Training Neural Networks – Part 2

Table of Contents

[Learning Rate Schedulers 2](#_Toc138882049)

[Early Stopping 4](#_Toc138882050)

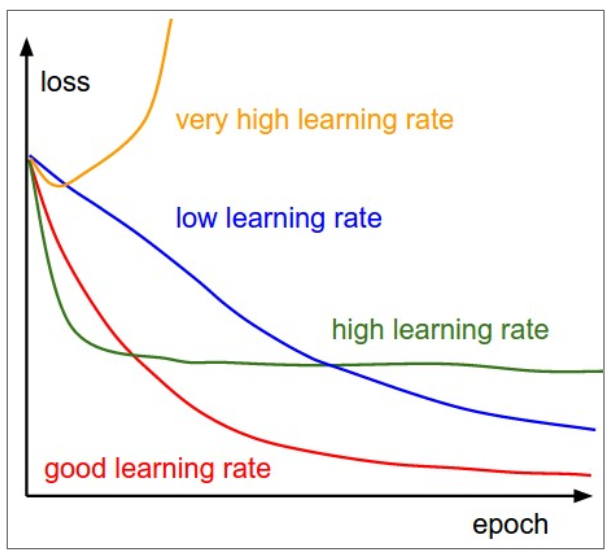
[Choosing Hyperparameters 5](#_Toc138882051)

[Track Ratio of Weight Update / Weight Magnitude 9](#_Toc138882052)

[Model Ensembles 9](#_Toc138882053)

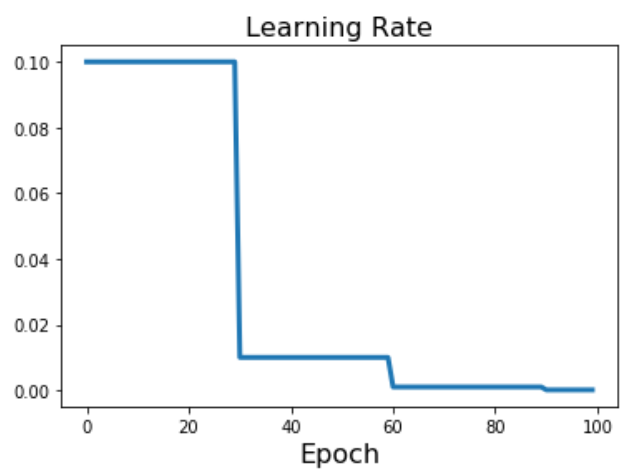
[Transfer Learning 10](#_Toc138882054)

## Learning Rate Schedulers

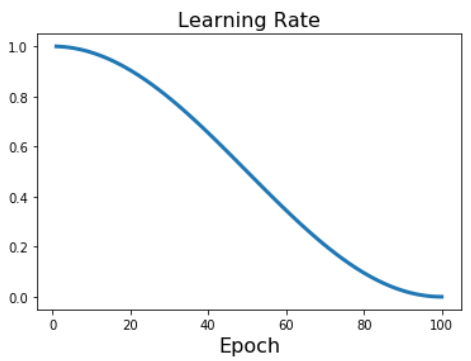


The graph above shows that we can end up with wildly different loss values depending on what our learning rate is. The idea scenario is to use a **high learning rate** towards the start of the training process and use a **low learning rate** towards the end. This will allow us to train quickly but also not overshoot the minima as we get close to it. This is where **learning rate schedulers** come in. They allow us to decay the learning rate using one of multiple schemes.

