CSCC11 Week 11 Notes

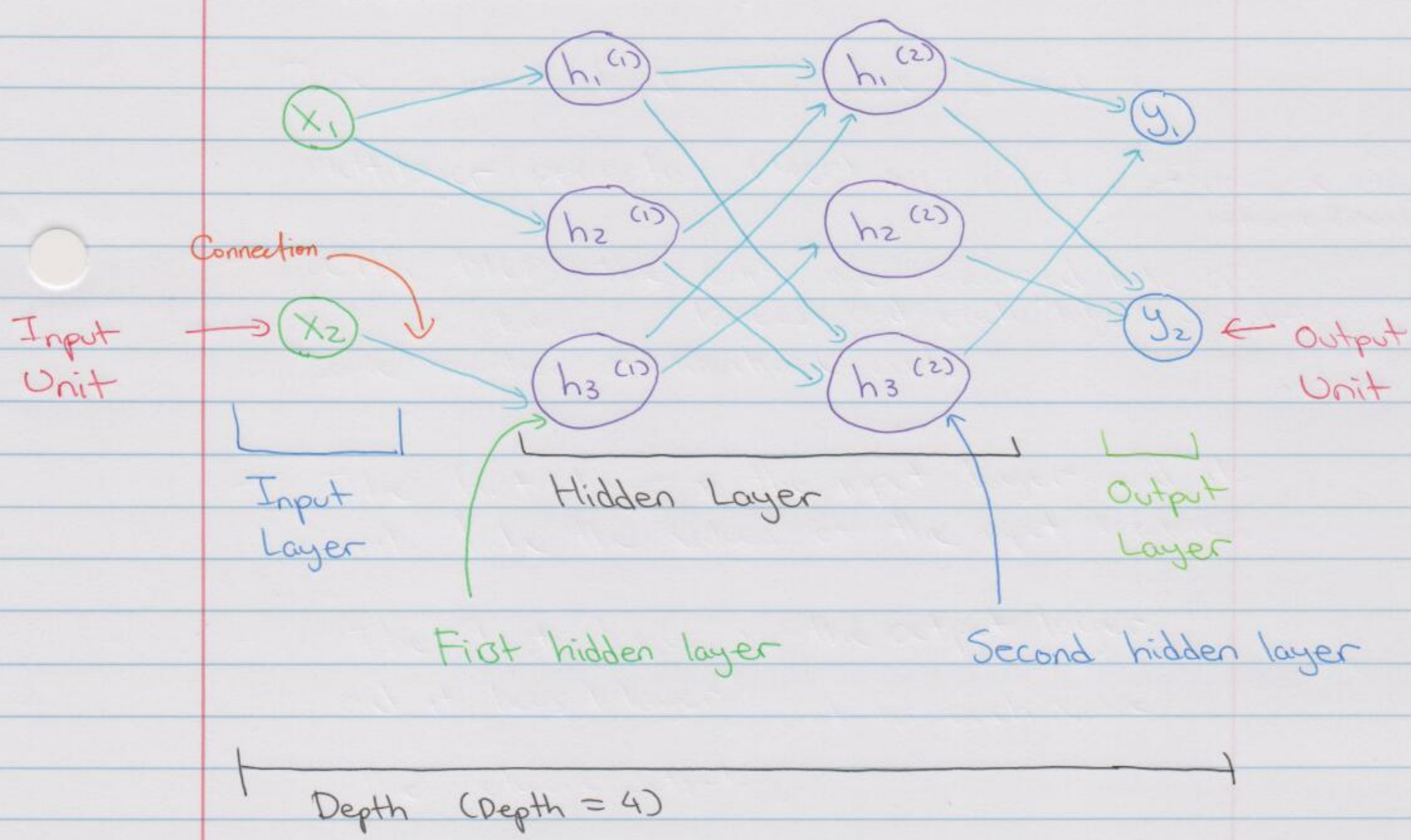
Multilayer Perceptrons (MLP):

- Consider $\phi\left(\sum_j w_j x_j + b\right)$. We've seen variations of it.
  For linear regression, $\phi(z) = z$.
  For logistic regression, ϕ is the logistic function $\sigma(z) = \frac{1}{1+e^{-z}}$
  A neural network is just a combination of these.
- The simplest kind of neural network is multilayer perceptron (MLP). MLP is a type of artifical neural network (ANN).
- With MLP, the units are arranged into a set of layers and each layer contains some number of identical units.
- The first layer is the input layer and its units take the values of the input features.
- The last layer is the output layer and it has 1 unit for each value the network outputs.
- All layers in btwn the input and output layers are known as hidden layers bc we don't know ahead of time what these units should compute and this needs to be discovered during learning.

