CSE 446 A4

Johan J How

0. Policies

0.1: List of Collaborators

0.2 List of Acknowledgements

0.3 I have read and understood these policies

1. Convexity: Linear and Logistic Regression

1. In the case of linear regression, the squared error will not converge to 0. This is because the data is linearly separable, so we create some margin that minimizes the squared error, but this margin is not perfect as it is linear, so we do have a non-zero squared error. In the case of logistic regression, the log loss is converging to 0 because it will find some decision boundary and increase its confidence on the boundary, causing the weights to go to infinity.
2. In the case of linear regression, the squared error will not converge to 0 for the same reason as in 1.1. In the case of logistic regression, the squared error will converge to 0 for the same reason as in 1.1 Because d >= n, the value that is converged to will be unique or one of many.
3. Once gradient descent hits a misclassification error of 0, the parameters will stop changing. This is because the weights will increase indefinitely as gradient descent grows in confidence.

2. Neural Networks and Non-convex optimization

2.1 Computational Complexity (and a little more linear algebra…)

1.

```
forwardPass($X_{d^0,1}$)
    $W^{(1)}_{d^1 d^0}$ = initializeWeights()
    $z^{(1)}_{d^0,1} = X_{d^0,1}$
    for layer $\ell$ in L
        $a^{(\ell+1)}_{d^{\ell+1},1} = W^{(\ell+1)}_{d^{\ell+1},d^\ell} * z^{(\ell)}_{d^\ell,1}$
        if $\ell+1$ is $L+1$
            return $a^{(\ell+1)}_{d^{\ell+1},1}$
        $z^{(\ell+1)}_{d^{\ell+1},1} = h(a^{(\ell+1)}_{d^{\ell+1},1})$
```

2.

```
forwardPassMinibatch($X_{d^0,m}$)
    $W^{(1)}_{d^1,d^0}$ = initializeWeights()
    $z^{(1)}_{d^0,m} = X_{d^0,m}$
    for layer $\ell$ in L
        $a^{(\ell+1)}_{d^{\ell+1},m} = W^{(\ell+1)}_{d^{\ell+1},d^\ell} * z^{(\ell)}_{d^\ell,1}$
        if $\ell+1$ is $L+1$
            return $a^{(\ell+1)}_{d^{\ell+1},m}$
        $z^{(\ell+1)}_{d^{\ell+1},m} = h(a^{(\ell+1)}_{d^{\ell+1},m})$
```

3.

$$O\left(\sum_{i=1}^{L}\left((d^{i+1} * d^{i} * m) + d^{i+1} C_h\right)\right)$$

4. O(forwardPass + L * C_h)
   This is because we run the forward pass first, then at each level we compute the gradient, which is within a constant of C_h.
5. It’s unreasonable to obtain a faster runtime because to compute the gradient of the parameters at the top level, we need to get all the other gradients as well, so no time is saved.

2.2 Saddle points and Symmetries

1. Not a saddle point. Consider if the data is linearly separable, then the MLP converges, which isn’t a saddle point. Consider if the data is not linearly separable, then the MLP doesn’t converge, which isn’t a saddle point either.
2. It is a saddle point. When the weights are 0, our tanh transfer function will always be 0. The hidden layers then always pass on 0 to the next layer, so the output will not change on different inputs. This is a stationary point, and we have a saddle point.
3. We do not start at a saddle point. Given an input of 0, sigmoid will be 1/2, which is the relation between each layer. The gradient continues to step here.
4. In a one hidden layer network, y_hat(x) will be the same. In a network with L hidden layers, if L is odd, y_hat(x) will be the same, but if it’s even, the y_hat(x) will be flipped.

2.3 Representation and Non-linear Decision Boundaries

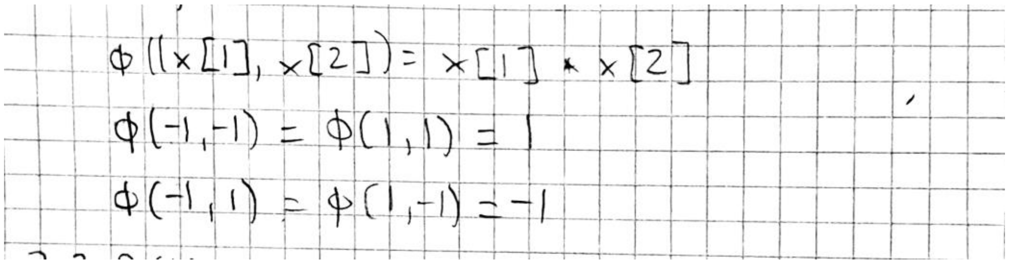
1.

a) We would use perceptron, as it can take the same inputs as a neural network. A perceptron gives us no saddle points, so all local optima are global optima.

b) Same as 2.3.1(a), except a^2 is the input. Same reasoning applies to satisfy 1) and 2).

