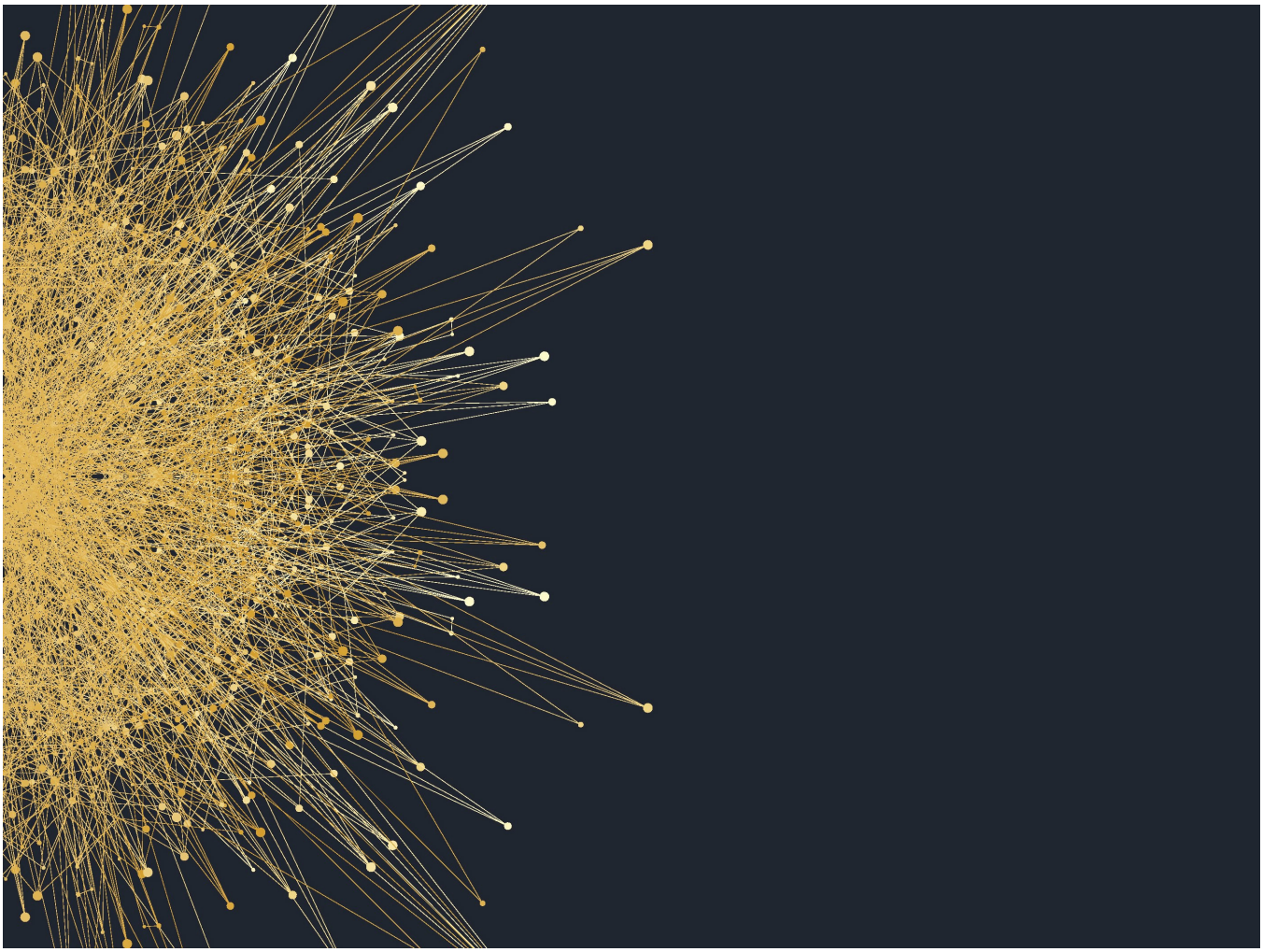
Guide to choosing Hyperparameters for your Neural Networks

Kaushik Bokka
Feb 18, 2019 · 8 min read



https://www.wired.com/2016/12/2016-year-deep-learning-took-internet/

In recent times, Deep Learning has created a significant impact in the field of computer vision, natural language processing, and speech recognition. Due to the large amounts of data being generated day after day, it could be used to train Deep Neural Networks and is preferred over traditional Machine Learning algorithms for higher performance and precision which has led to various successful commercial products.

