Parameter tuning:

Parameter tuning refers to finding optimal values for hyperparameters that can yield best model performance.There are many techniques such as grid search cv in which we go over every combination of values for hyperparameters and report combination which yields best performance. However, in deep learning it might be time consuming to do so. We discuss hyperparameter tuning techniques that are specifically targeted for neural networks.

Number of hidden layers:

According to [1], as we increase the number of layers inside a neural network, the model is able to learn more complex representations. For instance, 1 hidden layer can be used for simple regression where one continuous value maps to another continuous value. Two hidden layers can be used to learn decision boundaries for classification. With more than 2 hidden layers, models can actually learn complex representations such as high level features in an image (like white patch in a blood cell image which indicates Malaria infection).

Let's look at the following relationship between accuracy and epochs as we increase the number of hidden layers.

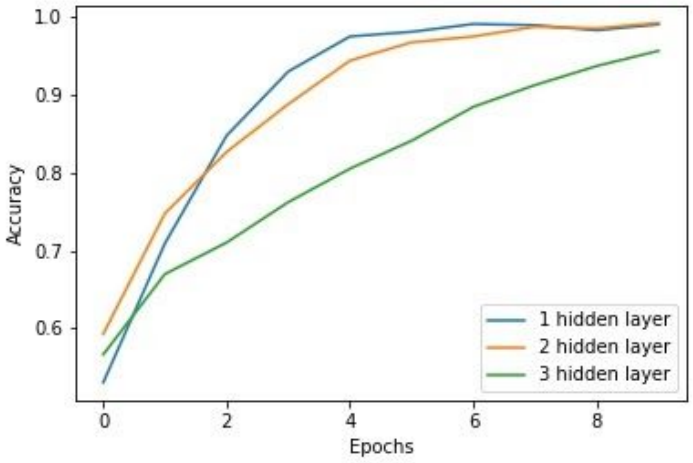


Fig 1. Training accuracy

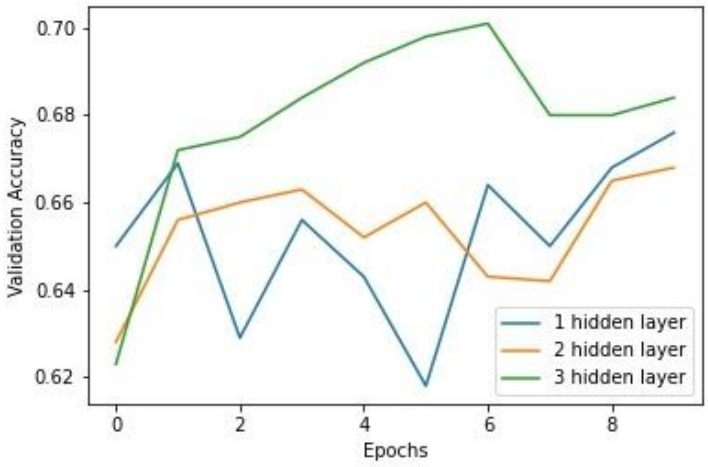
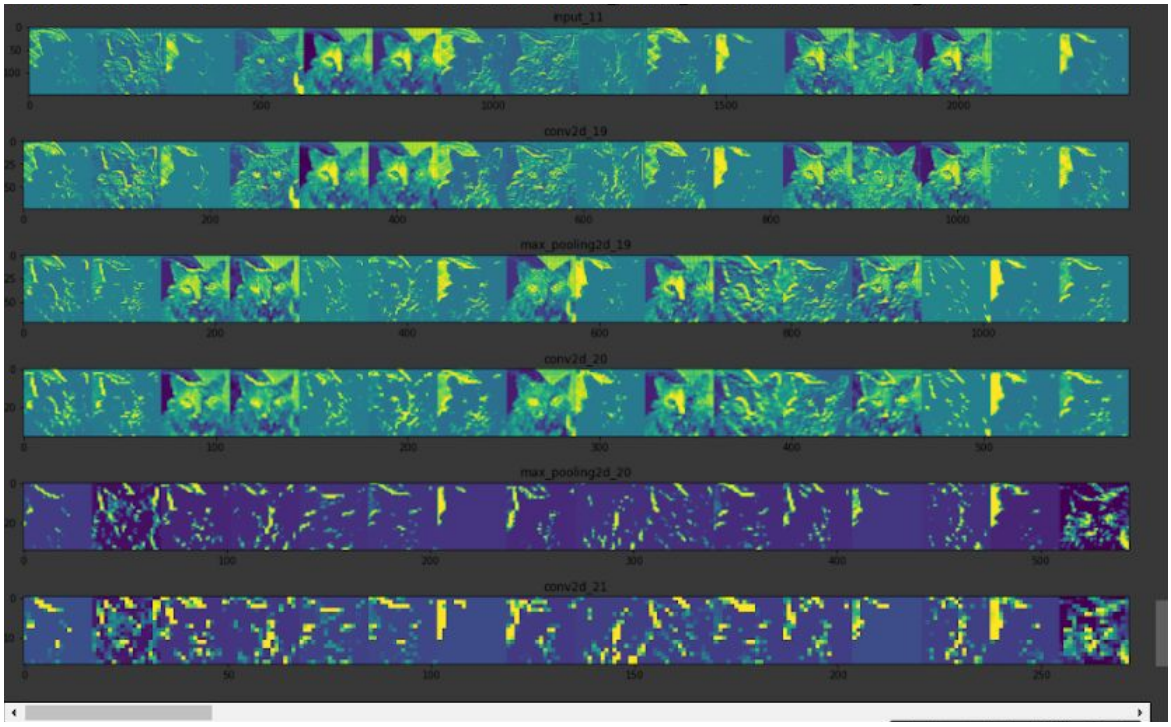


