Recitation 2

Computing Loss and Derivatives

Today:

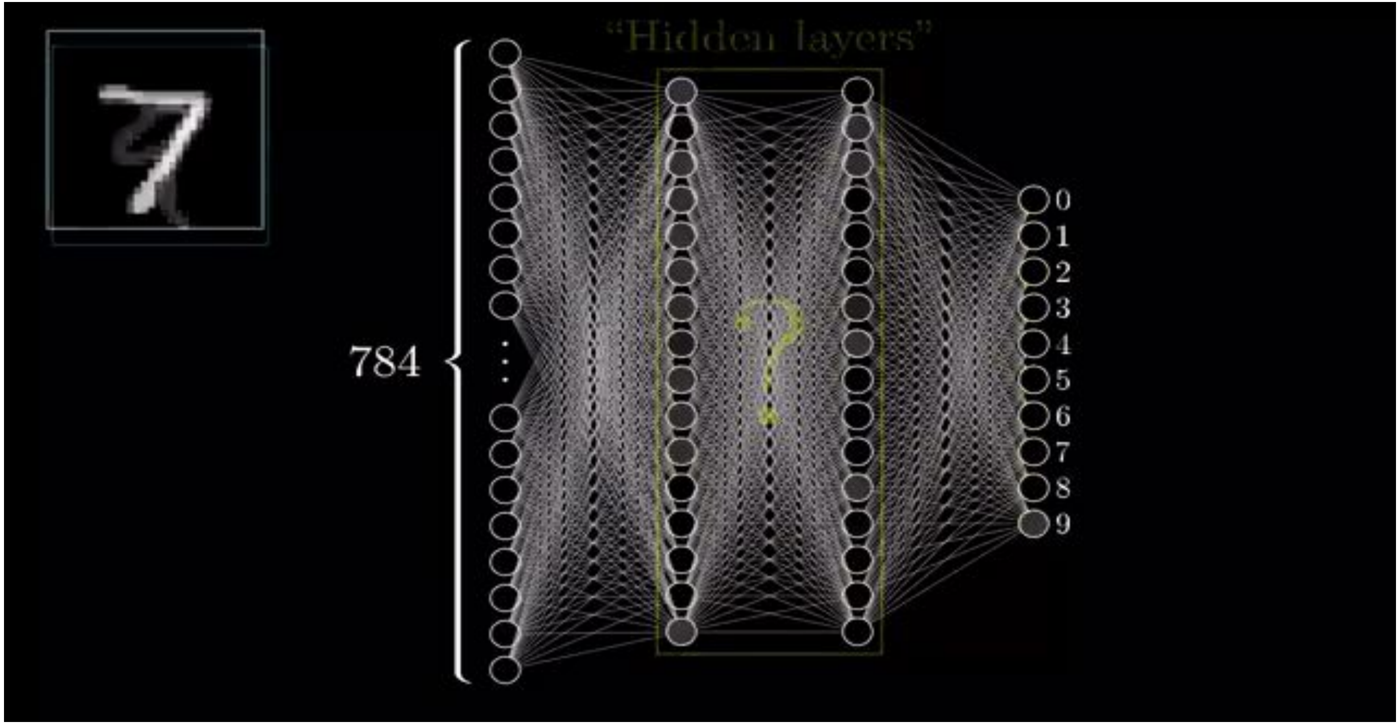
Goal:Conceptual understanding of the math behind backprop

- Neural network forward pass
- Concept of loss
- Motivation for backpropagation
- Why do we calculate gradients?
- How does PyTorch do this under the hood?

