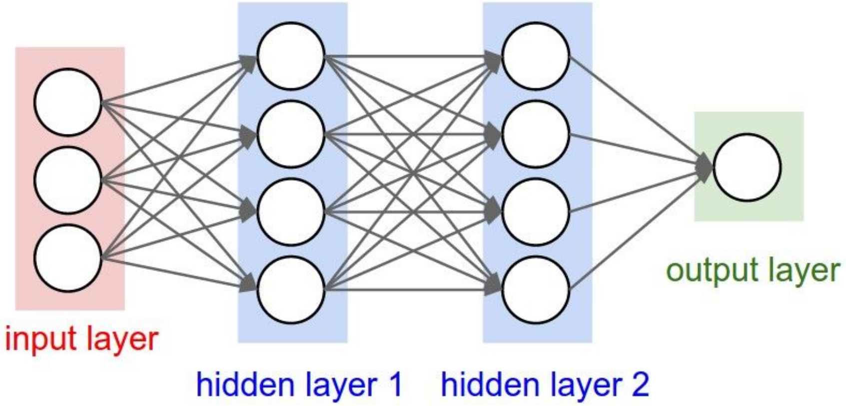
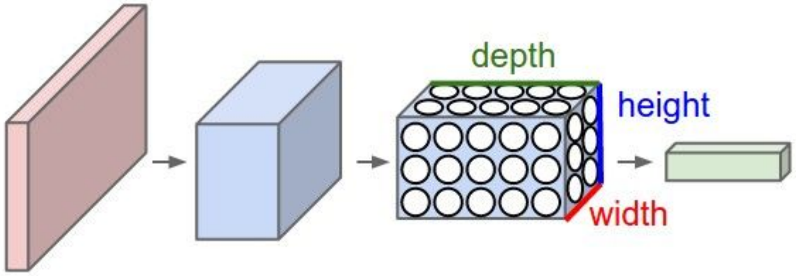
An Overview of Convolutional Neural Networks (CNNs)

Sofia Egan

What is a CNN?

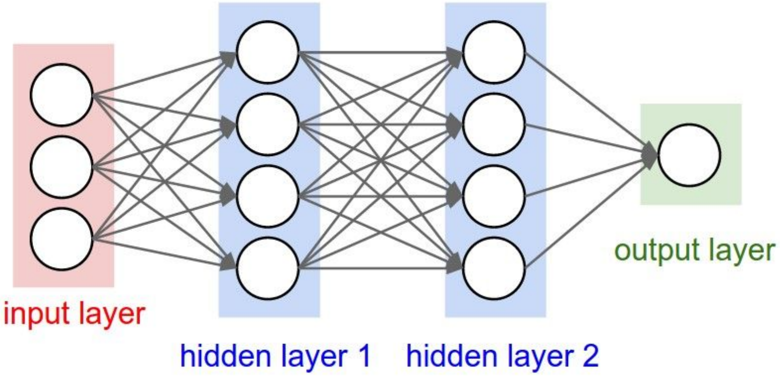
- Generally used for images
- Neural nets generally have fully connected layers
  - Inefficient for images
- CNNs preserve the dimensional properties of images
  - They make each layer of the neural net a tensor

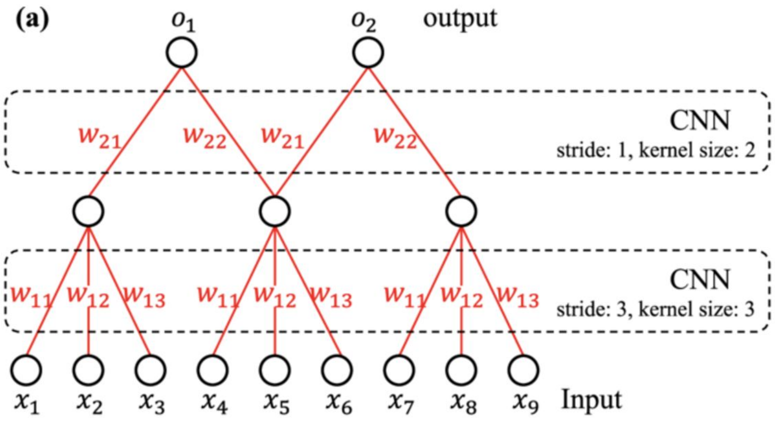




General Structure

- A regular neural net would be inefficient with images because you don't need everything to be fully connected
- We just want every "node" to just take in the information from the pixels around it
- To do this, we'll use convolutions





Why Convolutions?