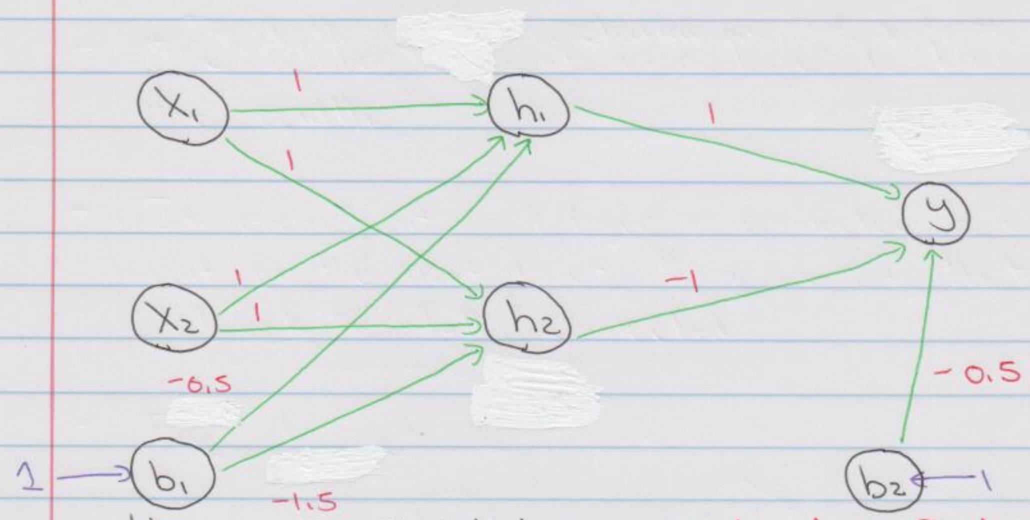
- The number of layers is known as the depth.
  If depth =1, we have shallow nn. If depth >1, deep nn.
- The number of units in a layer is known as the width.

- If every unit in 1 layer is connected to every unit in the next layer, then we say that the network is fully connected.

- E.g. 1



- E.g. 2 This is an MLP that computes XOR



Here, we will introduce an activation function.
An activation function is just a non-linear function applied to a node to get its output value from its inputs.

For this example, we'll define the activation function to be:

$$\phi(x) = \begin{cases} 1, & \text{if } w_1 x_1 + w_2 x_2 + b_1 \cdot w b_1 \geq 1 \\ 0, & \text{if } w_1 x_1 + w_2 x_2 + b_1 \cdot w b_1 < 1 \end{cases}$$

Consider $x_1 = 0$ and $x_2 = 0$.

$h_1 = x_1 w_1 + x_2 w_2 + b_1 w b_1$
$= (0)(1) + (0)(1) + (1)(-0.5)$
$= -0.5$
$= 0$ ← Bc of activation function

$h_2 = x_1 w_1 + x_2 w_2 + b_1 w b_1$
$= (0)(1) + (0)(1) + (1)(-1.5)$
$= -1.5$
$= 0$ ← Bc of activation function

$y = h_1(1) + h_2(-1) + b_2(-0.5)$
$= (0)(1) + (0)(-1) + (1)(-0.5)$
$= -0.5$
$= 0$ ← Bc of activation function

Now, consider $x_1 = 1$ and $x_2 = 0$
$h_1 = x_1(1) + x_2(1) + b_1(-0.5)$
$= 1 + 0 - 0.5$
$= 0.5$
$= 1$

