grid search would likely yield similar results within similar amounts of time. However, more consideration is needed for the neural net application. As seen in section 2.5, there are multiple hyperparameters that are taken into consideration when designing a model. The number of hidden layers, number of neurons in each layer, the activation function used, learning rate can all vary. These are the only the ones we have discussed so far, there will be more that are discussed in the neural networks part of the course such as momentum and dropout.

The time complexity and computing power needed for both types of search must be considered. Random search is trivial - it will take $\mathcal{O}(n)$ time, where **n** is the number of iterations that will be run. For grid search, the resources needed increase in a multiplicative way based on the number of hyperparameters and their ranges. If there are two hyperparameters to optimize, each with α and unique values, then the resources needed will be $\mathcal{O}(\alpha\beta)$. If we add a third hyperparameter with γ unique values, then the resources will scale to be $\mathcal{O}(\alpha\beta\gamma)$, as one may infer.

Unfortunately, there is not yet a clear answer on when to use which method. If a similar model, based on similar data has been constructed before, then the model architect may have an idea of what range the hyperparameters should fall in. In such a case, a grid search may prove to be effective. In scenarios where the

[5]https://www.jeremyjordan.me/nn-learning-rate/

model may be a little more novel, random search may be more effective. A combination of the to may also be used, where random search is used to find a ”ballpark” for an optimal hyperparameter combination and then grid search can be used to refine it. This technique may be better for problems with higher dimensionality. This is due to the fact that as seen in figure 6, grid search only tests three distinct values for each parameter while random search tests 9 distinct values for each. [6]

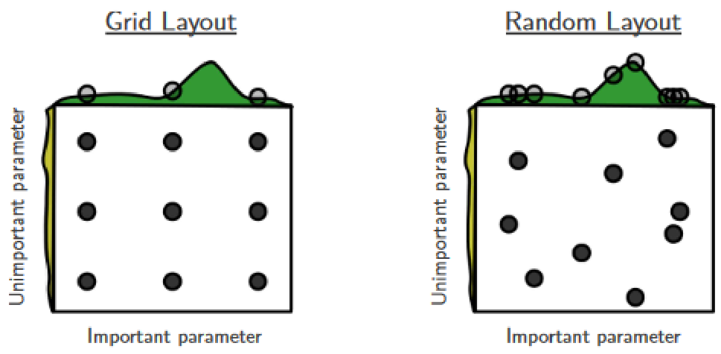


Figure 6: Visualization of Grid Search and Random Search [7]

There is still a lot of research going on in this area. In later classes, you will also be introduced to other hyperparameter optimization methods such as Bayesian Optimization, but that is currently out of scope for this class.

[6]More about this can be read in Bergstra and Bengio’s paper.
[7]Image from Bergstra Bengio’s paper