METU EE 496
Introduction to Computational Intelligence

Training Multi-Layer Perceptron

Homework 1

Report

31.03.2019

Muttalip Caner TOL

2031466

1. Basic Concepts

1.1. Which Function?

MLP with one hidden layer and one hidden neuron learns the function in f(x) given in Equation 1, where W_1 and W_2 are the weights of the input layer and hidden layer, and b_1 and b_2 are the biases of the hidden and output layers respectively. "g" is the hyperbolic tangent function. However, if the number of neurons gets larger, the function that MLP approximates would be more complex.

$$f(x) = W_2 g(W_1^T x + b_1) + b_2$$

Equation 1

For classification MLP Classifier uses "Cross entropy loss function", which is given in the Equation 1.

$$Loss(\hat{y}, y, W) = -y \ln \hat{y} - (1 - y) \ln (1 - \hat{y}) + \alpha ||W||_2^2$$

Equation 2

1.2. Gradient Computation

Gradient of the loss is given in the Equation 3.

$$(\nabla_\omega \mathcal{L}|_{\omega=\omega_k}) = \frac{\omega_k - \omega_{k+1}}{\gamma}$$

Equation 3

1.3. Some Training Parameters and Basic Parameter Calculations

1.3.1. **Batch** is a hyperparameter that indicates after how many samples the weights are updated while training the model.

Epoch is a hyperparameter that indicates how many times the learning algorithm passes over the whole training samples.

1.3.2.

$$Number\ of\ batches = \frac{Number\ of\ samples}{Batch\ size} = {}^{N}/_{B}$$

1.3.3.

$$Number\ of\ iterations = Number\ of\ epochs\ x \frac{Number\ of\ samples}{Batch\ size} = {}^{E.N}/_{B}$$

1.4. Computing Number of Parameters of an MLP Classifiers

$$\#\ of\ neurons = \sum_{k=1}^{K} H_k + D_{out}$$

$$\#\ of\ weights = H_1 * D_{in} + \sum_{k=2}^{K} (H_k * H_{k-1}) + H_K * D_{out}$$

$$\#\ of\ parameters = H_1 * D_{in} + \sum_{k=2}^{K} (H_k * H_{k-1}) + H_K * D_{out} + \sum_{k=1}^{K} H_k + D_{out}$$

2. Experimenting MLP Architectures

2.1. Experimental Work

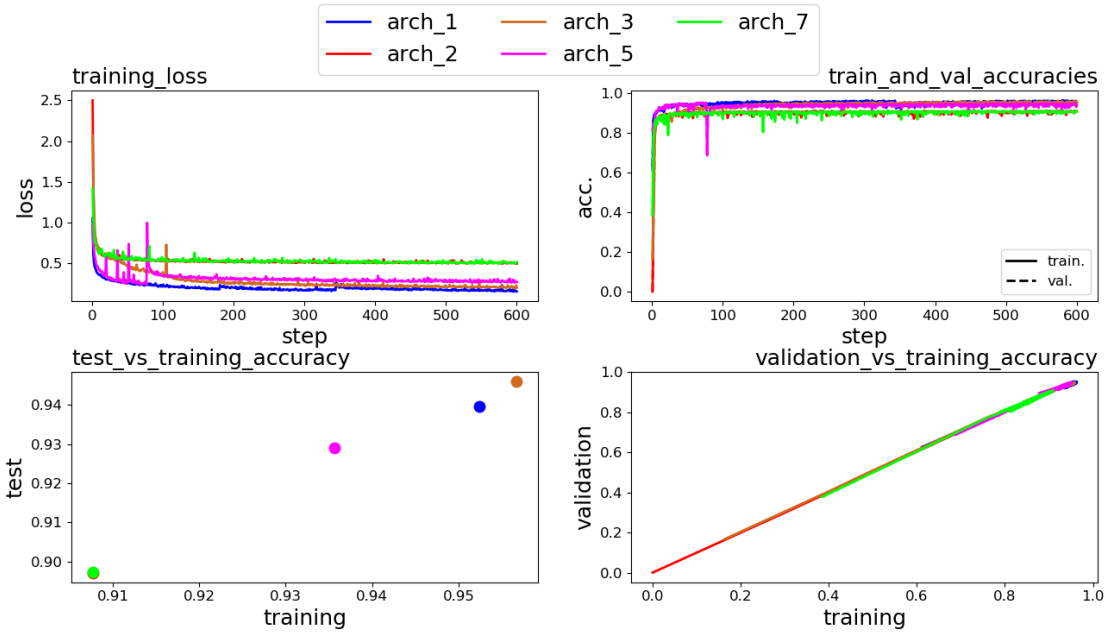


Figure 1 (Top left) Training loss curve, (Top right) Training and validation accuracy curves (Bottom left) Test vs Training accuracies, (Bottom right) Validation vs Training accuracies

2.2. Discussions

2.2.1. Generalization performance of a classifier indicates how well a classifier predicts a dataset on which it is never trained.

2.2.2. Test vs training accuracy graph on Figure 1 is informative to inspect generalization performance since we have never trained the model using the test dataset.

2.2.3. Generalization performances of the architectures can be ordered as follows:

Arch3 > Arch1 > Arch5 > Arch7 >Arch2

2.2.4. There is no general rule about it. However, if the number of parameters is unnecessarily large for the dataset, overfitting can occur.