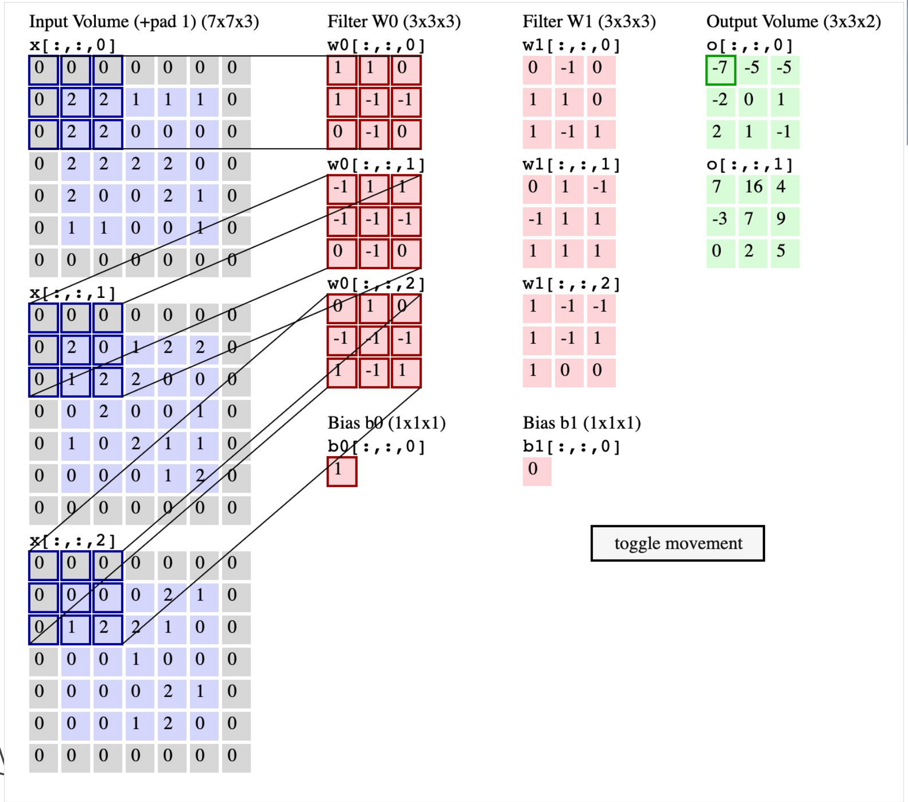
- Beyond reducing the computational power needed, convolutions are useful because they can highlight aspects of an image

