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.

How the weights are initialized plays an important part in the training of neural networks. In the case where the hidden units have sigmoid functions, if the weights are too small then the nonlinearity of the sigmoid funciton is lost. This is because the derivative of the sigmoid function is approximately linear for inputs close to zero. If the weights are too large, then the derivative of the sigmoid approaches zero, which can negatively impact the gradient descent, causing it to converge to a suboptimal value, or to take too long to train. One approach to handle this problem is to initialize the weights to come from a Gaussian distribution with zero mean and variance of $1/m$ where m is the number of units in the previous layer. For a linear neuron, we have that $y = \sum_i w_i x_i + b$. Calculating the variance gives us Var(y) = Var($\sum_i w_i x_i$). Var($w_i x_i$) = $E(w_i)^2$Var(w_i) + $E(w_i)^2$Var(x_i) + Var(w_i)Var(x_i). If we assume that the inputs have zero mean, then the expression simplifies to Var($w_i x_i$) = Var(w_i)Var(x_i). Substituting into the originial equation gives us Var(y) = $\sum_i Var(w_i)Var(x_i)$. Since they are identically distributed, we can rewrite it as Var(y) = mVar(w_i)Var(x_i). Thus, setting Var(w_i) = $1/m$ guarantees that the variance of the input will be equal to the variance of the output under these assumptions. This helps stabilize the signals that are propagated through the network.

Neural networks often overfit to the training data becuase they have many parameters. One way to stop the model from overfitting is to use regularization. Regularization aims to prevent overfitting by penalizing the size of the weight vector. This is implemented by adding the term $\lambda(\sum_j \left\| w^{(j)} \right\|_F^2)$ to the cost function, where w^j represents the weights at layer j. Integrating this with our implementation only involved adding a λw term to the gradient of the loss function at every layer. Thus, the gradient of the loss function with respect to the weights at layer i is $\frac{\partial C}{\partial w_i} + w_i$. The learning rule for stochastic gradient descent becomes $w \rightarrow (1 - \eta\lambda)w - \eta\frac{\partial C}{\partial w}$ where C is the cross entropy loss function.
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 sized 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 fxf squares in the nxn input image to generate an output image. Similar

to NNs, these outputs then also undergo some nonlinear transformation. Many filters can be applied in parallel to the same input and the outputs of filters can be fed as inputs of other filters. In this manner, CNNs boost a diverse set of possible architectures and have enabled 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 Set Up

Paintings were shrunk and 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 larger and more complex patterns.

2.1.1 Baseline

The architecture consists of one input layer, two convolutional layers, one fully connected layer and one output layer. The fully connected layer contains 64 hidden units and helps the output layer with classification. All nodes use RELU activation and output layer uses softmax. The loss function is softmax cross entropy loss(See Section 1 for equation). 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 98%. This means that our model is severely overfitting the training set and is not generalizing well.

2.2 Pooling

The ReLu activation functions in the CNN alone are oftentimes not good enough at isolate the important features in an image. Max pooling is an idea that aims to improve the selectivity of a CNN by actively amplifying high activation signals and throwing away low activation noise. After every convolution layer, a pooling layer, like filtering, acts on a fxf window to outputs not a linear combination of the nodes but the max. This way, only the large activations survive.

We tested out pooling on our CNN with diffent filter sizes and stride lengths. For simplicity, we used the baseline architecture and applied the same pooling after both the convolutional layers.

	Size = 2	Size = 3	Size = 4
Stride = 1	69.1	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

As you can see in the table above, as stride length and the filter size increase, the accuracy goes down. This is because the max pooling becomes too aggressive and throws away too much data. We found that for correctly chosen values, the classification accuracy improved substantially over our baseline, from 63% to 67%. The optimal stride length was 1 and filter size was 2.

2.3 Filter Size and Stride Length

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

	Filter = 3	Filter = 5	Filter = 9	Filter = 15
Stride = 1	16	69	68	N/A
Stride = 2	67	69	70	68
Stride = 3	66	68	69	68
Stride = 4	63	59	64	66

The accuracy also seems to improve with filter size but not by much past 5. The more significant tradeoff here was runtime. Stride lengths of 1 took a really long time to train. Small filter sizes and large stride length take less time to train. For filter size 15 and stride 1, the CNN took an impractical amount of time. Since there is not much reason to use filter sizes larger than 5 and for most filters, stride lengths of 1 and 2 had roughly the same performance, we decided the optimal filter size and stride length combinations for convolutional layers were $(5, 2)$ and $(7, 2)$.

