1 Introduction to Computation Graphs

1.1 Motivation for Computation Graphs

Most modern machine learning methods - in particular, deep neural networks, rely on using gradient-based optimization techniques to iteratively improve their performance. While analytically computing gradients of a loss function with respect to the parameters of simple models like linear models might be relatively straightforward, the complexity of neural networks forces us to use more sophisticated techniques like back-propagation. While backpropagation might initially seem arcane, the goal of this recitation is to improve our understanding of the backward flow of gradients through a neural network by introducing the notion of computation graphs.

1.2 Recap - Chain Rule:

If we have two functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, then

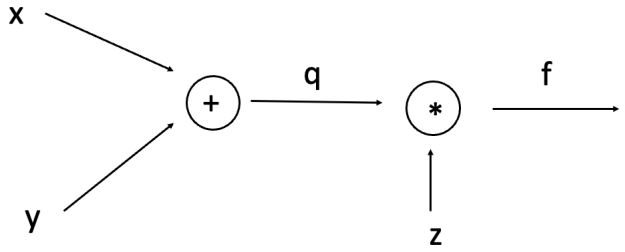
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}.$$

In the multivariable case, if we have functions $g : \mathbb{R}^n \to \mathbb{R}^m$ and $f : \mathbb{R}^m \to \mathbb{R}$, then

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial f}{\partial g_j}\frac{\partial g_j}{\partial x_i}.$$

1.3 Basic Computation Graphs

[1] Consider the function $f(x, y, z) = (x + y)z$. We could very well compute the gradient of this by expanding the expression out, but let's try visualizing this using a computation graph.



Let's call $q = x + y$, so $f = qz$. Then $\frac{\partial f}{\partial z} = q$. Now, note that the derivative of f with respect to z only depends on x and y through the intermediate value q. This will become much more useful when we consider more complicated expressions.

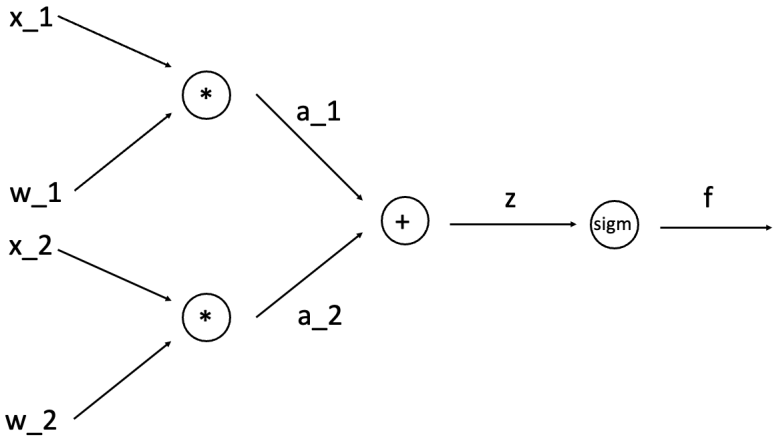
1.4 More Complex Graphs

If we observe the graph, we can see that the edges represent some kinds of data or parameters that are involved in the function we are trying to compute. In this case, it's the values x, y, and z, but in a neural network (as we'll see later), it can be things like parameter vectors or neural network inputs.
On the other hand, the nodes represent some function that is applied to the in-flowing edges. For example, at the second multiplication node, the inputs are $q = x + y$ and z. Crucially, if the node represents a function that is differentiable in its inputs, then we can compute the derivatives of this function with respect to its inputs, which by the chain rule, will be useful when computing the derivatives of earlier inputs.

[1] Much of this material is inspired by content from Stanford's CS231n course and UC Berkeley's CS182 course.

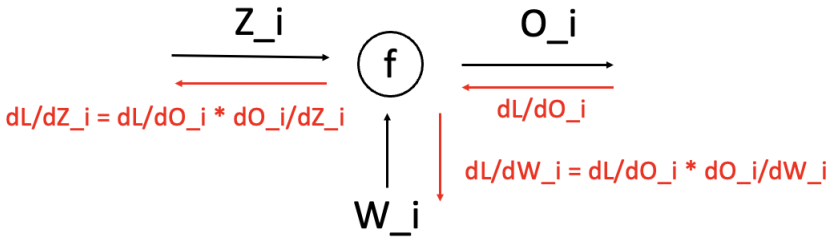
1.5 Exercise: Sigmoid Functions

Let's consider the sigmoid function $f(z) = \frac{1}{1+e^{-z}}$ whose inputs are given by the output of a linear model $w_1x_1 + w_2x_2 = w^Tx$. We can draw this computation graph as follows:

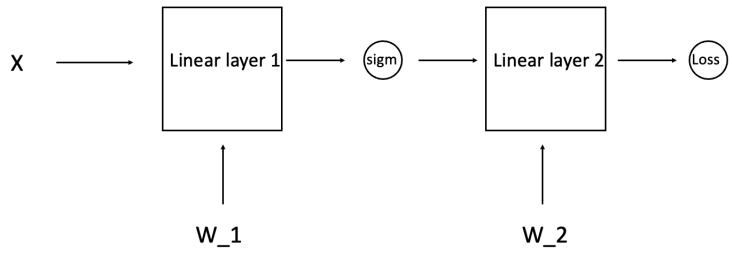


1.6 Backpropagation as a Computation Graph

Let's recall the chain rule again. We can consider some functions $f(x, y) = z$ and $g(z) = l$ such that $g(f(x, y)) = l$. If we pretend that l is a loss function, we want to compute the gradient with respect to a loss, so we want $\frac{\partial l}{\partial x}$ and $\frac{\partial l}{\partial y}$. Using the chain rule, $\frac{\partial l}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial l}{\partial z}$. This means that the derivative of l with respect to the inputs of the function f are the derivative of the value l with respect to the output of f, times the derivative of f with respect to its inputs. Let's see how we can view this in a neural network.



1.7 Graph of General Neural Network



As the image shows, we can see that the neural network itself can be represented as a function that induces a computational graph - it composes a series of differentiable operations that take in the activations and a certain set of parameters. If we use the same method for taking derivatives - ie computing the local derivative at each node, we can recover backpropagation.

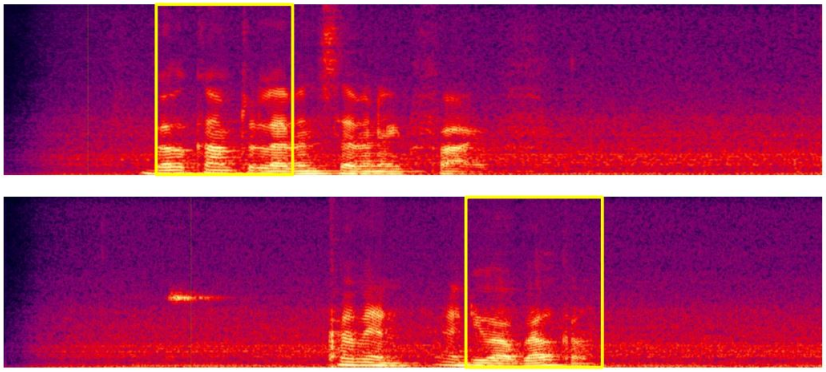
2 Convolutional Neural Networks

2.1 Motivation

1. Convultional Neural Networks are modeled after the mammalian vision cortex
2. Compared to other classification techniques, there is much less data pre-processing involved to classify an image using a CNN (this is because CNNs are position invariant)
3. Compared to traditional Neural Networks, CNNs have far fewer weights to be learned

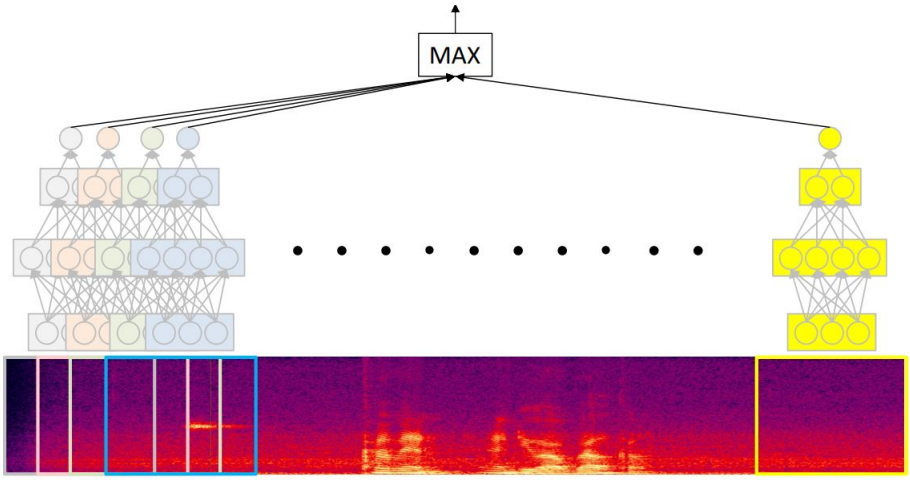
2.2 CNN as a "Scanning" Neural Network

[2] Task: We want to identify the word "welcome" in these spectrogram images.