Even though Deep Learning has been a booming field, there are certain practical aspects of it which remains a black box like choosing the optimal hyperparameters for your Neural Networks.

You need to understand that Applied Deep Learning is a highly iterative process. There are various hyperparameters you need to keep in your mind before training your model. It is almost impossible to get the correct values of hyperparameters as it requires years of experience and expertise and as well as extensive trial and error to find the optimal values for your model which becomes a huge problem for the people starting out in the field.

In this guide, I'm going to describe powerful and effective ways of choosing the optimal hyperparameters for your model. The hyperparameters are-

> ***Learning Rate***
>
> ***Batch Size***
>
> ***Momentum***
>
> ***Weight Decay***

I am going the divide this guide into two parts. In this part, I will talk about the learning rate and the batch size.

But before getting into hyperparameters, we need to cover some core concepts and tradeoffs which play a major role in the performance of the model and how the hyperparameters impact these concepts and it would help us to interpret the training process. After reading this post, you will understand the important machine learning concepts such as Bias-Variance tradeoff, Overfitting, and Underfitting and practical and effective ways of choosing the parameters which are -

Generalization

We have trained a classification model on 10,000 images with their labels. We test the model out on the original dataset, it is able to predict labels with a mindblowing accuracy of 99%

But when we try the same model on a new unseen dataset, we only get an accuracy of 50% and fails to performs well.

This is the case of overfitting as our model wasn't able to generalize well on the unseen dataset.

Our goal while training a network is to have it generalize well on the training data which means capturing the true signal of the data rather than memorizing the noise in the data. In statistics, it is termed as "Goodness of Fit" which refers to how well our predicted values match to the true (ground) values.

ML is also you learning from the algorithm and getting it right!

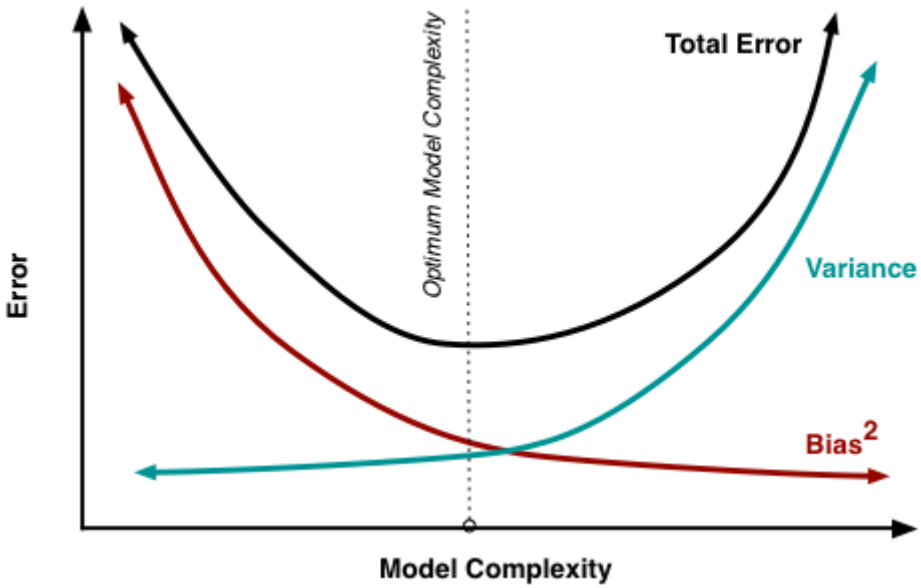
Bias-Variance Tradeoff

Bias-Variance Tradeoff is one of the important aspects of Applied Machine Learning. It has simple and powerful implications around the model complexity and its performance.