2.

a)

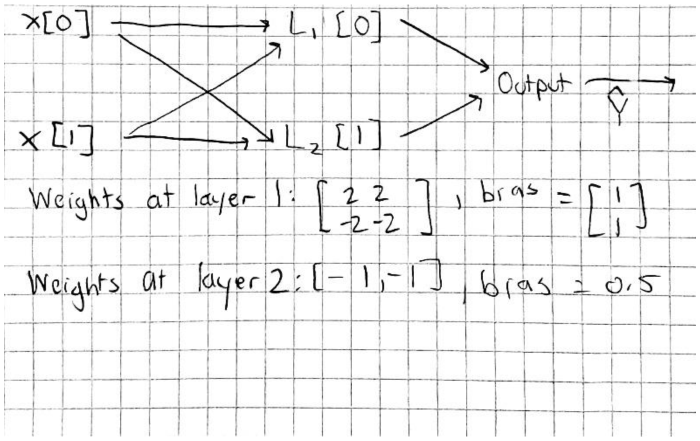


$\phi([x[1], x[2]]) = x[1] * x[2]$

$\phi(-1,-1) = \phi(1,1) = 1$

$\phi(-1,1) = \phi(1,-1) = -1$

b)



Weights at layer 1: $\begin{bmatrix} 2 & 2 \\ -2 & -2 \end{bmatrix}$, bias = $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Weights at layer 2: $[-1, -1]$, bias = 0.5

Proof that the networks predictions matches the XOR function:

a. Input (-1, -1)
   i. $a^{(1)} = W^{(1)} * X^{(1)} + B^{(1)} = [-3, 5]^T$
   ii. $\text{Tanh}(a^{(1)}) = [-1, 1]^T$
   iii. $a^{(2)} = W^{(2)} * X^{(2)} + B^{(2)} = 0.50$
   iv. $\text{y\_hat} = \text{Sign}(a^{(2)}) = 1$
b. Input (1, 1)
   i. $a^{(1)} = W^{(1)} * X^{(1)} + B^{(1)} = [5, -3]^T$
   ii. $\text{Tanh}(a^{(1)}) = [1, -1]^T$
   iii. $a^{(2)} = W^{(2)} * X^{(2)} + B^{(2)} = 0.50$
   iv. $\text{y\_hat} = \text{Sign}(a^{(2)}) = 1$
c. Input (-1, 1)
   i. $a^{(1)} = W^{(1)} * X^{(1)} + B^{(1)} = [1, 1]^T$
   ii. $\text{Tanh}(a^{(1)}) = [0.76, 0.76]^T$
   iii. $a^{(2)} = W^{(2)} * X^{(2)} + B^{(2)} = -1.0$
   iv. $\text{y\_hat} = \text{Sign}(a^{(2)}) = -1$
d. Input (1, -1)

i. $a^{(1)} = W^{(1)} * X^{(1)} + B^{(1)} = [1, 1]^T$

ii. $\mathrm{Tanh}(a^{(1)}) = [0.76, 0.76]^T$

iii. $a^{(2)} = W^{(2)} * X^{(2)} + B^{(2)} = -1.0$

iv. $\hat{y} = \mathrm{Sign}(a^{(2)}) = -1$

3. MLPs: Let's try them out on MNIST

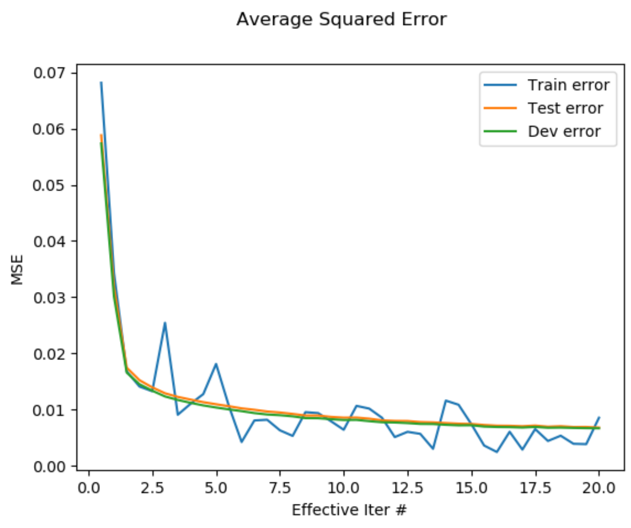
3.1 Try to learn a one hidden layer network

1.

a) Mini-batch size = 50, no regularization parameter, learning rate = 0.40, weight decay = 0.00001, decay every step. Initialized due to tuning on the dev set, seeing slower decreases in error for <0.40 learning rate.

