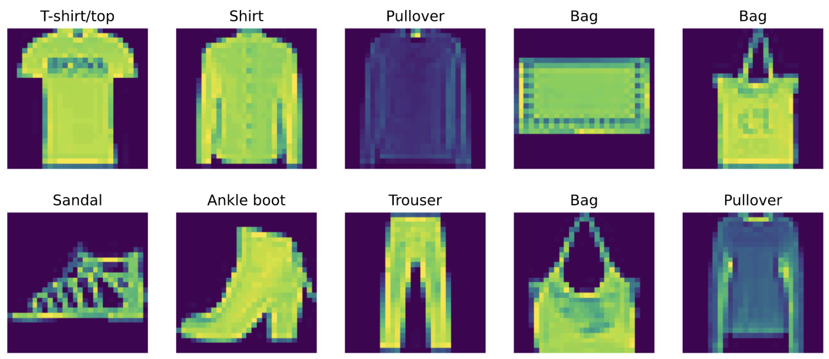
Project 0 Mark Palmer

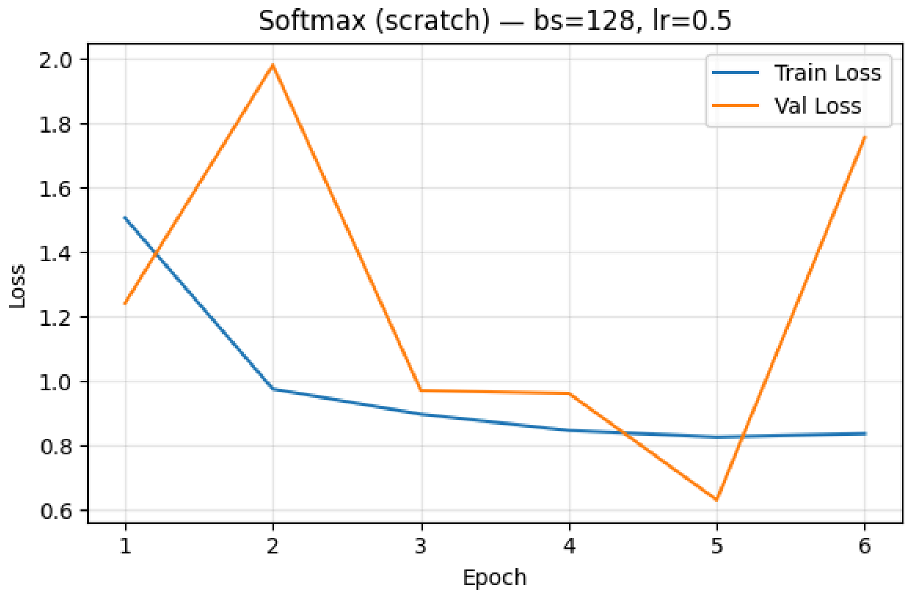
Part 1

Q1) Visualize 10 images from the dataset.



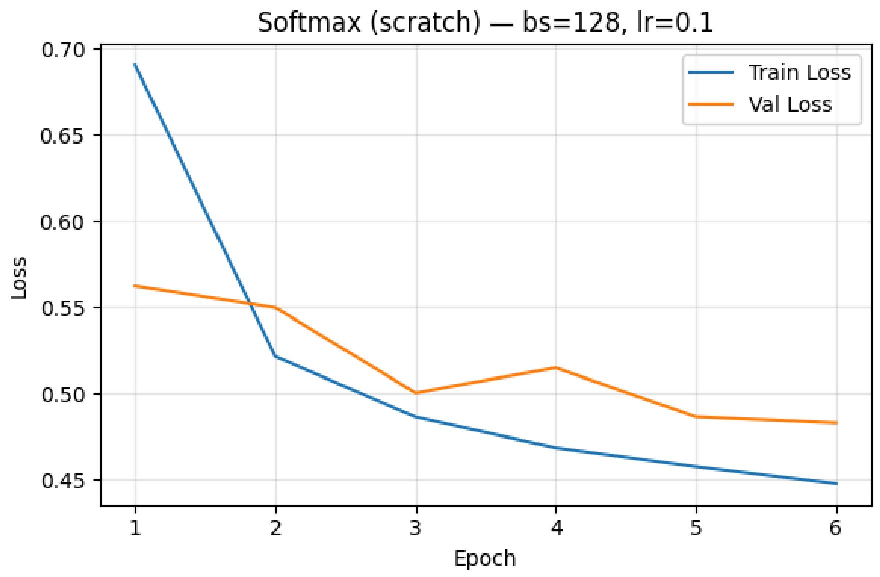
Q2.1) Try to use different learning rates, plot how validation loss changes.