$h_2 = x_1(1) + x_2(1) + b_1(-1.5)$
$= -1.5$
$= 0$

$y = h_1(1) + h_2(-1) + b_2(-0.5)$
$= 1 - 0.5$
$= 0.5$
$= 1$

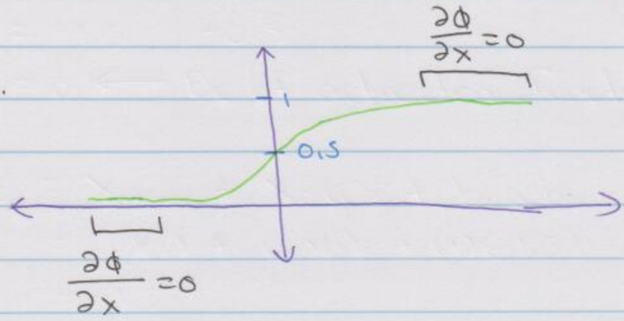
- Some activation functions are:

1. Sigmoid Function:

$$\phi(x) = \frac{1}{1+e^{-x}}$$

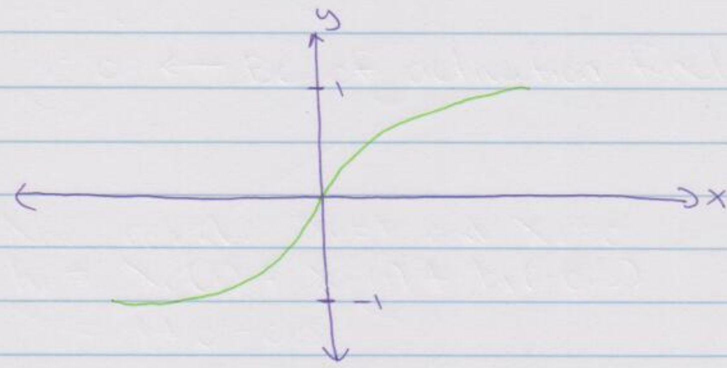
This isn't very good bc the derative of the tail is 0.

I.e.



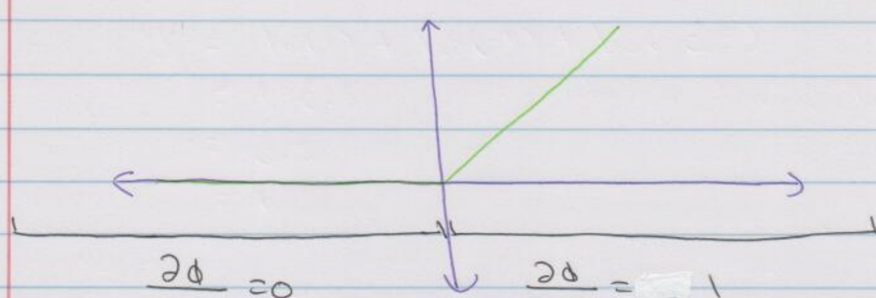
2. tanh function:

$$\phi(x) = \tanh(x)$$



3. Rectified Linear Unit (ReLU)

$$\phi(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$



$\frac{\partial \phi}{\partial x} = 0$ $\qquad$ $\frac{\partial \phi}{\partial x} = 1$

This is a popular function

4. Swish Function:

$$\phi(x) = x \cdot \text{sigmoid}(x)$$

- For hidden layer 1:

$$h_i^{(1)} = \phi_i^{(1)}\left(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)}\right)$$

E.g.

$$h_1^{(1)} = \phi_1^{(1)}\left(\sum_j w_{1j}^{(1)} x_j + b_1^{(1)}\right)$$

Note: We can "move" b into w. If we do,

$$h_i^{(1)} = \phi_i^{(1)}\left(\sum_j w_{ij}^{(1)} x_j\right)$$ but now,

$$w = [w_{11}^{(1)}, \ldots, w_{1N}^{(1)}, \underbrace{b_1^{(1)}}_{\text{New term}}]$$

$$x = [x_1, \ldots, x_N, \underbrace{1}_{\text{New term}}]$$

Suppose the width $= k$.
I.e. $k = \#$ of hidden units

$$h^{(1)} = \phi^{(1)}(W^{(1)} x)$$ where

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & \ldots & w_{1N}^{(1)} & b_1^{(1)} \\ w_{21}^{(1)} & \ldots & w_{2N}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{k1}^{(1)} & \ldots & w_{kN}^{(1)} & b_k^{(1)} \end{bmatrix}$$

$$\phi^{(1)} = \begin{bmatrix} \phi_1^{(1)} \\ \vdots \\ \vdots \\ \phi_k^{(1)} \end{bmatrix}$$

