Deep Learning (Lab)

Lab 3

윤 준 영 (202252001)

1. Define NNs with 5 layers. Compare performance with/without weight regularization and dropout.

   Base Model:

   (input): Linear(in_features=784, out_feature=512, bias=True)

   (hidden1): Linear(in_features=512, out_feature=512, bias=True)

   (hidden2): Linear(in_features=512, out_feature=512, bias=True)

   (hidden3): Linear(in_features=512, out_feature=512, bias=True)

   (out): Linear(in_features=512, out_feature=10, bias=True)

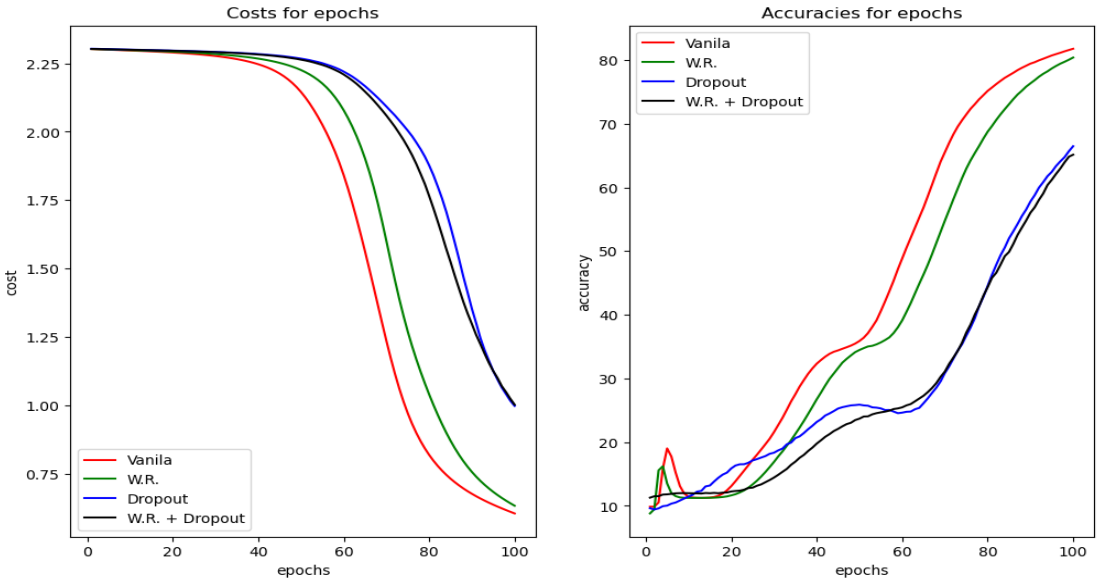
   Optimization Function: ReLU

   Learning Rate: 0.1

[Result]

Model	Train Accuracy (%)	Test Accuray (%)
Vanilla	81.79	82.37
Weight Decay (λ: 1e-3)	80.41	81.3
Dropout (Prob: 0.3)	66.48	73.54
Weight Decay (λ: 1e-3) + Dropout (Prob: 0.3)	65.15	73.17

*Dropout was adopted in front of all hidden and out layers.



[Conclusion]

문제 4번까지 시행한 결과, 학습이 Overfitting 되려면, 좀 더 복잡한 7 Layer를 사용하거나, Epoch 수를 늘려야 할 것으로 보인다. 100 Epochs를 사용하였을 때에는 Overfitting이 일어

나지 않았고, Weight Decay나 Dropout은 오히려 성능이 떨어지는 모습을 보인다. 하지만 Weight Decay나 Dropout을 사용하면서 Epoch을 늘려주면 해결될 문제로 보이며 (Costs for epochs graph가 계속해서 감소하는 방향으로 진행되고 있기 때문에), 그렇게 된다면 Overfitting을 줄이면서 성능을 높이는 방법이 될 수 있을 것으로 보인다.

2. Define NNs with 7 layers. Compare performance with different initializations such as random normal and He initializations. You can try other initializations as well.