This word can appear anywhere in the input image. Therefore, we need a position invariant model that can detect the word "welcome" anywhere in the input image.
Using a traditional Neural Network model, we would have to:

1. train a Neural Network to recognize the "welcome" pattern
2. string these Neural Networks together (each one corresponding to a different location of the input image)
3. take the maximum output value of all these sub-NNs (maximum works as an or-gate)



[2] "Scanning" Neural Network Concepts Attributed to Dr.Bhiksha Raj's 11-785 lecture "Convolutional Networks II" at Carnegie Mellon University

2.3 Recap- Terminology

1. Position Invariant: The result of the model does not change based on the position of the object in the input
2. Filter: A mask applied to an image to produce a "feature mapping"
3. Pooling: "Summarizes" the features in the feature map produced by applying the filter in the convolutional layer
4. Stride: Number of pixels the filter is shifted by

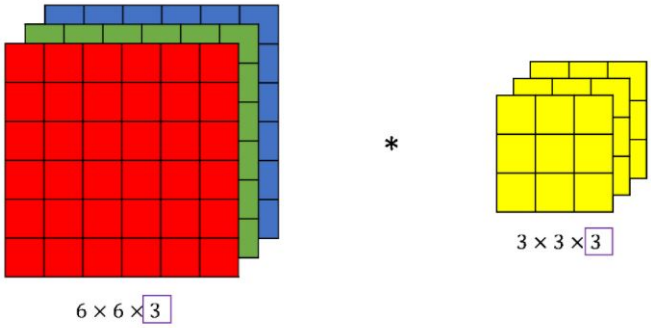
2.4 CNN Forward Pass

2.4.1 Convolutional Layer

The convolutional layer applies a "filter" which contains the weights the network learns over the layer's input. An activation function is applied after the convolution. How to Apply a filter:

1. Take the dot product between the filter and the corresponding section of the input image
2. Shift by stride

We apply a filter over a three-dimensional input, an image with RGB channels for example, the same way we would with a two-dimensional input.



What is the size of the feature mapping after this (3 x 3 x 3) filter has been applied to an input of size (6 x 6 x 3) with stride 1?

$$\text{Input Image} = \begin{bmatrix} [1,2,3] & [4,5,6] & [7,8,9] \\ [1,0,0] & [0,1,0] & [0,0,1] \\ [2,4,6] & [4,6,8] & [6,8,10] \end{bmatrix} \quad \text{Filter} = \begin{bmatrix} [-1,0,1] & [1,0,-1] \\ [-2,0,2] & [2,0,-2] \end{bmatrix}$$

What is the feature mapping after this filter has been applied to the input image with a stride of 1?

2.4.2 Pooling

The purpose of pooling is to summarize the feature mapping generated in the convolutional layer. When we use a pooling layer, we are reducing the dimension of the feature map. The type of pooling used is a hyperparameter, so no part of the pooling layer is learned by the network.

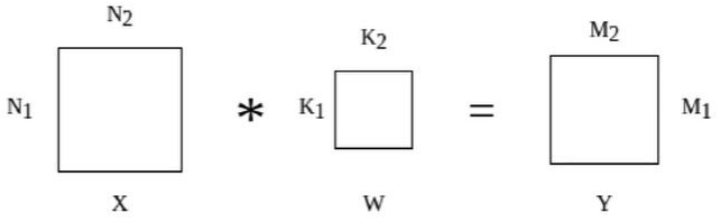
Common Pooling Functions:

1. Max Pooling: Take the maximum element of each feature map (selecting the most prominent feature)
2. Mean Pooling: Take the mean of each feature map (average of features in a patch)

What is the size of the output of applying mean pooling on the feature map in the image? Compute the output

2.5 Backpropagation through Convolutional Layer

[3] The filter used in the convolutional layer is composed of weights learned by the network. Consider a single convolutional filter $W^{k_1 \times k_2}$ applied to an image $X^{n_1 \times n_2}$ to form the output $Y^{m_1 \times m_2}$



$Y[i, j]$ is computed by

$$Y[i,j] = \sum_{a=0}^{k1-1} \sum_{b=0}^{k2-1} X[i-a, j-b]W[a,b] \tag{1}$$

When performing backpropagation through the convolutional layer there are two gradients which need to be computed:

1. $\partial L/\partial W$ in order to update the weights of the filter
2. $\partial L/\partial X$ for backpropagation in previous layers

2.5.1 Finding $\partial L/\partial W$

Any individual weight $\partial L/\partial W$[a', b'] affects every pixel in the output Y because each pixel in Y is computed by taking a sum of part of the input image multiplied by the weights.