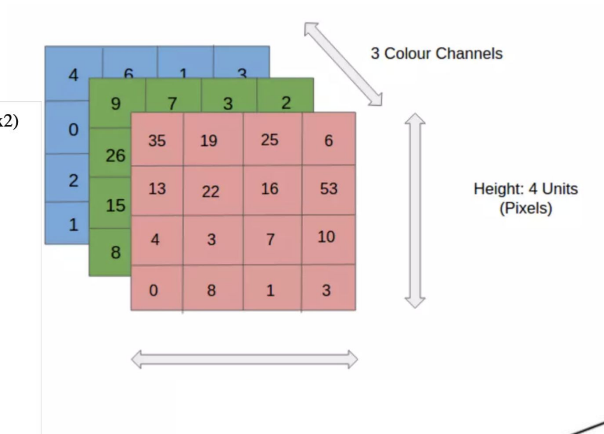
*

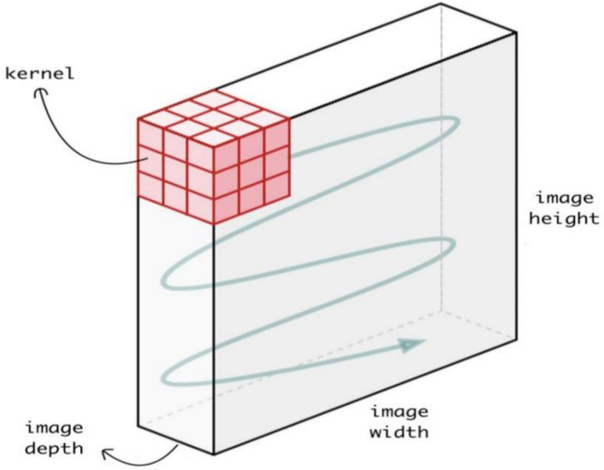
	0	1	0	
	1	-4	1	
	0	1	0	

=

Convolutions



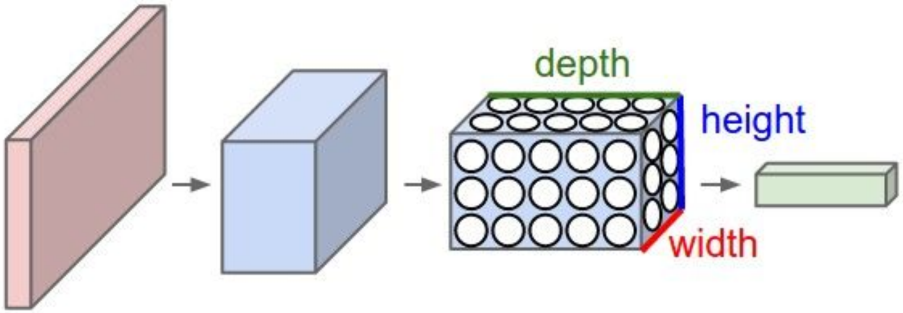




Structure

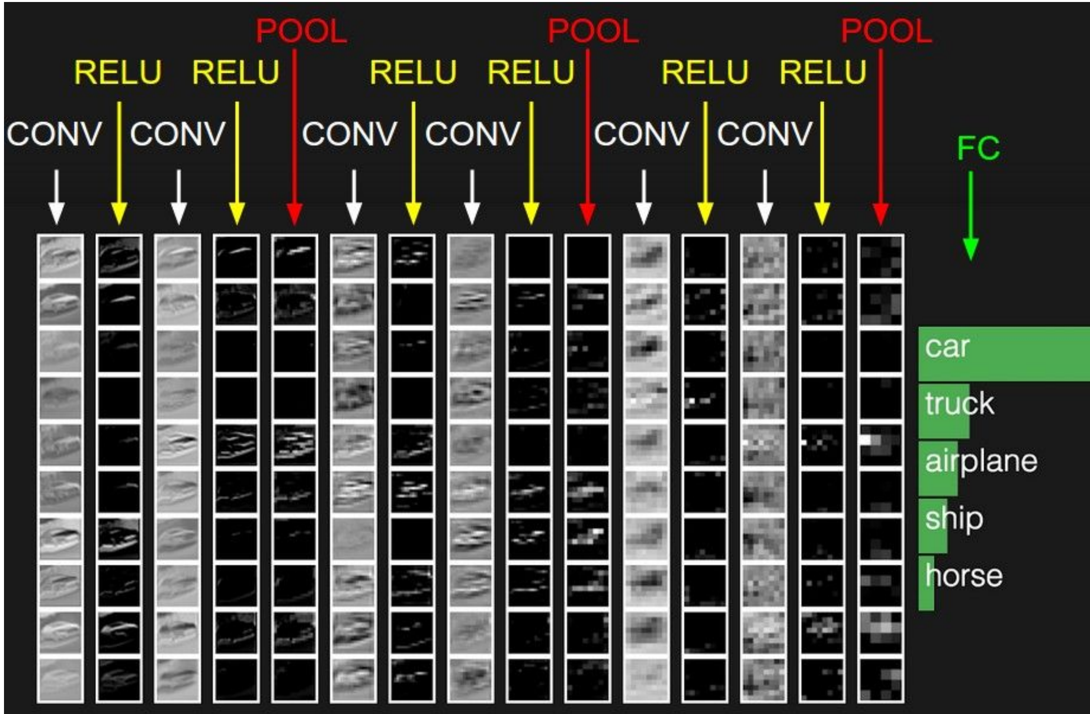
- Each layer is a tensor
- The depth is equal to the number of kernels used on the previous layer
- At the end, there’s one or more fully connected layers and (for classification) a softmax to make the output a probability distribution

