Programming Assignment 1

CS 640 Programming Assignment 1

Author: Alec Hoyland

Date: 2019-10-1 13:28

How my neural network classifies data

My neural network consists of an input layer, a hidden layer, and an output layer. I use a cross-entropy cost loss function with L2 regularization. The network is fully-connected and feedforward, with a hyperbolic tangent activation function.

There are three hyperparameters: the number of nodes in the hidden network, the learning rate (epsilon), and the regularization coefficient (lambda).

Data are passed into the first layer, which has one node per feature of the input data. The values are passed from the input nodes to the hidden layer nodes according to some weights. This is a dot product between a weight matrix and a value vector. The hidden nodes perform a nonlinear transformation by the activation function, tanh. Then, a second set of weights maps the output of the hidden layer to the output layer.

I compute the output probability using a soft argmax function. These probabilities are degrees of belief that the input is in class 1 or class 2, or so on. There is one probability value per class.

Forward propagation

The inputs are dot-multiplied by weight matrix between the first and second layers, to which the second layer bias is added. This is transformed by the tanh activation function. This process is repeated, where the new vector is dot-multiplied by the weight matrix between the second and third layers, and the third layer bias is added. Probabilities are computed using the soft argmax function.

```
# Forward propagation
z1 = X.dot(W1) + b1
a1 = np.tanh(z1)
z2 = a1.dot(W2) + b2
exp_scores = np.exp(z2)
probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
```

Backpropagation

Each individual weight must be changed by an amount equal to the learning rate times the partial derivative of the loss function by the individual weight. Since this is a multi-layer network, the error derivatives propagate backwards according to the chain rule.

I compute the error by taking the output probabilities and subtracting the real result. The second to third layer weight “gradient” is computed as the dot product between the second layer activated values and the third layer error. The first to second layer weight “gradient” is computer as a dot product between the third layer weight “gradient”, the second to third layer weights, and the derivative of the activation function evaluated on the second layer activated values.

The bias terms also change values, but simply by the sum of the computed errors, since the bias terms are added (they aren’t multiplied by the weight matrix).

Regularization affects the weight error, by penalizing when weights get large.

All weights and biases are then decreased by the learning rate times the backpropagated error.

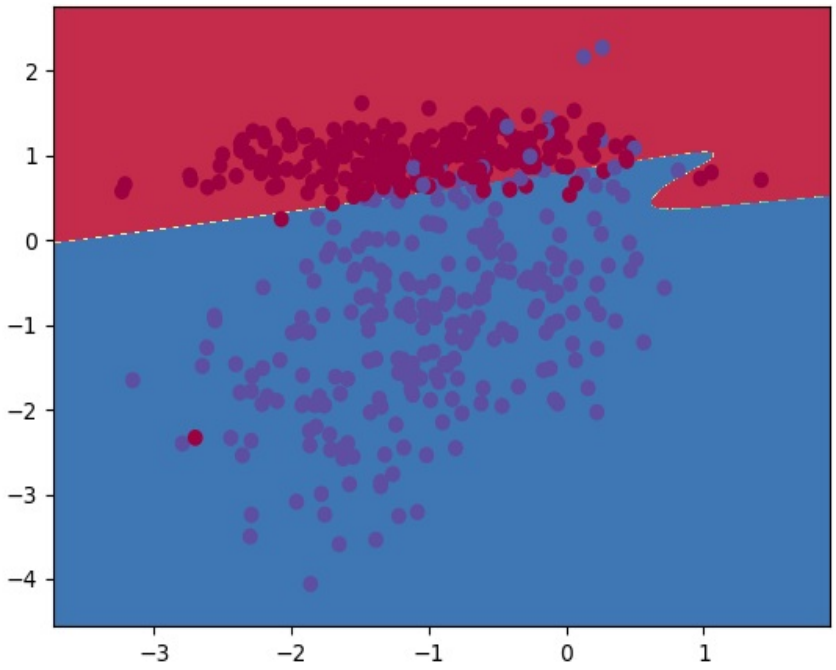
```
# Backpropagation
delta3 = probs
```

```
delta3[range(num_examples), y] -= 1
dW2 = (a1.T).dot(delta3)
db2 = np.sum(delta3, axis=0, keepdims=True)
delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))
dW1 = np.dot(X.T, delta2)
db1 = np.sum(delta2, axis=0)

# regularization
dW2 += config.reg_lambda * W2
dW1 += config.reg_lambda * W1

# Gradient descent parameter update
W1 += -config.epsilon * dW1
b1 += -config.epsilon * db1
W2 += -config.epsilon * dW2
b2 += -config.epsilon * db2
```