Step1: Forward Propagation

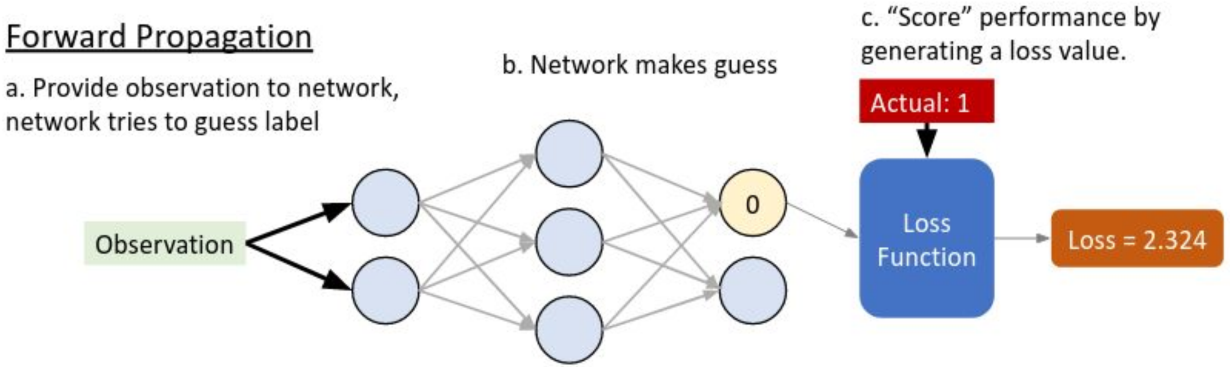
- Composed of 2 elements
  - Affine combination
  - Activation function
- The affine combination is the result of the product of weights with the corresponding inputs summed with a bias.
  - $y(\vec{x}) = W\vec{x} + \vec{b}$
- The activation function introduces non-linearity thus allowing us to learn complex decision boundaries.

What does forward pass look like on a high level?



What is Loss?

- Loss function is a measure of how good your prediction model does in terms of being able to predict the expected outcome(or value).
- We are converting the learning problem into an optimization problem define a loss function and then optimize the algorithm to minimize the loss function.
- The loss function will be the starting point of our back propagation

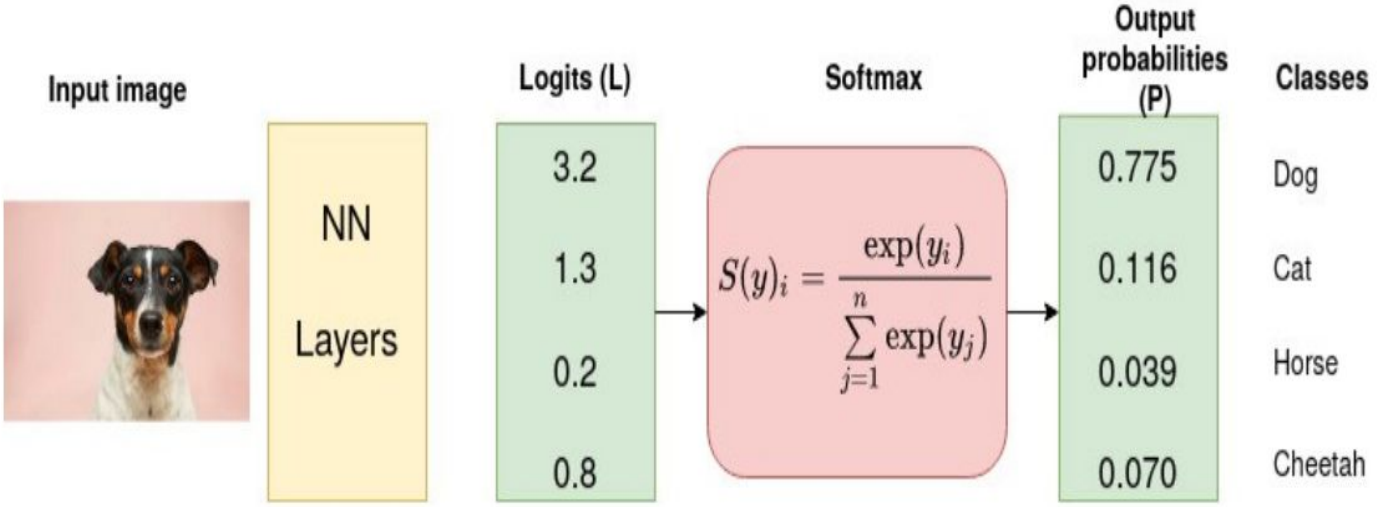


Popular choices of loss functions

1. Regression Loss Functions
    1. Mean Squared Error Loss
    2. Mean Squared Logarithmic Error Loss
    3. Mean Absolute Error Loss
2. Binary Classification Loss Functions
    1. Binary Cross-Entropy
    2. Hinge Loss
    3. Squared Hinge Loss
3. Multi-Class Classification Loss Functions
    1. Multi-Class Cross-Entropy Loss
    2. Sparse Multiclass Cross-Entropy Loss
    3. Kullback Leibler Divergence Loss