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$



Layers

- **Convolution (Conv) Layer**
  - Performs n convolutions with n kernels and outputs a tensor with depth n
- **Pooling Layer**
  - Reduces the dimensions of the image to prevent overfitting and reduce computation

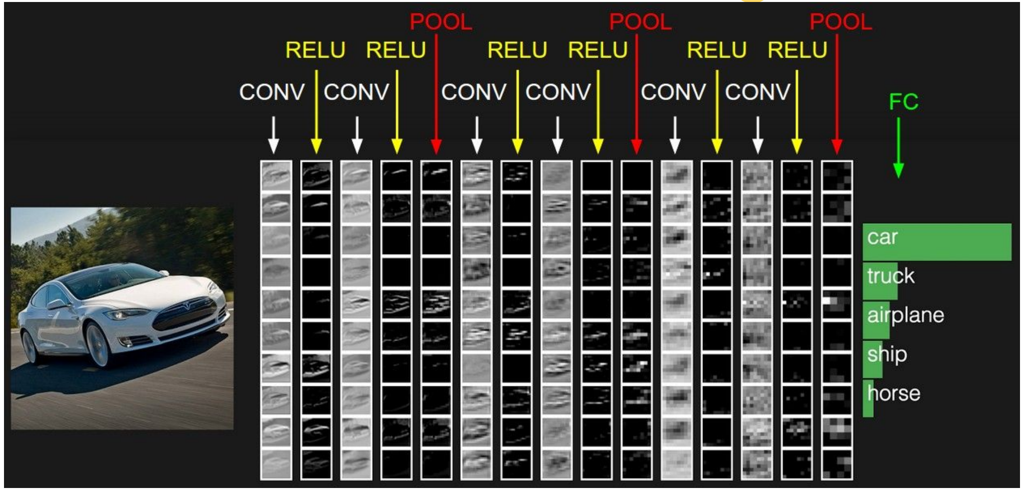


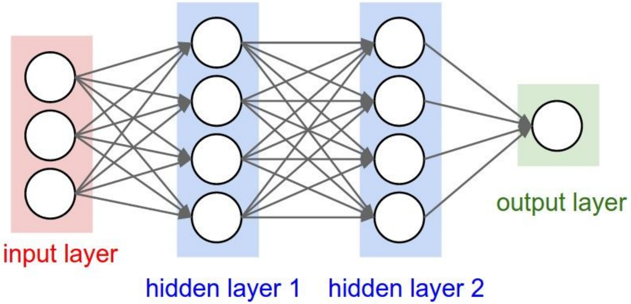
Layers

- **Activation Function**
    - Commonly, ReLU (rectified linear unit)
    - ReLU(x) = max(0,x)
    - Introduces non-linearity
    - Or, softmax (commonly at the end for classification

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

- **Fully Connected**
    - The same as in a neural net
    - Used to compute the output - a probability distribution for classification

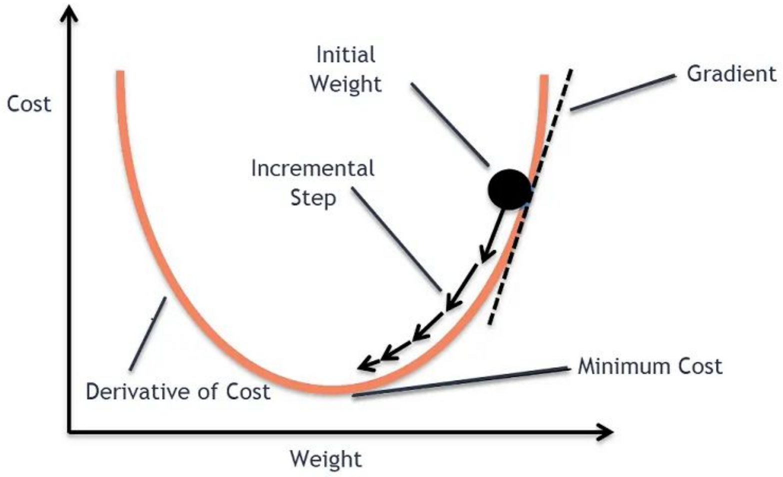




Training

Overview

- First, we need to know how well the model did
  - How far off it is from the correct answer is the **loss**
- In order to figure out how the weights need to change, we want to get $\frac{\partial L}{\partial w_i}$
- This tells us in what direction w_i needs to change in order to reach the minimum possible loss