Assuming $\partial L/\partial Y$ is computed through backpropagation of the last layer, using the chain rule:

$$\frac{\partial L}{\partial W[a',b']} = \sum_{i=0}^{m1-1} \sum_{j=0}^{m2-1} \frac{\partial L}{\partial Y[i,j]} \frac{\partial Y[i,j]}{\partial W[a',b']} \tag{2}$$

Where $\partial Y[i, j]/\partial W[a', b']$ is still unknown

Substituting $Y[i, j]$ into this expression:

$$\frac{\partial Y[i,j]}{\partial W[a',b']} = \frac{\partial(\sum_{a=0}^{k1-1} \sum_{b=0}^{k2-1} X[i-a, j-b]W[a,b])}{\partial W[a',b']} = \frac{\partial(X[i-a', j-b']W[a',b'])}{\partial W[a',b']} = X[i-a', j-b'] \tag{3}$$

[3] Adapted from Dr. Vineeth Balasubramanian's "Backpropagation in CNNs" Lecture at IIT-Hyderabad

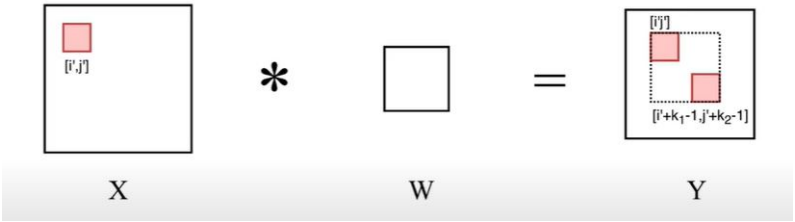
Substituting $\partial Y[i,j]/\partial W[a',b']$ back into the equation for $\partial L/\partial W[a',b']$:

$$\frac{\partial L}{\partial W[a',b']} = \sum_{i=0}^{m1-1}\sum_{j=0}^{m2-1} \frac{\partial L}{\partial Y[i,j]} X[i-a', j-b'] \tag{4}$$

This is a convolution of X over $\partial L/\partial Y$!

2.5.2 Finding $\partial L/\partial X$

We follow a similar procedure to find $\partial L/\partial X$. To compute $\partial L/\partial X[i',j']$, we must consider which pixels in Y are affected by $X[i',j']$



Fill in the blank bounds to define the region of all output pixels affected by $X[i',j']$

$$\frac{\partial L}{\partial X[i',j']} = \sum_{a=0}^{\_\_\_}\sum_{b=0}^{\_\_\_} \frac{\partial L}{\partial Y[i'+a, j'+b]} \frac{\partial Y[i'+a, j'+b]}{\partial X[i',j']} \tag{5}$$

Once again, we assume $\partial L/\partial Y[i'+a, j'+b]$ is known through backpropagation from the last layer.

To compute $\partial Y[i'+a, j'+b]/\partial X[i',j']$, recall

$$Y[i',j'] = \sum_{a=0}^{k1-1}\sum_{b=0}^{k2-1} X[i'-a, j'-b]W[a,b] \tag{6}$$

This can be re-written as

$$Y[i'+a, j'+b] = \sum_{a=0}^{k1-1}\sum_{b=0}^{k2-1} X[i',j']W[a,b] \Rightarrow \tag{7}$$

What is $\frac{\partial Y[i'+a,j'+b]}{\partial X[i',j']}$?

Substituting into (6)

$$\frac{\partial L}{\partial X[i',j']} = \sum_{a=0}^{k1-1}\sum_{b=0}^{k2-1} \frac{\partial L}{\partial Y[i'+a, j'+b]} W[a,b] = \frac{\partial L}{\partial Y} * flip_{180^\circ}(W) \tag{8}$$

Thus, $\partial L/\partial X$ can be expressed as another convolution operation using a 180 degree rotation of W!

