CS155 Set 4

Timothy Liu

February 1, 2018

1 Problem 1

1.1 Problem A

1.1.1 Problem i

The first neural network has weights initialized to what appear to be random numbers between -0.5 and 0.5 while the second neural network has weights all initialized to zero. After 250 iterations, the first network has a test loss of 0.001 while the second network that has weights initialized to 0 has test loss of 0.508. The ReLu function is non-differentiable at 0 and has derivative of zero for values less than 0. If the neural network weights are initiated at 0, then back propagation won't have a gradient to go down because it's sitting at the discontinuity. This means initiating weights at 0 will leave it stuck and unable to improve.

1.1.2 Problem ii

The neural net with weights initialized to zero again have the same problem. Because the weights are 0, when fed into the neural net all of the neurons go to zero. After 400 iterations, the neural net has a test loss of 0.374, which is much larger than the loss from ReLu. This is because the gradient of the sigmoid far from 0 is very small, so the weights change by very small amounts.

1.2 Problem B

If the neural net is only trained on the negative examples, then the weights will be pushed to the regime where the output is negative and the ReLU is flat. The nonlinearity will always output a zero so even when the positive examples are input the nonlinearities will still output a zero.

1.3 Problem C

1.3.1 i

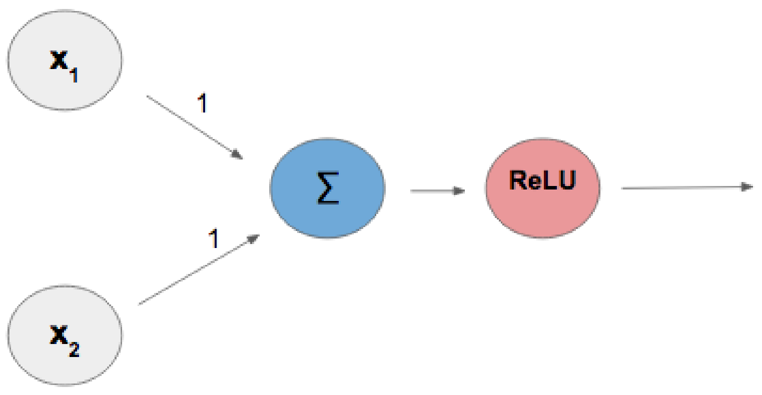


Figure 1: OR function. If either input is 1 then the output will be 1.

1.3.2 ii

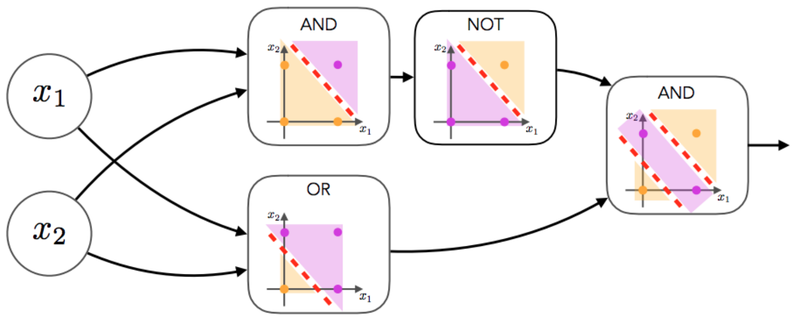


Figure 2: XOR function.

A minimum of 3 ReLU layers are needed. To make an XOR we need to AND together an inverted AND along with an OR. The AND, OR, and NOT operations each require one layer, and three must be stacked to create an exclusive OR.

2 Problem 2

2.1 Problem A

Tensorflow version: 1.5.0 Kearas version: 2.1.3

2.2 Problem B

2.2.1 i

Each input example is a 2D 28 by 28 array. The elements in each array index are 8 bit grayscale value. 0 corresponds to black and 255 corresponds to white. There are 60,000 input examples, so the input array Xtrain has dimensions 60,000 by 28 by 28.

2.2.2 ii

The new shape of the training input is a 2D array with dimensions 60,000 by 784.

2.3 Problem C

The neural net has 100 hidden units organized in 2 layers with ReLU activation and no dropout. Batch size is set at the default of 32, run for 10 epochs, and using the Adam optimizer. The test accuracy is 0.9729.