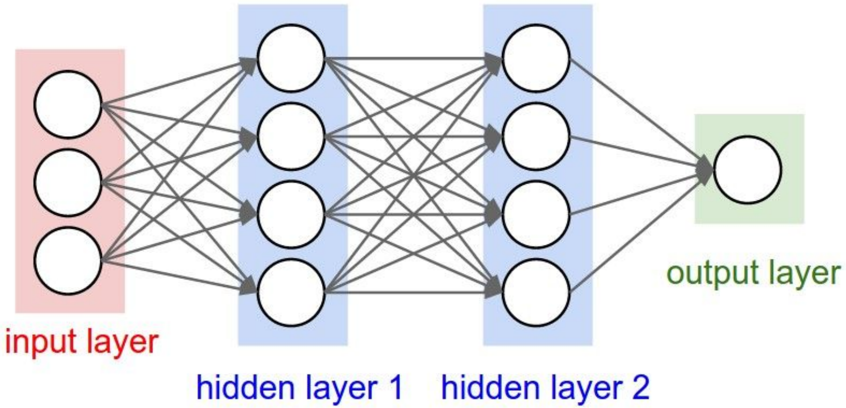


Loss Function

- To know if the model did a good job, we need to calculate how far it was from the correct answer

- We do this using a loss function - typically cross-entropy loss for classification

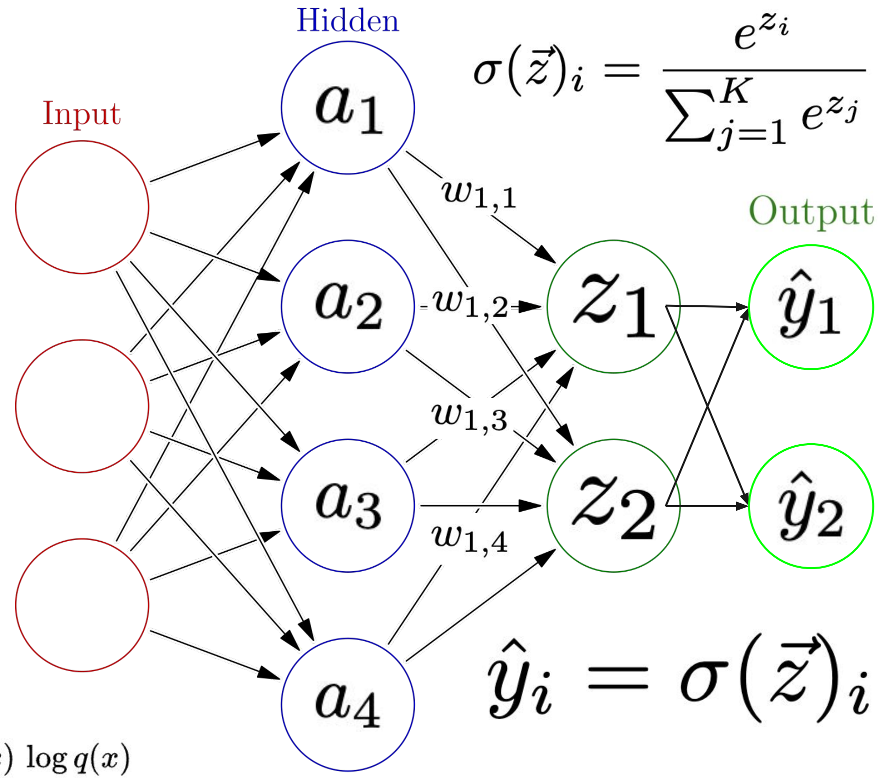
$$H(p,q) = -\sum_{x \in \mathcal{X}} p(x) \, \log q(x)$$



Backpropagation – Fully Connected

- For all the weights, we want $\frac{\partial L}{\partial w_{j,i}}$
- First, we find $\frac{\partial L}{\partial \hat{y}_i}$ and $\frac{\partial \hat{y}_i}{\partial z_j}$
- Because of the chain rule,