Linear Dataset

I used my neural network on the linear dataset provided. I used a learning rate of 0.01, regularization lambda of 0.01, and 3 nodes in the hidden layer. There were 20,000 epochs over 80% of the dataset.

```
true positives  231
true negatives  242
false positives  8
false negatives  19
precision  0.9665271966527197
recall  0.924
F-1 score  0.9897750511247444
accuracy  0.946
```

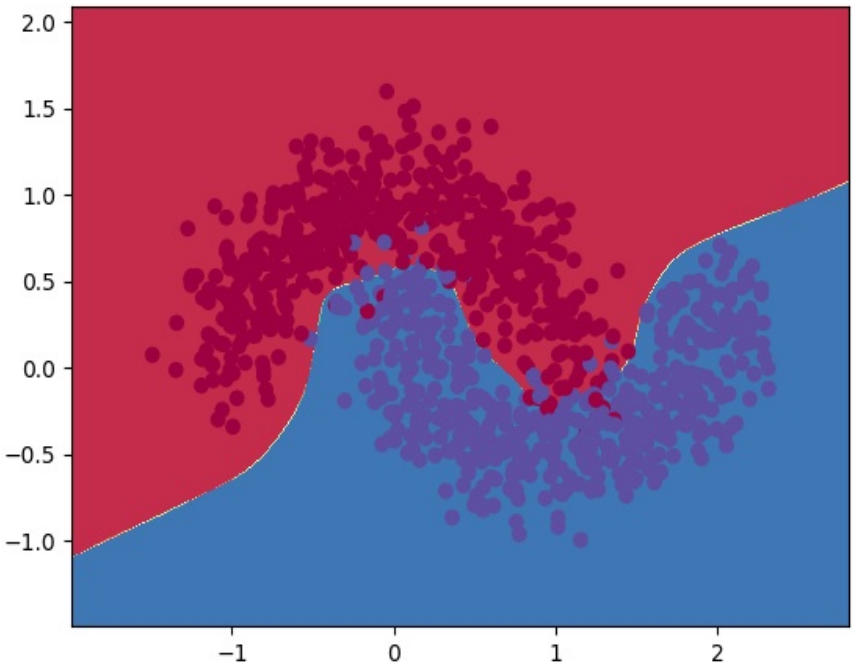


Nonlinear Dataset

I used my neural network on the nonlinear dataset provided.

Varying the number of neurons

```
Number of Neurons:  1
-----------------
true positives:  452
true negatives:  413
false positives:  87
false negatives:  48
precision:  0.8385899814471243
recall:  0.904
F-1 score:  0.794995187680462
accuracy:  0.865

Number of Neurons:  3
-----------------
true positives:  379
true negatives:  474
false positives:  26
false negatives:  121
precision:  0.9358024691358025
recall:  0.758
F-1 score:  1.047513812154696
accuracy:  0.853

Number of Neurons:  10
-----------------
true positives:  477
true negatives:  493
false positives:  7
false negatives:  23
precision:  0.9855371900826446
recall:  0.954
F-1 score:  1.0020325203252032
accuracy:  0.97

Number of Neurons:  30
-----------------
true positives:  468
true negatives:  493
false positives:  7
false negatives:  32
precision:  0.9852631578947368
recall:  0.936
F-1 score:  1.0112820512820513
accuracy:  0.961

Number of Neurons:  100
-----------------
true positives:  498
true negatives:  485
false positives:  15
false negatives:  2
precision:  0.9707602339181286
recall:  0.996
F-1 score:  0.9575518262586377
accuracy:  0.983
```