Example: Cross-Entropy Loss

- Task: classifying dogs and cats



Input image source: Photo by Victor Grabarczyk on Unsplash . Diagram by author.

- The desired output is [1,0,0,0] for the class dog but the model outputs [0.775, 0.116, 0.039, 0.070]

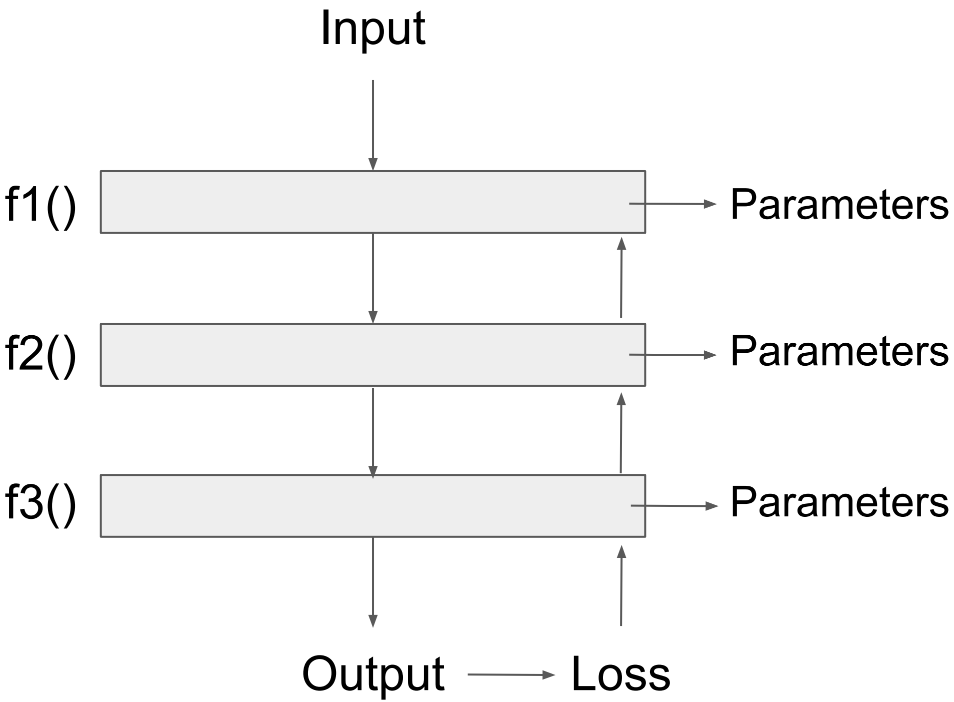
The categorical cross-entropy is computed as follows

$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -\left[1 \log_2(0.775) + 0 \log_2(0.126) + 0 \log_2(0.039) + 0 \log_2(0.070)\right] \\
&= -\log_2(0.775) \\
&= 0.3677
\end{aligned}
$$

Backpropagation through Layers

Backpropagate Loss

1. Forward
2. Calculate Loss
3. Pass Gradient with respect to output
4. Update Parameters
5. Continue



$$\text{Output} = f3(\ f2(\ f1(\ x\ )\ )\ )$$

Single Layer Backward: Linear

$$\frac{d\mathbf{L}}{d\mathbf{x}}$$

$$\frac{d\mathbf{L}}{d\mathbf{x}} = \frac{d\mathbf{L}}{d\mathbf{y}}\frac{d\mathbf{y}}{d\mathbf{x}}$$

