Scientific Computing

Monday, April 20

Announcements

* Homework 6 due next Monday, April 27

* Final Exam: Monday, 5/4,
  1pm-3pm
  Johnston Hall 417

Office Hours:
Mon, 9:30-10:30
Fri, 2:00-3:00

Cudahy 307

Topic 17 - Backpropagation (Training NNs)

Big picture:

* Start with a NN with random weights and biases
* Feed all of the known inputs (training data) through in one batch and get all their outputs
* Compute the loss between the expected outputs (y) and the actual outputs ($\hat{y}$)

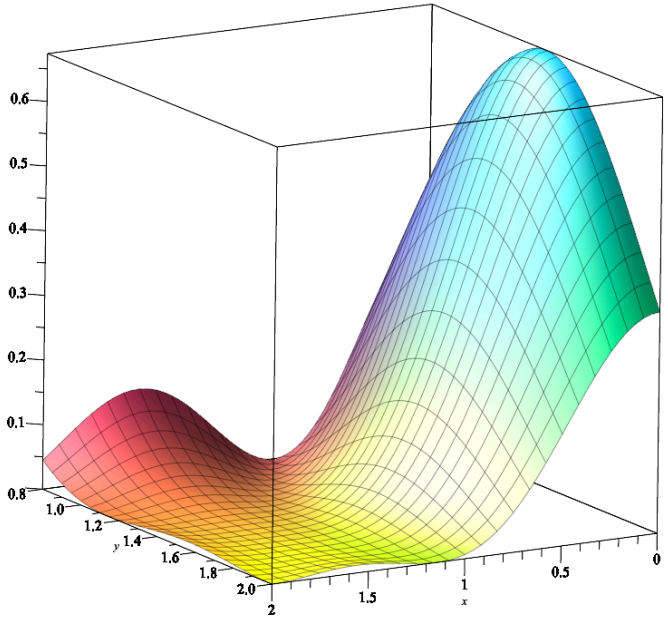
* Use **CALCULUS** to figure out how to tweak the weights + biases a little to make the loss go down a little.

* Repeat until your NN is good enough.

CALCULUS

Let's recall the idea of Gradient Descent.

Suppose you have a function $f(x,y)=z$.



The vector $\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \nabla f$ tells you the direction of steepest ascent from any point (x,y).

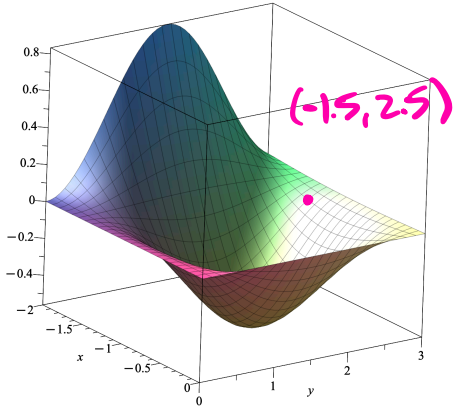
The negative version $-\nabla f = \begin{bmatrix} -\frac{\partial f}{\partial x} \\ -\frac{\partial f}{\partial y} \end{bmatrix}$ is the direction of steepest descent.

CALCULUS

The negative version $-\nabla f = \begin{bmatrix} -\frac{\partial f}{\partial x} \\ -\frac{\partial f}{\partial y} \end{bmatrix}$ is the direction of <u>steepest descent</u>.

This means if you're only going to walk 1 step in the mountains, then one step in <u>that direction</u> will get you as low as possible among all possible one steps.

<u>Ex:</u> $f(x,y) = x\cos(x)(\sin(y))^2$.



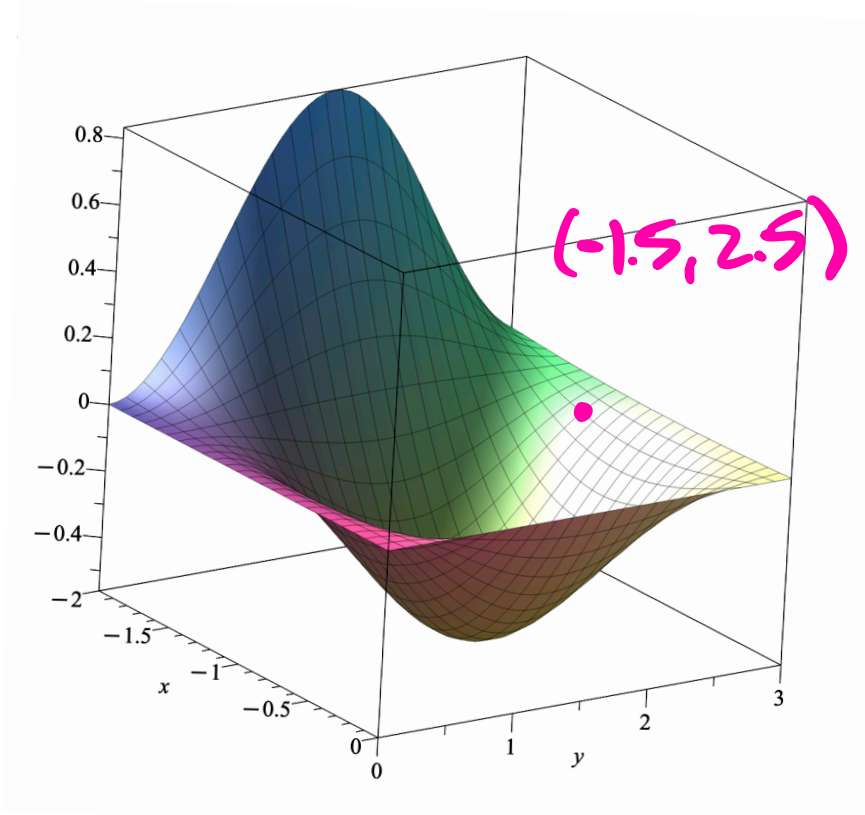
$$-\nabla f = \begin{bmatrix} -\frac{\partial}{\partial x}\left(x\cos(x)\cdot(\sin(y))^2\right) \\ -\frac{\partial}{\partial y}\left(x\cos(x)(\sin(y))^2\right) \end{bmatrix}$$