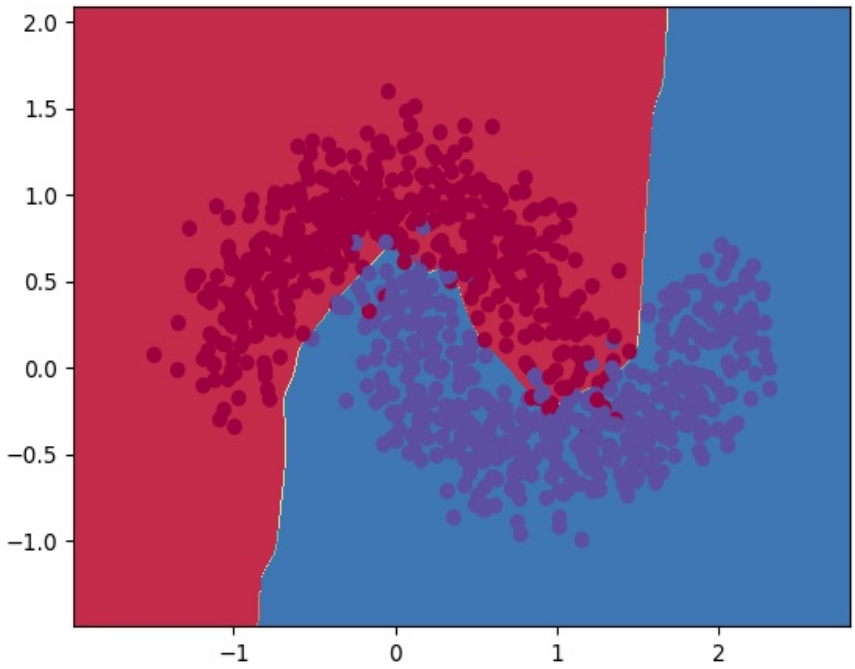
10 hidden nodes

Varying the regularization coefficient

```
Regularization Coefficient:  0.001
-----------------
true positives:  485
true negatives:  491
false positives:  9
false negatives:  15
precision:  0.9817813765182186
recall:  0.97
F-1 score:  0.9879275653923542
accuracy:  0.976

Regularization Coefficient:  0.003
-----------------
true positives:  491
true negatives:  483
false positives:  17
false negatives:  9
precision:  0.9665354330708661
recall:  0.982
F-1 score:  0.9583333333333334
accuracy:  0.974

Regularization Coefficient:  0.01
-----------------
true positives:  477
true negatives:  493
false positives:  7
false negatives:  23
precision:  0.9855371900826446
recall:  0.954
F-1 score:  1.0020325203252032
accuracy:  0.97

Regularization Coefficient:  0.03
-----------------
```

```
true positives:  469
true negatives:  494
false positives:  6
false negatives:  31
precision:  0.9873684210526316
recall:  0.938
F-1 score:  1.0133333333333334
accuracy:  0.963

Regularization Coefficient:  0.1
-----------------
true positives:  467
true negatives:  495
false positives:  5
false negatives:  33
precision:  0.989406779661017
recall:  0.934
F-1 score:  1.0185185185185186
accuracy:  0.962
```