We say there's a **bias** in a model when the algorithm is not flexible enough to generalize well from the data. Linear parametric algorithms with low complexity such as Regression and Naive Bayes tend to have a high bias.

Variance occurs in the model when the algorithm is sensitive and highly flexible to the training data. Non-Linear non-parametric algorithms with high complexity such as

Decision trees, Neural Network, etc tend to have high variance.



Bias vs Variance

So, while training a model we need to keep in mind of this tradeoff because algorithms with high bias and a low variance perform well consistently but on the average, their accuracy is lower. Whereas, algorithms with low bias and high variance have better accuracy but are inconsistent and as well as this tradeoff depends on the dataset it's trained on.

There are various ways to find the balance of bias and variance for each algorithm family by using methods such as regularization, pruning, etc.

But in this post, I'm going to only talk about finding the balance for the Neural Networks.

Overfitting vs Underfitting

Do you know what's one mistake that could single handly ruin the performance of your model?

It's one of the trickiest problems one faces in applied machine learning.

It is Overfitting. Because, when your machine learning model is tested on the unseen data in the "real" world, it will come across data which wouldn't be in the training data. Therefore, it is necessary for the model to generalize well on the data.

Algorithms overfit on the training data when it memorizes the noise instead of the data. Usually, complex algorithms such as Neural Networks are prone to overfitting.

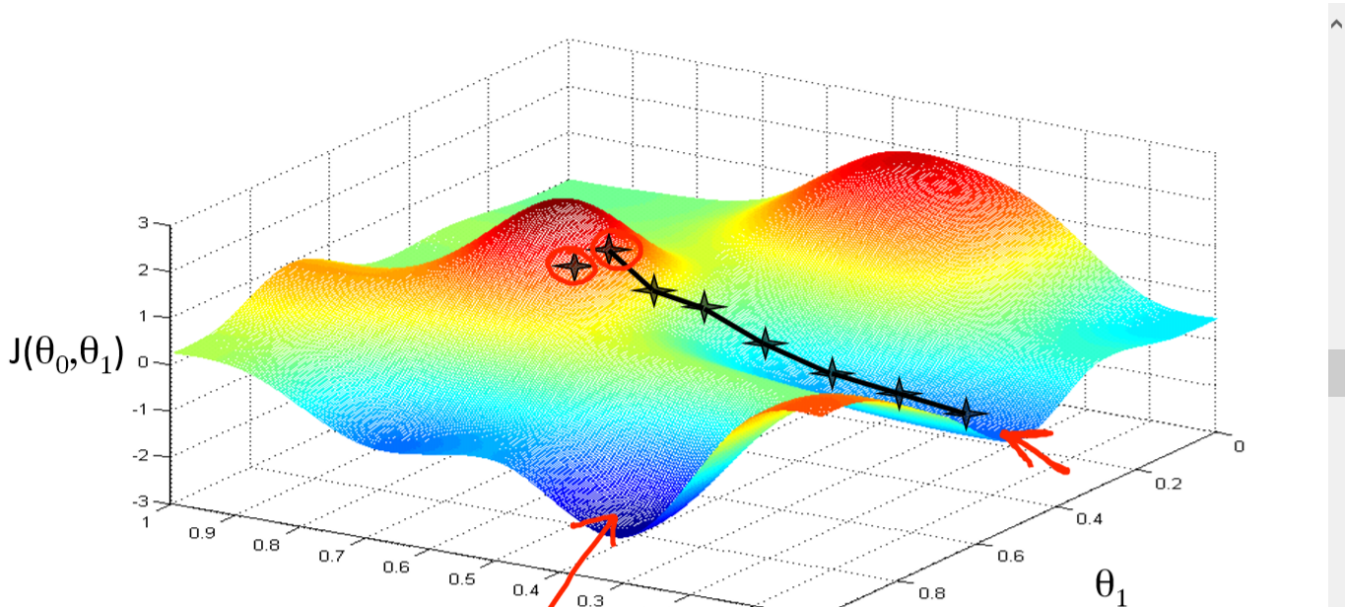
Algorithms underfit on the training data when it is not able to capture the true signal from the data. Underfitted models have bad accuracy in training as well as on the test data.