CALCULUS

Ex: $f(x,y) = x\cos(x)(\sin(y))^2$.

$$\frac{\partial f}{\partial x} = (\sin(y))^2 \cdot (\cos(x) - x\sin(x))$$

$$\frac{\partial f}{\partial y} = x\cos(x) \cdot 2\sin(y)\cos(y)$$



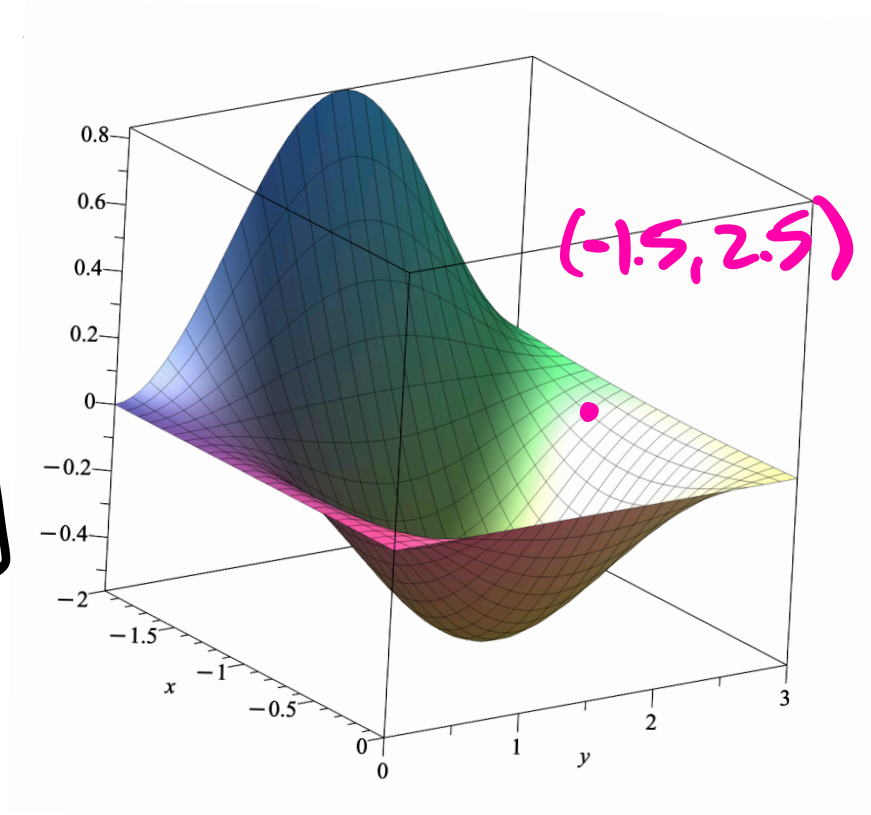
$$(-\nabla f)(-1.5, 2.5) = -\begin{bmatrix} \sin(2.5)^2 \cdot (\cos(-1.5) + 1.5\sin(1.5)) \\ (-1.5)\cos(-1.5) \cdot 2 \cdot \sin(2.5)\cos(2.5) \end{bmatrix}$$

$$= \begin{bmatrix} 0.5105\ldots \\ -0.1017\ldots \end{bmatrix}$$

CALCULUS

Ex: $f(x,y) = x\cos(x)(\sin(y))^2$

$$(-\nabla f)(-1.5, 2.5) = \begin{bmatrix} 0.5105\ldots \\ -0.1017\ldots \end{bmatrix}$$



This doesn't mean you move +0.51 on the x-axis an −0.10 on the y-axis. That would be a huge step

Instead you move a small step, maybe $\frac{1}{100}$ of that.

$$\begin{bmatrix} -1.5 \\ 2.5 \end{bmatrix} + \begin{bmatrix} 0.0051\ldots \\ -0.0010\ldots \end{bmatrix} = \begin{bmatrix} -1.5051 \\ 2.4990 \end{bmatrix}$$

−1.4995...