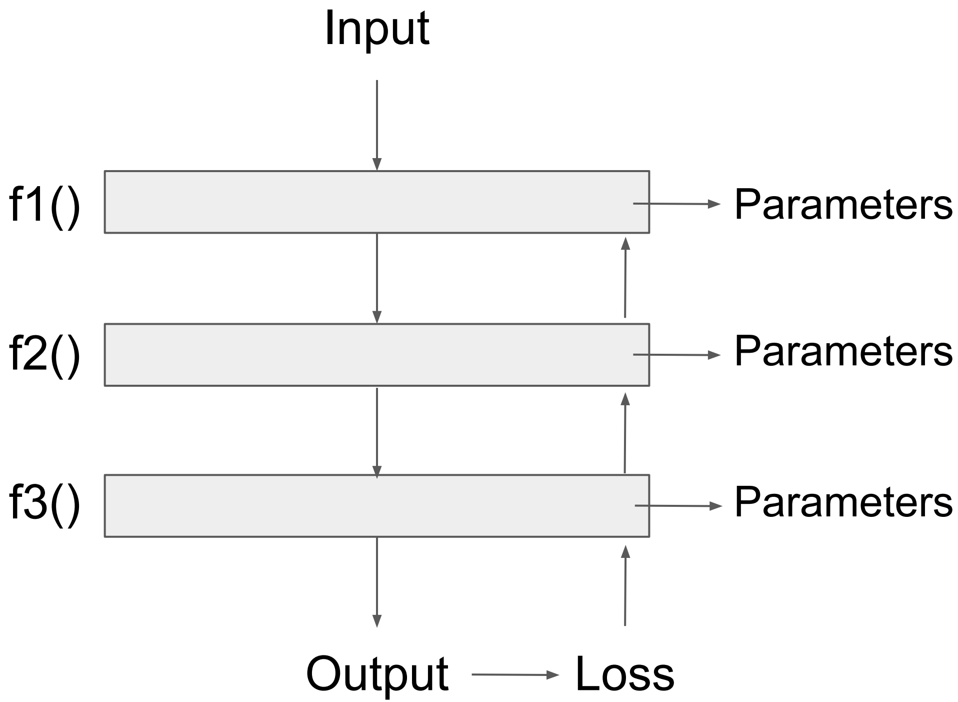
$$\frac{d\mathbf{L}}{d\mathbf{W}} = \frac{d\mathbf{L}}{d\mathbf{y}}\frac{d\mathbf{y}}{d\mathbf{W}}$$ Update W

$$\frac{d\mathbf{L}}{db} = \frac{d\mathbf{L}}{d\mathbf{y}}\frac{d\mathbf{y}}{db}$$ Update b