   Base Model:

   (input): Linear(in_features=784, out_feature=512, bias=True)

   (hidden1): Linear(in_features=512, out_feature=512, bias=True)

   (hidden2): Linear(in_features=512, out_feature=512, bias=True)

   (hidden3): Linear(in_features=512, out_feature=512, bias=True)

   (hidden4): Linear(in_features=512, out_feature=512, bias=True)

   (hidden5): Linear(in_features=512, out_feature=512, bias=True)

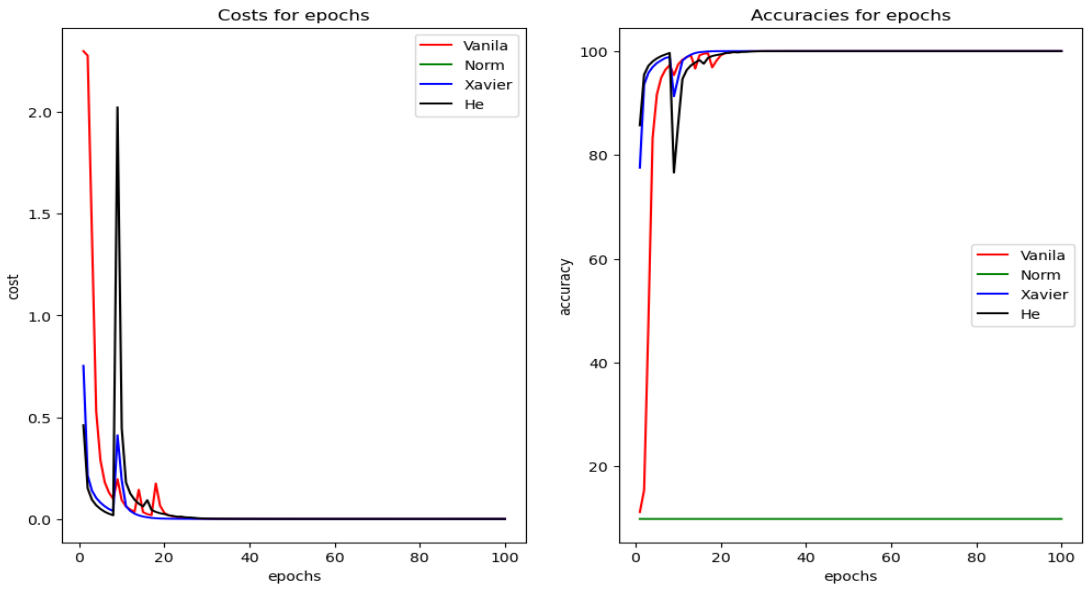
   (out): Linear(in_features=512, out_feature=10, bias=True)

   Optimization Function: ReLU

   Learning Rate: 0.1

[Result]

Model	**Train Accuracy (%)**	**Test Accuray (%)**
Vanilla	100.00	97.85000000000001
Normal (mean: 0, std: 1)	9.87	9.8
Xavier Normal	100.00	98.17
He Normal	100.00	97.74000000000001



[Conclusion]

Weight를 Normal하게 Initialization 하였을 때에는 학습이 진행되지 않는 모습을 보였고, 다른 방법들에서는 큰 성능 차이가 존재하지 않게 학습이 진행되었다. ReLU를 사용하였을 때에는 He 를 일반적으로 사용한다고 하였으나, 큰 차이가 없는 것 같다.

3. Define NNs with 7 layers. Compare performance with different optimization methods such as stochastic gradient descent and Adam optimization. You can try other optimization methods as well.

   Base Model:

   (input): Linear(in_features=784, out_feature=512, bias=True)

   (hidden1): Linear(in_features=512, out_feature=512, bias=True)

   (hidden2): Linear(in_features=512, out_feature=512, bias=True)

   (hidden3): Linear(in_features=512, out_feature=512, bias=True)

   (hidden4): Linear(in_features=512, out_feature=512, bias=True)

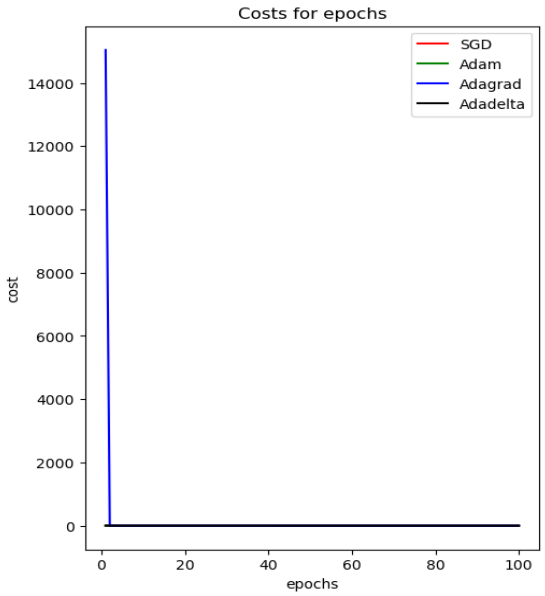
   (hidden5): Linear(in_features=512, out_feature=512, bias=True)