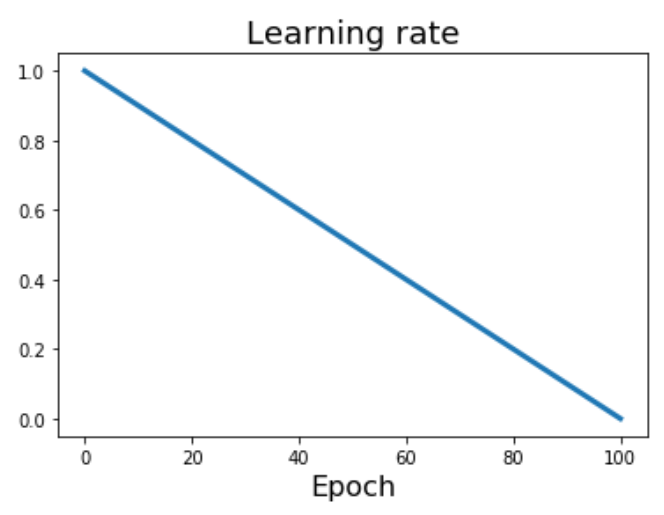
* **Step Decay** - Reduce by a fixed amount at specific intervals. The amount by which we want to decay and the intervals at which the decays should take place are hyperparameters, which adds to the complexity of choosing hyperparameters.



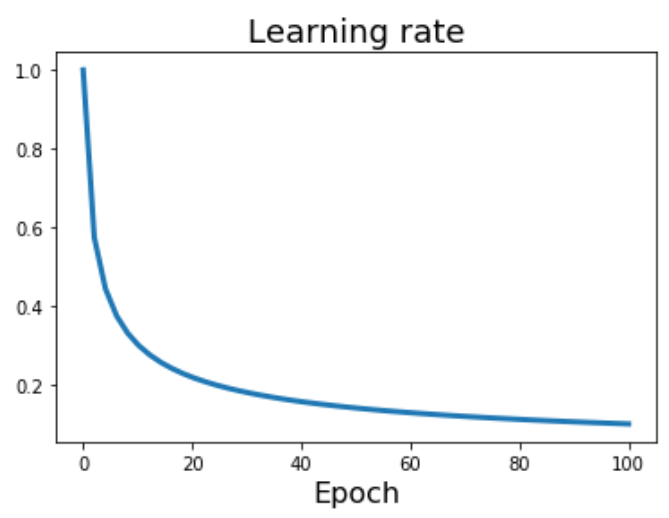
* **Cosine Decay** - . The benefit to this method is that there are no new hyperparameters.



* **Linear Decay** -



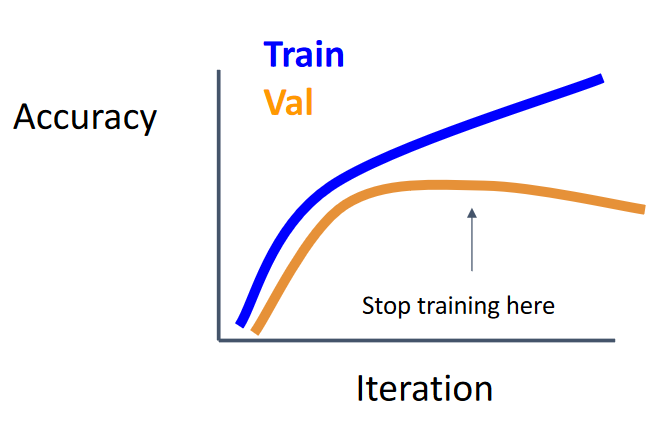
* **Inverse Square Root** - . This method has the learning rate drop very quickly though, which makes it less popular.



* **Constant** -

## Early Stopping

We should also stop training the model when we find that the error on the validation set has increased. This is called **early stopping**. Alternatively, we could store the checkpoints for our model at every iteration and go back and use the model that had that least error on the validation set.