LR .5



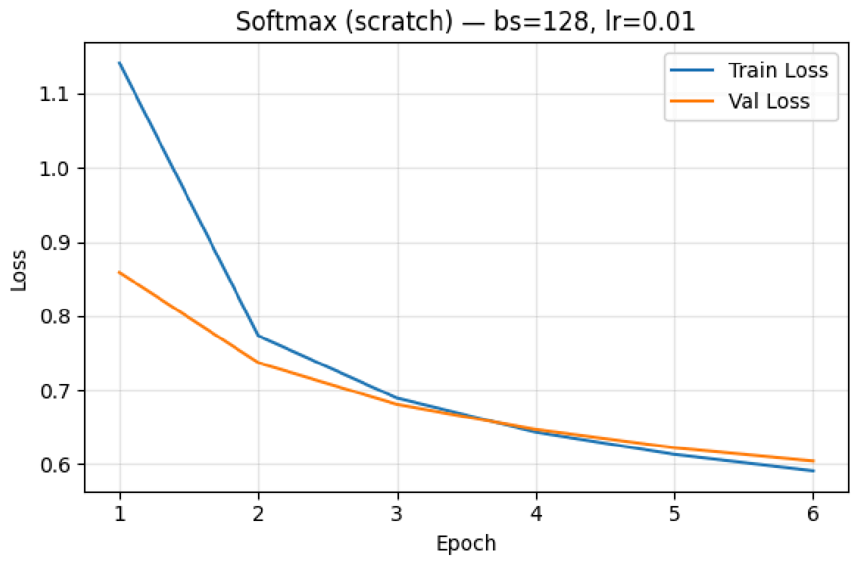
The validation loss varies greatly from epoch to epoch, verifying the instability of this high learning rate.

LR .1



The validation loss is much more consistent than a learning rate of 0.5 and lower than a learning rate of 0.1. Best trade off option