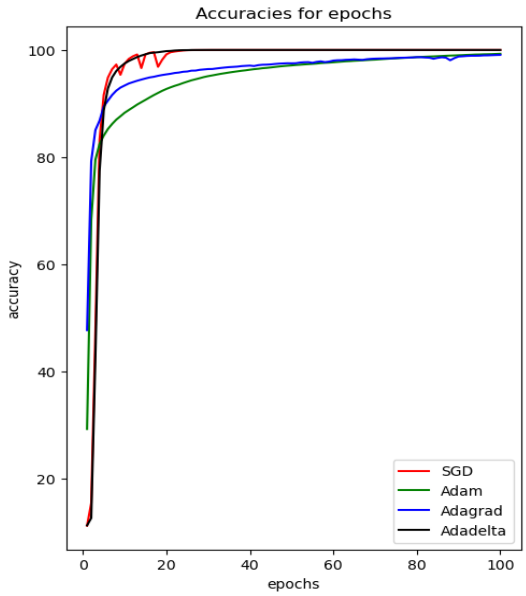
   (out): Linear(in_features=512, out_feature=10, bias=True)

   Optimization Function: ReLU

[Result]

Model	Train Accuracy (%)	Test Accuray (%)
SGD (lr: 0.1)	100.00	97.85000000000001
Adam (lr: 1e-5)	99.29	96.96000000000001
Adagrad (lr: 0.1)	99.08	93.78
Adadelta (lr: 0.1)	100.00	97.6

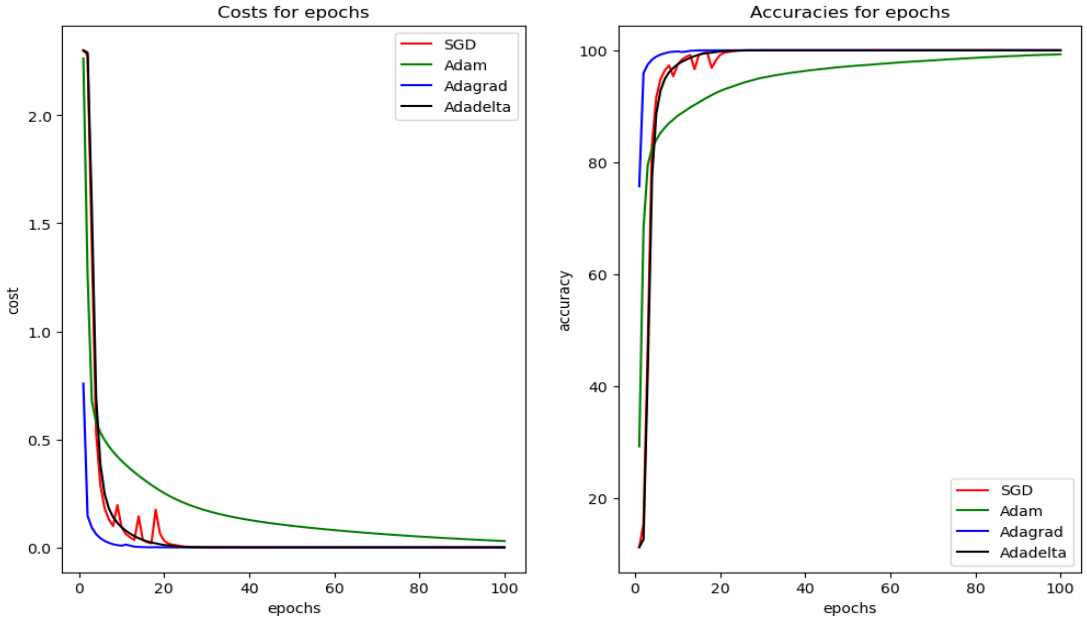




[Conclusion]

MNIST dataset은 그렇게 복잡하지 않은 데이터이기에 어떤 Optimization method를 쓰는 지에 큰 영향을 받지 않는 것으로 보인다. Adadelta, SGD, Adagrad, Adam 순으로 빠르게 Saturation 되는 모습이 보이지만, 모델마다 다른 Learning Rate를 사용하였기 때문에 정확한 비교라고 보기 힘들다. 하지만 같은 Learning Rate를 사용한 SGD, Adagrad, Adadelta를