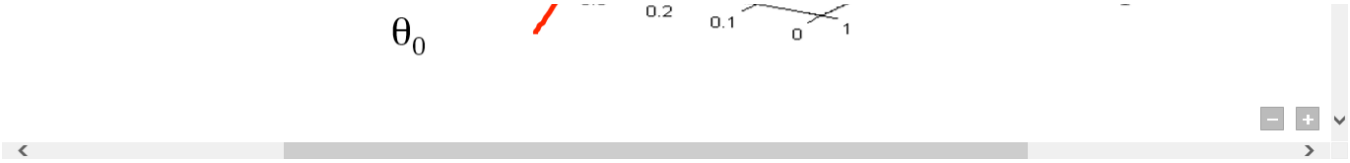
We are going to talk more about it later and will see how choosing the optimal parameters helps in finding the right balance to get the right predictions which minimize total error.

Gradient Descent Algorithms

Gradient Descent is one of the most popular and used algorithms for weights optimization of the models. Gradient Descent helps in minimizing the cost function by updating the parameters which are weights in the opposite direction of the gradient of the cost function. It has a learning rate, a hyperparameter, which helps us control the adjustment of weights for our network to our loss gradient or in simpler terms, we can say the learning rate refers to the size of steps the gradient descent takes to reach the local optima. That's our goal to find the optimal weights. We will talk about choosing the right learning rate for our network later as it plays an important role.

There are in total of three variants of Gradient Descent. They differ by the way they take data for computing the gradient of the cost function. There's a tradeoff over here depending upon the Gradient Descent variant we choose, which is the accuracy of the update of weights and the time it takes for each epoch i.e each iteration for the weights update.

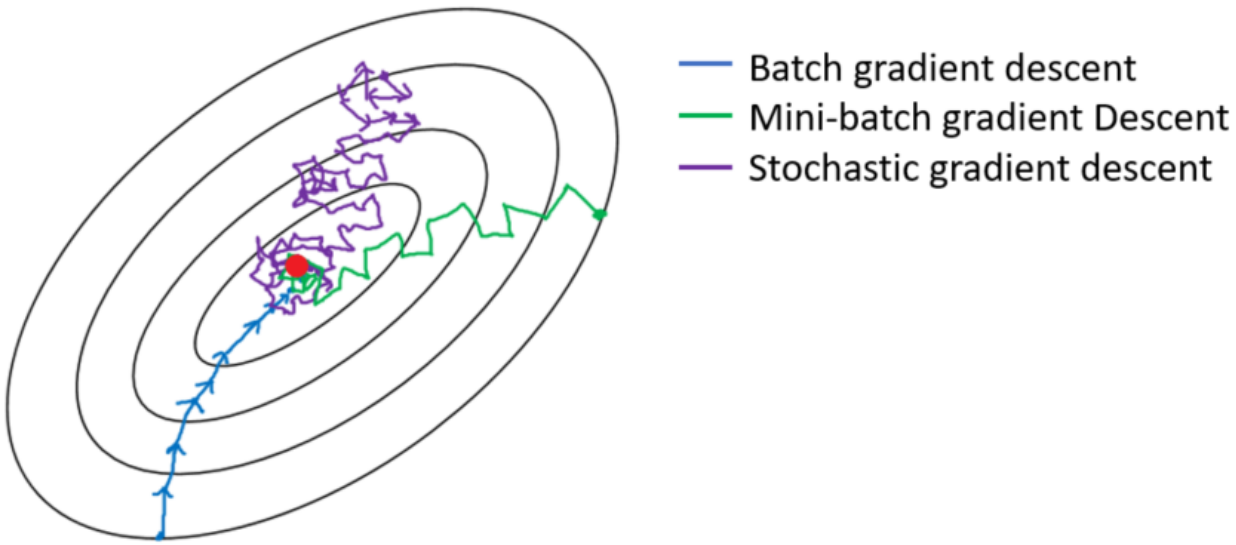




Search for local optima to get the best weights

Our main goal while training a neural network is to find the optimal weights to get accurate results. The distinct feature that makes Deep Learning better than traditional machine learning is the fact that we don't need to do any feature engineering.

