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.

2.1.1 Baseline

Our architecture consisted of one input layer, two convolutional layers, one fully connected layer and one output layer. The fully connected layer contains 64 hidden units and is there to help the output layer with classification. All nodes use RELU activation and output layer uses softmax. The loss function is softmax cross entropy loss ???

hi

The network was trained with stochastic gradient descent with batch size of 10.

In this paper, we only explore the effects of changing the convolutional layers. The baseline contains was two convolutional layers, both with a filter size of 5, stride of 2, and a depth of 16. This achieved a classification accuracy of 63% and a training accuracy of 97%. This means that our model is severely overfitting the training set.

2.1.2 Pooling

The ReLu activation function on each node in the CNN is often times not selective enough to isolate the important features in an image. Max pooling is an idea that aims to improve selectivity by amplifying high activation and throwing away low activation. After every convolution layer, we add a pooling layer. Max pooling, like filtering, also acts on a fxf window but outputs not a linear combination of the nodes but the max. In this manner, only the large activations survive. Pooling also has a stride length. We applied the same pooling after all the layers.

	Size = 2	Size = 3	Size = 4
Stride = 1	70.1	67.8	66.7
Stride = 2	64.4	63.2	65.2
Stride = 3	50.6	54.4	54.2
Stride = 4	51.7	49.6	50.2

Notice that as the stride length increase and the filter size increases, the max pooling becomes more and more aggressive, meaning it throws away more and more data. As you can see in the table above, the classification accuracy gets worse. We found that for two layers, the optimal stride length was 1 and filter size was 2 and it had an accuracy that was a significantly better than our baseline.

2.1.3 Filter Size and Stride Length

Next, we experimented with convolutional layers of different filter sizes and stride lengths. We kept the max pooling at window size 2 and stride 1 and used two identical convolutional layers. The results are summarized in the table below.

	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

As you can see, accuracy was consistantly the best when stride length 1. The trend with filter size is less convincing. The more significant tradeoff here is runtime. For filter size 15 and stride 1, the CNN took upwards of 15 minutes to train. We noticed however that increasing the filter sizes beyond 5 did not significantly improve perfomace and for large filter sizes, stride lengths of 1 and 2 had roughly the same performance. Thus we found the optimal filter size and stride length combinations for convolutional layers were $(3,1)$, $(5,2)$ and $(7,2)$.

2.1.4 3 Layers

We thought increasing the complexity of the architecture might improve our performance so we tried three convolution layers. Unfortunately at first, the jump did not yield any notable improvement. Using three uniform layers with filter sizes of 5 and depths of 16, we achieved a classification accuracy of .

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 significantly outperform 2 layer architectures but took much more time to train. One idea was to stack the

2.2 Regularization

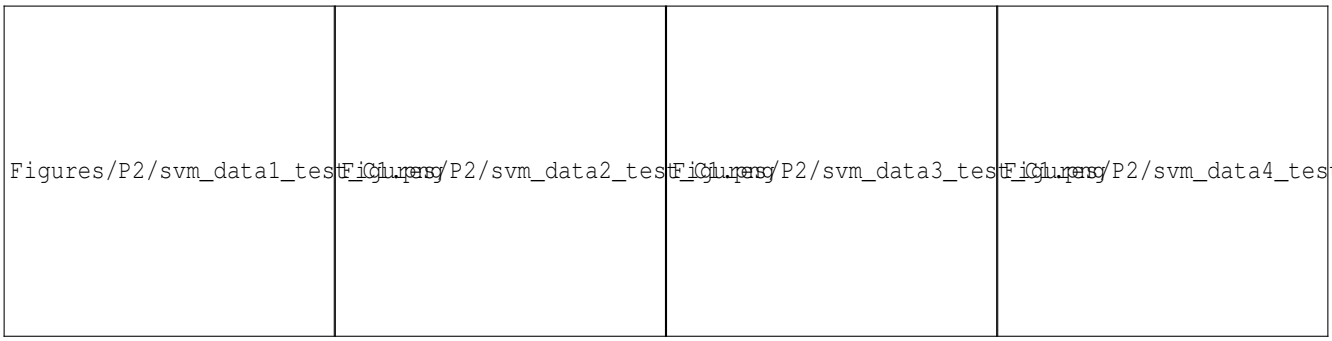
Across all architectures, overfitting the training data was a common theme. With no regularization, the CNNs commonly achieved training accuracy of close to 100%. Validation accuracy only ever reached the high 60s. Explain dropout.

	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



(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.