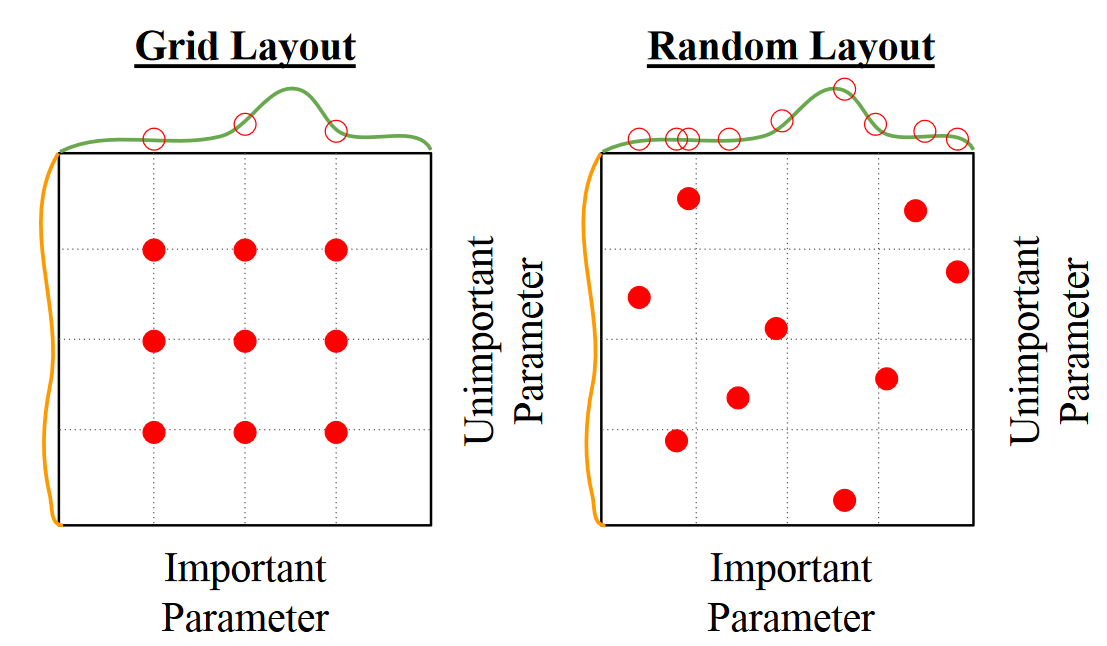


## Choosing Hyperparameters

Searching for the correct values for hyperparameters is a complicated process and there are two systematic ways to go about it: grid search and random search.

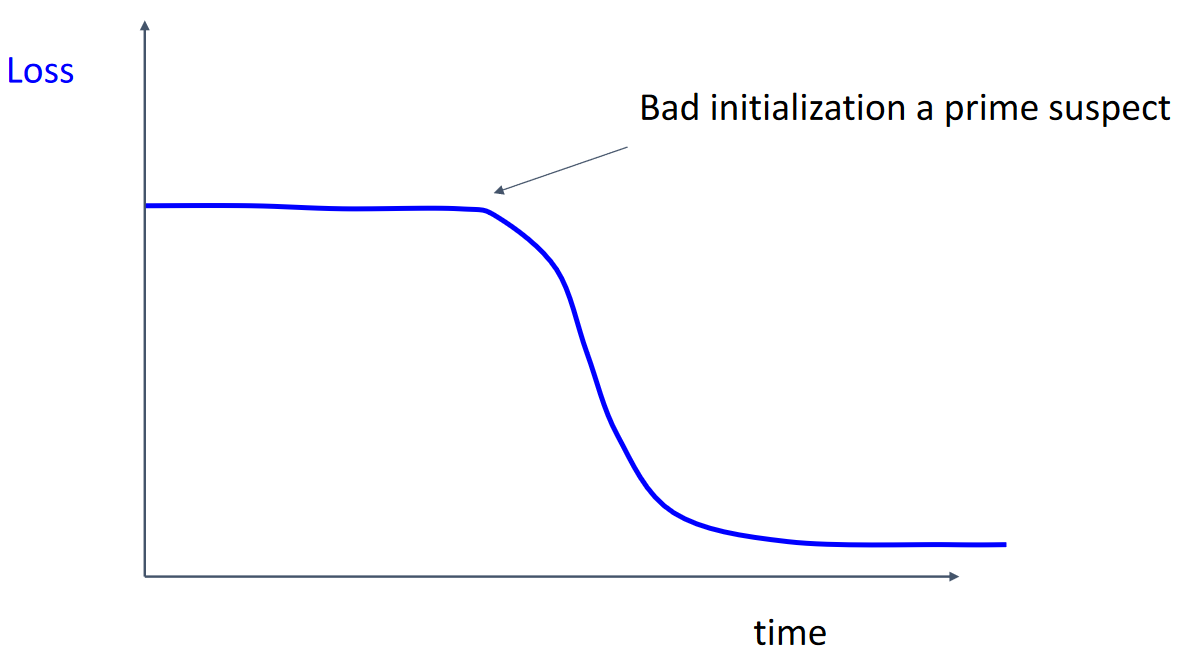
In **grid search**, we create a grid of possible combinations for hyperparameters and go over them one by one. For 2 hyperparameters, this would be a 2D matrix. The choices for values are often spaced log-linearly, e.g., , , .

