Homework 1: Regression, Regularization, Over-Parameterization

01:198:462

In this assignment, we will build a simple data set, and explore fitting models to it.

1 Generating the Data

Before we fit a model, we need data to fit it to. The data set will be a set of points of the form $\{(\underline{x}^1, y^1), (\underline{x}^2, y^2), \ldots, (\underline{x}^N, y^N)\}$, where each $\underline{x}^i \in \mathbb{R}^{101}, y^i \in \mathbb{R}$. To generate the data points $(\underline{x}, y)$, the following instructions lay out how to generate $(x_0, x_1, x_2, \ldots, x_{100})$, and then y based on that $\underline{x}$ vector.

> Note: In the following, I am using $\mathcal{N}(\mu, \sigma^2)$ to indicate a random value sampled from a *normal distribution* with mean μ and variance σ^2. In order to generate random values, you need to specify a distribution (see **CS-206**), and normal distributions are fairly standard functionality for RNG packages. I am performing a slight abuse of notation when I write (expression) $+\mathcal{N}(\mu, \sigma^2)$, but this should be interpreted as adding a small random bit of noise sampled from that distribution to the value of (expression). All noise terms should be taken as independent. These terms are present to represent noise or fluctuations in the measurements that you collect from the real world.

- $x_0 = 1$.
- $x_1, \ldots, x_{50}$ are i.i.d. random values, with distribution $\mathcal{N}(0, 1)$.
- For $i = 51, 52, \ldots, 60$, $x_i = x_1 + 0.5 * x_{i-50} + \mathcal{N}(0, 0.1)$.

  > Note: The way to interpret this is that, for example, $x_{55} = x_1 + 0.5 * x_5 +$ (a little bit of noise).

- For $i = 61, 62, \ldots, 70$, $x_i = x_{i-60} - x_{i-50} + x_{i-40} + \mathcal{N}(0, 0.1)$.

  > Example: $x_{65} = x_5 - x_{15} + x_{25} +$ (a little bit of noise).

- For $i = 71, 72, \ldots, 80$, $x_i = x_{6*(i-70)} + 3 * x_{i-70} + \mathcal{N}(0, 0.1)$.
- For $i = 81, 82, \ldots, 90$, $x_i = 5 - x_{i-10}$.
- For $i = 91, 92, \ldots, 100$, $x_i = 0.5 * x_{50+(i-90)*4} + 0.5 * x_{50+(i-90)*3} + \mathcal{N}(0, 0.1)$.

The output data will be generated in the following way. For any input $\underline{x}$, the output is given by

$$y = \left(\sum_{i=1}^{50} (-0.88)^i x_{2i} \right) + \mathcal{N}(0, 0.01). \tag{1}$$

Note that with this setup, none of the odd-indexed components of $\underline{x}$ 'explicitly' show up in the value of y.

> Note: This specifies that the value of y is based on $\underline{x}$ via $y = \underline{w}.\underline{x} +$ (a little noise), and that the 'true' weights are given by
>
> $$\underline{w} = (0, 0, -0.88, 0, (-0.88)^2, 0, (-0.88)^3, 0, \ldots, 0, (-0.88)^{50}). \tag{2}$$

Problem 1: Write a function that takes as input a number of data points N, and gives as output a data matrix X as a PyTorch tensor, with N rows and 101 columns, where each row is an $\underline{x}$ vector as above, as well as a PyTorch tensor $\underline{y}$ with N rows and 1 column, where each entry of $\underline{y}$ is the corresponding y value for that row of X.

Some functions that may be useful:

- **torch.zeros**
- **torch.randn**

2 Fitting a Linear Model

Problem 2: Write the following functions.

- A function that takes as input a data tensor X and vector of weights $\underline{w}$, and returns a tensor of predictions $\underline{y}^{\text{pred}}$ based on a linear model, $\underline{w}.\underline{x}$ for each data point.
- A loss function that takes as input a tensor of actual y-values, $\underline{y}$, and a vector of predicted y-values, $\underline{y}^{\text{pred}}$, and returns (as a result of tensor operations) the *average* squared error between them. (Average in this case being over the number of data points in the tensors.)

A function that may be useful: **torch.matmul**.

Problem 3: Given a *training* data set $(X^{\text{train}}, \underline{y}^{\text{train}})$, and a *testing* data set $(X^{\text{test}}, \underline{y}^{\text{test}})$, and a stepsize $\alpha > 0$, write code that calculates a linear model of best fit on the training data using a stepsize of α. Keep track of the loss on the training set and the loss on the testing set over the course of training, to plot at the end of training. *What condition should you put to end your training loop?*

Problem 4: For a training set of size $N = 1000$ and a test set of size $N = 200$, consider fitting a linear model at various stepsizes α.

- What range of α lead to convergence, what don't? How can you tell that an α is too large?
- For an α with convergence, plot the loss on the training data and the loss on the testing data as a function of training time. How do the training and testing error compare over time? Is the model learning?
- For a good α, trained to convergence - does the resulting model generalize to the testing data well?
- How does the fit model compare to the 'actual' model?

Bonus: Show that there are multiple models that produce the same output, and therefore there is no 'best' model. Does this contradict there being a 'true' formula, as indicated above?