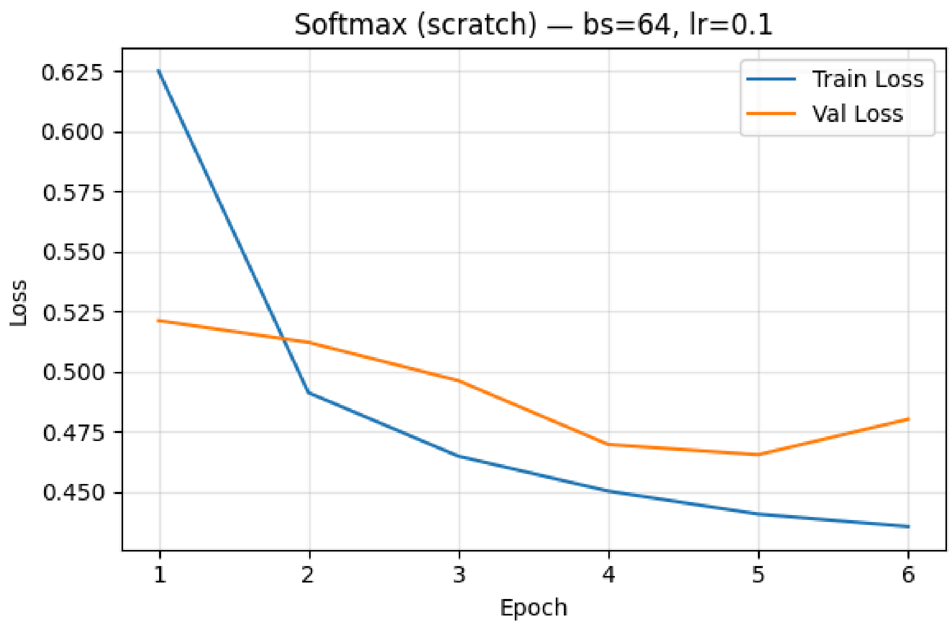
LR .01



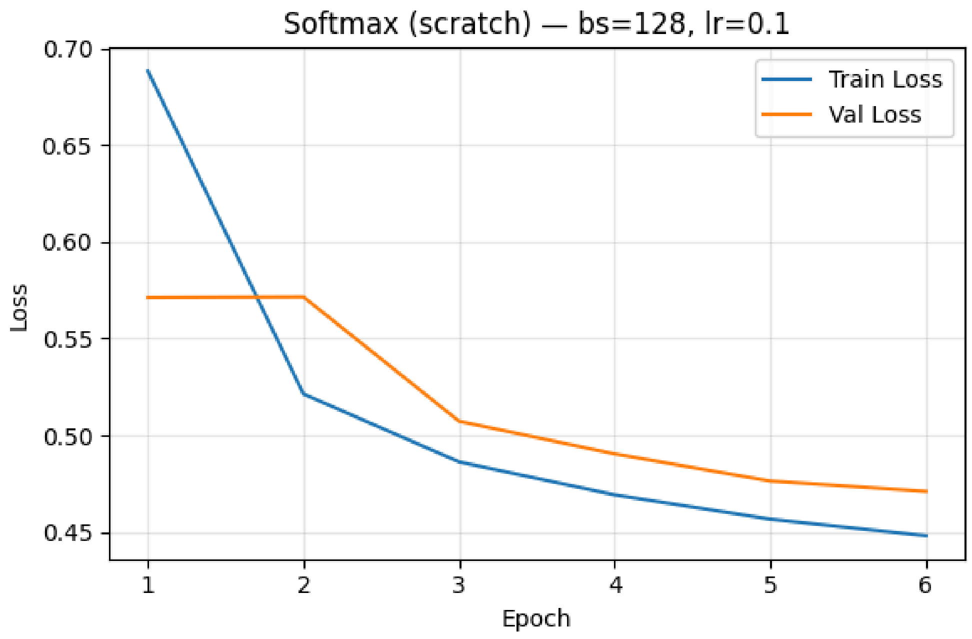
The validation loss is very consistent but the validation loss is much greater than a learning rate of 0.1, and this model is probably underfitting

Q2.2) Try to use different batch sizes, how validation and training loss changes?

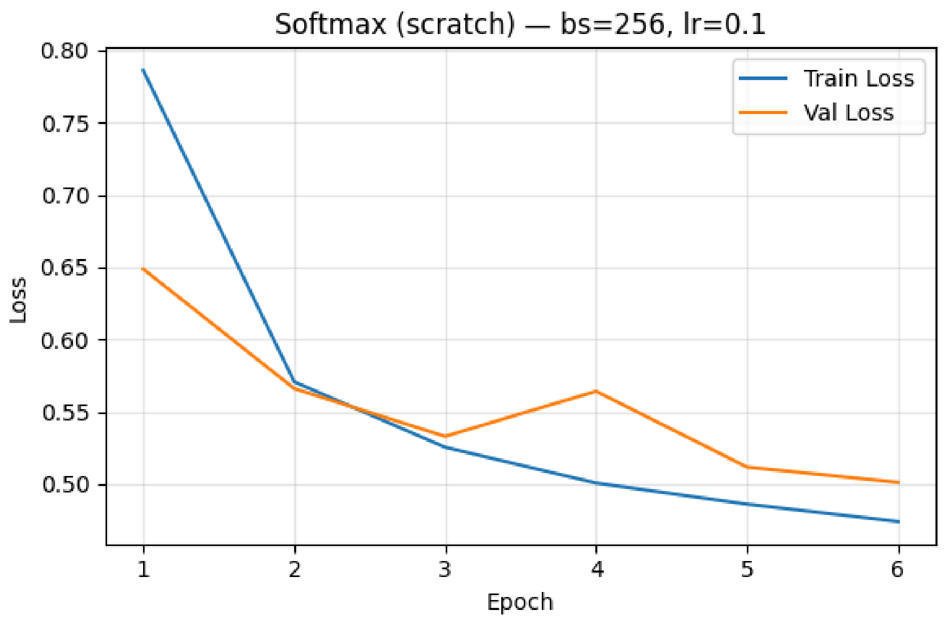
BS 64


Softmax (scratch) — bs=64, lr=0.1
Train Loss
Val Loss
Loss
Epoch
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0.450
1
2
3
4
5
6

BS 128


Softmax (scratch) — bs=128, lr=0.1
Train Loss
Val Loss
Loss
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
1
2
3
4
5
6

BS 256



Q3) Explain the overflow and underflow problems in softmax computation

Numeric overflow can be caused when exponentiating or taking logarithms of large positive or negative numbers. Underflow is caused when taking the log(0) and getting a NaN value in either the numerator or denominator of the softmax function.