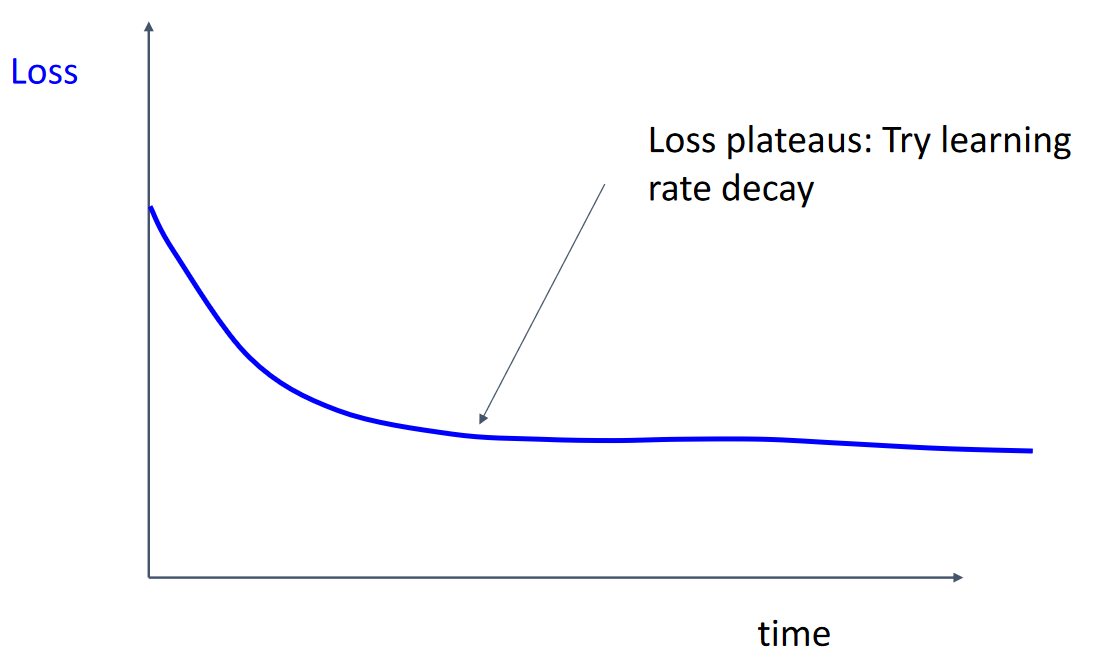
By contrast, in **random search**, we choose combinations randomly within specified ranges. This gives us the benefit of allowing us to go through more values for important parameters and using values anywhere in the range instead of at specific points.