- For hidden layer 2:

$$h^{(2)} = \phi^{(2)}(h^{(1)} W^{(2)T})$$
$$= \phi^{(2)}(W^{(2)} h^{(1)})$$
$$= \phi^{(2)}(W^{2}(\phi^{(1)}(W^{(1)} x)))$$

Note: If $\phi^{(i)}$'s are linear, then we'll just have a bunch of matrix multiplications. Furthermore, if you don't have an activation function or all the activation functions are identity, then you get linear regression.

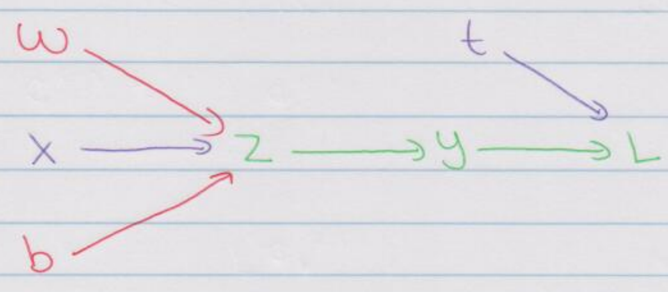
- For hidden layer L:

$$h^{(L)} = \phi^{(L)}(W^{(L)} h^{(h-1)})$$

- Now, we'll talk about how we can learn $W^{(i)}$'s.

Consider a regression problem with 1D input/output (x, t) with loss function L. We'll construct a model $z = wx + b$ followed by an activation function ϕ s.t. $y = \phi(z)$.

I.e.



— are params
— are input/output
— are operations

Forward pass is to compute the loss (L).
Backward pass/ Backpropagation is to compute the gradient.

We need to use gradient descent to learn the $w^{(i)}$'s.

For the gradients, we want to compute $\frac{\partial L}{\partial w}$, and $\frac{\partial L}{\partial b}$.

First, we have to do forward pass:

1. $z = wx + b$
2. $y = \phi(z)$
3. $L = \frac{1}{2}(y-t)^2$

Now, we can do backpropagation:

1. $\frac{\partial L}{\partial y} = y - t$
2. $\frac{\partial y}{\partial z}$ ← We don't know what ϕ is, so can't say much for now.
3. $\frac{\partial z}{\partial w} = x$

$\frac{\partial z}{\partial b} = 1$ ← Because b is a constant

We need to do these 3 steps bc $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$ and similar for $\frac{\partial L}{\partial b}$

Chain Rule

Convolutional Neural Networks:

- Artifical Neural Network (ANN) and Multilayer perceptron (MLP) in particular is not very good with analyzing visual images.

The first reason is that with MLP, because it usually involves fully connected networks. (I.e. Each unit in one hidden layer is connected with each unit in the next layer.) This makes it prone to overfitting.

The second reason is that it uses too many parameters and computations. Consider a 224x224 pixel image such that each pixel can be one of 3 colors. We'll need 224 x 224 x 3 or 150, 528 weights.

- CNN is used for classification of images and computer vision tasks.

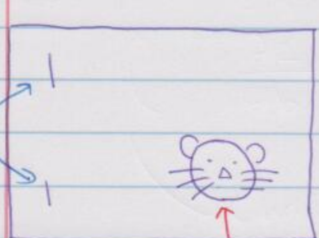
This assumption is called inductive bias.

- CNN assumes that the inputs are images and uses this fact to find patterns. We can use these patterns to reduce the number of weights.

E.g. Suppose we have an image like the one shown below and we want to see if there's a picture of 1 in the image.