Fig 2. Validation accuracy

We trained Cat dog classification models by varying hidden convolutional layers between the range 1 to 3 with each of them consisting of 16 units. As we can see, model with 1 hidden layer learns quickly but does not perform well on the validation data. On the other hand, a model with 3 hidden layers learns slowly and fits significantly better on validation data than a model with 1 hidden layer.

From another perspective, shallower layers in the CNN generally represent raw pixels but as we go deeper, more class-dependent abstract and compact features are extracted. Please refer to the following diagram which shows CNN activations for the cat image.



Number of units in each layers

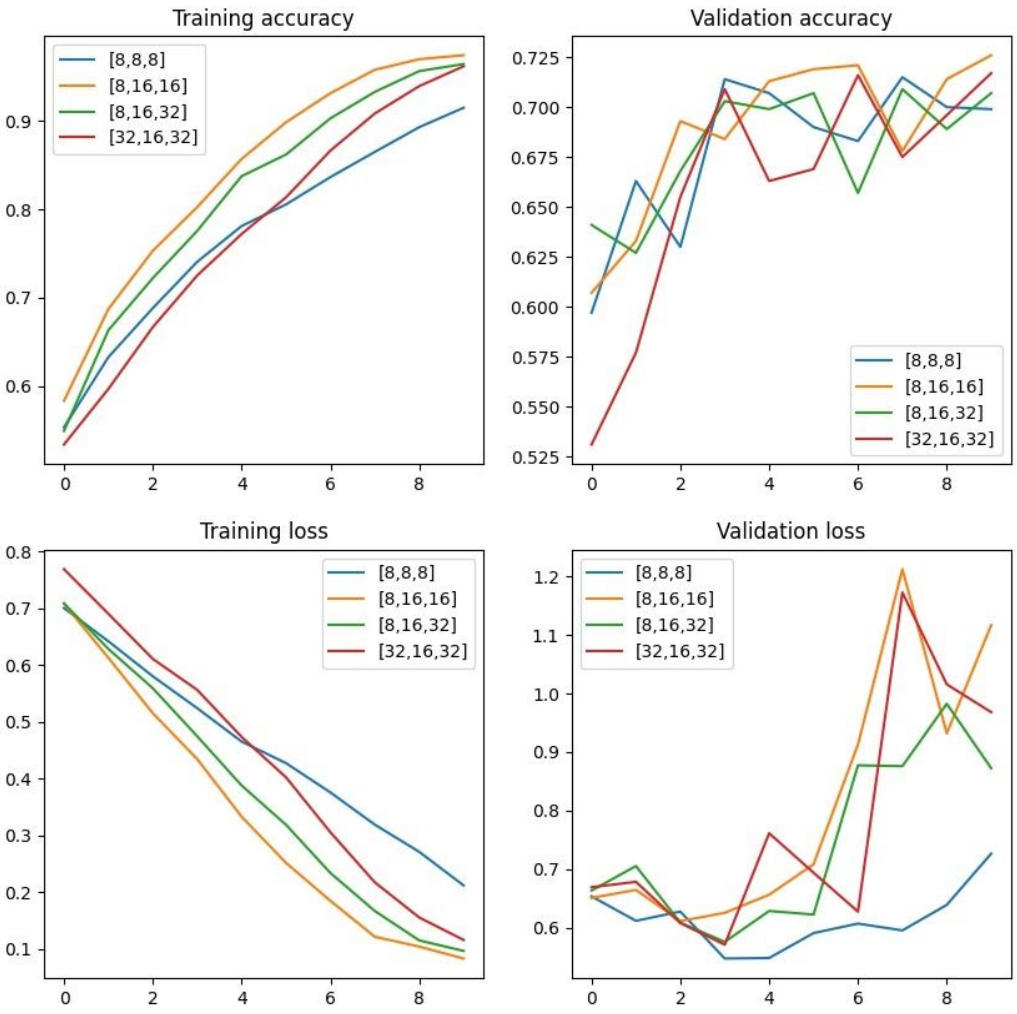
We experimented with 4 different configurations of hidden units:

[8,8,8]

[8,16,16]

[8,16,32]

[32,16,32]



As we can see, using too few neurons can cause underfitting and too many neurons can cause overfitting. According to [1], following are some thumb rules to consider while deciding number of hidden neurons:

1. The number of hidden neurons should be between the size of the input layer and the size of the output layer.
2. The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
3. The number of hidden neurons should be less than twice the size of the input layer.

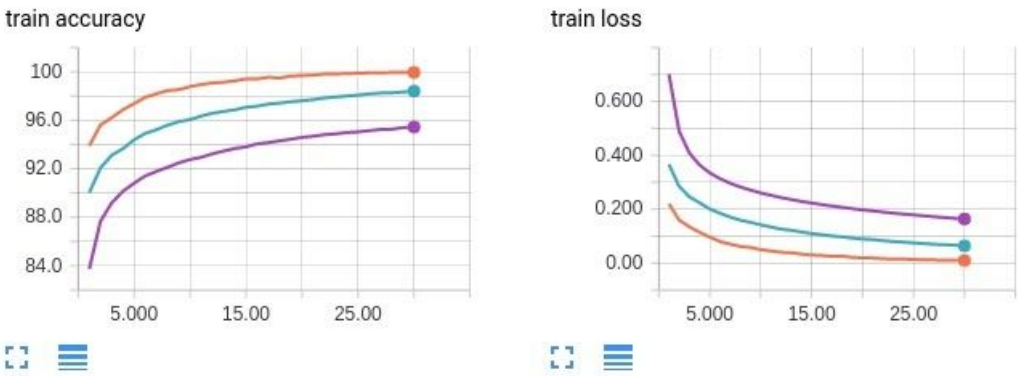
G. Batch size

Batch size is the number of examples from the training dataset used in the estimation of the error gradient and is an important hyperparameter that influences the dynamics of the learning algorithm.