2.4 Problem D

The neural net has 200 hidden units organized in 2 layers with ReLU activation and no dropout. Batch size is set at the default of 32, run for 10 epochs, and using the Adam optimizer. The test accuracy is 0.9808.

2.5 Problem E

The neural net has 1000 hidden units organized in 4 layers with ReLU activation and no dropout. Batch size is of 1024, run for 30 epochs, and using the Adam optimizer. The test accuracy is 0.9833.

3 Problem 3

3.1 Problem A

One benefit of zero-padding is that the result has the same dimensions and the same size as the original. You are neither upsampling nor downsampling. The disadvantage is that the image effectively has a black order around the edge, which may affect the results and disrupt the training.

3.2 Problem B

3.2.1 i

The number of weights is the size of the filter 5 x 5 x 3 + 1 multiplied by the 8 filters. This is 608 weights.

3.3 ii

The output tensor is 30x30x8 because since there is no zero-padding and the stride is 1, we have a little downsampling. The depth is the same as the number of filters.

3.4 Problem C

3.4.1 i

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.25 \end{bmatrix}, \begin{bmatrix} 0.5 & 1 \\ 0.25 & 0.5 \end{bmatrix}, \begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.25 \\ 1 & 0.5 \end{bmatrix}$$

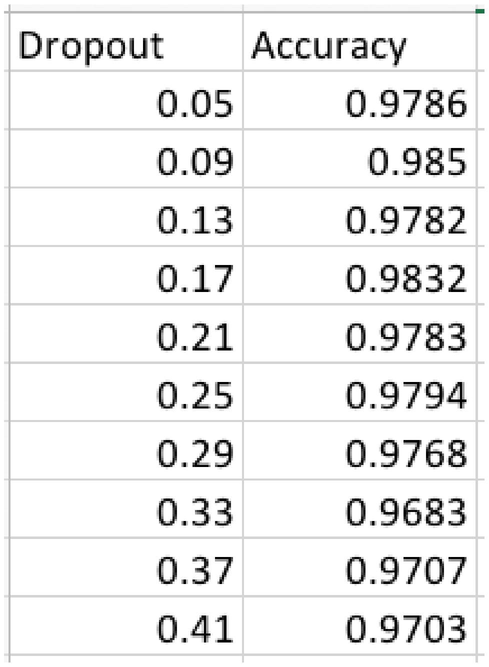
3.4.2 ii

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

3.4.3 iii

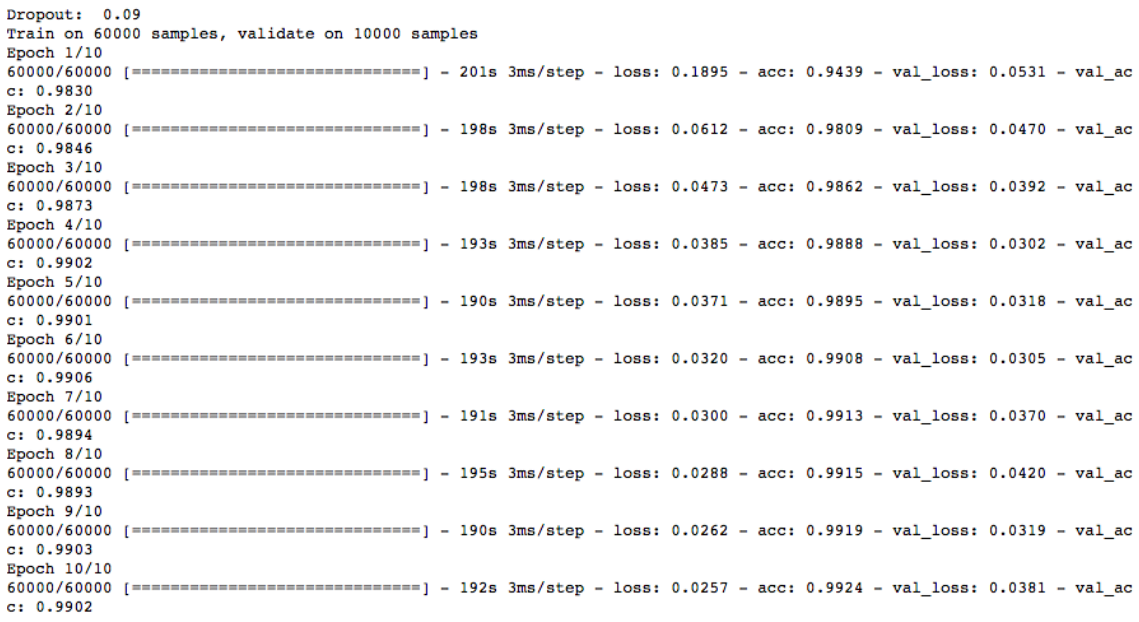
Pooling is advantageous because averaging several pixels together will smooth the image out and reduce noise.

