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 α_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 δ 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 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 mxn weight matrix. Choosing to intiialize 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

¡¡¡¡¡¡¡ HEAD This would impact the pseudocode because now we must also incorporate this additional regularization term when doing the stochastic gradient descent update. Specifically, the update step before regularization was:

$$\theta \leftarrow \theta - \eta \frac{\delta l(y, F(x;\theta))}{\delta\theta}$$

With regularization, the update step now becomes:

$\theta \leftarrow \theta - \eta\frac{\delta J}{\delta\theta}$ where $J(\theta) = L(\theta) + R(\theta)$ and $R(\theta) = \lambda * (\sum_{ij} w_{ij}^{(1)2} + \sum_{ij} w_{ij}^{(2)2})$. Furthermore, $J'(\theta) = L'(\theta) + R'(\theta)$. $L'(\theta)$ has already been calculated and $R'(\theta) = 2 * \lambda * \sum_{ij} w_{ij}$ ======= 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. ¿¿¿¿¿¿¿ 19bccd1f93e56c5a76074bc3d45db80909a1b5a3

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

	$l = 1, n = 5$	$l = 1, n = 100$	$l = 2, n = 5$	$l = 2, n = 100$
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	0.9[illegible]

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

	$l = 1, n = 5$	$l = 1, n = 100$	$l = 2, n = 5$	$l = 2, n = 100$
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

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 initializaation.

1.5. Multi-class classification

2. Convolutional Neural Networks

In this section, we explore the application of convolutional neural networks in performing image classification.

2.1. Convolutional filter receptive field

The dimensions of the receptive field for a node in Z_2 is $7x7$. With this in mind, we see that it is effective to build convolutional networks deeper in order to increase the receptive fields for subsequent nodes. This is effective because it allows the network to learn additional features that take into account larger and larger amounts of the original image's pixels.

2.2. Run the Tensorflow conv net

In the "define_tensorflor_graph" graph function we see that there could be a maximum of 6 layers and a minimum of 4 layers. Only 2 of the layers are convolutional. 2 other ones are regular hidden ones and the 2 optional ones are pooling layers. The relu activation function is used on the hidden nodes. The softmax loss function with cross entropy is being used to train the network. The loss is being minimized with gradient descent. When the convolutional network is run, we get a batch training accuracy of 100% and a validation accuracy of 63.2%. What this means is that the network is classifying extremely well on the training data, while not so well on the validation datasets, which points towards overfitting.

2.3. Add pooling layers

By adding pooling layers to our convolutional network, our results definitely changed. Specifically, we used a layer 1 pool filter size of 5, a layer 1 pool stride of 2, a layer 2 pool filter size of 2, and a layer 2 pool stride size of 2. After the 1500th step, this achieved a batch training accuracy of 100% and a validation accuracy of 69%. Interestingly enough, after the 1400th step, the training accuracy was 70% and the validation accuracy was also 70%. With these results it appears that max pooling marginally reduces the training accuracy while improving the validation accuracy. In a sense, it almost serves as a form of regularization that is able to prevent overfitting because now instead of the final layer processing information from all the original pixels the pooling allows clusters of pixels to be expressed as one value which enables better generalization.

2.4. Regularize your network!

2.4.1. Dropout

2.4.2. Weight regularization

2.4.3. Data augmentation

2.4.4. Early stopping

2.5. Experiment with your architecture

2.6. Optimize your architecture

2.7. Test your final architecture on variations of the data