$$\frac{d\mathbf{L}}{d\mathbf{y}}$$

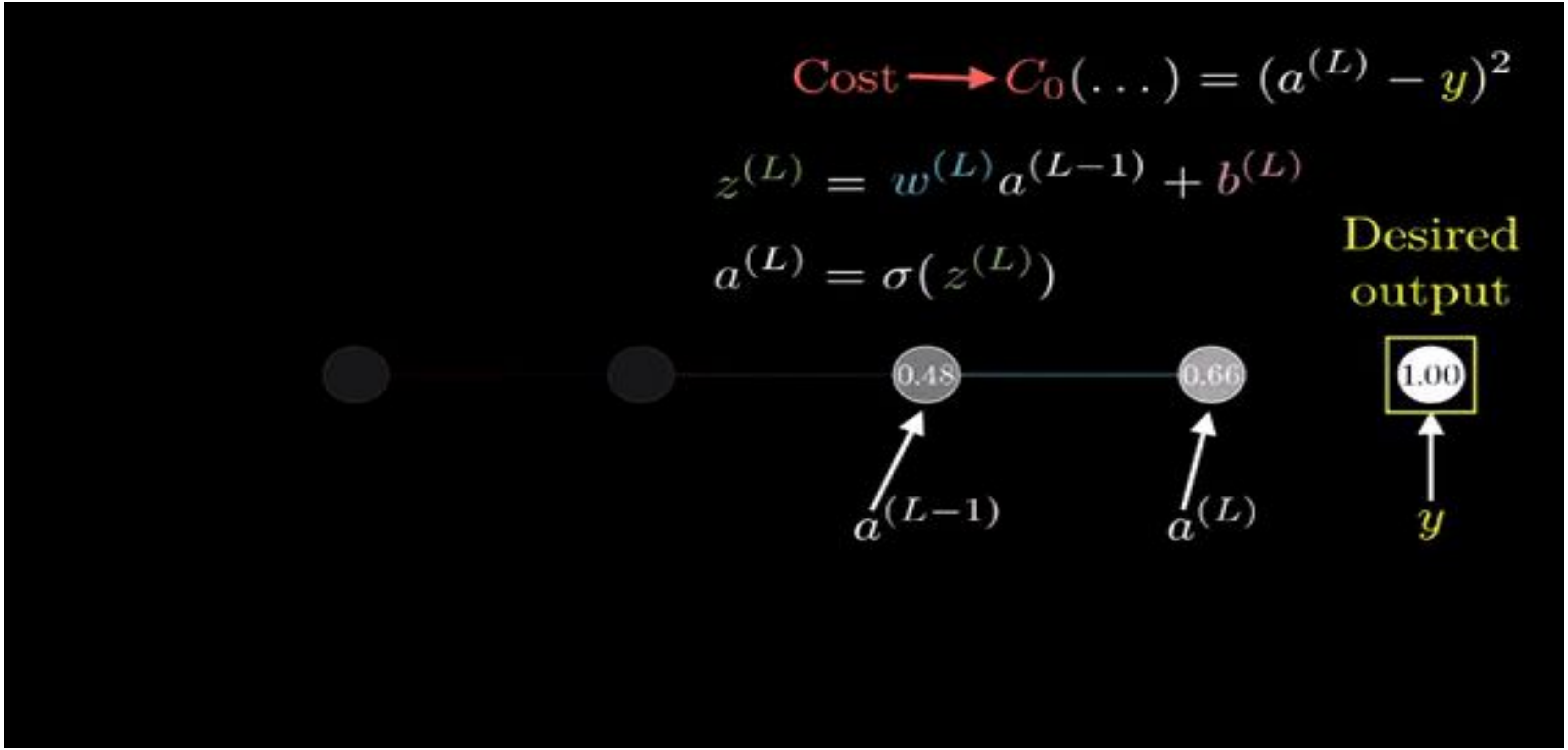
Remember...

- Layers are nested functions
  - Use Chain Rule
- Update parameters as we go
- Check Gradient Shapes (transpose?)
- Gradients follow influence
  - What if something affects more than 1 output?



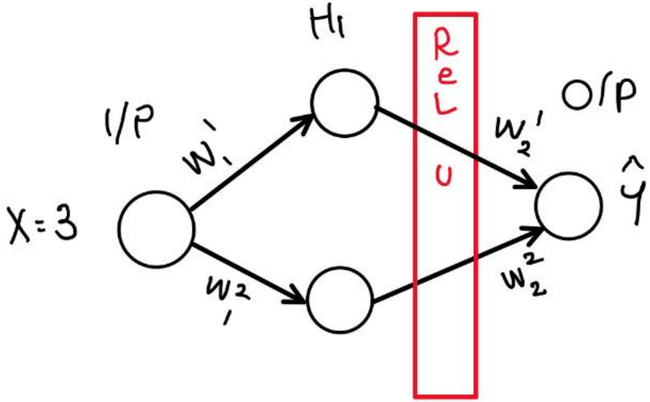
Output = f3(f2(f1(x)))

Lets visualize the above mentioned process...



MLP Example!