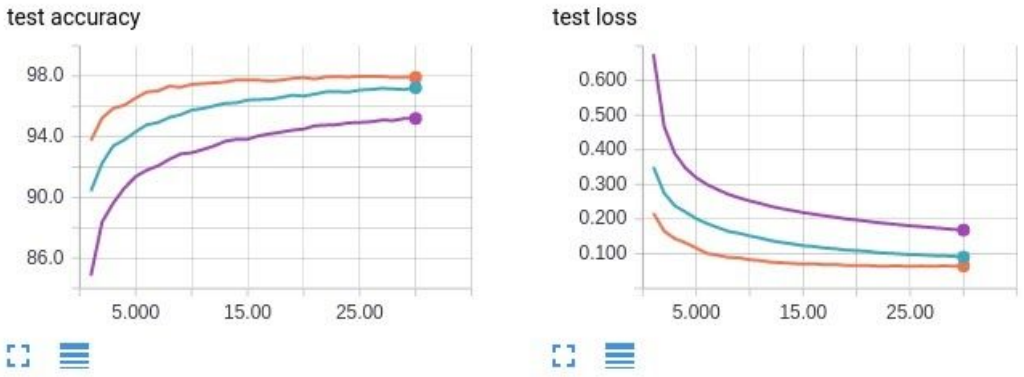
For example, A batch size of 32 means that 32 samples from the training dataset will be used to estimate the error gradient before the model weights are updated.

- **Batch Gradient Descent**. Batch size is set to the total number of examples in the training dataset.
- **Stochastic Gradient Descent**. Batch size is set to one.
- **Minibatch Gradient Descent**. Batch size is set to more than one and less than the total number of examples in the training dataset.

In the below graph we will see the effect of batch size on training/testing accuracy and loss:



Training loss and accuracy when the model is trained using different batch sizes.



Testing loss and accuracy when the model is trained using different batch sizes.

Where,

Orange curves : 64 batch size
Blue curves : 256 batch size
Purple curves : 1024 batch size

As we can see 64 is the optimal batch size. The accuracy is more and loss is less when a batch size of 64 is used.
It is generally accepted that there is some “sweet spot” for batch size between 1 and the entire training dataset that will provide the best generalization. This “sweet spot” usually depends on the dataset and the model at question.

Effects of larger batch size:

- Larger batch size to train model allows computational speedups from the parallelism of GPUs.
- Too large of a batch size will lead to poor generalization
- Using a batch equal to the entire dataset guarantees convergence to the global optima.

Effects of smaller batch size:

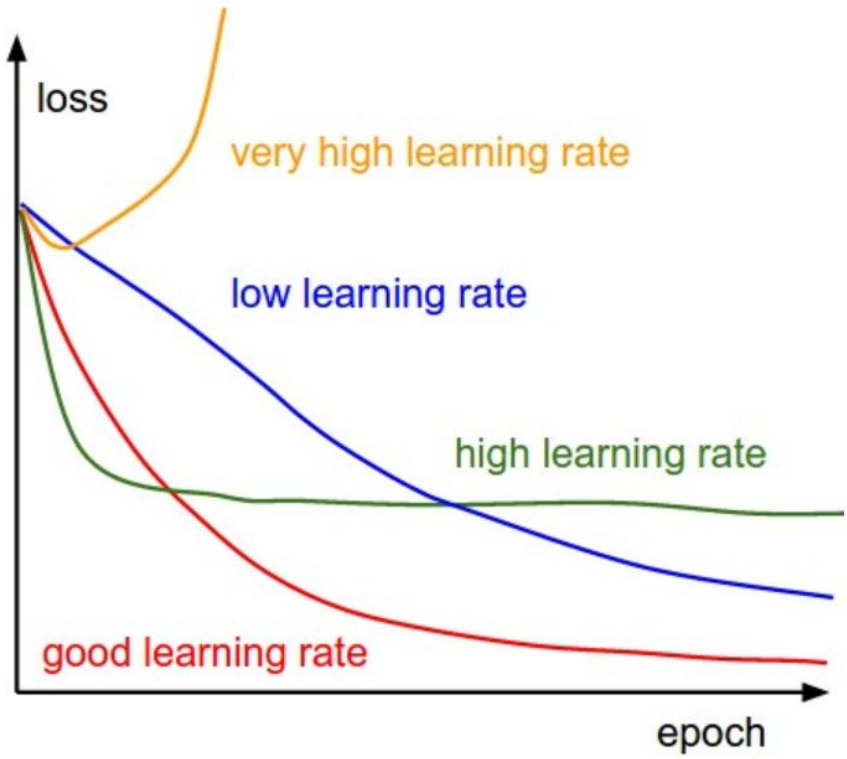
- Using smaller batch sizes leads faster convergence to “good” solutions.
- Smaller batch sizes allow the model to “start learning before having to see all the data.”
- The downside of using a smaller batch size is that the model is not guaranteed to converge to the global optima.
- Smaller batch sizes make it easier to fit one batch worth of training data in memory (i.e. when using a GPU).