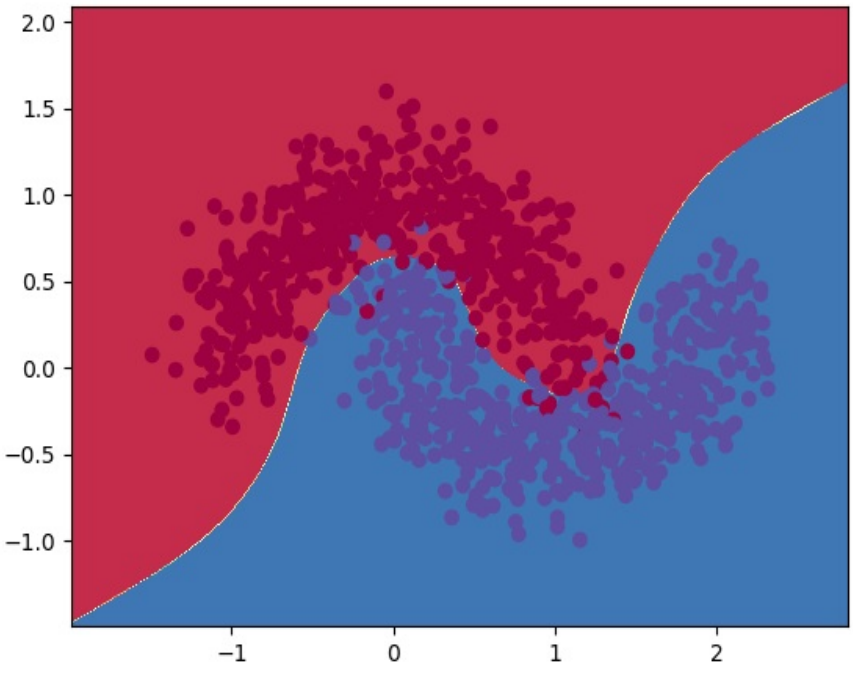
regularization coefficient of 0.001



Varying the learning rate

```
Learning Rate:  0.001
-----------------
true positives:  486
true negatives:  486
false positives:  14
false negatives:  14
precision:  0.972
recall:  0.972
F-1 score:  0.972
accuracy:  0.972

Learning Rate:  0.003
-----------------
true positives:  485
true negatives:  484
false positives:  16
false negatives:  15
precision:  0.9680638722554891
recall:  0.97
F-1 score:  0.967032967032967
accuracy:  0.969

Learning Rate:  0.01
-----------------
true positives:  477
true negatives:  493
false positives:  7
false negatives:  23
precision:  0.9855371900826446
recall:  0.954
F-1 score:  1.0020325203252032
accuracy:  0.97

Learning Rate:  0.03
-----------------
```

```
true positives:  472
true negatives:  486
false positives:  14
false negatives:  28
precision:  0.9711934156378601
recall:  0.944
F-1 score:  0.9858012170385395
accuracy:  0.958

Learning Rate:  0.1
-----------------
true positives:  436
true negatives:  428
false positives:  72
false negatives:  64
precision:  0.8582677165354331
recall:  0.872
F-1 score:  0.8492063492063492
accuracy:  0.864
```

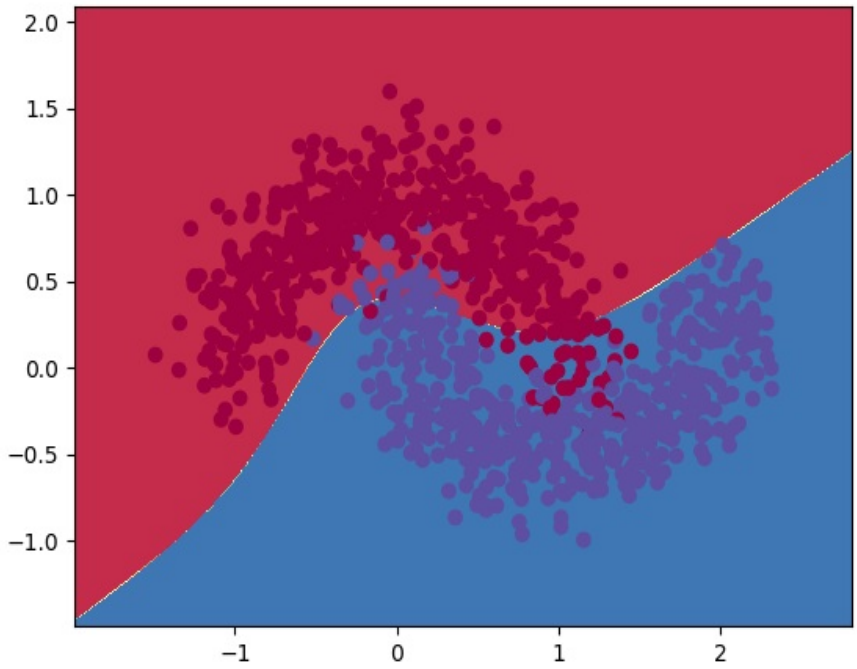


learning rate of 0.001

Varying the number of epochs

```
Number of Epochs:  100
-----------------
true positives:  421
true negatives:  456
false positives:  44
false negatives:  79
precision:  0.9053763440860215
recall:  0.842
F-1 score:  0.9450777202072539
accuracy:  0.877

Number of Epochs:  200
-----------------
true positives:  437
```

```
true negatives:  443
false positives:  57
false negatives:  63
precision:  0.8846153846153846
recall:  0.874
F-1 score:  0.8913480885311871
accuracy:  0.88

Number of Epochs:  2000
-----------------
true positives:  492
true negatives:  315
false positives:  185
false negatives:  8
precision:  0.7267355982274741
recall:  0.984
F-1 score:  0.5352591333899746
accuracy:  0.807

Number of Epochs:  10000
-----------------
true positives:  467
true negatives:  393
false positives:  107
false negatives:  33
precision:  0.813588850174216
recall:  0.934
F-1 score:  0.7318435754189944
accuracy:  0.86

Number of Epochs:  20000
-----------------
true positives:  436
true negatives:  428
false positives:  72
false negatives:  64
precision:  0.8582677165354331
recall:  0.872
F-1 score:  0.8492063492063492
accuracy:  0.864
```



number of epochs: 100

Digit detection

I applied the neural network to digit detection. In this task, there are 10 possible classes, one for each digit, and the input dimensionality is 64.