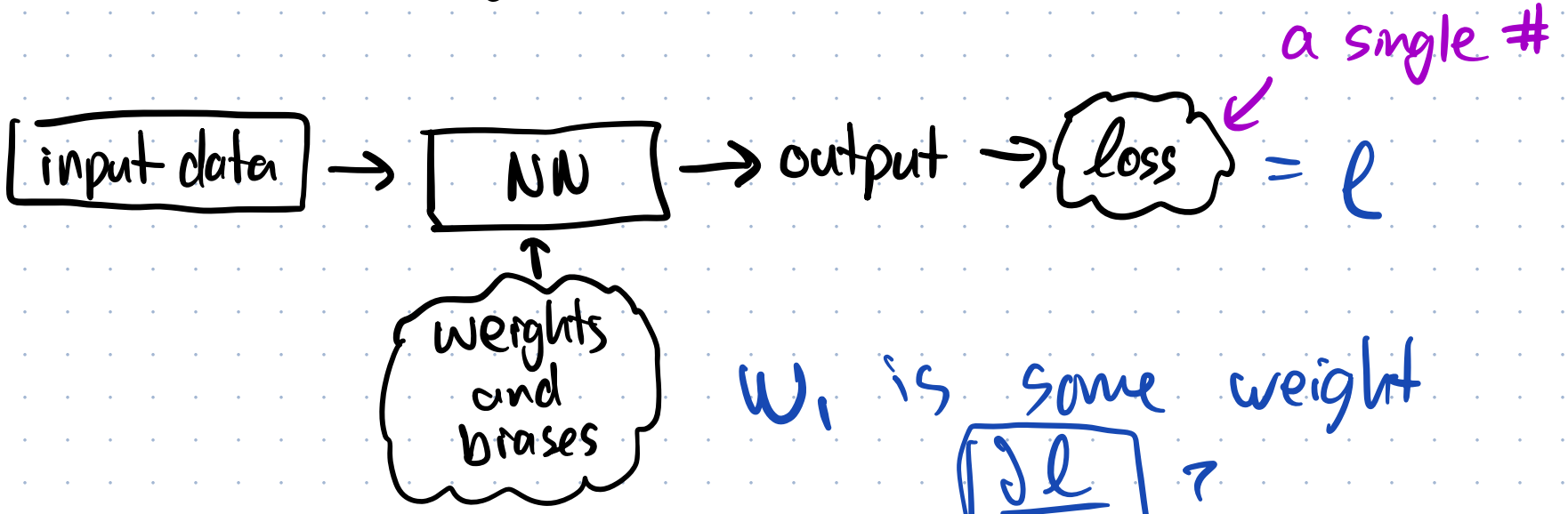
and repeat.

$-\nabla f$ is the same! Just new points.

* Gradient Descent Demo

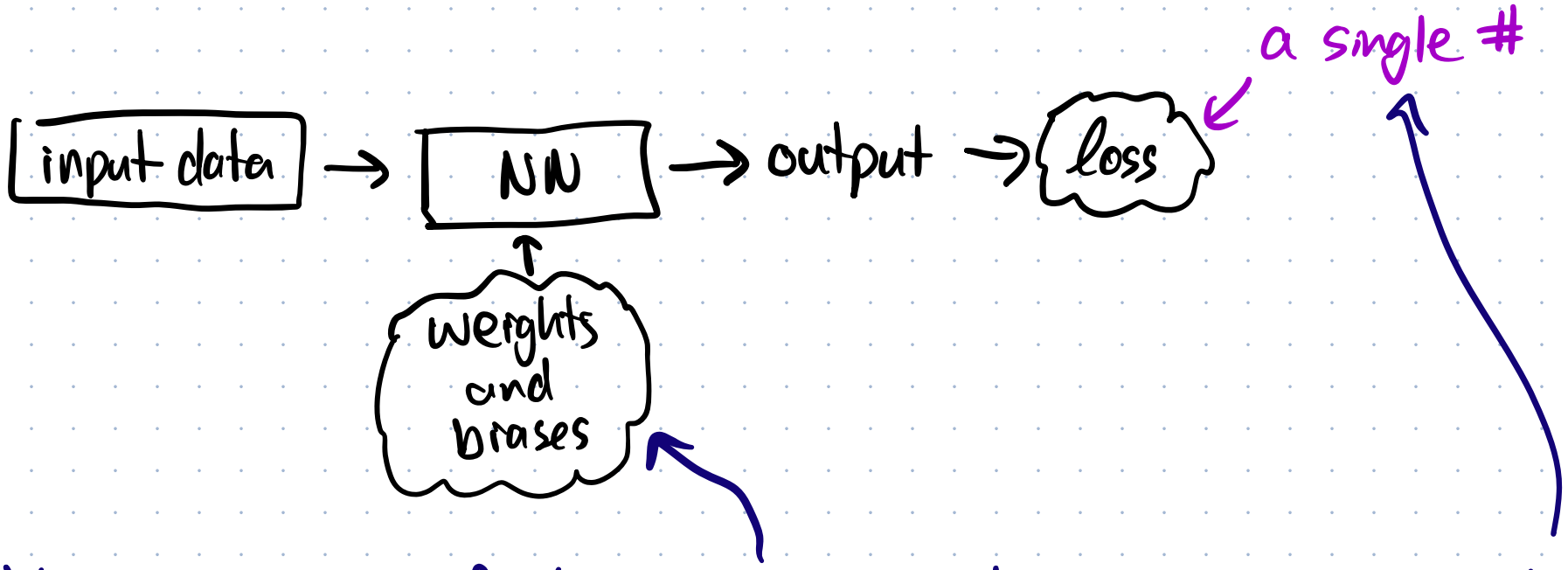
We have been thinking about Neural Networks as big functions