Lets solve the following MLP:



$$W_1 = \begin{bmatrix} w_1^1 \\ w_1^2 \end{bmatrix} = \begin{bmatrix} 4 \\ -2 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} w_2^1 \\ w_2^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$Y = 10$$

Forward Pass

Let outputs of H_1 be H_1^1 and H_1^2

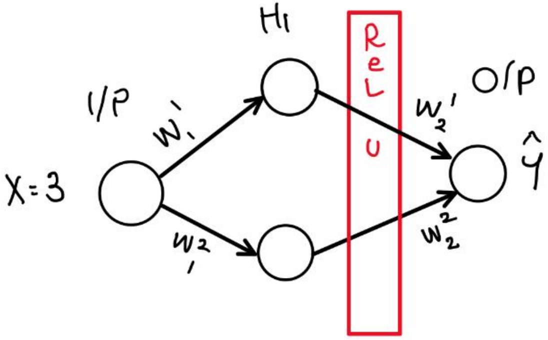
$H_1^1 = W_1^1 X = 4 \times 3 = 12$

$H_1^2 = W_1^2 X = -2 \times 3 = -6$

$\therefore$ After activation:

$H_1^1 = \text{ReLU}(H_1^1) = 12$

$H_1^2 = \text{ReLU}(H_1^2) = 0$



Similarly

$\hat{y} = \text{ReLU}(W_2^1 H_1^1 + W_2^2 H_1^2) = \text{ReLU}(1 \times 12 + 0 \times 0) = 12$

Back propagation:

Lets calculate $\partial C_0 / \partial W_2^1$

Notations: Affine combination $= z$

Activated affine combination $= F(z)$

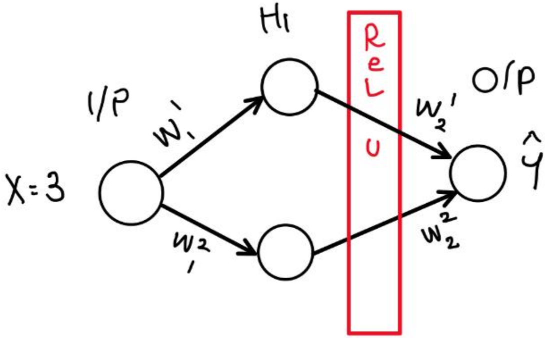
$$\therefore \frac{\partial C_0}{\partial W_2^1} = \frac{\partial C_0}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial F(z)} \times \frac{\partial z}{\partial W_2^1}$$

$$\frac{\partial C_0}{\partial \hat{y}} = 2(\hat{y} - y) = 2(12 - 10) = 4$$

$$\frac{\partial \hat{y}}{\partial F(z)} = \text{ReLU}'(W_2^1 H_1^1) = 1$$

$$\frac{\partial z}{\partial W_2^1} = H_1 = 12$$

$$\therefore \frac{\partial C_0}{\partial W_2^1} = 4 \times 1 \times 12 = 48$$



similarly: $\frac{\partial C_0}{\partial W_2^2} = 0$

Now for the weights b/w i/p layer & layer H_1

$$\frac{\partial C_0}{\partial W_1^1} = \frac{\partial C_0}{\partial H_1^1} \times \frac{\partial H_1^1}{\partial F(z)} \times \frac{\partial z}{\partial W_1^1}$$

Lets calculate

$$\frac{\partial C_0}{\partial H_1^1} = \frac{\partial C_0}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial F(z)} \times \frac{\partial z}{\partial H_1^1}$$

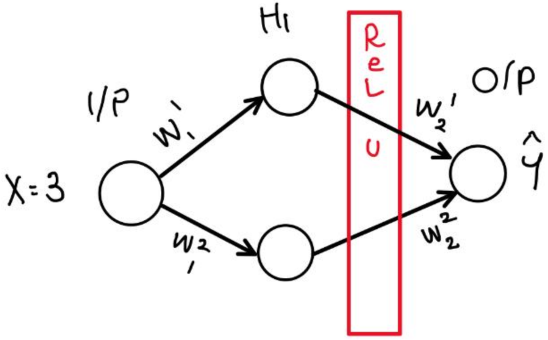
we have the product $\frac{\partial C_0}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial F(z)} = 4 \times 1 = 4$

$$\frac{\partial z}{\partial H_1^1} = \frac{\partial}{\partial H_1^1}(W_2^1 H_1^1) = W_2^1 = 1$$