Image

A pic of 1 in the image



Pic of cat

Called filter/kernel

We can overlap  on the image and see if it matches anywhere on the pic. If it does, we know that place has a 1.

- The layers in CNN have the neurons arranged in 3 dimensions (length, width, depth) where they refers to the RGB values of the image
- There are 3 main layers in CNN:

1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer (FC Layer)

Convolutional Layer:

- In this layer, we take the input data and a filter/kernel.
- The filter/kernel is a 2-D array of weights that represents a part of the image.
- The filter is applied over an area of the image and we take the dot product btwn the input pixels and the filter. The final output from the series of dot products is called the feature map/activation map/convolved feature.
- In the above point, we shift the filter by a stride until the entire input is covered.

The stride determines the amount of movement for the filter.

- E.g. Input = $\begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{bmatrix}$, filter = $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 1\times1 + 2\times1 + 4\times1 + 5\times1 = 12$$

$$\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 2\times1 + 3\times1 + 5\times1 + 6\times1 = 16$$

$$\begin{bmatrix} 3 & 1 \\ 6 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 3\times1 + 1\times1 + 5\times1 + 6\times1$$
$$= 11$$

$$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 4\times1 + 5\times1 + 7\times1 + 8\times1$$
$$= 24$$

$$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 5\times1 + 6\times1 + 8\times1 + 9\times1$$
$$= 28$$

$$\begin{bmatrix} 6 & 1 \\ 9 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 6\times1 + 1\times1 + 9\times1 + 1\times1$$
$$= 17$$

Feature map $\rightarrow \begin{bmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{bmatrix}$

Here are the strides:

$$\begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{bmatrix}$$

The red circle is the first step.
I.e. We apply the filter over the red part first.
We then shift to the blue part, then the purple part, then the light-blue part, then the turquoise part and finally the green part. Afterwards, we've covered the entire input array.

- Note: The stride doesn't have to one single part of the input array. It can be $\begin{bmatrix} 1 & 1 \\ 7 & 1 \end{bmatrix}$ for example.

- E.g. Input $= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, filter $= \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$

1. $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0\times0 + 1\times1 + 0\times0 + 1\times1 = 2$

2. $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 1\times0 + 0\times1 + 1\times0 + 0\times1 = 0$

3. $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

4. $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 2$

Feature Map

$$\begin{bmatrix} 2 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

5. $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

6. $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

7. $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 1$

8. $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

9. $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

Pooling:

- Is used to reduce the dimensionality of the feature map.
- It combines a set of values into a smaller number of values.
- This layer serves 2 purposes:
  1. Reduce the number of parameters/weights.
  2. Control overfitting
- An ideal pooling method is expected to extract only useful info and discard irrelevant details.
- There are 2 types of pooling:

1. Average Pooling:

- We divide the feature map into rectangular sections/ rectangular pooling regions and computing the avg values of each region.

- E.g.

Feature Map

1	4	2	7
2	6	8	5
3	4	0	7
1	2	3	1

$\longrightarrow$

5.5

3.25	
2.5	2.75

Region 1: $\frac{(1+2+4+6)}{4} = \frac{13}{4} = 3.25$

Region 2: $\frac{(2+5+7+8)}{4} = \frac{22}{4} = 5.5$

Region 3: $\frac{(1+2+3+4)}{4} = \frac{10}{4} = 2.5$

Region 4: $\frac{(1+3+7)}{4} = \frac{11}{4} = 2.75$

2. Max Pooling:

- We divide the feature map into rectangular pooling regions and get the max of each region.

- E.g.

Feature Map

$$\left[\begin{array}{cc|cc} 1 & 4 & 2 & 7 \\ 2 & 6 & 8 & 5 \\ \hline 3 & 4 & 0 & 7 \\ 1 & 2 & 3 & 1 \end{array}\right] \longrightarrow \left[\begin{array}{c|c} 6 & 8 \\ \hline 4 & 7 \end{array}\right]$$

Note: There could be some overlaps for the rectangular pooling regions.

Note: The window size of the pooling region is just another hyperparameter. If we use an extremely large region, we may lose out on some key information.