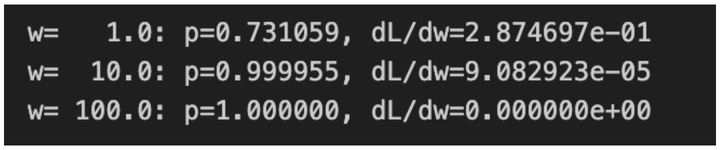
To solve overflow, because softmax is shift-invaritant, we can subtract both the numerator and the denominator by the max value we can keep the exponents between (0,1].

By subtracting the max we can solve the overflow problem but if oj-ox(max) = 0 then we still get a NaN value due to underflow. To solve this we can combine max-shifting with a log-sum-exp formulation.

Part 2

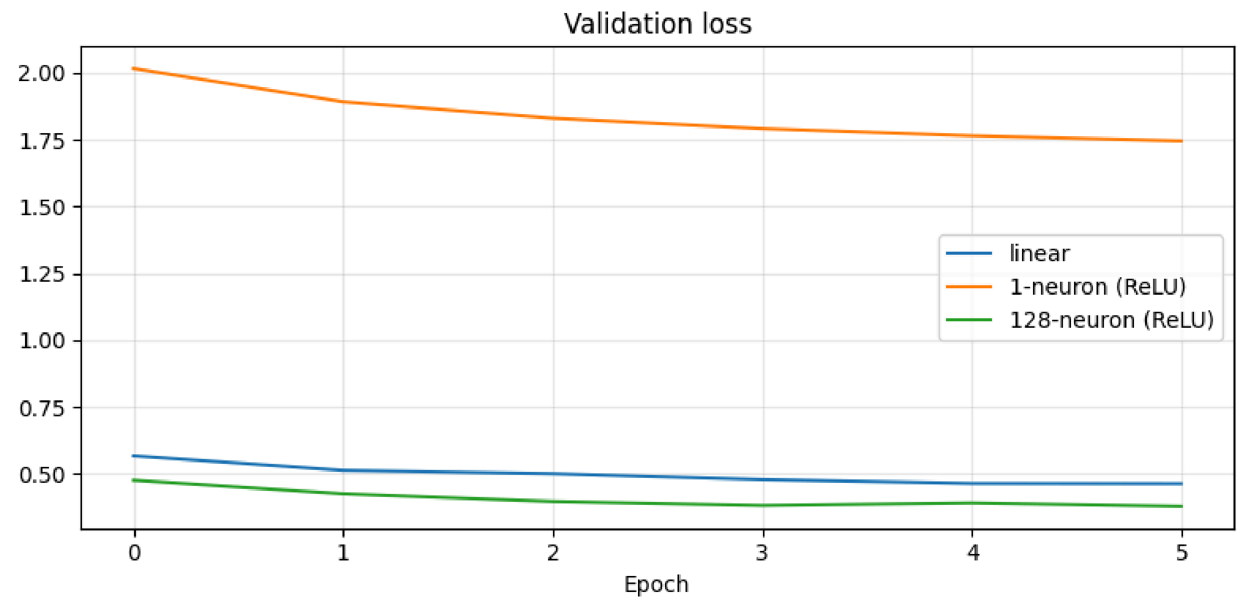
Q1) Come up with an example that the gradients vanish for the sigmoid activation function.