3.5 Problem D



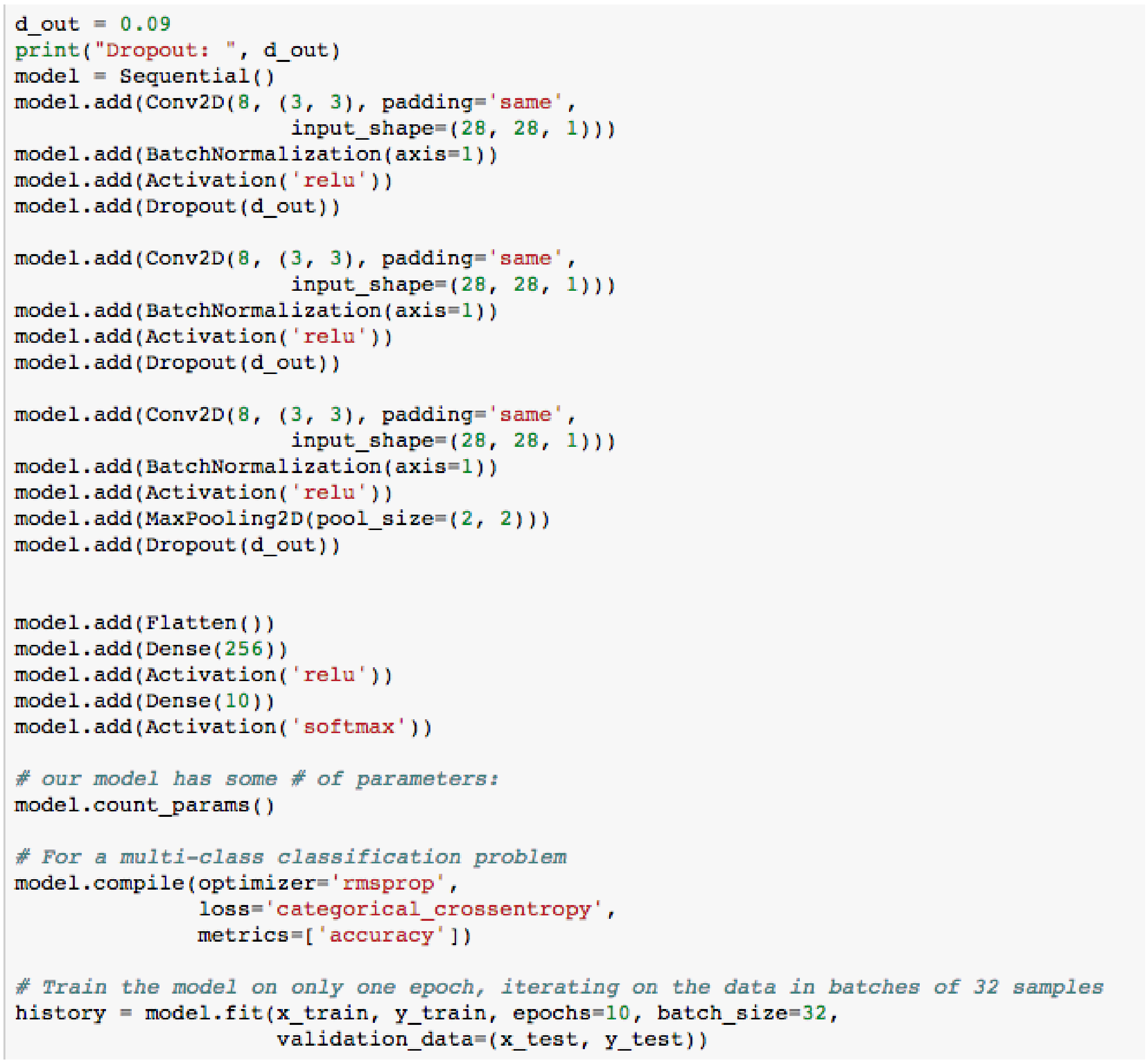
Dropout	Accuracy
0.05	0.9786
0.09	0.985
0.13	0.9782
0.17	0.9832
0.21	0.9783
0.25	0.9794
0.29	0.9768
0.33	0.9683
0.37	0.9707
0.41	0.9703

