CS7150.37338.202130 Homework 2

Eda Aydin Oktay, Aishwarya Vantipuli

TOTAL POINTS

5.1 / 6

QUESTION 1

1 Question 1 **1 / 1**

✓ **- 0 pts** Correct

- 0.15 pts Show better performance on test set

- 0.15 pts Show batchnorm can permit a higher learning rate

- 0.05 pts Show test accuracy for higher learning rate.

- 0.05 pts Legends should include learning rate

QUESTION 2

Question 2 2 pts

2.1 a **0.45 / 1**

- 0 pts Correct

✓ **- 0.25 pts Incorrect Derivation for the Gradient**

✓ **- 0.1 pts Incorrect Closed Form**

- 0.5 pts No Work

✓ **- 0.2 pts Additional Parameter Theta**

- 0.25 pts Did not solve for y*

- 0.05 pts Calculation Error

There no parameter theta in the objective function, we are trying to minimize y. This is essentially the linear regression derivation for theta.

2.2 b **0.9 / 1**

- 0 pts Correct

- 0.05 pts Give an explanation of the plot

- 0.2 pts Unclear Terminal Error

- 0.05 pts The plot is relative to x not y*

- 0.3 pts Not the effect the problem is looking for

- 0.1 pts Smaller learning rate does not converge

- 0.1 pts Need to plot on log scale to see the effect

✓ **- 0.1 pts Scaling is off**

QUESTION 3

Question 3 3 pts

3.1 a **1 / 1**

✓ **- 0 pts** Correct

3.2 b **1 / 1**

✓ **- 0 pts** Correct

- 0.15 pts Momentum should converge first

3.3 c **0.75 / 1**

- 0 pts Correct

✓ **- 0.05 pts Need to explain why f1 ad f2 are different (they rotate by 45 degrees)**

✓ **- 0.1 pts Unclear explanation why gd and gd with momentum is the same for f1 and f2.**

✓ **- 0.1 pts Unclear why adaptive learning optimization schemes are different for f1 and f2.**

- 0.05 pts Incorrect reason for f1 and f2 being the same for gd and gd with momentum.

- 0.1 pts Analyzed incorrect optimizers.

- 0.05 pts Adaptive learning optimization performance is dependent on the curvate of function.

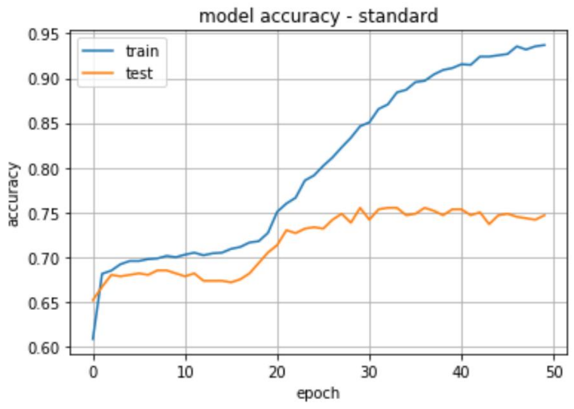
Looking for gd and gd with momentum commute with rotation and the adaptive learning methods (adam and rmsprop) do not.

Question 1. *Perform experiments that show that BatchNorm can (a) accelerate convergence of gradient based neural network training algorithms, (b) can permit larger learning rates, and (c) can lead to better neural network performance. You may select any context you like for your experiments.*

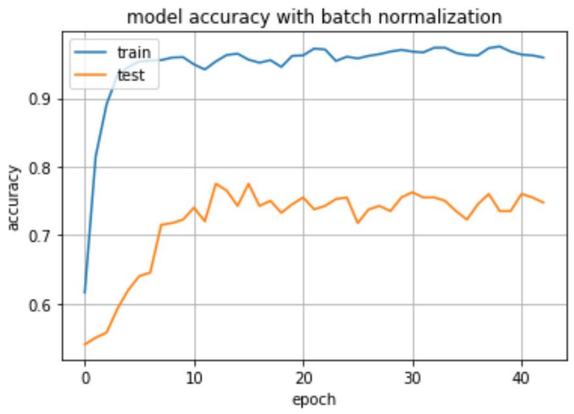
Response:

In this question we used 2k tweet data about stock market labeled as 'buy alert' or 'not an alert'. We trained Feed forward neural network (FFNN) with Tf-IDF and Doc2Vec feature representations for this binary classification task to predict if a given tweet is a 'buy alert'.

a) Experiment 1: To test how does Batchnorm can accelerate convergence of gradient based neural network training algorithms, we trained a feed forward neural network with Adam optimizer (See Appendix for Model implementations) with and without BatchNorm. We observe that model without Bathnorm reaches the maximum test accuracy of 75.5% at epoch 31 (Figure 1). The same model trained with same feature vectors reaches a test accuracy of 77.8 % at epoch 11 (Figure 2) when we add BatchNorm to all layers of the NN. We observe that Bathnorm provides almost 3 times faster convergence.

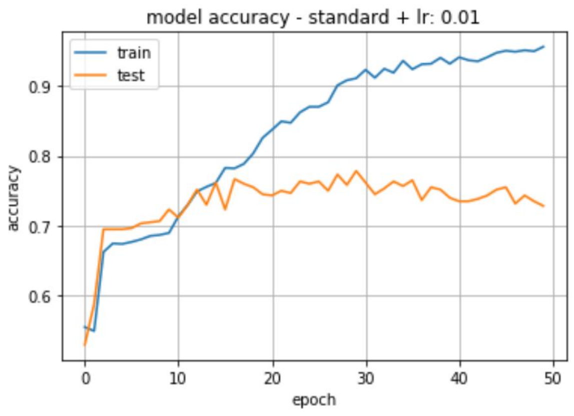


(a) Figure 1 : Training without BatchNorm

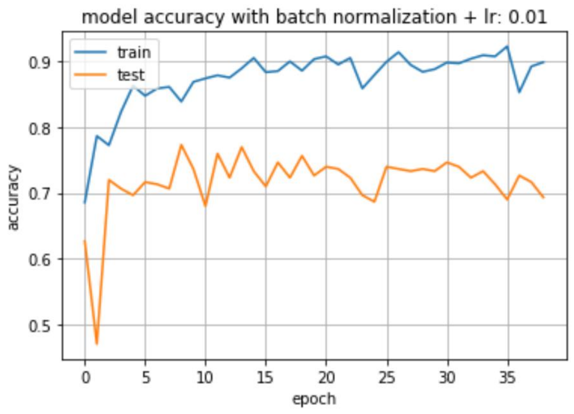


(b) Figure 2: Training with BatchNorm

b) Experiment 2: In the second experiment we trained a FFNN using SGD optimizer with a user defined momentum, and learning rate then we trained the mode with and without Batchnorm. In the first setup we set the learning rate to 0.01 in both models. From Figure 3 and Figure 4 we observe that both models learn an accuracy of about 77% (only with different number of epoches). Then we increased the learning rate to a high value which is 1 to see how do each model behaves in this case. From Figure 5 we see that model without batchnorm cannot converge anymore. On the other hand model with Batchnorm can still learn and reach to a testing accuracy of 75%. Although the model fluctuates a lot more we see that Batchnorm is permitting a larger learning rate, while the standard model fails to converge.