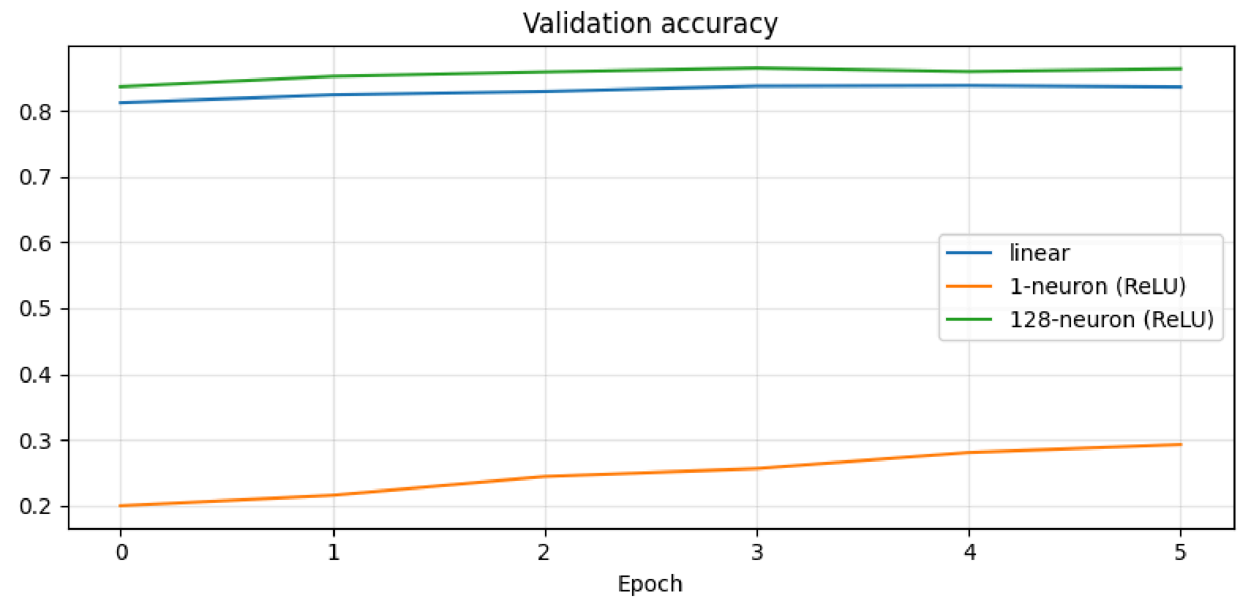
Using MSE for a loss function and a one layer sigmoid model, as the weights increase and the gradients vanish (dL/dw), with weight of 100 the gradient completely vanishes



```
w=   1.0: p=0.731059, dL/dw=2.874697e-01
w=  10.0: p=0.999955, dL/dw=9.082923e-05
w= 100.0: p=1.000000, dL/dw=0.000000e+00
```

Q2) Give it a try to add one hidden layer, show how it affects the results. What if adding a hidden layer with one single neuron, what's the results? Please discuss what you get and why.





Training with a hidden layer decreased the model's original loss. But training with a single neuron layer significantly increased the validation loss for the model.
Overall, the benefits are clear for adding hidden layers with multiple neurons.

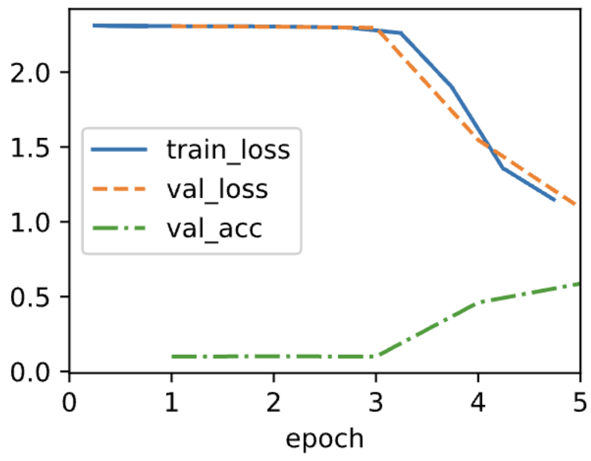
Q3) What's the memory footprint for training and prediction in the model described in the above example?
The memory footprint based on parameters, training and prediction (adam)

	Params	Training	Prediction
Linear =	(784 x 10 + 10)	520KB	430KB
1 Layer 1 Neuron =	805	410KB	401KB
1 Layer 128 Neurons =	100,000	2MB	.86MB

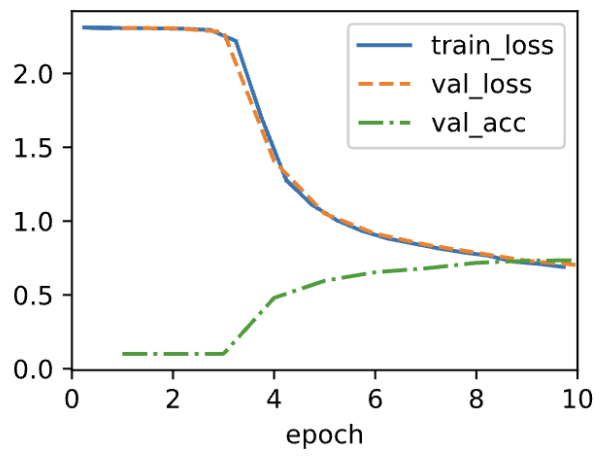
Part 3

Q1) Use different epochs (5, 10, 20) to train your LeNet, plot the training loss, validation loss, and validation accuracy of each training process, and discuss how changing training epochs affects the performance.