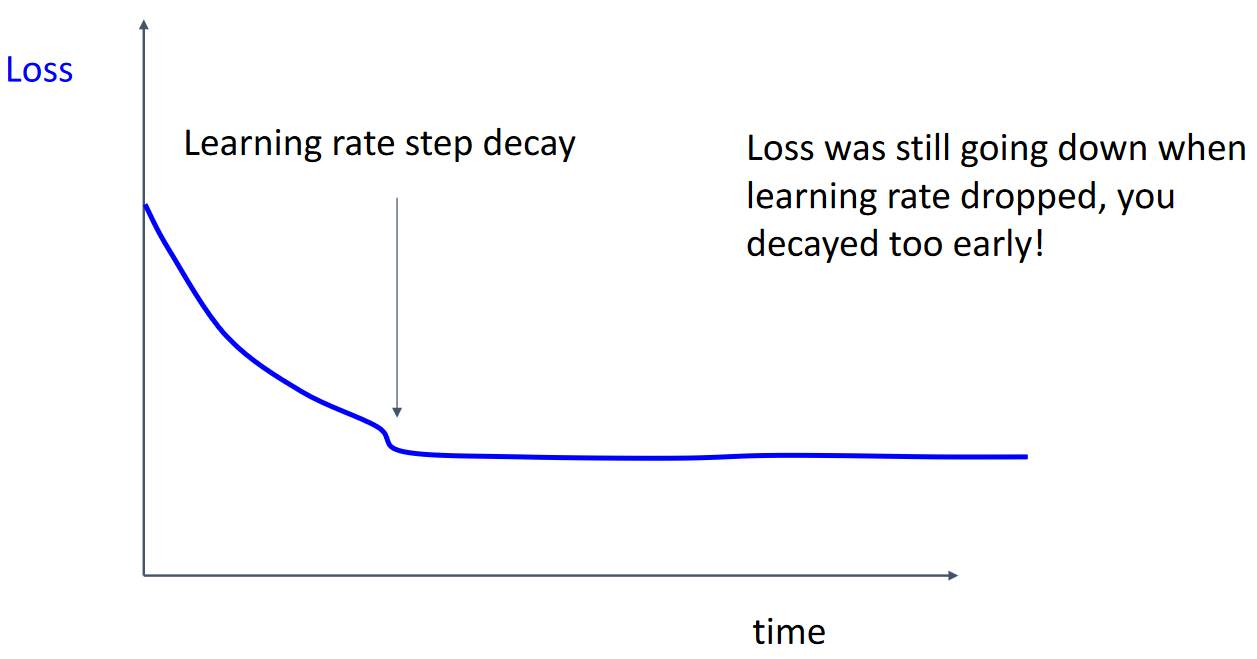


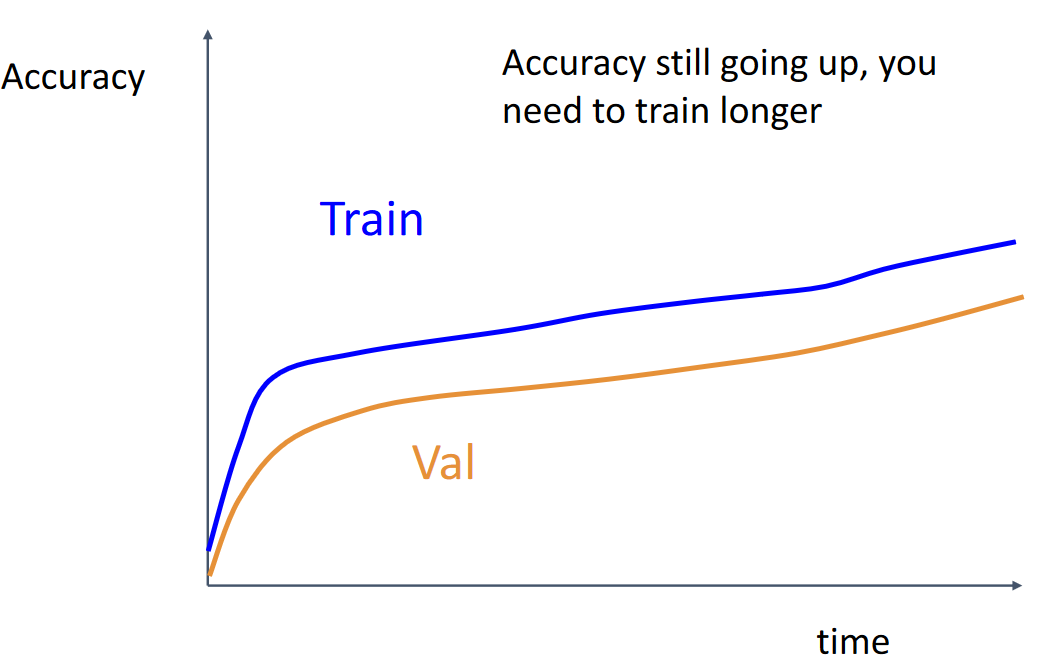
Using grid search or random search requires training the models again and again and typically requires lots of GPUs. For the average person, there are several steps we should be going over with regards to finding the correct hyperparameters.