  $\frac{\partial \hat{y}_i}{\partial z_j} \cdot \frac{\partial L}{\partial \hat{y}_i} = \frac{\partial L}{\partial z_j}$
- We can show $\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i$
    - Proof here



$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\hat{y}_i = \sigma(\vec{z})_i$$

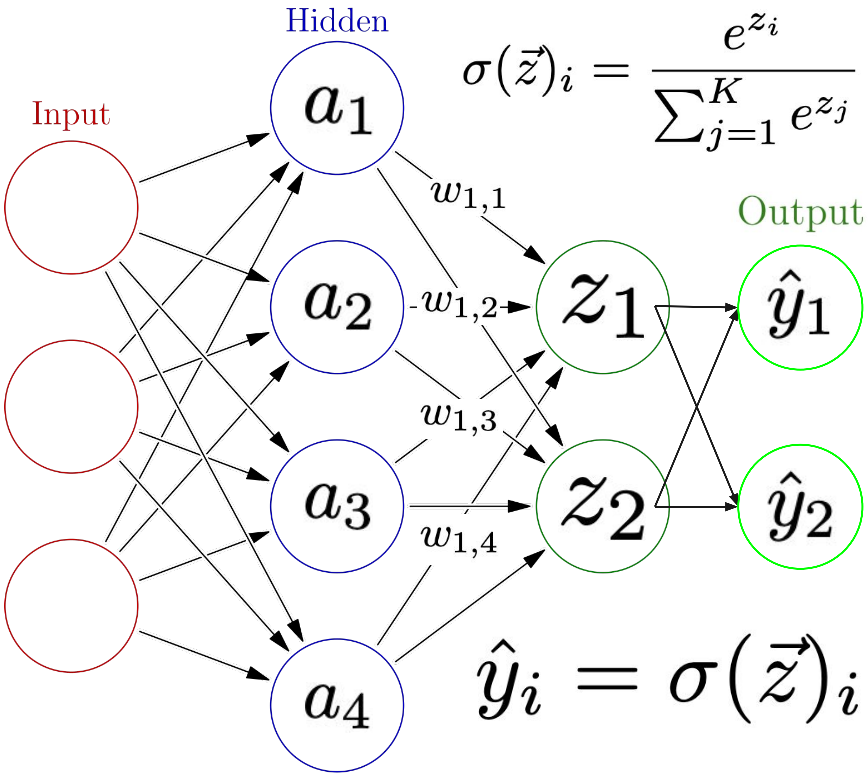
$$H(p,q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

Backpropagation – Fully Connected

- We can show $\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i$
    - Proof here
- $z_i = \sum_{k=1}^{|w|} w_{j,k} \cdot a_k \longrightarrow \frac{\partial z_i}{\partial w_{j,k}} = a_j$
- Now, using the chain rule, we can do

$$\frac{\partial L}{\partial w_{j,k}} = \frac{\partial L}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_{j,k}} = a_j(\hat{y}_i - y_i)$$

- Now, we need $\frac{\partial L}{\partial a_j}$ for the previous layers
- $\frac{\partial L}{\partial a_j} = \sum_i \frac{\partial L}{\partial z_i} \cdot \frac{\partial z_i}{\partial a_j} = \sum_i (\hat{y}_i - y_i) \cdot w_{j,i}$
- $z_i = \sum_j w_{j,i} \cdot a_j \longrightarrow \frac{\partial z_i}{\partial a_j} = w_{j,i}$

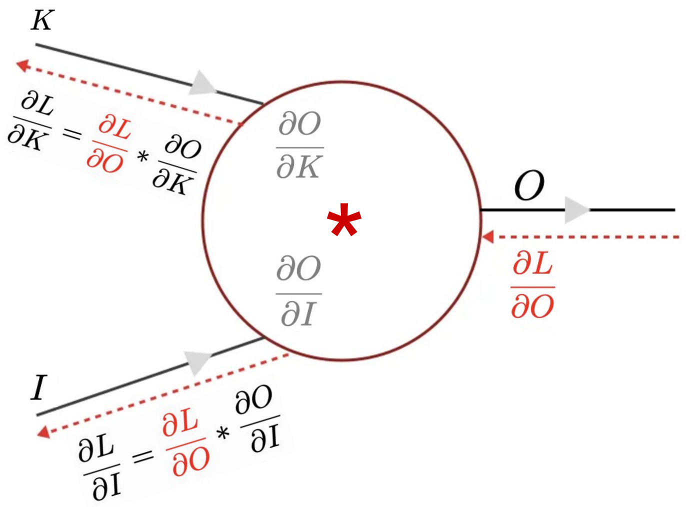


$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\hat{y}_i = \sigma(\vec{z})_i$$

