ECGR 5105 Homework 5

Owen Bailey-Waltz (801488178)
GitHub Link: https://github.com/obaileyw-uncc/ecgr5105/tree/main/hw05_pytorch

Problem 1: Temperature model

(a & b) Training for quadratic regression

Trainings using learning rates of 0.1, 0.01, and 0.001 all fail to converge when applying the same input feature scaling (multiplication by 0.1) as the example from the text. A learning rate of 0.0001 leads to a convergent result for our quadratic model, however the training takes longer than the linear model to converge (the linear model is convergent shortly after 2500 epochs while the quadratic takes the full 5000) and gives a weaker convergence (training loss of 3.86 in the quadratic model compared to 2.93 in the linear model). Comparing our quadratic model with the linear model from the example with some example values also shows clearly that the quadratic model underperforms the linear model, particularly on the higher end where values begin to diverge rapidly.

Input temperatures for test

Features were divided by 10 on the way in to match the example model from the text.
32 °F, 50 °F, 68 °F, 86 °F, 104 °F, 122 °F
(0 °C, 10 °C, 20 °C, 30 °C, 40 °C, 50 °C)

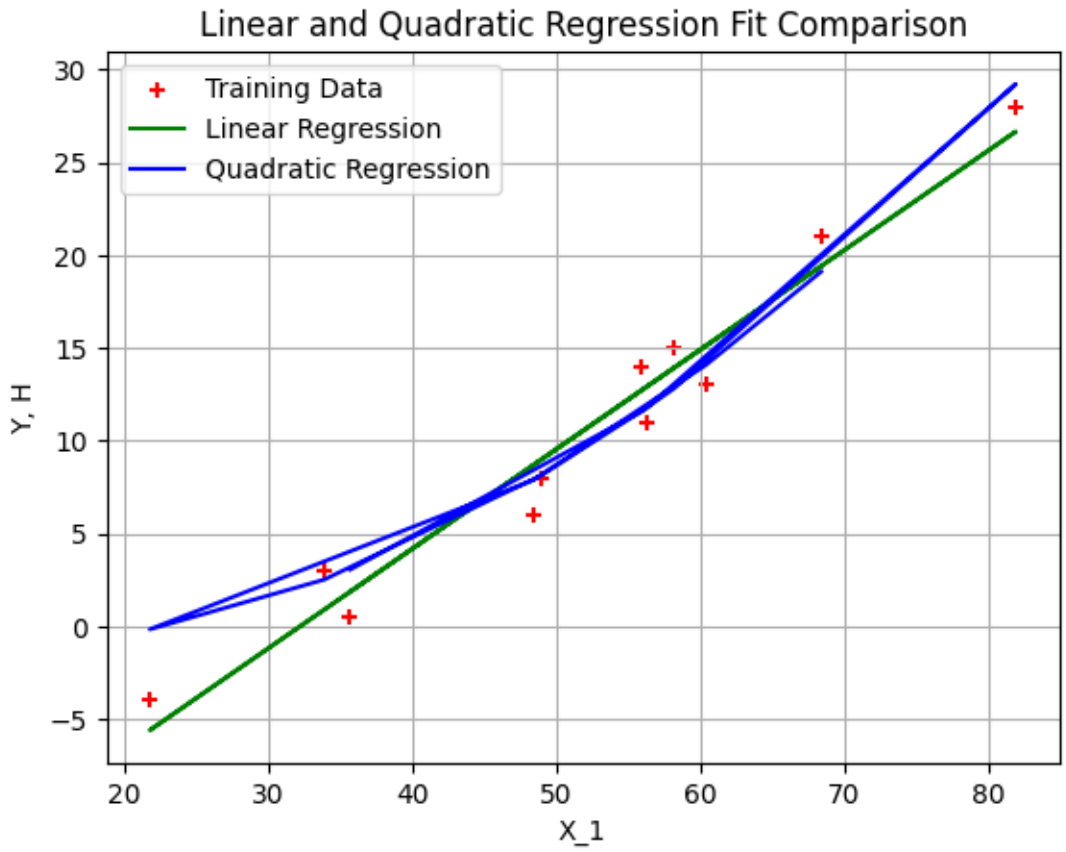
Results from quadratic model

1.99 °C, 8.61 °C, 18.84 °C, 32.68 °C, 50.13 °C, 71.19 °C

Results from linear model

-0.126 °C, 9.53 °C, 19.20 °C, 28.86 °C, 38.52 °C, 48.18 °C

(c) Comparison with linear model



Visual comparison of the quadratic and linear regression results shows that the quadratic regression model fits tightly in the central region and moves to match what it perceives as a nonlinearity on the upper end but loses fit quality near the lower end.

Problem 2: PyTorch housing dataset linear regression

(a) Multi-variable gradient descent

$$h(\boldsymbol{x}) = 100636 + 534548x_1 - 145352x_2 + 154401x_3 - 150380x_4 + 192145x_5$$

This traditional linear regression model obtained via 5000 epochs of vanilla PyTorch gradient descent.

(b) Variation in learning rate

Training for 5000 epochs causes the model's loss figures to apparently wrap around multiple times per training, possibly due to complications with floating point numbers interfering with gradient descent by providing extremely large values. Higher learning rates

cause the model's loss to wrap around more, yet the examples with higher learning rates converged to lower validation loss amounts and therefore represented better convergences. The model specified above is from the training with $\alpha = 0.1$.

Final training losses with different values of α

α = 0.1 1.459×10^{13}
α = 0.01 1.540×10^{13}
α = 0.001 1.827×10^{13}
α = 0.0015 1.885×10^{13}
α = 0.0001 3.203×10^{12}

Final validation losses with different values of α

α = 0.1 1.377×10^{13}
α = 0.01 1.503×10^{13}
α = 0.001 1.807×10^{13}
α = 0.0015 1.931×10^{13}
α = 0.0001 3.185×10^{13}

Problem 3: Fully connected neural network for the housing dataset

(a) Fully connected NN with one sigmoid-activated hidden layer

200 epochs of training on a neural network with a single hidden layer converges quickly (less than a second). The validation loss for this solution, at 1.454×10^{13}, is approximately the same as the best traditional linear regression example with vanilla gradient descent for this dataset.

(a) Fully connected NN with three sigmoid-activated hidden layers

Adding two more layers of eight neurons reduces the validation loss from 1.37×10^{13} to 1.36×10^{13}, providing a minor increase in performance with the added model precision. Increasing the model precision by doubling the number of neurons leads to a further reduction in validation loss to 8.54×10^{12}. Doubling the neurons again decreases validation loss to 2.89×10^{12} at the cost of a longer training time, which reaches about two seconds.

For the example with 1024 features per hidden layer, the model, which had been increasing in accuracy to this point, observably begins to overfit. The 1024-feature-per-layer example

and the 2048-feature-per-layer example each feature lower training losses than their lower-resolution counterparts, however the validation loss for both exceed the 256-feature example. Training time also significantly increases, taking approximately one minute for the 2048-feature example and six minutes for the 4096-feature example, illustrating the principle of how increases in feature count cause exponential increases in training time. Doubling the feature count one more time to 8192 leads to a training time of 24 minutes. The 8192-feature-per-layer example also features a comparable validation loss to the 256-feature-per-layer example.