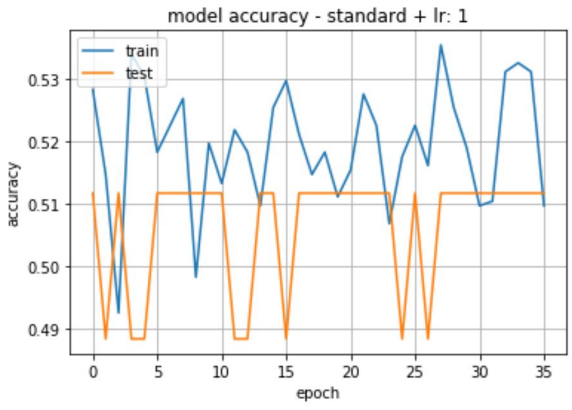


(a) Figure 3: Training without BatchNorm with learning rate = 0.01

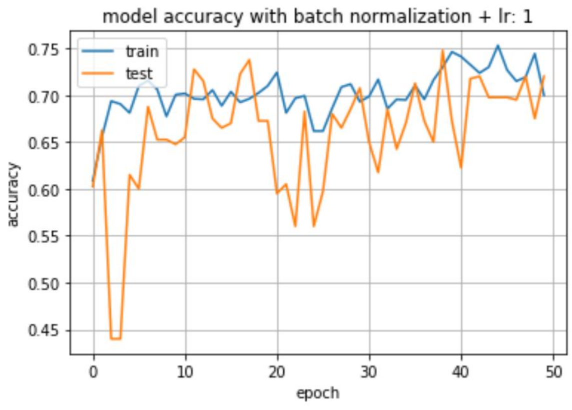


(b) Figure 4: Training with BatchNorm with learning rate = 0.01

(c) The first experiment shows that BatchNorm added model yields about 2% more accuracy along with a benefit of faster convergence. In the second experiment accuracies were very similar but the converges time was again a lot faster with Batchnorm (8 epochs vs 18 epochs). Both cases indicates a better neural network performance with Batchnorm.



(a) Figure 5: Training without BatchNorm with learning rate =1



(b) Figure 6: Training with BatchNorm with learning rate = 1

Question 2. *Stochastic Gradient Descent*

In this problem, you will build your own solver of Stochastic Gradient Descent. Do not use built-in solvers from any deep learning packages. In this problem, you will use stochastic gradient descent to solve

$$\min_y \frac{1}{n}\sum_{i=1}^{n} |y - x_i|^2, \tag{1}$$

where x_i is a real number for $i = 1 \dots n$.

(a) Using calculus, derive a closed-form expression for the minimizer y^*.

Response:

Given Objective function:

$$J(\Theta) = \frac{1}{2}\sum_{i=1}^{n}(x_i^T\Theta - y_i)^2$$

1 Question 1 1 / 1

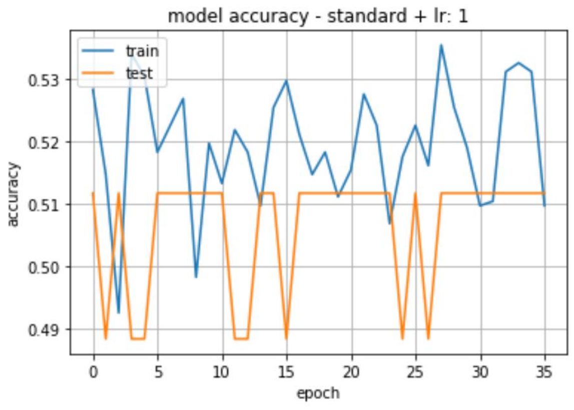
✓ **- 0 pts** Correct

- 0.15 pts Show better performance on test set

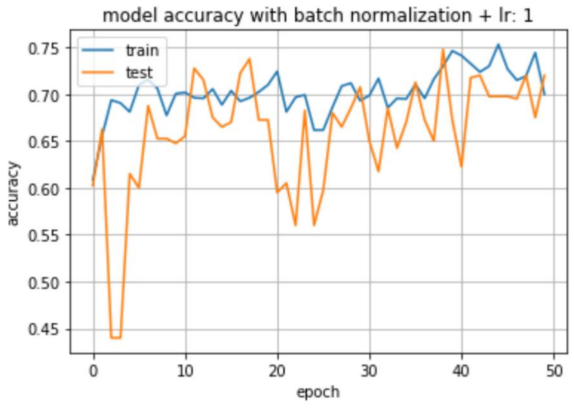
- 0.15 pts Show batchnorm can permit a higher learning rate

- 0.05 pts Show test accuracy for higher learning rate.

- 0.05 pts Legends should include learning rate



(a) Figure 5: Training without BatchNorm with learning rate =1



(b) Figure 6: Training with BatchNorm with learning rate = 1

Question 2. *Stochastic Gradient Descent*

In this problem, you will build your own solver of Stochastic Gradient Descent. Do not use built-in solvers from any deep learning packages. In this problem, you will use stochastic gradient descent to solve

$$\min_y \frac{1}{n}\sum_{i=1}^{n} |y - x_i|^2, \tag{1}$$

where x_i is a real number for $i = 1 \dots n$.

(a) Using calculus, derive a closed-form expression for the minimizer y^*.