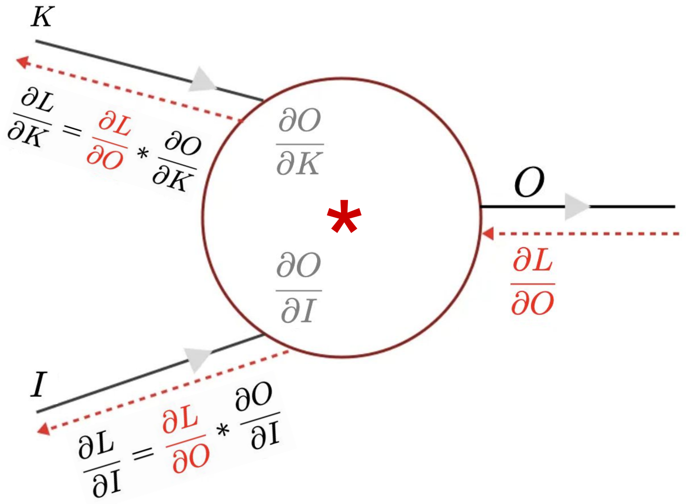
Backpropagation – Convolution

- K is the kernel
- The new values in each kernel are calculated using $\frac{\partial L}{\partial K}$
- So, we're looking for $\frac{\partial O}{\partial K}$ because $\frac{\partial L}{\partial K} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial K}$
- $\frac{\partial L}{\partial K_i} = \sum_j \frac{\partial L}{\partial O_j} \cdot \boxed{\frac{\partial O_j}{\partial K_i}}$
- We need these



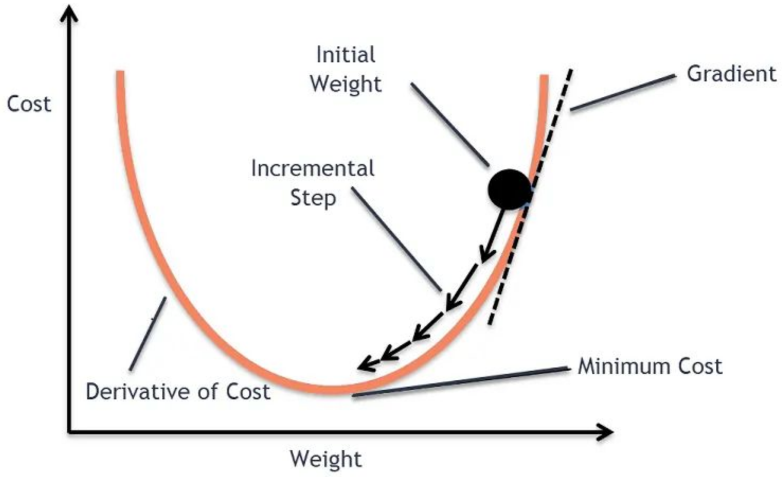
Backpropagation – Convolution

- $$O_i = \sum_{i,j} I_i \cdot K_j$$
  - Every value in K is multiplied by a value in I (maybe padding)
- $\frac{\partial O_j}{\partial K_i} = I_i$ = a value in I



Gradient Descent

- Now that we know $\frac{\partial L}{\partial w_i}$ for all values in the kernels and fully connected layers, we can figure out how to change each one to optimize the output
- $$w_{j,i,new} = w_{j,i} - \eta \frac{\partial L}{\partial w_{j,i}}$$
  - Changes more when the slope is steeper



Thanks!

Image Sources

https://cs231n.github.io/convolutional-networks/#conv

https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c

https://sarosijbose.github.io/files/talks/A%20General%20Overview%20of%203D%20Convolution%20.pdf

https://www.slideshare.net/slideshow/a-brief-survey-of-tensors/72717067

https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/

https://en.wikipedia.org/wiki/Neural_network_%28machine_learning%29