6.867 Machine Learning Homework 3

1 NEURAL NETWORKS

In this paper, we explore the performance of Neural Networks on different classification tasks. Neural Networks were invented to try to model how the neurons work in the human brain. A neural network consists of multiple layers of neurons, each of which takes a linear combination of its inputs and uses that to compute an output.

1.1 Implementation

In our implementation of a neural network, we represented the the weights of each layer as an m x n matrix. Where m is the size of the previous layer and n is the size of the current layer. Each column in the matrix represented the weights for a single neuron in the layer. The bias term for each neuron was captured in a separate n x 1 vector. We used ReLU activation functions for the neurons in the hidden layers and a softmax activation function for the last layer. The loss function we used was the cross entropy loss function, which is $\sum_i -y_i log(f(z)_i)$. Where $f(z)$ is the output of the softmax function. We used stochastic gradient descent to train the parameters of the network, which in this case are the weights and biases of each layer. Computing the gradient of a neural network with respect to the weights is a very complicated operation. Fortunately, the derivative can be found using backpropagation. Backpropagation finds the gradient of the loss function with respect to each layer by propagating the error back through the network. The first step in this is computing the graident of the loss function with respect to the output layer. A convenient property of the combination of a softmax output layer and the cross entropy function is that the derivative of the cost function with respect to the softmax function simplifies to $f(z) - t$ where $f(z)$ is the output of the softmax and t is the target vector. smaller.

1.2 Section 2

1.2.1 sub sub sub section

1.3 Install Latex

2 CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks(CNNs) are similar to feed forward neural networks but are better suited for the task of image recognition. Traditional neural networks use fully connected layers and hidden units to classify inputs. For even small 50x50 pixel images though, the squared factor of weights between layers that need to be trained quickly makes this problem untractable for any reasonably size dataset. CNNs solves this problem by recycling parameters with a concept called filters. A filter is a fxf matrix of weights that gets applied to every possible fxf square in the nxn input image to generate another nxn output. Similar to NNs, these outputs then also undergo some nonlinear transformation. Many filters can be applied in parallel to the same input and the output of filters can be fed into the input of other filters. In this manner, CNNs boost a diverse set of possible architectures have made remarkable progress in image recognition.

In this section, we will explore the effect of different filter sizes, architectures, pooling, and regularization techniques on CNNs. The task was classifying images of paintings to their artist.

2.1 Architecture

Paintings were padded to 50x50 pixel RGB images and inputed as a three layer image. We fed these images through various CNNs implemented with TensorFlow. A CNN can consist of multiple layers and each layer takes three arguments, filter size, stride, and depth. Stride is how many pixels the filter shifts each time and is typically 1 or 2. Depth is the number of filters working in parallel at that layer. Note that if a layer has a depth of 8, a filter of size 3 in the next layer actually has $9*8$ nodes feeding into it. The receptive field of a node is defined as the number of pixels in the original image that contribute to it. For example, if a image was fed through a 5x5 filter layer and then a 3x3 filter layer, the receptive field of a node in the last layer would be 7x7. This means that that nodes can only activate in response to features in the original image of size up to their receptive field. Adding more layers increases the receptive field size and allows the last layer to decide to activate based on more complex patterns.

After the convolutional layers, we also added one fully connected layer with 64 hidden units that was also fully connected to the output to help with classification. The network was trained with stochastic gradient descent with batch size of 10.

2.1.1 Baseline

Our baseline was two layers, both with a filter size of 5, stride of 2, and a depth of 16. This achieved a classification accuracy of 63%.

2.1.2 Pooling

2.1.3 Filter Size and Stride Length

We first started experimenting with different filter sizes and stride lengths while keeping both layers the same.

	Filter = 3	Filter = 5	Filter = 9	Filter = 15
Stride = 1	0	0	0	0
Stride = 2	0	0	0	0
Stride = 3	0	0	0	0
Stride = 4	0	0	0	0

We found that decreasing the stride length to 1 dramatically increased runtime without much gain in accuracy while increasing the stride length could quickly decreased accuracy. Increasing the filter sizes of both layers did not significant improve perfomace past 5 but increased runtime.

Thus for any layer, the optimal filter size and stride length combinations for good performance and runtime were $(3,1)$, $(5,2)$ and $(7,2)$.

2.1.4 3 Layers

At first, the jump to 3 layers did not yield any notable improvement. Using three uniform layers, with filter sizes of 5 and depth of 16,

We then tried different architectures and the results are documented in the table below. Note that a filter size of 5 implies a stride length of 2, etc.

Layer 1	Layer 2	Layer 3	Classification Accuracy	Training Accuracy	Training Time
(5, 16)	(5,16)	(5,16)	0	0	40 sec
(7, 16)	(7,16)	(7,16)	0	0	40 sec
(7, 8)	(5,16)	(3,32)	0	0	40 sec
(3, 4)	(5,8)	(7,16)	0	0	40 sec
(3, 8)	(5,16)	(7,32)	0	0	40 sec

The uniform architectures took did not significant outperform 2 layer architectures but took much more time to train. One idea was to stack the

2.2 Regularization

Explain dropout.

	Dropout = 0.5	Dropout = 0.7	Dropout = 0.9	Dropout = 1.0
Classification Accuracy	0	0	0	0
Training Accuracy	0	0	0	0

Dropout not very effective. Data augmentation only effective

2.3 Results

Our optimal CNN architecture is a 3 layer pyramid. Words

Figures/P2/svm_data1_test_C1.png Figures/P2/svm_data2_test_C1.png Figures/P2/svm_data3_test_C1.png Figures/P2/svm_data4_test_C1.png

(a) Dataset 1, C=1 (b) Dataset 2, C=1 (c) Dataset 3, C=1 (d) Dataset 4, C=1

Figure 1: Linear SVM perfomance of different datasets. Datasets 1 and 3 are relatively linearly separable so this SVM performs well on them. Dataset 2 and 4 are definitely not linearly separable so this SVM is not the right model for the data.