

---

## 6.867: Homework 3

---

### 1. Neural Networks

In this section, we explore neural networks and various choices for the number of hidden layers, the number of neurons per hidden layer, regularization, binary classification, and multiclass classification. We investigate how our choice of these variables impacts the overall classification rates and how these decisions are incorporated into our implementation of neural networks.

#### 1.1. ReLU + Softmax

We have here implemented a neural network in which all the hidden units have a ReLU activation function and the final output layer has a Softmax activation function. This is incorporated into the implementation where in the output layer, the activation  $\alpha_i$  for class  $i$  is determined by the Softmax function:

$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

Thus, for a  $k$ -class classification problem, the output layer has  $k$  neurons each with Softmax activation. The prediction is therefore given by the argmax of the  $\alpha$  values calculated by the Softmax for each of the  $k$  neurons.

The cross entropy loss is the loss function used here and is given by:

$$-\sum_{i=1}^k y_i \log(f(z)_i)$$

This formulation of the loss function is essentially a calculation of our confidence in the correct classification. In other words, the loss is the log likelihood of the correctly classified class by the weights assigned in our training of the neural network model. This loss is incorporated into the implementation via the output error. This output error is then involved in backpropagation to determine  $\delta$  values for each layer in the neural network, which is subsequently used to calculate the gradient used for stochastic gradient descent. Thus, our choice of the Softmax function as the output activation function and the cross entropy loss as the loss function in this case is critically important in

	$l = 1, n = 5$	$l = 1, n = 100$	$l = 2, n = 5$	$l = 2, n = 100$
1	0.718	0.997	0.75	0.995
2	0.503	0.782	0.805	0.825
3	0.512	0.953	0.885	0.94
4	0.5	0.944	0.676	0.94

Table 1. Training accuracy,  $l$  layers,  $n$  neurons/layer

our overall implementation. Different choices of these functions could fundamentally change the training and thus classification accuracy of our overall model.

#### 1.2. Initialization

We can initialize the weights to be randomly selected from a Gaussian distribution with zero mean and standard deviation  $\frac{1}{\sqrt{m}}$  for an  $m \times n$  weight matrix. Choosing to initialize weights to these values is a reasonable decision because  $m$  represents the number of input neurons in each of our layers. This is selected to control the variance of the distribution in order to ensure that each layer in the neural network is agnostic to the number of inputs it receives, and instead depends on the values of the inputs.

#### 1.3. Regularization

This would impact the pseudocode because now we must also incorporate this additional regularization term when taking the gradient of the loss function. Since this gradient is used in the stochastic gradient descent update it ultimately impacts the entire algorithm.

#### 1.4. Binary classification

We test our implementation on the 2D datasets provided in HW2. The accuracies for the training and testing data sets are presented for different architectures of the neural network in Tables 1 and 2. It is important to note that the accuracies varied significantly between multiple runs of the algorithm. This is because of the randomly initialized weights, so the accuracies presented are some examples of the results of our algorithm, but may reflect slight variation due to the random initialization.

	$l = 1, n = 5$	$l = 1, n = 100$	$l = 2, n = 5$	$l = 2, n = 100$
1	0.71	0.99	0.74	0.995
2	0.49	0.785	0.805	0.815
3	0.48	0.945	0.89	0.955
4	0.5	0.945	0.683	0.938

Table 2. Testing accuracy,  $l$  layers,  $n$  neurons/layer

### 1.5. Multi-class classification

## 2. Convolutional Neural Networks

In this section, we explore various versions of the dual form of support vector machines, first with slack variables and then with generalized kernel functions.

### 2.1. Convolutional filter receptive field

### 2.2. Run the Tensorflow conv net

### 2.3. Add pooling layers

### 2.4. Regularize your network!

### 2.5. Experiment with your architecture

### 2.6. Optimize your architecture

### 2.7. Test your final architecture on variations of the data