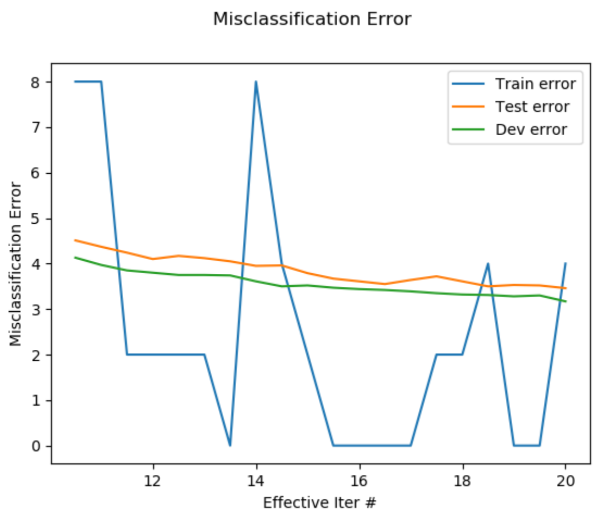
b) At about a learning rate of 0.05, there is a non-trivial decrease in error. At a learning rate of about 5 there was divergence, but I did not get NaNs, because the function sigmoid is unable to get NaNs because the denominator will never be 0 for any input.

c)

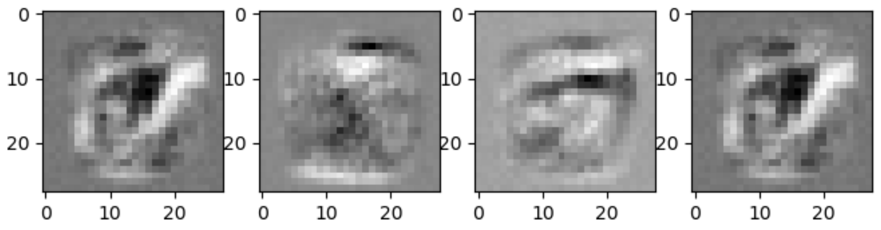


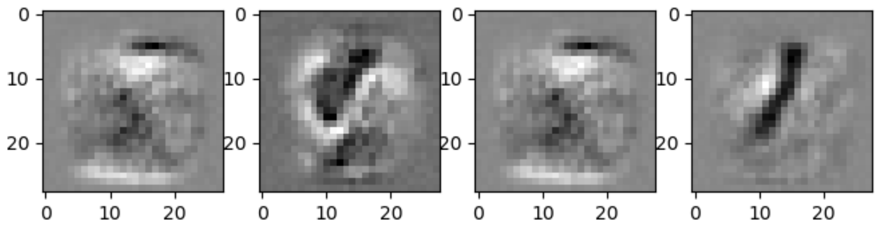
d)

Minimum Test Error: 3.33%



e)





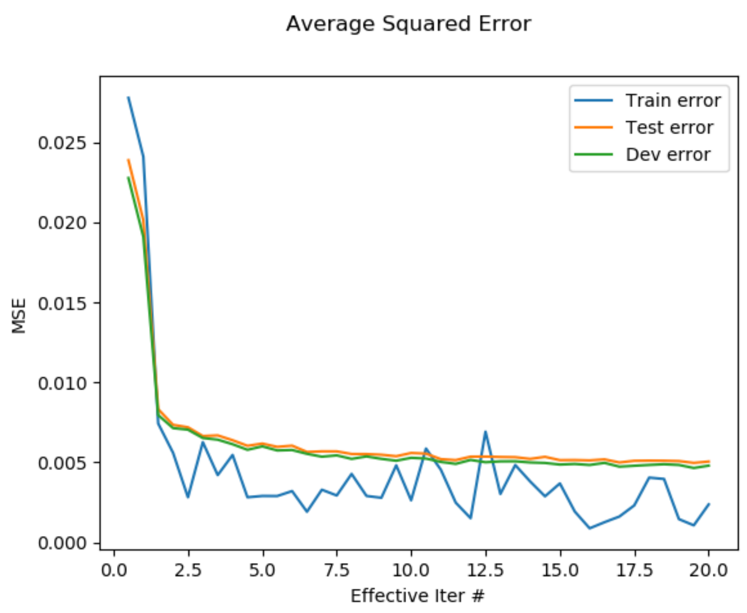
As we can see in the plots, there isn’t a clear image forming, because we are using SGD with a randomized batch every iteration. The weights around the edges of the images are mostly the same colors, while weights near the middle of the images give light and dark patterns. The interpretation that we can gather from this is that weights in the middle are more important for our classification. Additionally, weights that help with prediction also are lighter.