Learning rates: (Naveen Kumar)

Learning rate is one of the hyperparameters used in Gradient descent algorithm for backtracking and updating the weights of the neural networks. While learning the weights of the model the gradient descent tries to update the weights in order to reduce the error. The hyperparameter learning rate is the factor by which the weights are updated at each epoch. Selecting the right learning rate for gradient descent is important to ensure that the algorithm converges at an optimal solution. Having too large a learning rate poses the risk of exploding gradients and the algorithm may never converge.

However, there is no one optimal value for learning rate. Although, having a small learning rate can ensure that gradient descent converges, it can also slow down the learning as the algorithm makes little progress with each epoch and can take longer to converge. It is often best practice to come up with a range of smaller learning rates and perform training and validation to see which learning rate gives the best results.

Below graph shows the impact of different learning rates on the convergence of gradient descent [2].



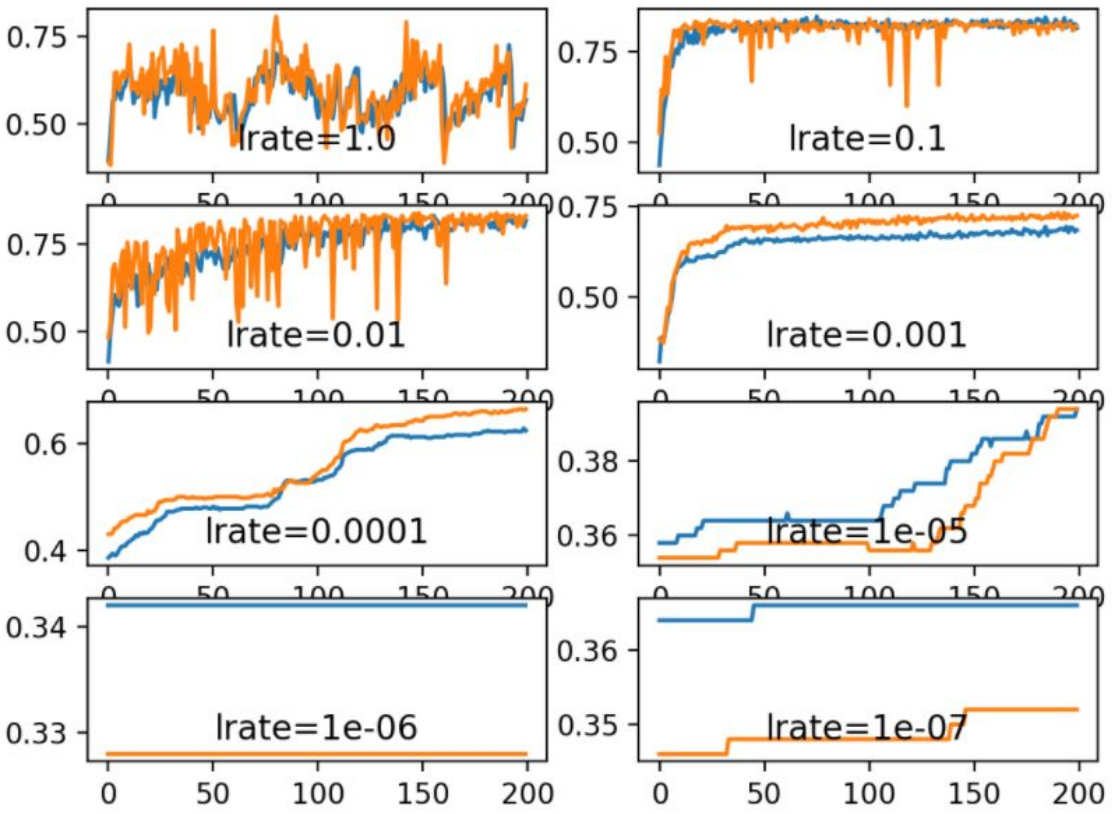
Effect of various learning rates on convergence (Img Credit: cs231n)

Another graph shows the range of learning rates and their respective effect on performance training and test data. The graph shows training performance using blue lines and test performance using orange lines [3].