Problem 5: In this problem, we want to look at the effect of the amount of data. For $N = 1000, 500, 250, 100, 50$, consider training a model on training sets of these sizes (keeping the test set fixed at size $N = 200$). Plot the training and testing loss for each experiment over the course of training. Analyze and describe your results.

Bonus: How does the optimal α seem to depend on the number of training data points? Be thorough.

3 Regularization: Ridge and Lasso

Frequently, we are interested in finding 'simpler' models over more complex ones. This is essentially Occam's Razor (simpler models $\rightarrow$ better models), but complexity can mean different things. Two possible examples of this in regression:

- When there are multiple sets of weights that produce the same output, we generally prefer models with smaller weights, so as not to amplify any noise in the data, and to not depend too heavily on any single component of the data. Simpler $\rightarrow$ smaller weights.
- When a model has weights that are close to zero - it may indicate that the components those weights are scaling are negligible, possibly random noise, that should be excluded. We could consider 'pruning' those components or features, or pushing those weights all the way to zero. Simpler $\rightarrow$ weights that are too small are deleted / set to zero.

To achieve both of these goals, we utilize what's known as regularization. In this case, we augment the loss function so that we penalize models that are more complex, and reward models that are more simple.

- **Ridge Regression:** In this case, we take the loss function to be

$$\text{RidgeLoss}(\underline{w}) = \text{Loss}(\underline{w}) + \lambda \sum_{i=1}^{100} w_i^2. \tag{3}$$

  This loss will be smaller when not only is the loss on the data set small, but the weights are also small. Models with smaller weights will be preferred over models with the same error but larger weights. The constant $\lambda > 0$ determines exactly how strong the pressure toward small weights will be.

- **Lasso Regression:** In this case, we take the loss function to be

$$\text{LassoLoss}(\underline{w}) = \text{Loss}(\underline{w}) + \beta \sum_{i=1}^{100} |w_i|. \tag{4}$$

  By adding this penalty term, weights are pressured to be smaller - but weights that would be large without it are pressured to be slightly smaller, and weights that would be small without it are pressured all the way to zero. The threshold for this behavior is set by β - the larger β is, the more weights are pushed all the way to zero. We'll talk about this more in class and recitation, but it serves to automatically 'prune' features or components of $\underline{x}$ that are below a certain significance.

Regularization Bonus: Why is it worth keeping w_0 out of the regularization penalty terms?

Problem 6: For $N = 300$ training data, and $N = 200$ testing data, consider fitting a model by minimizing the ridge regression loss, for various values of $\lambda > 0$. Plot the testing loss (without the ridge penalty) as a function of λ. Are you able to improve generalization of your model? What range of λ seems useful?

Problem 7: For $N = 300$ training data, $N = 200$ testing data, consider fitting a model by minimizing the lasso regression loss, for various values of $\beta > 0$. Show that as β goes up, the number of non-zero terms in the fitted model go down. What features or components of the data are most persistent (surviving across a range of β)? How does that compare with what we know about how y actually depends on $\underline{x}$?

For this problem, you should generate at least these two plots:

- As a function of $\beta > 0$, plot the number of non-zero weights that survive after training.
- For each $i = 0, 1, 2, 3, \ldots, 100$, plot the β where that weight goes to 0, killing that feature.

Bonus: Note that the lasso regression term does not have a well defined gradient everywhere, because of the sharp point of the absolute value function. How does PyTorch handle this when calculating gradients? Do some research.

4 Under-Parameterization and Over-Parameterization

In the previous section, we had more data points than features in our data, i.e., we were looking at $N > 100$. This tends to be the ideal situation, since we need to find an unknown weight for each feature, and this gives us enough information to determine each weight (similar to how two data points are enough to find the slope and intercept, the two unknowns, of a line).

Sometimes, however, we may have fewer data points than we have features - this makes it difficult to determines how the underlying model should depend on each feature. We just don't have enough data. In the following problems, consider a training data set of size $N = 50$ and a test data set of size $N = 50$.

Problem 8: Let A be a matrix of random values, with k rows and 101 columns, where each entry sampled from a $\mathcal{N}(0, 1)$ distribution. Note that for any input vector $\underline{x}$, $A\underline{x}$ will be a vector of k values. We could then consider performing linear regression on the data points $(A\underline{x}, y)$ rather than $(\underline{x}, y)$. Note that if $k < 50$, this transformed data set will have fewer input features than we have data points in our data set, and thus we restore linear regression to working order.

Plot over k from 1 to 50 the testing error when, for a given k, you pick a random A to transform the input vectors by, then do linear regression on the result. You'll need to repeat the experiment for a number of A, for each k, to get a good plot. What do you notice? Does this seem to be a reasonable trend?

Problem 9: Notice that there's nothing stopping us from continuing to increase k. This puts us in a region over over-parameterization (we have more features in our data than data points), and in fact increasingly over-parameterization, if we were bold enough to take $k > 100$. One possible solution is to, when performing linear regression on the transformed $A\underline{x}$ data, do ridge regression, introducing the ridge penalty into the loss we are minimizing.

Continue the experiment, for $k = 50, 51, 52, \ldots, 200$, plotting the resulting testing error (averaged over multiple choices of A). How did you choose a good λ value? (Note that the number of weights we need to find changes with k - should this influence λ?) What do you notice?

Bonus: Why does this happen?