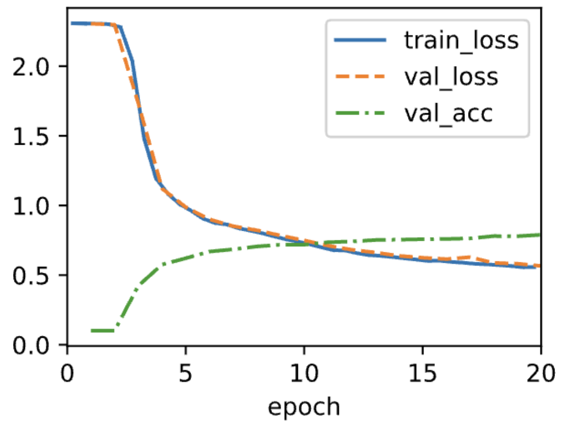
Epoch 5



Epoch 10

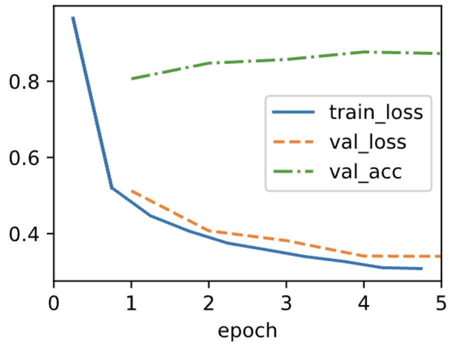


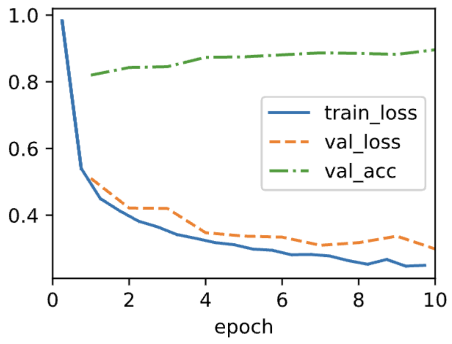
Epoch 20

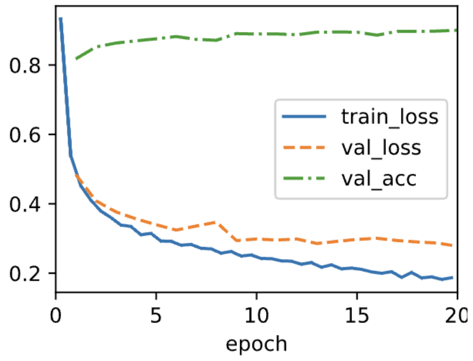


The validation accuracy is very low, the model took till epoch 5 to learn, adding epochs helps learn early, ideal epoch for this model would be 10-15

Q2) Modifying LeNet: Replace average pooling with max-pooling, replace the `Sigmoid` layer with ReLU. After doing so, redo the above experiment (Using different epochs 5, 10, 20), plot the training loss, validation loss, and validation accuracy for each training process. Discuss the changes compared to the results of original LeNet training.

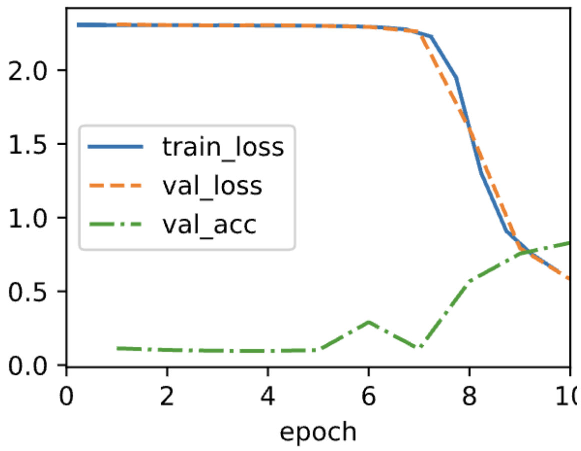






These results are much better, the max pool boosted validation accuracy, validation accuracy starts much higher with this model.

Q3) Use a different dataset, MNIST(PyTorch, TensorFlow), to do training and plot training loss/accuracy and test accuracy, and discuss the results.



The model takes almost 7 epochs before it starts to learn, the activation could be saturated and the model is early in training. The train loss is also steep which could mean slight over fitting but its because the dataset is less complex. The ReLU implementation would take the model farther.