$\therefore \frac{\partial C_0}{\partial H_1^1} = 4$ & similarly $\frac{\partial C_0}{\partial H_1^2} = 0$

$$\therefore \frac{\partial C_0}{\partial W_1^1} = 4 \times 1 \times 3 = 12$$

& $\frac{\partial C_0}{\partial W_1^2} = 0$



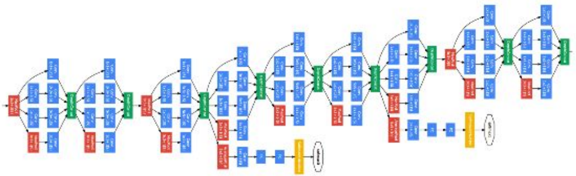
Exercise for self…

- Introduce a bias in the network and compute the derivatives
- Do the same for the input.
- Lecture slides should provide enough support for these exercises...

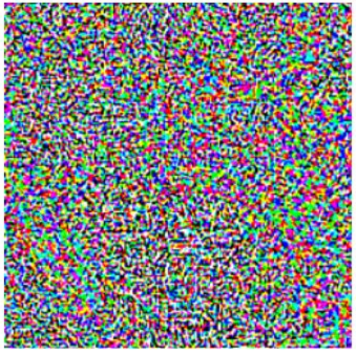
So…
How does Pytorch do it?

WHO WOULD WIN?

A deep convolutional network with 5 million parameters trained on 64 GPUs on 1 million images



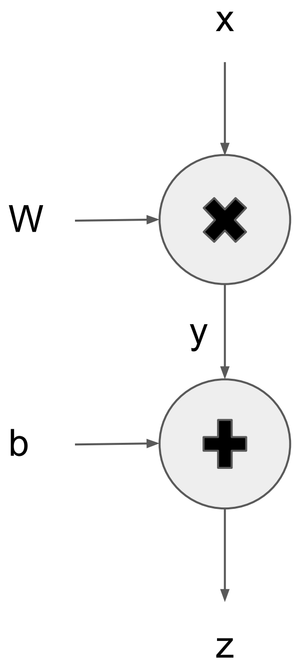
One small gradient boi



How does Pytorch take derivatives and backpropagate?

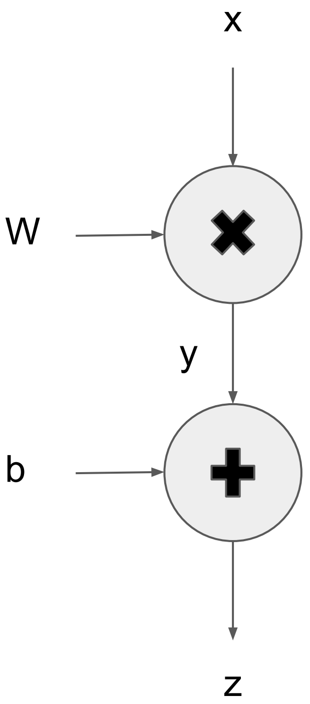
Auto-differentiation:

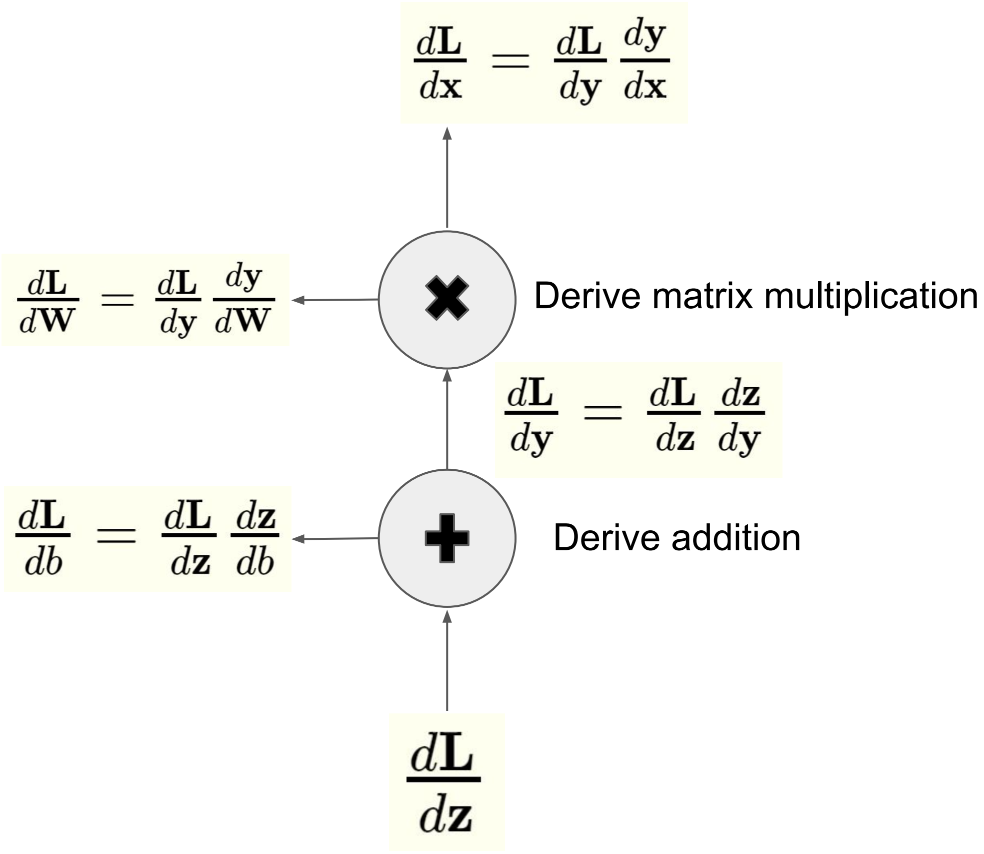
- All of the functions can be rewritten into basic operations
  - True for all computer based calculations
- Sequence of operations instead of a layers
- Each operation is differentiable



$$\mathbf{z} = \mathbf{W}\mathbf{x} + b \longrightarrow \begin{array}{l} \mathbf{y} = \mathbf{W}\mathbf{x} \\ \mathbf{z} = \mathbf{y} + b \end{array}$$

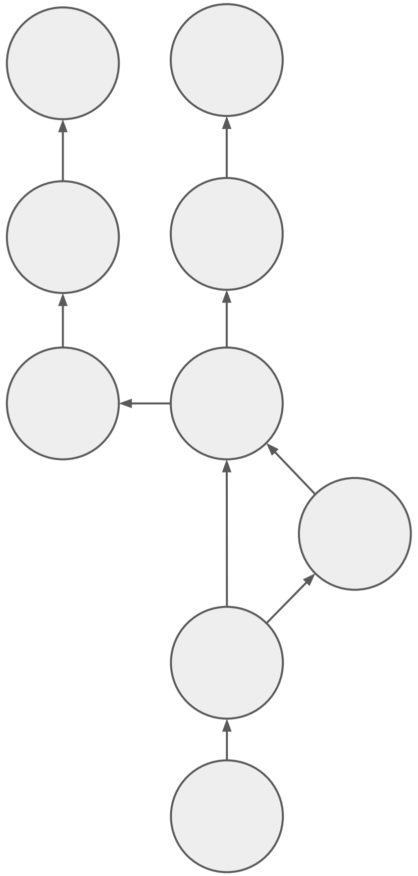
Operational List





Finally… Pytorch

- Pytorch Tensor Class (https://pytorch.org/docs/stable/tensors.html)
  - Keeps track of gradients
  - Points to parent and derivative function
- Autograd - not a list
  - Computational graph (directional, acyclic)
  - Backpropagation = graph traversal
  - loss.backward() = kick off backpropagation
  - optimizer.step() = update parameters
- New this semester: Surprise Bonus
  - Still a list of operations
  - HOW???
  - Coming soon to a bonus HW near you.



Visualizations

Credits(3Blue1Brown YouTube channel)

1. MNIST forward propagation GIF: https://gfycat.com/deadlydeafeningatlanticblackgoby-three-blue-one-brown-machines-learning
2. Backpropagation GIF:https://gfycat.com/adolescentidioticgoldeneye

Enjoy!


TENSORFLOW
PYTORCH
DYNAMIC GRAPH
EAGER EXECUTION
imgflip.com