Quick review: training gradients and influence

Prof. Nick Vincent

Simon Fraser University

2024-02-07

Agenda

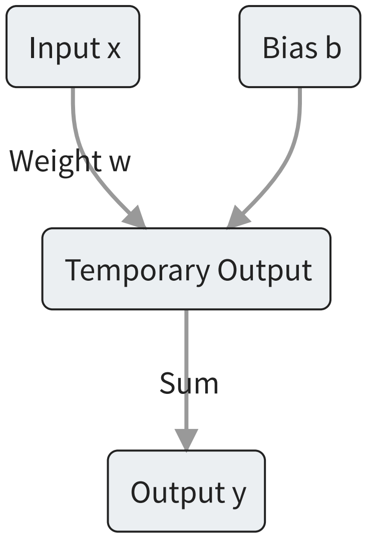
- Review training gradients
- Quick overview: retraining based data values
- Quick overview: gradient based data values

Quick review of training gradients

Imagine we're trying to predict a single output value y from a single input value x using a simple neural network. The network's prediction $\hat{y}$ is given by $\hat{y} = wx + b$

Where:

- w is the weight of the connection between the input and output neuron.
- b is the bias.
- x is the input.


Input x
Bias b
Weight w
Temporary Output
Sum
Output y

Objective of our training

- want to adjust w and b to get $\hat{y}$ close to real y
- measure how we're doing with *loss*
- example choice: Mean Squared Error (MSE)
- $L = \frac{1}{2}(y - \hat{y})^2$

Training Gradients in this example

Training gradients indicate in which direction (and by how much) we should adjust our params (w and b) to minimize loss

To find these gradients, we use backpropagation, which involves calculating the derivative of the loss function with respect to each parameter

Note on terms

In our reading, we use θ to refer to a vector of arbitrary number of params (cardinality is p).

So in this example we can assume

$\theta =< w, b >$ and $p = 2$ (there's just two params)

Gradient wrt w

$$\frac{\partial L}{\partial w} = -(y - \hat{y}) \cdot x$$

tells how a small change in w affects loss.

If $\frac{\partial L}{\partial w}$ is positive, increasing w will increase the loss, so we should decrease w to reduce the loss.

How did we get gradient wrt w?

- plugin $\hat{y}$ into L
- $L = \frac{1}{2}(y - (wx + b))^2$
- or, just apply chain rule
- $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w}$

How did we get gradient wrt w?

- $\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}(\frac{1}{2}(y - \hat{y})^2) = -(y - \hat{y}) = \hat{y} - y$
- $\frac{\partial \hat{y}}{\partial w} = \frac{\partial}{\partial w}(wx + b) = x$

Chain rule helps out

- Note that we can also just multiply everything out and compute partial derivatives without the chain rule
- Chain rule is just convenient for composite function. Here $\hat{y}$ is a function of w, x, and b.
- Note that if we have an activation function, it's even more helpful…

Example with actual values, 1/n

Suppose

- Input value $x = 2$
- Actual output value $y = 5$
- Weight $w = 1$
- Bias $b = 1$

Want gradient of L wrt w still

Example with actual values, 2/n

$(x = 2, y = 5, w = 1, b = 1)$

- $\hat{y} = wx + b = 1 * 2 + 1 = 3$
- $L = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(5 - 3)^2 = 2$
- $\frac{\partial L}{\partial w} = -(y - \hat{y})x = -(5 - 3) * 2 = -4$

Example with actual values, 3/n

- Small increase in w will decrease loss, so adjust w upwards
- We make our update based on learning rate, η

Gradient wrt b

$\frac{\partial L}{\partial b} = -(y - \hat{y})$

tells us how a small change in b affects the loss. Similarly, if $\frac{\partial L}{\partial b}$ is positive, increasing b will increase the loss, so we should decrease b to reduce the loss.

How did we get gradient wrt b

- $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b}$
- $\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y})$
- $\frac{\partial \hat{y}}{\partial b} = \frac{\partial}{\partial \hat{y}}(wx + b) = 1$
- so, $\frac{\partial L}{\partial b} = -(y - \hat{y})$

Updating params

We update w and b using a learning rate η (a kind of step size, how far we adjust things based on the direction of our gradients):

$$w = w - \eta \frac{\partial L}{\partial w}$$

$$b = b - \eta \frac{\partial L}{\partial b}$$

Iteration

This process is repeated for many iterations (or epochs) over the training data, gradually reducing the loss and making the predictions $\hat{y}$ closer to the actual outputs y

Other resources

- We ignored activation function here
- Exercise for the reader! What if we add ReLu or logistic activation function (hint: now we have $z = wx + b$ and $\hat{y} = a(z)$, so we have to do a 3-piece chain rule)
- Worth reviewing longer backprop materials[1]

Relevance back to training data influence

- Gradient based methods rely on the idea that: training data only influences the final model via the gradients produced
- We should be able to keep track of these gradients to understand how a training instance affected a test instance

For a data valuator

- We can assume gradients are being calculated throughout training
- pytorch: `requires_grad=True`
- another example: https://fluxml.ai/Flux.jl/stable/models/basics/