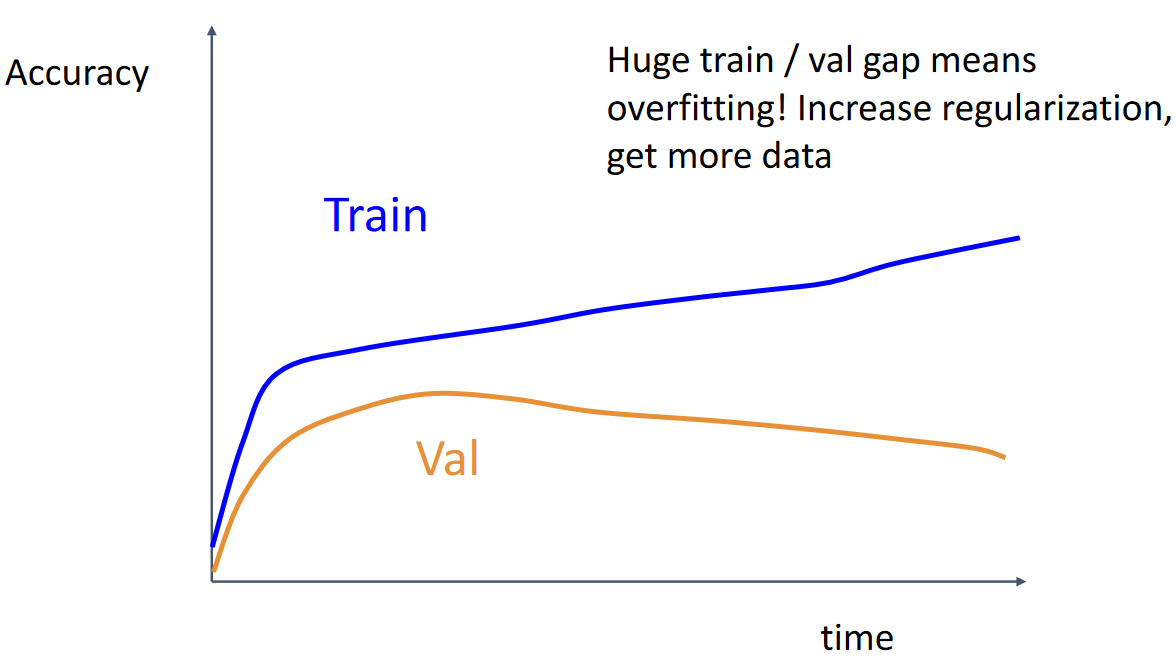
1. **Check Initial Loss** – This is done as a sanity check to find the baseline loss of the model on the data itself. We do not have any weight decay at this stage and the model has not been trained.
2. **Overfit a Small Sample** – Try to train to 100% accuracy. You can fiddle with the architecture, learning rate and weight initialization. Do not use regularization. If the loss is not going down, it means the learning rate is too low or the weight initialization is poor. If the loss explodes, it means the learning rate is too high or the weight initialization is poor.
3. **Find a Learning Rate that Decreases Loss** – Using all the training data, find a learning rate that causes the loss to decrease significantly within 100 epochs. Use a small weight decay.
4. **Coarse Grid** – Next, we need to find a combination of other hyperparameters that work well. Choose a few values for learning rate and weight decay around the ones that worked in the previous stage and train for 1-5 epochs.
5. **Refine Grid** – Pick the best setup from the previous step and train for longer (10 – 20 epochs) without any learning rate decay.
6. **Look at the Learning Curves** –

