CSCI 5922 Fall 2022–HW2

Abulitibu Tuguluke

September 17, 2022

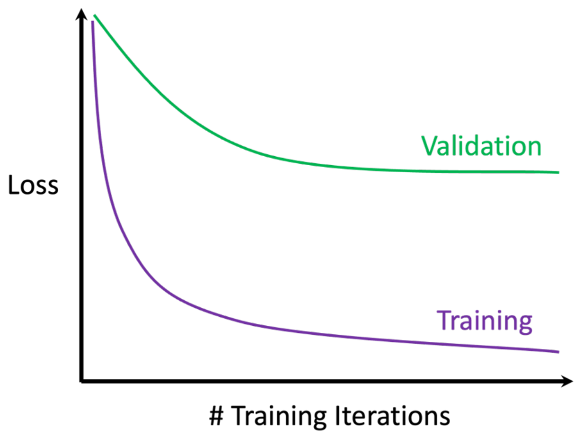
1 Model Parameters versus Hyperparameters

Parameter are function's 'coefficients' that machine will learn during mode training, such as weight $\bar{\boldsymbol{W}}$ and bias $\bar{\boldsymbol{\beta}}$. While Hyper-parameters are part of the prefix that need to tune and seleted rather than learning through iterations, which includes **Number of filters, including height and width of each, Strides, Padding type Activation function**.[1]

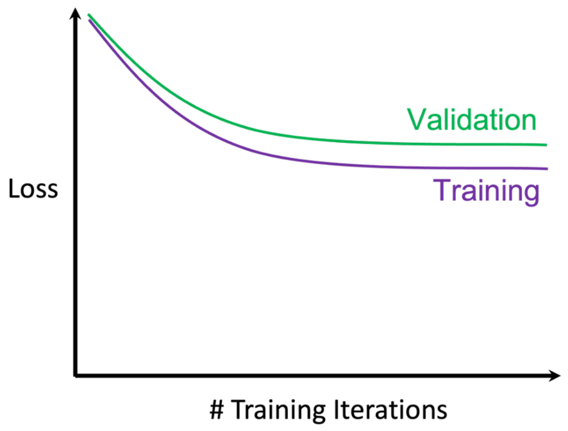
2 Dataset Splits

Splitting data into Training/Test set is just a basic requirement to get a result, yet we may still run into the issues of over-fitting which leads to bad result, but if we start consider more than weight and bias and introducing optimization techniques for hyperparameters tuning, then 'carving out' a portion of the training set and make it a validation for comparing the best performed architecture, having that 'carved' validation set is a must. It is based on the logic of Training/Test split with one more safety-nets for 'damage control'.

3 Overfitting versus Underfitting



(a)



(b)

Figure 1: Learning curves for two deep learning models trained and tested with the same training and validation datasets are shown in (a) and (b). The scales of the x-axis and y-axis in both plots are the same.

[1]Lecture slides

(a)

Sub-figure (b) is the representative of underfitting, due to remaining high loss[2] after many iterations, the model failed to learn the data features.

(b)

Sub-figure (a) is the representative of overfitting, due to disparity between training and validation loss after many iterations, which is a clear indication of losing knowledge for unobserved data in validation set.

4 Optimization

In numerical differential equations, gradient descent is an algorithm for finding local minimum, due to gradient represent the perpendicular direction to function's vector space, hence getting the result speedily, it has huge application in error analysis.
Since the computational cost is roughly $\mathcal{O}(kn^2)$, it will get super expensive and slow when training with huge m samples, probably will kill a dolphin from the emission of training the model. The word 'batch' meaning using all the training examples in each iteration. Therefore, Stochastic(meaning randomly determined) comes into play: using the calculation from one example, to save cost and memory, but this can cause the update a little 'bouncy'. Hence, mini-batch: subset of training examples. In this case it combines the 'stable-ness' from batch and 'less cost-ness' from stochastic.