Varying the number of nodes

```
Number of Neurons:  10
-----------------
accuracy:  0.9287305122494433

Number of Neurons:  30
-----------------
accuracy:  0.9454342984409799

Number of Neurons:  100
-----------------
accuracy:  0.9398663697104677

Number of Neurons:  200
-----------------
accuracy:  0.9387527839643652

Number of Neurons:  300
-----------------
accuracy:  0.933184855233853
```

Varying the regularization coefficient

```
Regularization Coefficient:  0.001
-----------------
accuracy:  0.9409799554565702

Regularization Coefficient:  0.003
-----------------
accuracy:  0.9409799554565702
```

```
Regularization Coefficient:  0.01
-----------------
accuracy:  0.9398663697104677

Regularization Coefficient:  0.03
-----------------
accuracy:  0.9398663697104677

Regularization Coefficient:  0.1
-----------------
accuracy:  0.9420935412026726
```

Varying the learning rate

```
Learning Rate:  0.0001
-----------------
accuracy:  0.9432071269487751

Learning Rate:  0.0003
-----------------
accuracy:  0.9420935412026726

Learning Rate:  0.003
-----------------
accuracy:  0.8385300668151447

Learning Rate:  0.001
-----------------
accuracy:  0.9398663697104677

Learning Rate:  0.003
-----------------
accuracy:  0.8385300668151447
```

Varying the number of epochs

```
Number of Epochs:  100
-----------------
accuracy:  0.10133630289532294

Number of Epochs:  200
-----------------
accuracy:  0.09799554565701558

Number of Epochs:  2000
-----------------
accuracy:  0.8307349665924276

Number of Epochs:  10000
-----------------
accuracy:  0.8173719376391982

Number of Epochs:  20000
-----------------
accuracy:  0.8385300668151447
```

Conclusions

In general, it is good to have a good number of neurons, though too many seems to result in too many parameters to

train, and therefore worse results, unless other parameters are changed as well (such as the number of epochs).

Increasing the regularization coefficient decreases the amount of overfitting. Overfitting occurs when the model over-specializes on the training dataset, which make it very accurate for the training dataset, but less accurate against novel data. When your data are noisy, it's better to have a model that overfits less. In the nonlinear dataset, I saw that increasing the regularization coefficient resulted a less accurate model, but a higher F-1 score. For the digit dataset, picking a higher regularization coefficient resulted in higher accuracy. Overall, I think this parameter should be hand-tuned, and depends heavily on the quality of your training dataset.

A finer learning rate decreases the rate at which an optimization procedure converges, however it increases the likelihood that the optimization procedure will find a true minimum. A coarser learning rate therefore decreases the time it takes to train a model, but results an a less precisely-trained model.

In my neural network, I don't look for convergence of the optimization. Instead, I train for a fixed number of epochs. This means that the learning rate determines how closely I get to some minimum of the loss function – since too small of a learning rate means I don't get close, but too coarse of a learning rate means I might not find one at all.

For example, in the digit-learning task, I had to use a learning rate of less than 0.003, or my model would completely fail at the task (0% accuracy). Generally, a lower learning rate was helpful here.

Really, training a good model means having a tiny learning rate and a large number of epochs, but this is time-expensive, since more epochs mean more training time.

I think the effect works in the opposite direction with epochs and the regularization coefficient. This is because letting a model train for more epochs increases the degree to which it will overfit. I see this in my results with the nonlinear dataset, where the number of false positivess skyrockets as the number of epochs increase, before decreasing again with a large number of epochs.

Too few epochs will result in a poorly-trained model. This is shown by my 100-epoch digit-task training, with an accuracy a little over 10%.

Credits & Contributions

I worked on this assignment by myself, but I did get a lot of help from the internet/textbooks. I used Bishop's Machine Learning textbook for the backpropagation algorithm. I reproduced this function for the decision boundary. I also read these articles:

- https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6
- https://iamtrask.github.io/2015/07/12/basic-python-network/
- https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/