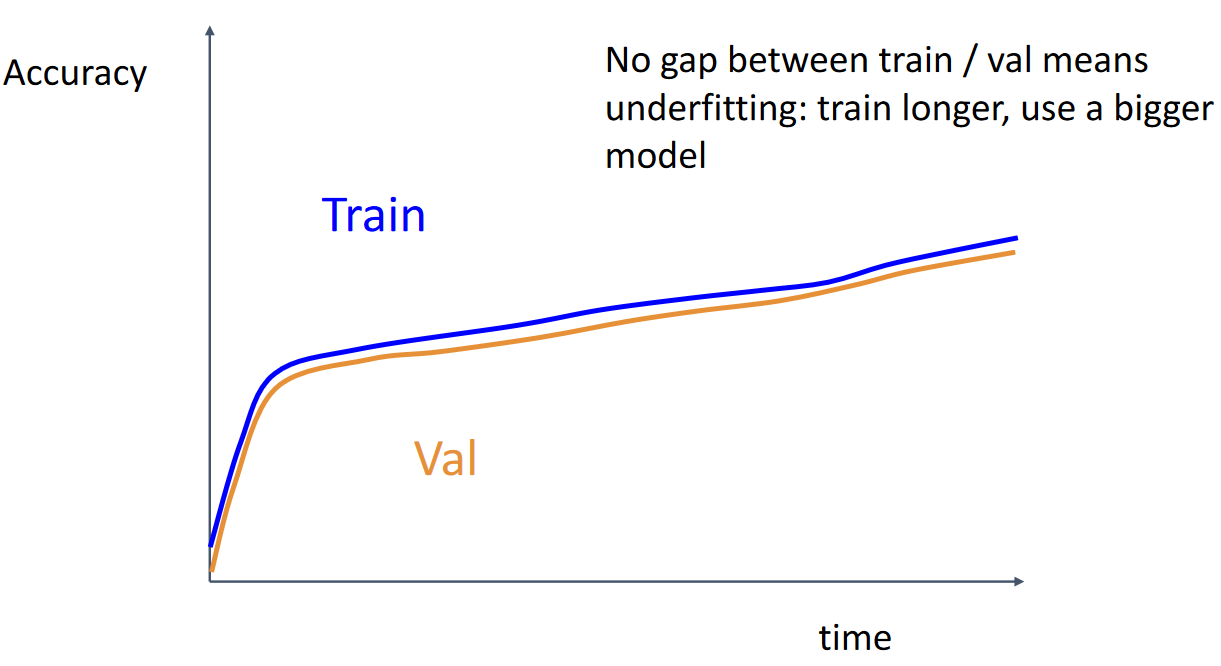












1. Go back to Step 5 and repeat until satisfied.

Typically, the hyperparameters you will end up working with most are the network architecture, learning rate, learning rate decay scheduling and regularization.

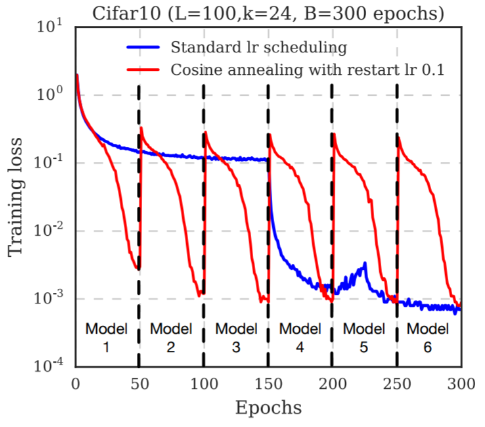
### Track Ratio of Weight Update / Weight Magnitude

Generally, we do not want our weights to be updated by an amount that is significantly larger than their magnitude. To avoid this, we should track these values and make sure that the ratio is somewhere in the range of 0.01 to 0.001.