Here the large learning rate 1 shows the model never converges for both training and test data and keeps oscillating. However the learning rate of 0.01 gives the optimal performance where it converges for both training and test data. Although, a very small learning rate shows down the learning curve.

The graph also shows for extremely small learning rate the model doesn't learn much and although the performance on training data looks good it does not perform well on the test data.

Below is a graph showing the performance on training and test data for different learning rates.



Line Plots of Train and Test Accuracy for a Suite of Learning Rates on the Blobs Classification Problem

Epoch (Savankumar Patel)

The number of **epochs** is a **hyperparameter** that defines the number times that the learning algorithm will work through the entire training dataset. One **epoch** means that each sample in the training dataset has had an opportunity to update the internal model parameters. An **epoch** is comprised of one or more batches.

You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over

each batch of samples, where one batch has the specified “batch size” number of samples.

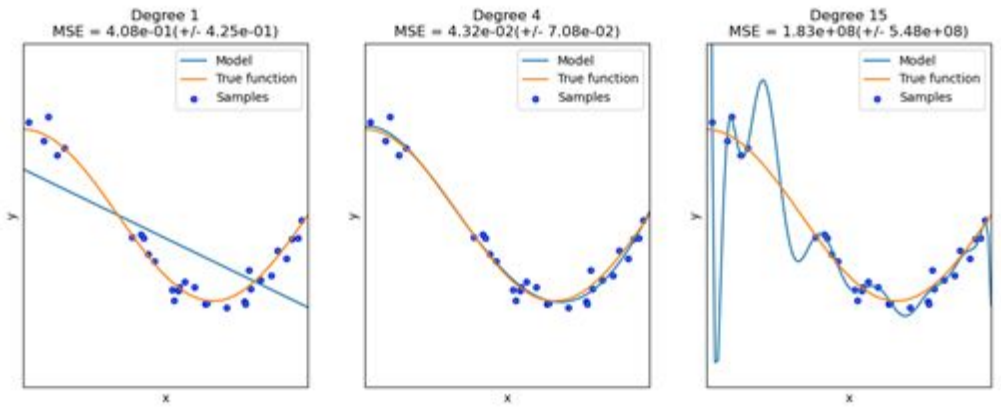
The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.

It is common to create line plots that show epochs along the x-axis as time and the error or skill of the model on the y-axis. These plots are sometimes called learning curves. These plots can help to diagnose whether the model has over learned, under learned, or is suitably fit to the training dataset.

If our training dataset has 1000 records, we could decide to split it into 10 batches (***100 records per batch — Batch size of 100***). Thus, 10 steps will be required to complete one learning cycle. Also if we decide to split the 1000 training set into 100 batches, we would then need 100 steps per each learning cycle. (***10 records per batch — Batch size of 10***).

The ***10*** or ***100*** steps are **I*terations***. And by the **E*nd of the 10th or 100th step***, we would have completed ***one Epoch,*** which is a complete learning cycle. By the end of an Epoch, the learning algorithm is able to compare or review actual outputs from the training data and optimize or make adjustments to its parameters in order to make better predictions in the next cycle. There is no guarantee that the Gradient Descent will be globally optimized or reach its best optimization by the end of the first optimization cycle(Epoch). Because of this, more than a few Epochs are often needed to achieve an ideal or high model accuracy. There is no set number of Epochs for optimizing a particular learning algorithm.

Just one Epoch can result to underfitting. However, too many Epochs after reaching global minimum can cause learning model to overfit. Ideally, the right number of epoch is one that results to the highest accuracy of the learning model.



Underfitting fit overfitting

While the concept of Epoch remains fundamental to optimization of a learning algorithm, its specific application to learning models such as Artificial Neural networks or Reinforcement learning can differ in terms of how the model is modified to perform better after each cycle.

References:

[1]. https://www.heatonresearch.com/2017/06/01/hidden-layers.html
[2].https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10
[3].https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/