2.2.5. There is no general rule about it. If the number of classes are large and dataset is very large and complicated, it is good for generalization performance to have a large depth of architecture. However, if the model's depth unnecessarily large for the dataset, overfitting can occur.

2.2.6.

2.2.7. Yes, units are specialized to specific classes.

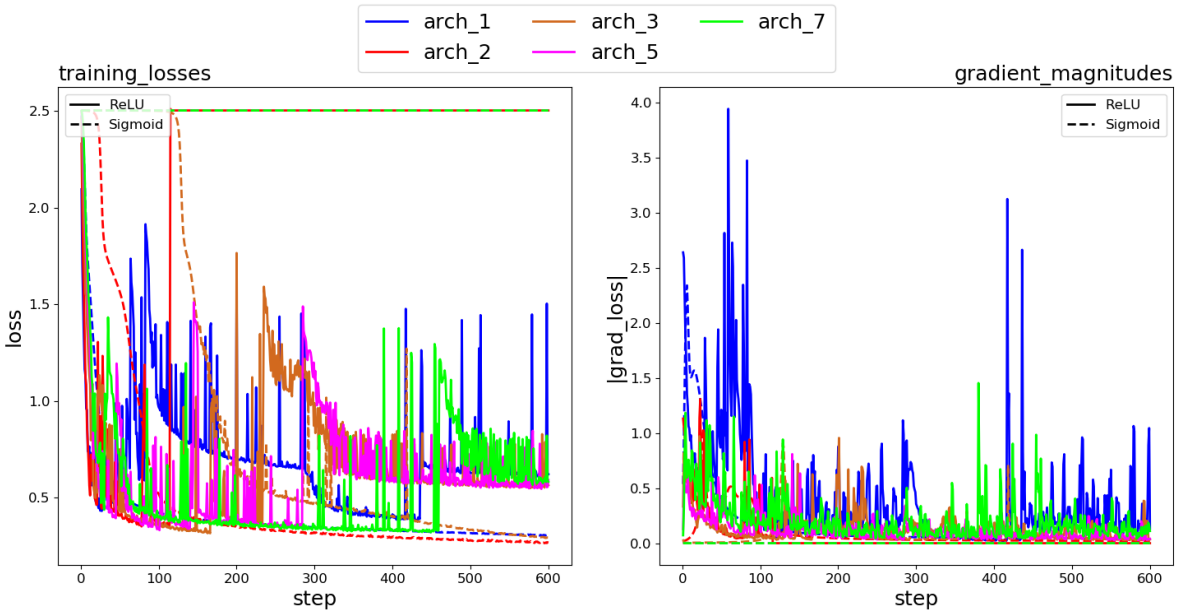
2.2.8.

2.2.9. Some of the architectures are designed to be symmetric, i.e. there is a separating hidden layer in the middle and there are equal number of hidden layers with same sizes on left and right sides of the separating hidden layer. Arch1, Arch3 and Arch7 have this property. Arch1 and Arch3 has similar accuracies.

2.2.10. I would pick Arch3 because it has the largest test accuracy. Moreover, it is one the models which has the lowest training loss and the largest training and validation accuracy.

3. Experimenting Activation Functions

3.1. Experimental Work



3.2. Discussions

3.2.1. When ReLU is used, the gradient in Arch1 has lots of large spikes and it looks like it is not converged.

However, when logistic sigmoid is used, gradients are more stable after a number of epochs. In Arch1, Arch2 gradient decreases smoothly and gets closer to zero. In Arch3, there are a few spikes around 6^{th} and 70^{th} epochs and after that gradient converges to zero. In Arch5 and Arch7 has almost zero gradient all of the times.

3.2.2. The gradient vanishes as the depth of the network gets larger. This problem is called in the literature as “Vanishing Gradient Problem”. Since the derivative of the sigmoid goes to zero when the input of the sigmoid activation function is large or small in the negative region as in Figure 2.

This is not a problem in the Arch1, Arch2 and Arch3 since they use the

activation function only a few times in the hidden layers.

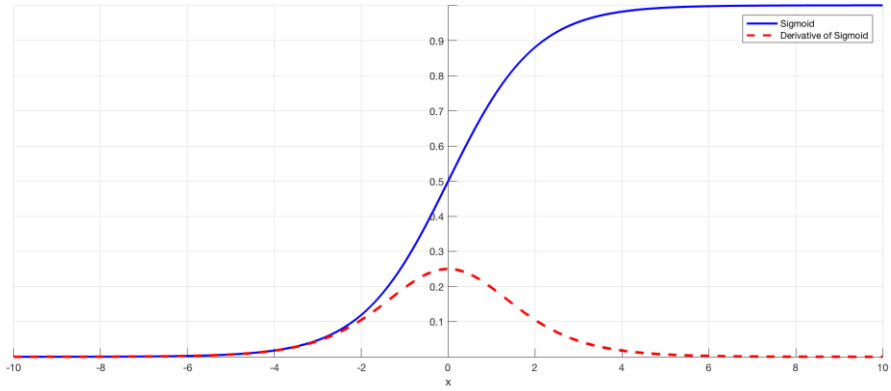


Figure 2 Sigmoid function and its derivative

This problem is not seen when ReLU activation function is used. Becase its derivative is constant.

3.2.3. If we do not scale the inputs between (-1,1), inputs with larger pixel values would be understood as more important and small pixel values would be understood as less important. Normalizing the inputs, therefore, uniformly distributes the importance to each of the pixels.

4. Experimental Learning Rate
   4.1. Experimental Work

   4.2. Discussions