## Model Ensembles

**Model ensembles** take multiple trained models and take their average results during test time. This results in slightly improved performance. There are a few additional tricks that can be used related to this method.

One way to get around using multiple models is to use **multiple checkpoints** of the same model. This tends to work well if we use a **cyclic learning rate scheduler**, where the learning rate is periodic (as shown below).

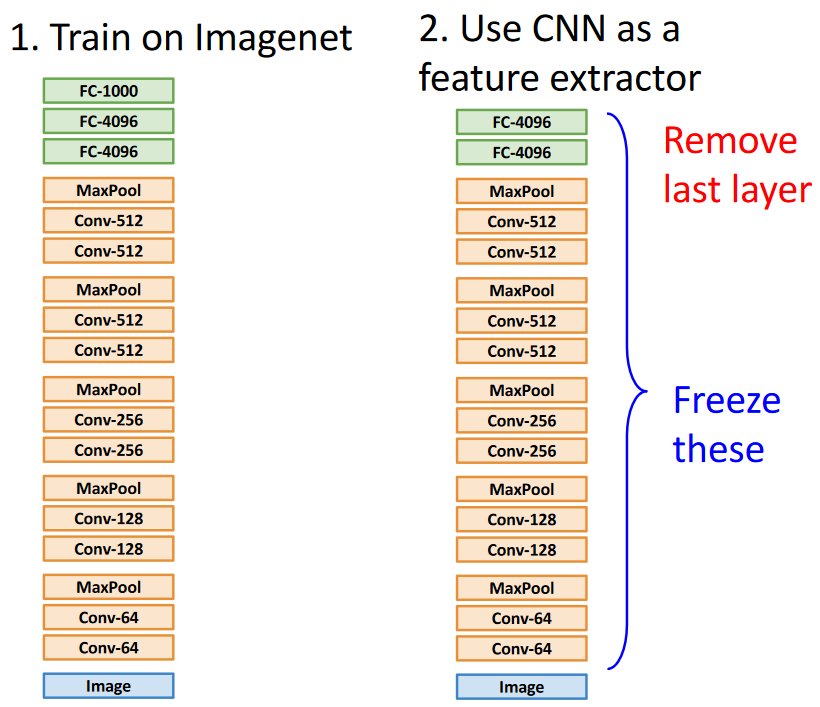


Another idea is to keep a **running average** of the model weights as we train, called **Polyak Averaging**. This average value is then used during test time.

## Transfer Learning

The concept of **transfer learning** allows us to take a model that has been trained on one task and use it for a different task. By doing this, we can bypass the need to train our model from scratch on a huge amount of data.

For example, we could take a model trained on ImageNet, freeze all the layers and remove the fully connected layers (because those are directly tied to the outputs). We can then use this model on something like Caltech-101 dataset, which will allow us to gain good performance despite not training on the correct dataset. This works because the main model has been trained on a much larger dataset and thus understands images much better. We do however have to add our own classifier to the head of the existing model.



For larger datasets for which directly using the existing model does not work well enough, we can still use the model weights as a starting point. This is called **fine-tuning** and allows us to bypass much of the initial learning phase. Note that none of the layers are frozen in this case, so they are all trained further.

A few pieces of advice when fine-tuning a model:

* You will most likely need to reduce the learning rate by a significant amount from the original model, to about 0.1 times the original value.
* It may still be beneficial to freeze at least the lower layers of the model which deal with the absolute fundamental features of what an image even is. These are unlikely to benefit from further finetuning.

Since the majority of models are initially trained on ImageNet, this gives us an easy chart to follow. The more data we have, the more layers we should fine-tune. The closer our data is to the ImageNet data (think similar looking objects being classified), the few layers we need to fine-tune. However if we do not have enough data and our data is different from ImageNet, transfer learning may not work too well.