Response:

Given Objective function:

$$J(\Theta) = \frac{1}{2}\sum_{i=1}^{n}(x_i^T\Theta - y_i)^2$$

can also be written as

$$= \frac{1}{2}(X\Theta - y')^T(X\Theta - y')$$

after expansion

$$= \frac{1}{2}(\Theta^T X^T X\Theta - \Theta^T X^T y' - y'^T X\Theta + y'^T y')$$

To minimize the cost function J, we take the derivative and equate it to zero

$$\nabla_\Theta J = \frac{1}{2}\nabla_\Theta tr(\Theta^T X^T X\Theta - \Theta^T X^T y' - y'^T X\Theta + y'^T y')$$

$$= \frac{1}{2}(\nabla_\Theta tr\Theta^T X^T X\Theta - 2\nabla_\Theta tr y'^T X\Theta + \nabla_\Theta tr y'^T y')$$

$$= \frac{1}{2}(X^T X\Theta - 2X^T y' + X^T X\Theta)$$

$$= X^T X\Theta - X^T y' = 0$$

Therefore closed form is

$$\Theta^* = (X^T X)^{-1} X^T y'$$

2.1 a **0.45 / 1**

- **- 0 pts** Correct
- ✓ **- 0.25 pts** **Incorrect Derivation for the Gradient**
- ✓ **- 0.1 pts** **Incorrect Closed Form**
- **- 0.5 pts** No Work
- ✓ **- 0.2 pts** **Additional Parameter Theta**
- **- 0.25 pts** Did not solve for y*
- **- 0.05 pts** Calculation Error

There no parameter theta in the objective function, we are trying to minimize y. This is essentially the linear regression derivation for theta.

(b) Generate points $x_i \sim \text{Uniform}[0,1]$ for $i = 1 \dots 100$. Use Stochastic Gradient Descent with a constant learning rate to solve (1). Use $G(y) = \frac{d}{dy}|y - x_i|^2$ for a randomly chosen $i \in \{1 \dots n\}$. Create a plot of MSE error (relative to y^*) versus iteration number for two different learning rates. Make sure your plot clearly shows that SGD with the larger learning rate leads to faster initial convergence and a larger terminal error range than SGD with the smaller learning rate.

Response:

Clearly we can see that SGD with large learning rate(0.01) converges faster than with less learning rate (0.001)