5 Model Size

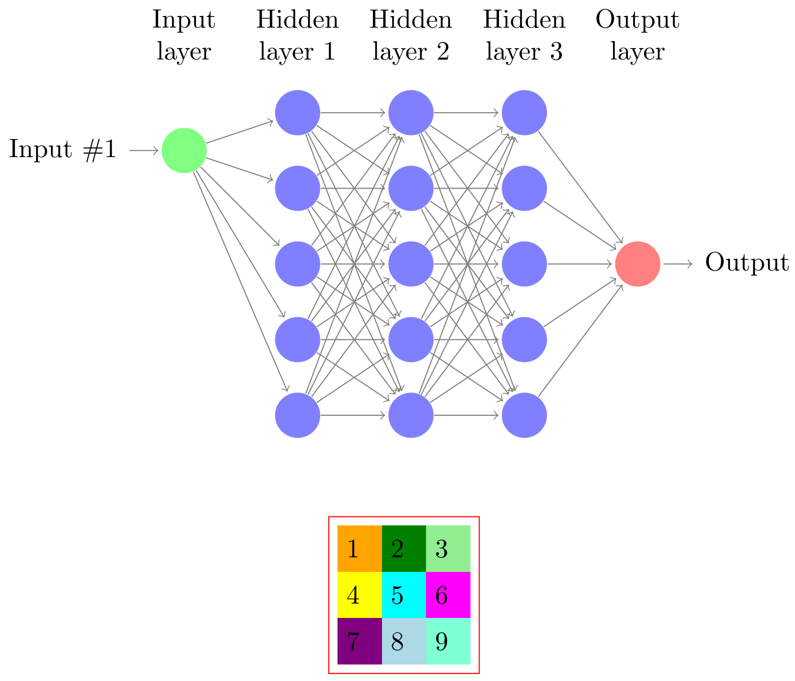


Figure 2: A 4 layer neural network (i.e., 3 hidden layers and 1 output layer) that takes as input a 64 × 64 matrix of integers and has 100 nodes at the output layer, with a 3 × 3 filters.

[2]A quantified measure of how bad it is to get an error of a particular size/direction.

(a) Fully connected with 5 nodes per hidden layer

pairs of layers \ parameters	Weights	Bias
input to hidden layer 1	$64 \times 64 \times 5 = 20480$	5
hidden layer 1 to hidden layer 2	$5 \times 5 = 25$	5
hidden layer 2 to hidden layer 3	$5 \times 5 = 25$	5
hidden layer 3 to output layer	100	100
Total sum	$20480 + 25 + 25 + 100 = 20630$	$5 + 5 + 5 + 100 = 115$

(b) Convolutional layers for all hidden layers with 3 3x3 filters per layer and a fully connected layer between the final hidden layer and last layer

pairs of layers \ parameters	Weights	Bias
input to hidden layer 1	$3 \times 3 = 9$	1
hidden layer 1 to hidden layer 2	$3 \times 3 = 9$	1
hidden layer 2 to hidden layer 3	$3 \times 3 = 9$	1
hidden layer 3 to output layer	100	100
Total sum	$9 + 9 + 9 + 100 = 127$	$1 + 1 + 1 + 100 = 103$

6 Convolutional Neural Net

1	1	0	2
4	0	8	1
6	4	2	3
8	7	4	2

Data type 1: Input

0	0.5	0
0.5	1	0
0	0	0

Data type 2: Filter

(a)

As we can see from last problem, there are less parameters (weights and bias) to learn, which translate to faster computing, further techniques such as pooling, and weight sharing, make it even faster while preserve the majority pattern (matrix/tensor) of inputs.
Another advantage CNN has, over other architectures, is the easy adaptation of parallelization, it is not really sequential feedback, which again, leads to speed up, especially if the nodes numbers are 2 exponential.

(b)

$$\begin{array}{|c|c|} \hline \begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ \hline 4 & 0 & 8 \\ \hline 6 & 4 & 2 \\ \hline \end{array} & \begin{array}{|c|c|c|} \hline 1 & 0 & 2 \\ \hline 0 & 8 & 1 \\ \hline 4 & 2 & 3 \\ \hline \end{array} \\ \hline \begin{array}{|c|c|c|} \hline 4 & 0 & 8 \\ \hline 6 & 4 & 2 \\ \hline 8 & 7 & 4 \\ \hline \end{array} & \begin{array}{|c|c|c|} \hline 0 & 8 & 1 \\ \hline 4 & 2 & 3 \\ \hline 7 & 4 & 2 \\ \hline \end{array} \\ \hline \end{array} \textbf{ Conv } \begin{array}{|c|c|c|} \hline 0 & 0.5 & 0 \\ \hline 0.5 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1*0.5+ 4*0.5 & 8*1 \\ \hline 6*0.5+4*1 & 8*0.5+ 4*0.5+2*1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 2.5 & 8 \\ \hline 7 & 8 \\ \hline \end{array}$$

(c)

0	0	0	0	0	0
0	1	1	0	2	0
0	4	0	8	1	0
0	6	4	2	3	0
0	8	7	4	2	0
0	0	0	0	0	0

Adding a zero padding layer

3×3 Stride:

0	0	0	0	0	0
0	1	1	0	2	0
0	4	0	8	1	0
0	6	4	2	3	0
0	8	7	4	2	0
0	0	0	0	0	0

Conv

0	0.5	0
0.5	1	0
0	0	0

=

1*1	2*1
6*0.5+8*1	3*0.5+ 4*0.5+2*1

=

1	2
11	5.5