For now, instead of thinking of the input data as the variables, we'll think of all the training data as a fixed constant and the <u>weights and biases</u> as the variables.



w_1 is some weight

$\frac{\partial \ell}{\partial w_1}$?



How can we find the inputs that minimize the output?

Gradient Descent!

Suppose the neural network has weights $w_1, w_2, \ldots, w_m$ and biases $b_1, b_2, \ldots, b_n$. Let ℓ be the mean loss over the whole batched input.

What is $-\nabla \underbrace{NN(w_1, w_2, \ldots w_m, b_1, b_2, \ldots, b_n)}$?

loss of output given these weights + biases + fixed input

$$\ell = NN(w_1, w_2, \ldots, w_m, b_1, b_2, \ldots b_n).$$

How will we compute

$$\nabla NN = \begin{bmatrix} \frac{\partial \ell}{\partial w_1} \\ \vdots \\ \frac{\partial \ell}{\partial w_m} \\ \frac{\partial \ell}{\partial b_1} \\ \vdots \\ \frac{\partial \ell}{\partial b_n} \end{bmatrix} ?$$

CALCULUS

specifically
the CHAIN RULE

the CHAIN RULE

$f'(3) = 5$

$\frac{df}{dx}(3) = 5$ means if you change

$x = 3$ to $x = 3 + [\text{a little more}]$

then $f(x)$ goes from

$f(3)$ to $f(3) + 5 \cdot [\text{a little more}]$.

the CHAIN RULE

Suppose $h(x) = g(f(x))$ (a composition).

Chain rule: $h'(x) = g'(f(x)) \cdot f'(x)$

or $\frac{dh}{dx} = \frac{df}{dx} \cdot \frac{dh}{df}$

how much h changes when x changes a little

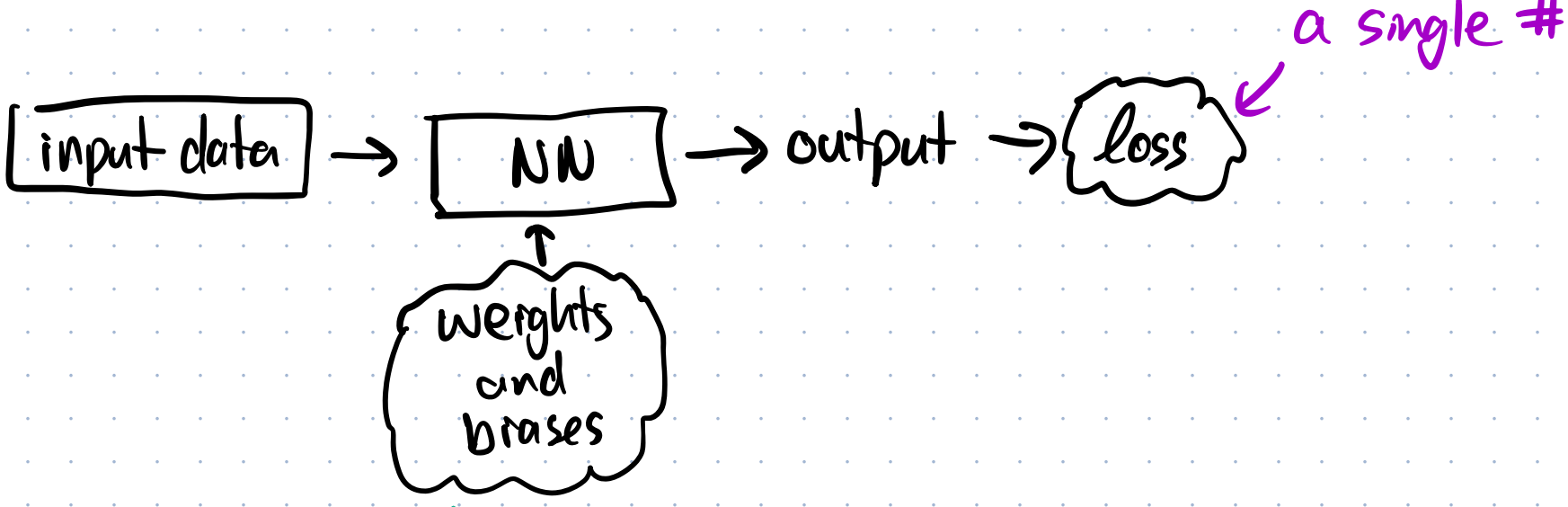
how much f changes when x changes a little