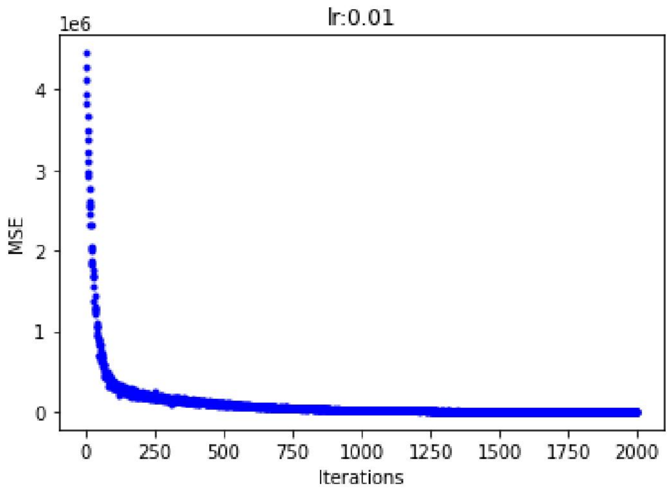


Figure 4: SGD with lr 0.01

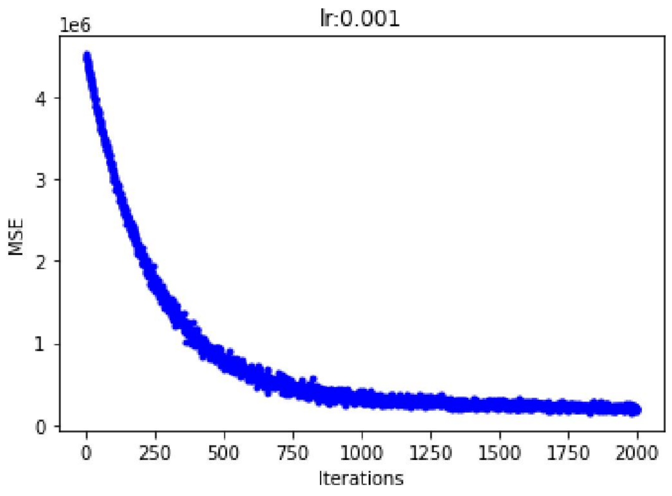


Figure 5: SGD with lr 0.001

2.2 b 0.9 / 1

- **- 0 pts** Correct
- **- 0.05 pts** Give an explanation of the plot
- **- 0.2 pts** Unclear Terminal Error
- **- 0.05 pts** The plot is relative to x not y*
- **- 0.3 pts** Not the effect the problem is looking for
- **- 0.1 pts** Smaller learning rate does not converge
- **- 0.1 pts** Need to plot on log scale to see the effect
- ✓ **- 0.1 pts** **Scaling is off**

Question 3. *Momentum, RMSProp, and Adam — revised*

Define

$$f_1(x,y) = x^2 + 0.1y^2,$$
$$f_2(x,y) = \frac{(x-y)^2}{2} + 0.1\frac{(x+y)^2}{2}.$$

In this problem you will minimize these two functions using four optimizers: GD, GD with momentum 0.9, RMSProp, and Adam with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For each, use learning rate 10^{-3}. When optimizing f_1, initialize at $(1,1)$. When optimizing f_2, initialize at $(\sqrt{2},0)$.

(a) Before numerically solving these 8 optimization problems, guess the ranking (from fastest to slowest) of the rates of convergence of the 8 solutions. Explain your reasoning. You might guess that some converge at the same rate as others. (This subproblem will be graded only on effort and not on correctness).

Response:

Ranking based of rate of convergence (highest to lowest)

1. Adam Optimizer (f_1)
2. Adam Optimizer (f_2)

Adam Optimizer should be the fastest to converge when compared to rest as it can handle sparse gradients and noisy problem. f1 may converge faster because of less complexity.

3. RMSProp (f_1)
4. RMSProp (f_2)

Rmsprop Optimizer will control the vertical oscillations hence works better than vanilla GD or GD with momentum. f1 may converge faster because of less complexity.

5. GD with Momentum (f_1)
6. GD with Momentum (f_2)

GD with an added momentum helps to accelerate gradients in right direction thus converging faster.f1 may converge faster because of less complexity.

7. GD (f_1)
8. GD (f_2)

One disadvantage of the SGD method is that its update direction depends entirely on the current data, so its update is very unstable. Hence it will be the slowest. f1 may converge faster because of less complexity.

3.1 a **1 / 1**

✓ **- 0 pts** Correct

(b) Numerically solve the 8 optimization problems described above. Plot the objective as a function of iteration count. Perform at least 2000 iterations. You are encouraged to use the built-in optimizers for these algorithms in PyTorch.

Response:

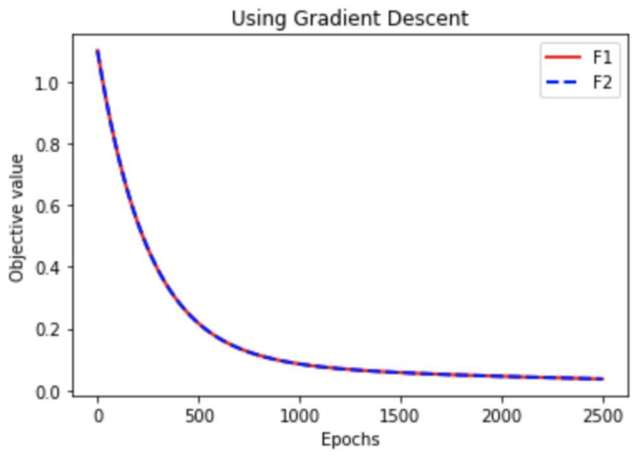


Figure 6: GD optimization

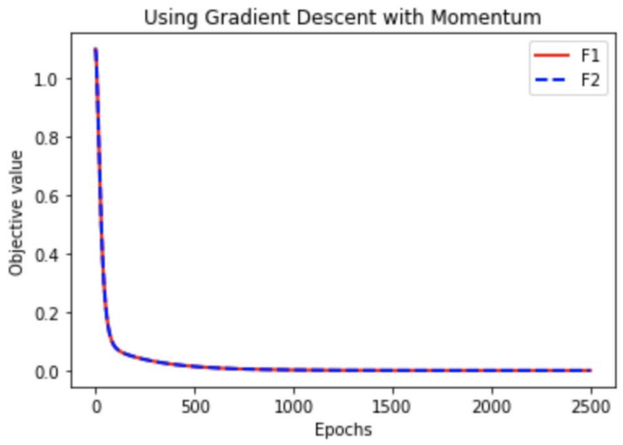


Figure 7: GD optimization with Momentum

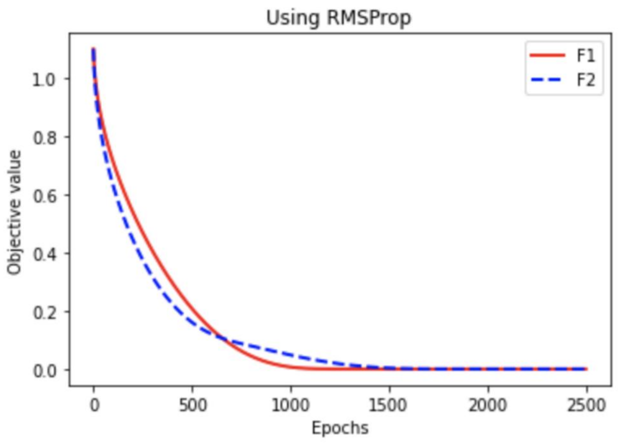


Figure 8: RMSProp Optimizer

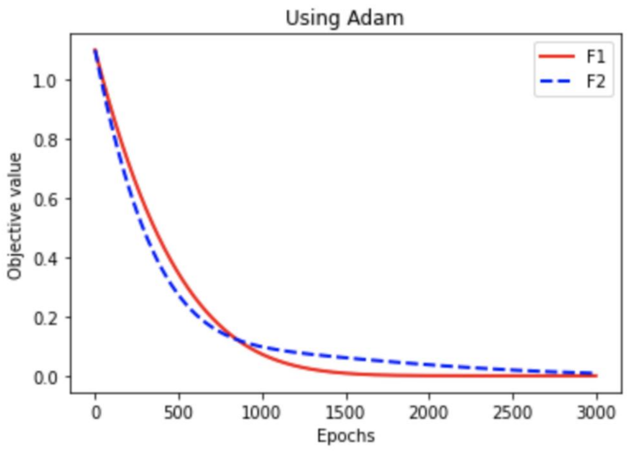


Figure 9: Adam Optimizer

3.2 b **1 / 1**

✓ **- 0 pts** Correct

- 0.15 pts Momentum should converge first

(c) For each optimizer, comment on whether convergence rate was the same for f_1 and f_2. Why was it the same or why was it different? Explain. You might find it useful break up the explanation in terms of a first phase and a second phase.

Response:

Convergence rate was similar with GD and GD with momentum for both the given functions. When we take the derivative of these function both have the same order and direction. Adam and RMSProp provides a faster convergence because it "smoothen" out these steps of gradient descent to follow a less noisy path and converge faster. RMSProp implicitly performs simulated annealing; heading towards the minima, we slow down in order not to overshoot the minima. RMSProp decreases the size of the steps towards minima when the steps are too large.

3.3 C **0.75 / 1**

- **- 0 pts** Correct
- ✓ **- 0.05 pts** **Need to explain why f1 ad f2 are different (they rotate by 45 degrees)**
- ✓ **- 0.1 pts** **Unclear explanation why gd and gd with momentum is the same for f1 and f2.**
- ✓ **- 0.1 pts** **Unclear why adaptive learning optimization schemes are different for f1 and f2.**
- **- 0.05 pts** Incorrect reason for f1 and f2 being the same for gd and gd with momentum.
- **- 0.1 pts** Analyzed incorrect optimizers.
- **- 0.05 pts** Adaptive learning optimization performance is dependent on the curvate of function.

Looking for gd and gd with momentum commute with rotation and the adaptive learning methods (adam and rmsprop) do not.