2.4 3 Convolutional Layers

With two layers, the best performance we achieved was only 68% validation accuracy. Thus, we tried increasing the complexity of our network. We used three convolutional layers as opposed to two and varied their parameters. The results are shown 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
(5, 16)	(5,16)	(5,16)	66	99
(7, 16)	(7,16)	(7,16)	67	99
(7, 8)	(5,16)	(3,32)	70	99
(3, 4)	(5,8)	(7,16)	62	99
(3, 8)	(5,16)	(7,32)	63	99

Initial tests were not promising. Adding another baseline convolutional layer to the network took more time to train and did not improve the classification accuracy at all. We then tried the pyramid structure for the convolutional layers. We gave the first layer a large filter size but a small depth, the second layer a medium filter size and a medium depth, and the third layer a small filter size with a large depth. This result is shown in row 3 and it performed marginally better at 70%.

The inverted pyramid with the same specification required much more time to run because the layer with the greatest depth had to operate on the full 50x50 pixel image. That layer also only had a stride of 1 so its output did not shrink the image for future layers. Even with the extra runtime though, the network did not perform notably.

2.5 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. For simplicity, all regularization techniques were tested on the baseline architecture.

2.5.1 Dropout

Dropout is a regularization technique where during training, with certain probability a node's contribution to the network is ignored. This makes it so many different nodes or filters have to agree and work together to track certain features, thereby making it more difficult to overfit to the training data. The table below documents the results. Unsurprisingly, the greater the dropout, the worse the training accuracy. Unfortunately, validation accuracy also appeared to drop and validation accuracy with dropout never outperformed validation accuracy without.

	Dropout = 0.5	Dropout = 0.6	Dropout = 0.7	Dropout = 0.8
Classification Accuracy	58.6	64.5	66.0	65.8
Training Accuracy	72.0	82.9	84.9	85.0

2.5.2 Weight Regularization

Weight regularization involves penalizing the size of the weights in the nodes. As penalization increases, training error drops monotonically but unfortunately so did validation accuracy. Again, this technique was of limited effectiveness.

2.5.3 Early Stopping

In our experience, after training for an extended period of time, the validation accuracy plateaus. For every architecture we tried, we erred on the side of excessive training and then decreased the iterations later.

2.5.4 Data Augmentation

Data augmentation was the only technique we found to have some, albeit marginal, effect. For most models, data augmentation had a $1 - 1.5\%$ benefit. Intuitively, this makes sense because by artificially introducing noise into your training data, the network can better train to recognize features in the painting from the artist that generalize better.

2.6 Results

In conclusion, our optimal CNN architecture was a 3 layer pyramid structure. The first layer had a filter size of 7, depth of 8 and a stride length of 2. The second layer had a filter size of 5, depth of 16, and a stride length of 2. The third layer had a filter size of 5, depth of 32, and a stride length of 1. It took 4.5 minutes to train with data augmentation achieved an validition accuracy of 71%. The test accuracy was 69%.

To better understand our CNN, we also wanted to figure out what kind of features our CNN focuses on to classify images. To do this, we took our test set performed certain transformations on the images. We translated, changed the brightness and the contrast, and even inverted the images and then recorded the classification accuracy.

Test Dataset	Test Accuracy
Normal Validation Data	69.0
Translated Data	20.7
Brightened Data	52.9
Darkened Data	63.2
High Contrast Data	52.9
Low Contrast Data	63.2
Flipped Data	52.9
Inverted Data	16.1

The CNN was very robust to changes in brightness and contrast. This is very good. We already knew the network would classifies images not by their absolute color but their relationship to their neighbors but this invariance tells us that the dependence on neighbors is also not absolute. The CNN is also invariant to being flipped so the relationship with the neighbors is not tied to a direction.

The network however could not handle the colors being inverted. We suspect this is because even though the features in the flipped colors are preserved through the linear combination with the weights, the nonlinear RELU functions are trained for normal colors. These activation functions would not trigger for the flipped colors and the information about the features would be not be passed on to the next layer.

Most surprisingly though, is the fact that the network could not handle translation. CNNs consider features relative to their neighbors so we originally thought the network should be most robust to it. We think what threw off the network was the padding. The unexpected placement of the black padding in the middle of the image may have been enough to misclassify the image.