how much h changes when f changes a little

the CHAIN RULE

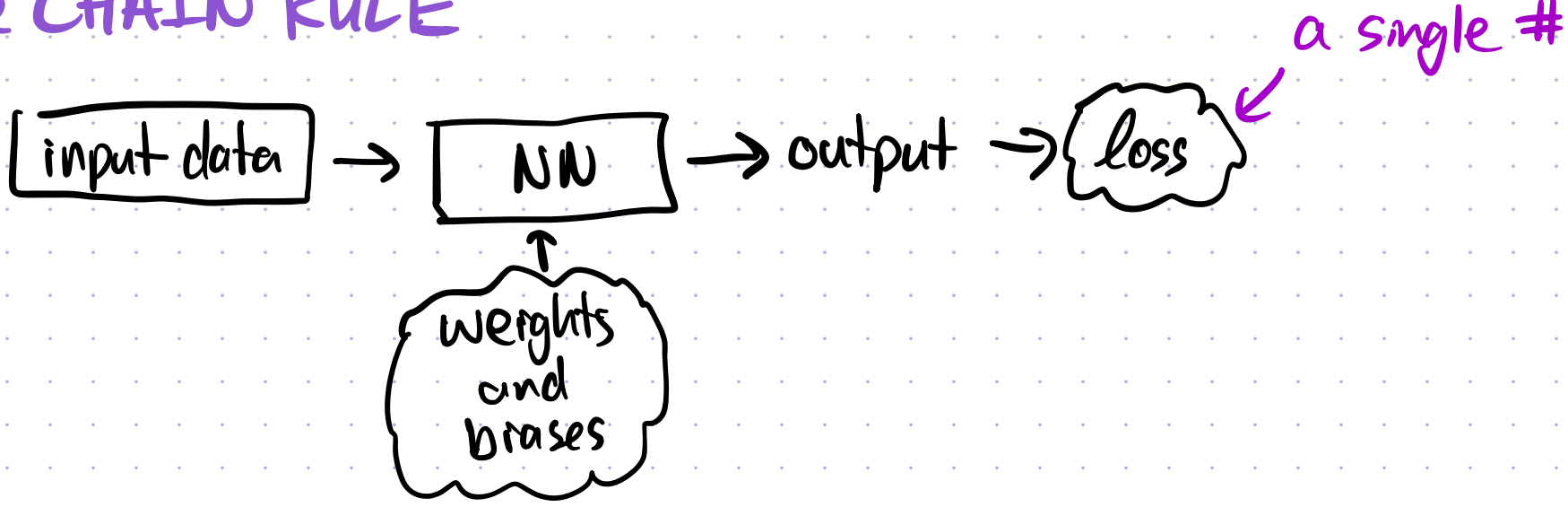
Neural Networks are just really big multivariate compositions of simple functions

(addition, multiplication, activation functions, loss functions)



We can apply the chain rule a lot to see how changing any weight or bias individually affects the loss

