Part 1 - Switching to a non-linear model

In class, we created a simple linear model ($w_1x + b$) in order to predict the equation for a Celsius to Fahrenheit conversion. This model worked really well, because the equation to convert between the two is indeed linear. Now, we are tasked with changing the structure of the model to be polynomial regression instead of just linear ($w_2x^2 + w_1x + b$). Changing this was pretty straightforward, as all I had to do was update the forward pass of the model and add one extra parameter tensor. The learning rate used for the graphs below was 0.1. Changing the learning rate to 0.01, for example, had almost no impact on the final loss and training sequence for the linear model. On the other hand, reducing the learning rate to 0.01 for the polynomial model caused it to train slower and not reach the loss reached by the 0.1 learning rate model. The final loss was around 14 compared to 2.6 for the 0.1 learning rate model.

Loss for Polynomial Model

Epoch	Loss
0	312798
500	4.003
1000	3.859
1500	3.727
2000	3.56
2500	3.363
3000	3.164
3500	3.131
4000	20.986
4500	2.734
5000	2.632

Loss for Linear Model

Epoch	Loss
0	1158.9
500	6.06
1000	2.97
1500	2.92
2000	2.92
2500	2.92
3000	2.92
3500	2.92
4000	2.92
4500	2.92
5000	2.92

Although the loss for the polynomial model was lower than the linear model at the end of training, the model overall is a worse for the data. If there were more data points to test on, this relationship would become more obvious. In reality, the linear model would do much better on a larger dataset. It's important to note that the w_2 parameter was basically zero in the final trained polynomial model.

Part 2 - Training on a real dataset

Using the same linear model from above, we can train on a real dataset. Data loading was taken care of by creating a class based on **torch.utils.data.Dataset** and implementing methods **len** and **get_item**, which returned the length of the dataset and the i^{th} datapoint, respectively. Data normalization of all columns was also handled here, and the resulting columns were stored as tensors on the GPU.

The training loop remained largely the same, with the only change being the shift to the data_loader class instead of the previous approach with only one input and output tensor. A batch size of 128 was used and the learning rate remained the same at 0.1, which seemed to be sufficient for training. The amount of epochs was reduced because the gains from training were almost nonexistent after **~250** iterations. Validation loss was also calculated after every 50 iterations.

0.1 learning rate:

Epoch 0, Loss: 0.06933233141899109, Validation Loss: 0.183798685669899
Epoch 50, Loss: 0.01373199000954628, Validation Loss: 0.29358336329460144
Epoch 100, Loss: 0.01858801580965519, Validation Loss: 0.2893568277359009
Epoch 150, Loss: 0.02208026312291622, Validation Loss: 0.2848132848739624
Epoch 200, Loss: 0.01798155903816223, Validation Loss: 0.3475337028503418
Epoch 250, Loss: 0.027690652757883072, Validation Loss: 0.3016890585422516

0.01 learning rate:

Epoch 0, Loss: 0.10799746215343475, Validation Loss: 0.625154435634613
Epoch 50, Loss: 0.015391075052320957, Validation Loss: 0.3070577383041382
Epoch 100, Loss: 0.012124455533921719, Validation Loss: 0.2979496419429779
Epoch 150, Loss: 0.018195969983935356, Validation Loss: 0.2907794713973999
Epoch 200, Loss: 0.011080420576035976, Validation Loss: 0.29371151328086853
Epoch 250, Loss: 0.008752954192459583, Validation Loss: 0.2952178418636322

0.001 learning rate:

Epoch 0, Loss: 0.047321565449237823, Validation Loss: 0.38668158650398254
Epoch 50, Loss: 0.04448213055729866, Validation Loss: 0.34275132417678833
Epoch 100, Loss: 0.03864359110593796, Validation Loss: 0.32555219531059265
Epoch 150, Loss: 0.02882867120206356, Validation Loss: 0.3140195906162262
Epoch 200, Loss: 0.021091708913445473, Validation Loss: 0.3014761507511139
Epoch 250, Loss: 0.014429564587771893, Validation Loss: 0.2985224723815918

The best learning rate for this model seems to be 0.01. That model ended with the lowest final loss and one of the lowest validation losses by the final epoch.

There are a couple major differences between this model and the one we built by hand in homework 2. But the results from both models are surprisingly similar. At the end of the day, the PyTorch model is a little more accurate, but took longer to train than the hand-built algorithm. The final parameters are only off by a small margin.

Part 3: The same, but with all parameters

Learning rate 0.1:

Epoch 0, Loss: 0.17915476858615875, Validation Loss: 0.2580113410949707
Epoch 50, Loss: 0.011121124029159546, Validation Loss: 0.3566964864730835
Epoch 100, Loss: 0.009300372563302517, Validation Loss: 0.3386739194393158
Epoch 150, Loss: 0.010756883770227432, Validation Loss: 0.4039651155471802
Epoch 200, Loss: 0.018699809908866882, Validation Loss: 0.3407701253890991
Epoch 250, Loss: 0.019792521372437477, Validation Loss: 0.3908078372478485

Learning rate 0.01:

Epoch 0, Loss: 0.03550412878394127, Validation Loss: 0.3764609098434448
Epoch 50, Loss: 0.01172765251249075, Validation Loss: 0.35122761130332947
Epoch 100, Loss: 0.02099722996354103, Validation Loss: 0.3570949137210846
Epoch 150, Loss: 0.010638284496963024, Validation Loss: 0.36363813281059265
Epoch 200, Loss: 0.00683095445856452, Validation Loss: 0.3544626533985138
Epoch 250, Loss: 0.0171627476811409, Validation Loss: 0.35210758447647095

Learning rate 0.001:

Epoch 0, Loss: 0.095384880900383, Validation Loss: 0.21835893392562866
Epoch 50, Loss: 0.042620524764060974, Validation Loss: 0.21314625442028046
Epoch 100, Loss: 0.022965336218476295, Validation Loss: 0.2717965245246887
Epoch 150, Loss: 0.008240364491939545, Validation Loss: 0.3095632493495941
Epoch 200, Loss: 0.018794963136315346, Validation Loss: 0.32853108644485474
Epoch 250, Loss: 0.013264286331832409, Validation Loss: 0.34004905819892883

The best learning rate here seems to be 0.001. This model has the lowest final loss and validation accuracy. Again, the parameters produced by this model are very similar to the ones produced by the gradient descent algorithm in homework 2.

Conclusion

Training each model to 5000 epochs would have taken a lot of time with almost no reward, so I capped all the training at 50. Beyond that, there was no real benefit to training, as in some cases the loss actually increased iteration over iteration.

https://github.com/Anu78/intro-to-ml-hw