보았을 때, 확실히 Adadelta가 높은 성능을 보이고 있고, Adam은 가장 큰 Learning Rate를 사용하였음에도 상대적으로 높은 성능을 보이는 것 같지는 않다. 또한, Training 시의 Cost는 Adagrad의 Cost 최대값이 상대적으로 크게 나와서 저런 문제가 발생한 것으로 보인다. Learning Rate 0.1이 큰 값인 것으로 보이며, 0.01로 변경하면 해당 문제가 발생하지 않는다. (아래의 그래프 참조) Adagrad는 Learning Rate가 어느 정도 큰 값이 들어가더라도 잘 대응하여 Converge 시키는 값을 찾아내는 좋은 Optimization 방법인 것 같다.



4. Define NNs with 7 layers. Compare convergence speed with/without batch normalization or layer normalization. We haven't learned the batch normalization yet, but coding is not difficult. Please refer to the last page of the class materials or browse webpages.

   Base Model:

   (input): Linear(in_features=784, out_feature=512, bias=True)

   (hidden1): Linear(in_features=512, out_feature=512, bias=True)

   (hidden2): Linear(in_features=512, out_feature=512, bias=True)

   (hidden3): Linear(in_features=512, out_feature=512, bias=True)

   (hidden4): Linear(in_features=512, out_feature=512, bias=True)

   (hidden5): Linear(in_features=512, out_feature=512, bias=True)

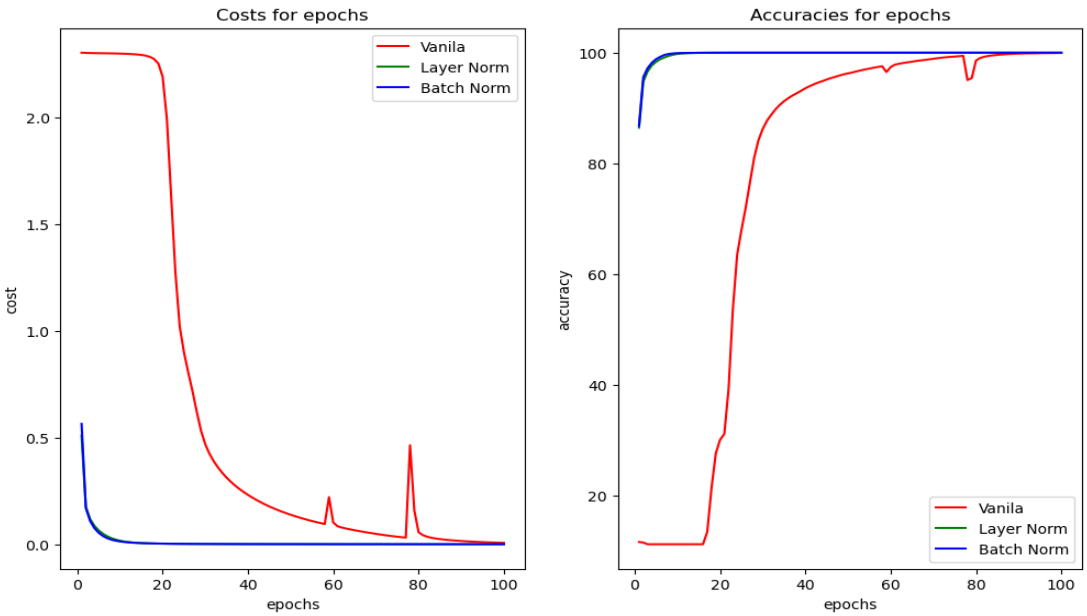
   (out): Linear(in_features=512, out_feature=10, bias=True)

   Optimization Function: ReLU

   Learning Rate: 1e-2

[Result]

Model	**Train Accuracy (%)**	**Test Accuray (%)**
Vanilla	99.95	96.86
Layer Norm	100.00	97.94
Batch Norm	100.00	97.86



[Conclusion]

Layer Norm과 Batch Norm을 하였을 때, Normalization을 하지 않은 Model보다 훨씬 빠르게 Saturation 되는 것을 볼 수 있다. 하지만 Layer Norm과 Batch Norm에서의 속도에는 큰 차이가 발생하지 않는데, 그 이유는 정확하게 모르겠다.