the CHAIN RULE

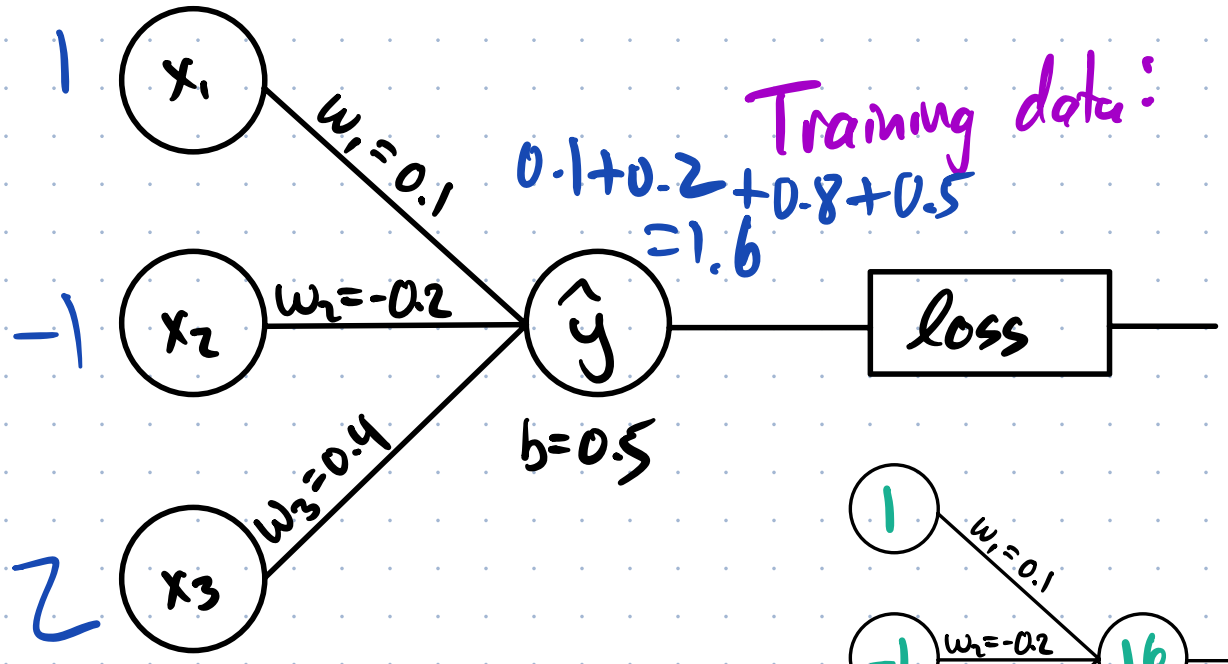


We can apply the chain rule a lot to see how changing any weight or bias individually affects the loss

$$\nabla NN = \begin{bmatrix} \frac{\partial \ell}{\partial w_1} \\ \vdots \\ \frac{\partial \ell}{\partial w_m} \\ \frac{\partial \ell}{\partial b_1} \\ \vdots \\ \frac{\partial \ell}{\partial b_n} \end{bmatrix}.$$

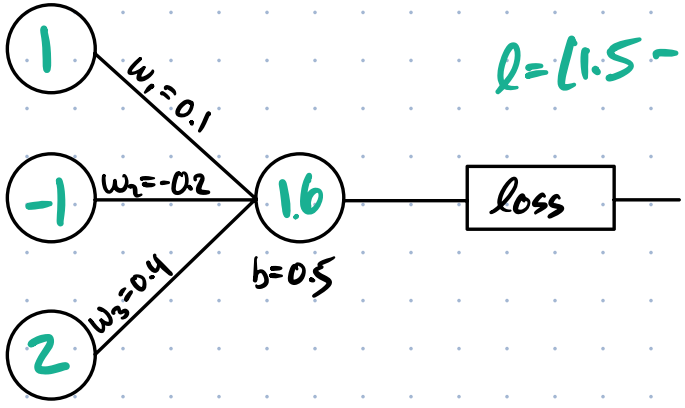
That means we'll know the gradient and we can do gradient descent.

First example: No AF, no hidden layers, 3 input, 1 output, only one point of training data



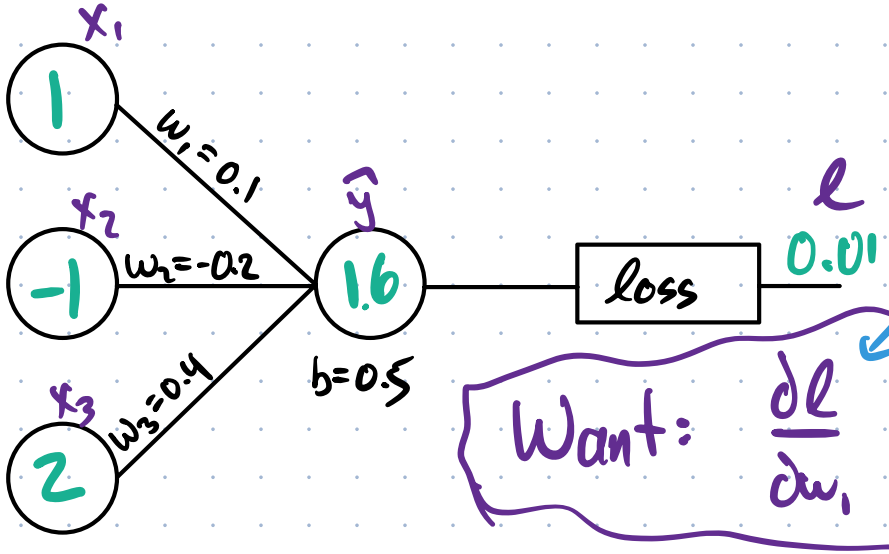
Training data: $(1, -1, 2) \rightarrow 1.5$

$0.1+0.2+0.8+0.5 = 1.6$



$\ell = (1.5-1.6)^2 = 0.01$

How do we adjust w_1, w_2, w_3, b to make ℓ go down for this training data?



Training data: $(1, -1, 2) \to 1.5$

When I change w_1 a little, how much does ℓ change?

Want: $\frac{\partial \ell}{\partial w_1}, \frac{\partial \ell}{\partial w_2}, \frac{\partial \ell}{\partial w_3}, \frac{\partial \ell}{\partial b}$

$\ell = (\hat{y} - 1.5)^2$.

So, $\frac{\partial \ell}{\partial \hat{y}} = 2(\hat{y} - 1.5)$.