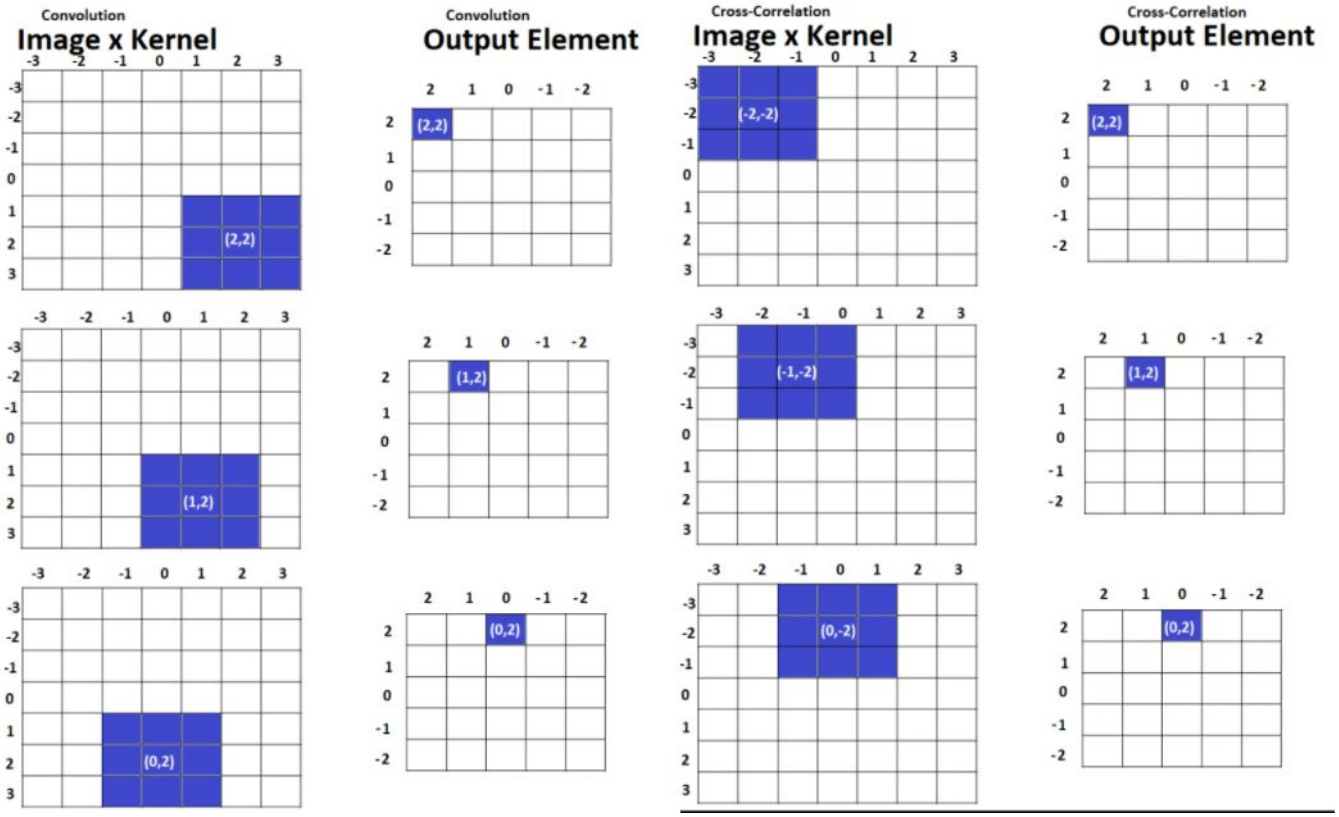
Why the 180 Degree Rotation?
This is because of Cross-Correlation! Recall (1) where **convolution** is computed as:

$$Y[i,j] = \sum_{a=0}^{k1-1}\sum_{b=0}^{k2-1} X[i-a, j-b]W[a,b] \tag{9}$$

Cross-Correlation, one the other hand, is computed as:

$$Y[i,j] = \sum_{a=0}^{k1-1} \sum_{b=0}^{k2-1} X[i+a, j+b] W[a,b] \tag{10}$$

Notice, the only difference here is how X is indexed. As a result, a cross-correlation operation acts as a "flipped" convolution.



In (7), the $\partial L / \partial Y[i'+a, j'+b]$ term indicates that this operation is a cross-correlation. In order to represent it as a convolution, we flipped the filter matrix W.