Figure 3: Test accuracy with varying dropout rates.



```
Dropout:  0.09
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 201s 3ms/step - loss: 0.1895 - acc: 0.9439 - val_loss: 0.0531 - val_ac
c: 0.9830
Epoch 2/10
60000/60000 [==============================] - 198s 3ms/step - loss: 0.0612 - acc: 0.9809 - val_loss: 0.0470 - val_ac
c: 0.9846
Epoch 3/10
60000/60000 [==============================] - 198s 3ms/step - loss: 0.0473 - acc: 0.9862 - val_loss: 0.0392 - val_ac
c: 0.9873
Epoch 4/10
60000/60000 [==============================] - 193s 3ms/step - loss: 0.0385 - acc: 0.9888 - val_loss: 0.0302 - val_ac
c: 0.9902
Epoch 5/10
60000/60000 [==============================] - 190s 3ms/step - loss: 0.0371 - acc: 0.9895 - val_loss: 0.0318 - val_ac
c: 0.9901
Epoch 6/10
60000/60000 [==============================] - 193s 3ms/step - loss: 0.0320 - acc: 0.9908 - val_loss: 0.0305 - val_ac
c: 0.9906
Epoch 7/10
60000/60000 [==============================] - 191s 3ms/step - loss: 0.0300 - acc: 0.9913 - val_loss: 0.0370 - val_ac
c: 0.9894
Epoch 8/10
60000/60000 [==============================] - 195s 3ms/step - loss: 0.0288 - acc: 0.9915 - val_loss: 0.0420 - val_ac
c: 0.9893
Epoch 9/10
60000/60000 [==============================] - 190s 3ms/step - loss: 0.0262 - acc: 0.9919 - val_loss: 0.0319 - val_ac
c: 0.9903
Epoch 10/10
60000/60000 [==============================] - 192s 3ms/step - loss: 0.0257 - acc: 0.9924 - val_loss: 0.0381 - val_ac
c: 0.9902
```

Figure 4: The final test accuracy after 10 epochs is 0.9902



```
d_out = 0.09
print("Dropout: ", d_out)
model = Sequential()
model.add(Conv2D(8, (3, 3), padding='same',
                     input_shape=(28, 28, 1)))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))
model.add(Dropout(d_out))

model.add(Conv2D(8, (3, 3), padding='same',
                     input_shape=(28, 28, 1)))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))
model.add(Dropout(d_out))

model.add(Conv2D(8, (3, 3), padding='same',
                     input_shape=(28, 28, 1)))
model.add(BatchNormalization(axis=1))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(d_out))


model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))

# our model has some # of parameters:
model.count_params()

# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model on only one epoch, iterating on the data in batches of 32 samples
history = model.fit(x_train, y_train, epochs=10, batch_size=32,
                    validation_data=(x_test, y_test))
```

Figure 5: Code for final model.

The most accurate model layered several convolutional networks with MaxPooling towards the end. Greater max pooling was found to damage the accuracy. The final dense layer was increased to 256 units, which helped to boost performance. A fairly low amount of dropout combined with batch normalization also improved accuracy.

A problem with this type of optimization is that there are too many variables to optimize. Different variables may also affect each other, so for example a different amount of dropout may be more appropriate for a different number of convolution layers.