0.2 $= 2 \cdot (0.1) = 0.2$

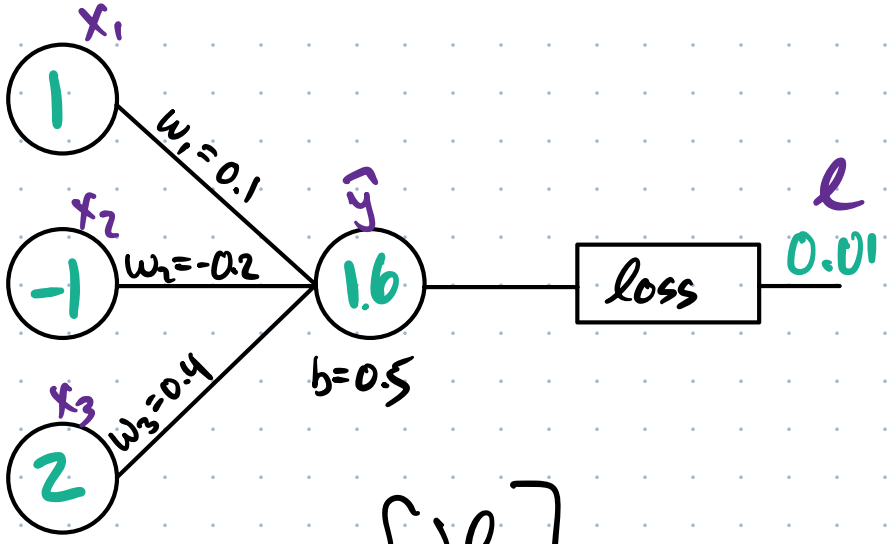
$\hat{y} = 1 \cdot w_1 - 1 \cdot w_2 + 2 \cdot w_3 + b$

So, $\frac{\partial \hat{y}}{\partial w_1} = 1$, $\frac{\partial \hat{y}}{\partial w_2} = -1$, $\frac{\partial \hat{y}}{\partial w_3} = 2$, $\frac{\partial \hat{y}}{\partial b} = 1$

$\frac{\partial \ell}{\partial w_1} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = 0.2 \cdot 1 = 0.2$

If $\hat{y}$ changes by δ, this causes ℓ to change by $0.2 \cdot \delta$

$\frac{\partial \ell}{\partial w_2} = -0.2$ $\frac{\partial \ell}{\partial w_3} = 0.4$ $\frac{\partial \ell}{\partial b} = 0.2$



Training data: $(1, -1, 2) \to 1.5$

$$-\nabla NN = -\begin{bmatrix} \frac{\partial \ell}{\partial w_1} \\ \frac{\partial \ell}{\partial w_2} \\ \frac{\partial \ell}{\partial w_3} \\ \frac{\partial \ell}{\partial b} \end{bmatrix} = \begin{bmatrix} -0.2 \\ +0.2 \\ -0.4 \\ -0.2 \end{bmatrix}$$

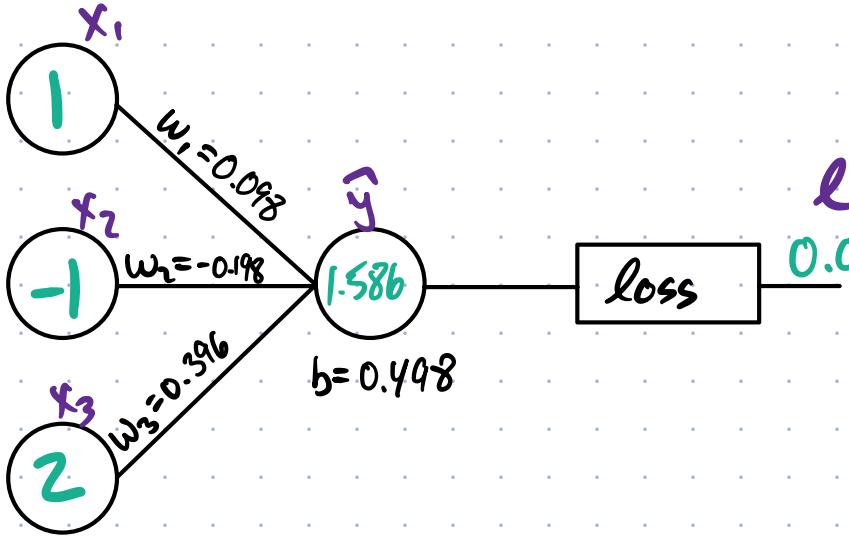
Now adjust each weight a small amount in this direction. Let's do $\frac{1}{100}$.