Weights are the parameters that we have no control on, we need to focus more on the network architecture, activation functions, and the hyperparameters which would help us find the optimal weights.



Variants of GD Algorithm

Learning Rate

> *Learning Rate is a parameter of the Gradient Descent algorithm which helps us control the change of weights for our network to the loss of gradient.*

The training of your model will progress very slowly if you keep your learning rate low as it would be making tiny adjustments to the weights in the network and it could also lead to overfitting. Whereas, when you set a high learning rate, the parametric function tend to have kinetic energy which makes it take huge leaps bouncing around chaotically, not letting it get narrower and deeper missing the local optima making the

training diverge. But having a large learning rate helps in regularizing the model well to capture the true signal.

We can do a random grid search to find the optimal learning rate that converges but there are easier ways as it is computationally heavy and time-consuming.

Learning Rate Schedule

The Learning Rate Schedule helps to adjust the learning rate while the training process by changing the learning rate through a predefined schedule. There are three variants of learning rate schedules such as time-based decay, exponential decay, and step decay.

Constant Learning Rate

We can use the default learning rate while training the network, which we could use as a baseline to compare our performances and to get a rough idea of how the network is behaving. We can set the momentum and time decay as default which is zero.

Adaptive Learning Rate Method

The drawback of using learning rate schedules is that we have to predefine the learning rates before the training process but we couldn't choose the right ones with intuition as it depends totally on the model we have and the domain problem we are working on.

There are variants of Gradient Descent Algorithms that are adaptive in nature such as Adagrad, Adadelta, RMSprop, Adam in which the learning rate adapts on the kind of data it is dealing with (sparse, less sparse, etc) and it doesn't require much of the manual work for tuning the learning rate.

In the deep learning frameworks, we can easily implement the above adaptive optimizers. it is recommended to keep the hyperparameters for these optimizers as default.

Cyclic Learning Rate

For using Cyclic Learning Rate, we need to define the minimum and maximum learning rate boundaries with the step size. The step size here is the number of epochs(iterations) taken for each step. The cycle consists of two kinds of step — one that linearly increases from minimum to maximum and the one that linearly decreases.

https://github.com/gaohuang/SnapshotEnsemble

Learning rate Range test

The training of the network by using the learning rate range test starts with a small learning rate which then linearly increases. We can obtain valuable information from it by a single run as when we choose a small learning rate, the cost function tends to converge well hitting the optima, but while increasing the learning rate, we will come across a threshold from which the validation loss starts increasing and the accuracy drops. That's how we get an idea about the optimal learning rate for our model and data

The amount of regularization we do should be balanced depending on the data and the model we are implementing.

Batch Size

> *The Batch Size refers to the number of training samples propagated through the network.*

For example, we have a training dataset of 1000 sample and if we set a batch size of 100, the network will consider every 100 samples for the training but process every epoch.

While practicing deep learning, it is our goal to obtain the maximum performance by minimizing the computational time required. While the value of the hyperparameter learning rate doesn’t affect the training time but the batch size does.

How Batch size impacts the performance

Some papers recommend using a larger batch size which could be supported by your system's memory. Some also suggest modifying the batch size rather than changing the learning rate. However, a larger batch size enables using a large learning rate. Larger batch sizes tend to have low early losses while training whereas the final loss values are low when the batch size is reduced.

. . .

In part two of this guide, I will cover momentum and weight decay as well as the code to reproduce the results. *Cheers!*

Machine Learning Data Science Deep Learning Artificial Intelligence