2.

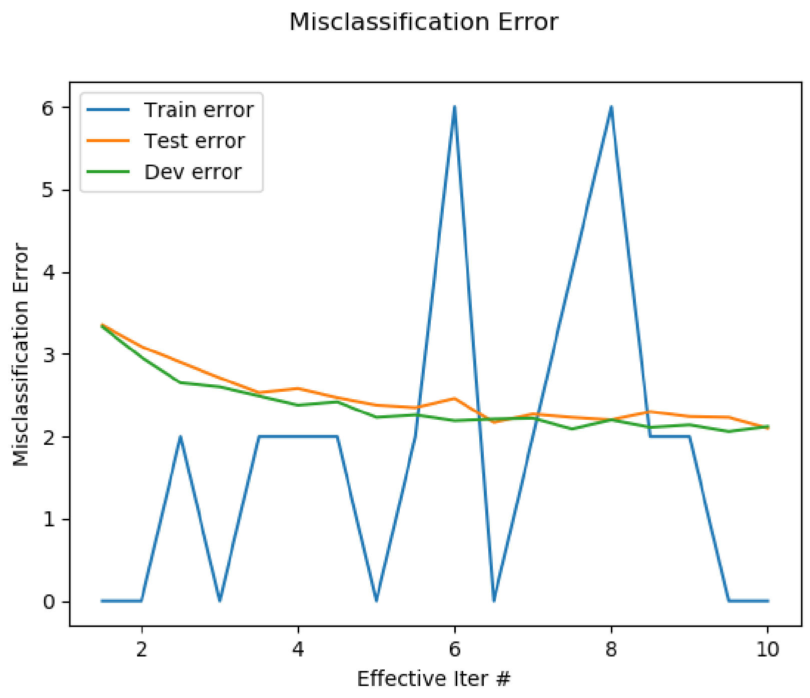
a) Mini-batch size = 500, no regularization parameter, learning rate = 0.2, weight decay = 0.00001, decay every step. Found these parameters by working with the sigmoid parameters in 3.1, then reduced learning rate by tuning on the dev set.

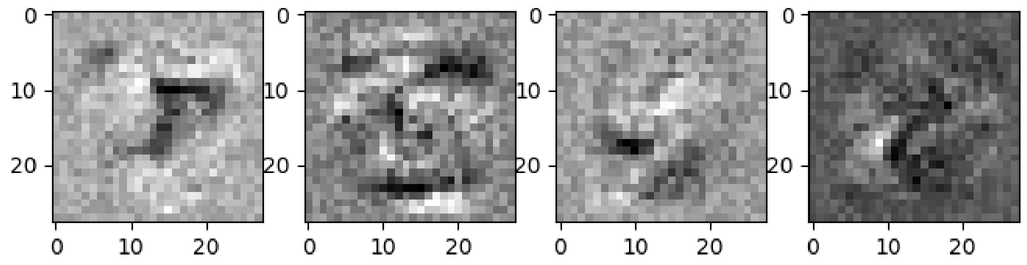
b) At about a learning rate of 0.05, there is a non-trivial decrease in error. At a learning rate of about 1 there was divergence, but I did not get NaNs. This is because ReLu goes toward the convergent value, and does not overstep if your step size is too large.

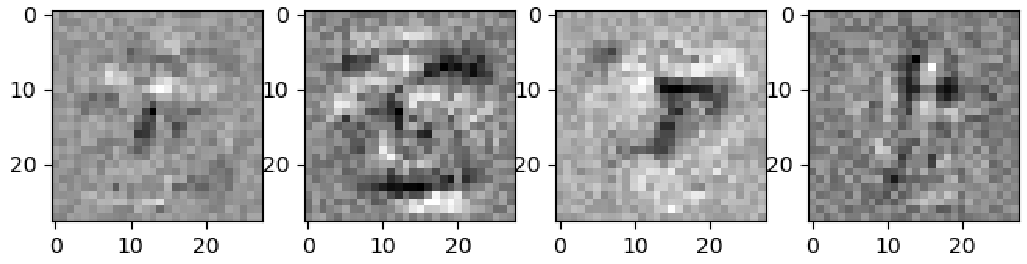
c)



d) Minimum Test Error: 1.82%







e)

Same as sigmoid, except there seems to be more noise in the visualizations. This is because ReLu is harder. The edges of the visualizations are not as “clean” as sigmoid.