w_1: $0.1 \to -0.098$

w_2: $-0.2 \to -0.198$

w_3: $0.4 \to 0.396$

b: $0.5 \to 0.498$



Training data: $(1, -1, 2) \to 1.5$

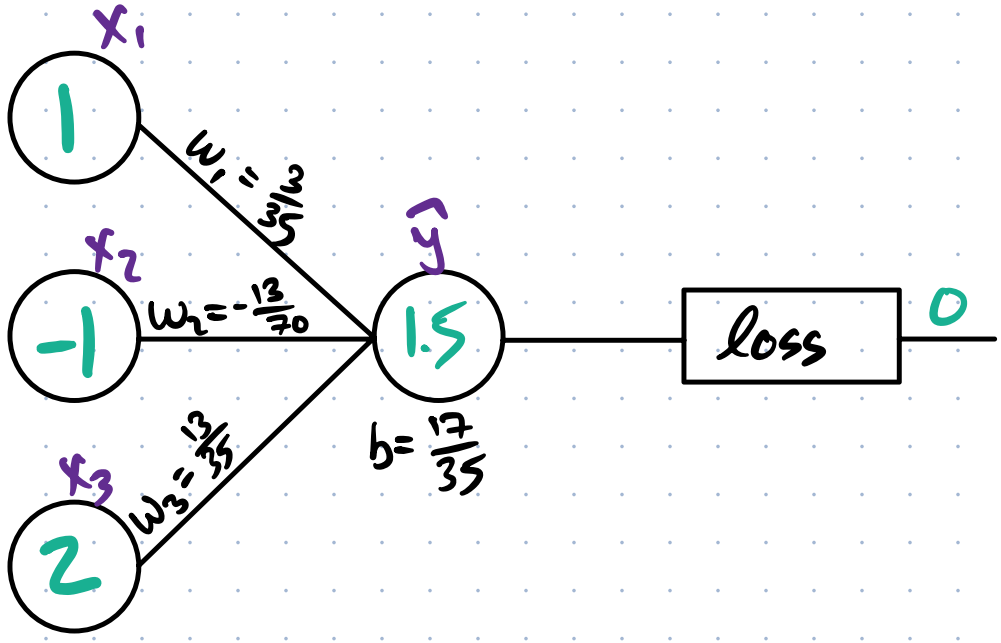
(prev. 0.01)

Now repeat. Compute $-\nabla NN$ with these altered weights, adjust by $\frac{1}{100}$ of it, and so on.

* Python demo.

Training data: $(1, -1, 2) \to 1.5$

After 100 or so repetitions:



Easy because only one piece of training data!

Example 2: Backpropagation on a Batch of data.

Training Data:

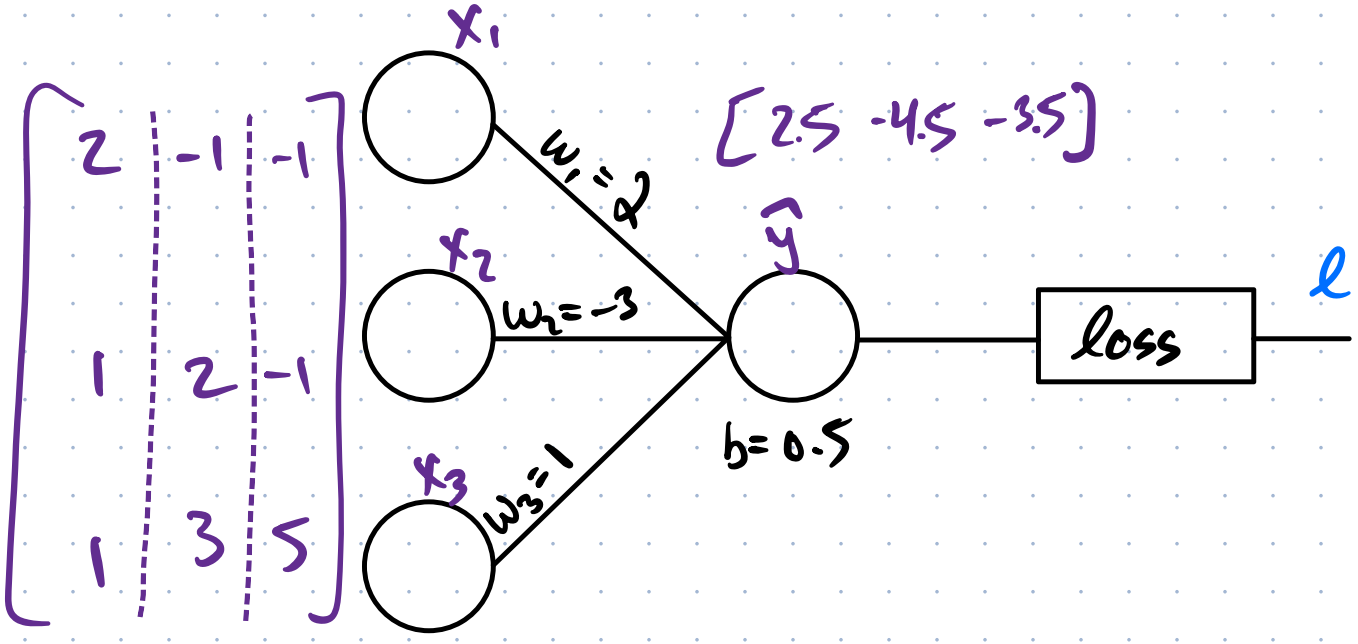
$(2, 1, 1) \to 3$

$(-1, 2, 3) \to -4$

$(-1, -1, -5) \to -5$

Still no AF

no hidden layers



$$\ell = \frac{1}{3}\left((2.5-3)^2 + (-4.5--4)^2 + (-3.5--5)^2\right)$$

$$= 0.916\ldots$$

(making the weights whole #s for now)