LEC-1

low level features

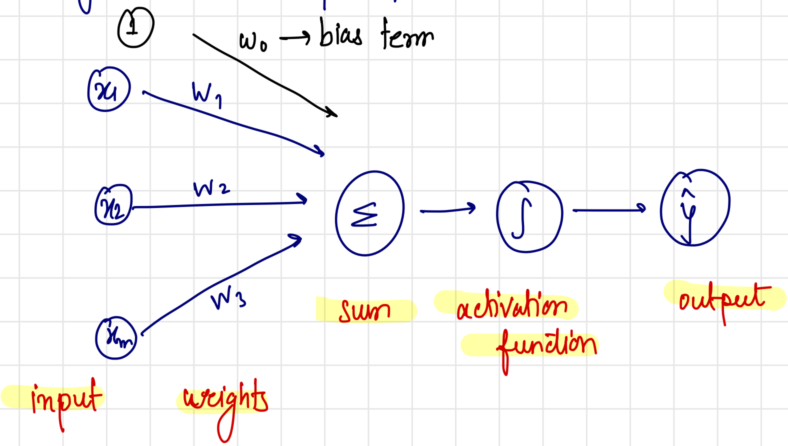
mid level features

high level features

Why now?

1) Big Data
2) Hardware advancements
3) Software

Single neuron → perceptron



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

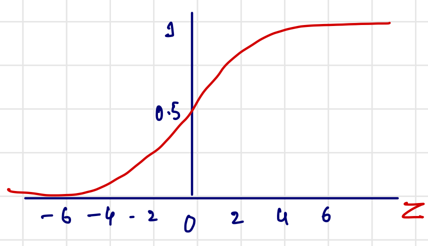
$$\hat{y} = g(w_0 + X^T W)$$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \qquad W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

$g \rightarrow$ activation function / non linear

example → **sigmoid function**

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



② **Hyperbolic tangent**

tf.math.tanh(z)

③ **Rectified Linear Unit (ReLU)**

tf.nn.relu(z)

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

$w_0 = 1 \quad w = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$\uparrow$

bias

$$\hat{y} = g(w_0 + X^T W)$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

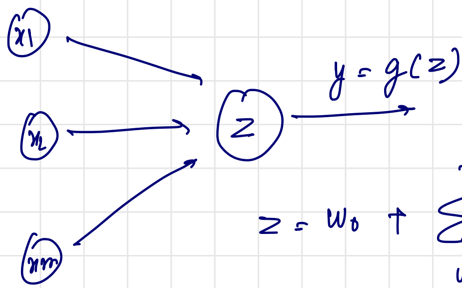
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

in 2d line

→ dot product # main process

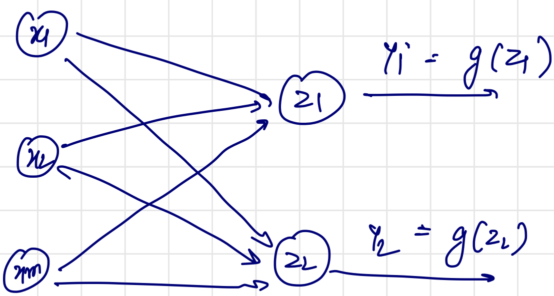
→ add bias

→ apply non linearity



$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

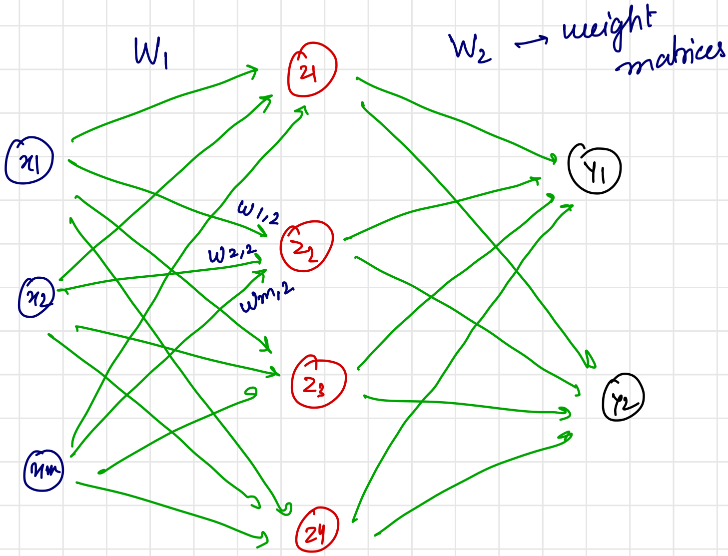
Multi Output Perceptron

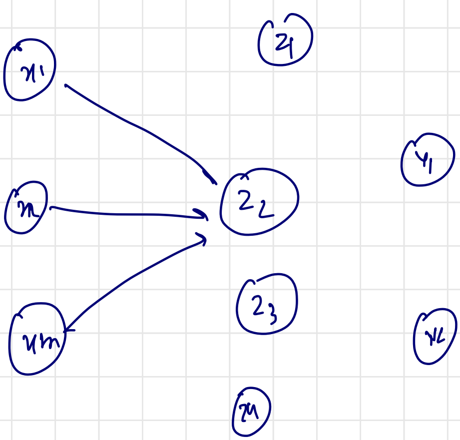


Because all inputs are directly connected to all outputs, these layers are known as Dense layers.

```
layer = tf.keras.layers.Dense(unit=2)
```

Single Layer Neural Network





$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)}$$

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(n),
    tf.keras.layers.Dense(2)])
```

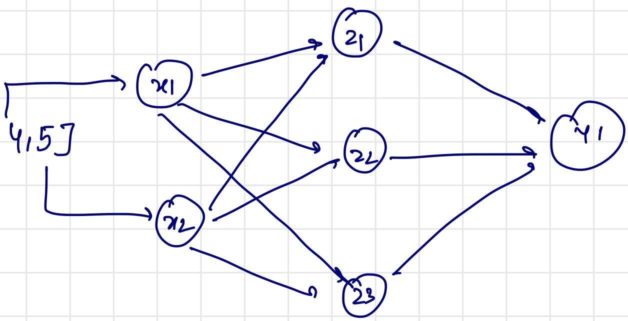
Applying Neural Networks

Q: Will I pass this class?

x_1 = no. of lectures attended

x_2 = hours spent on final project

$x^{(1)} = [4, 5]$



predicted: 0.1

10%

Actual = 1

Quantify Loss

$$\mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\underbrace{f\left(x^{(i)}; W\right)}_{\text{predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\right)$$

objective function

cost function

empirical Risk

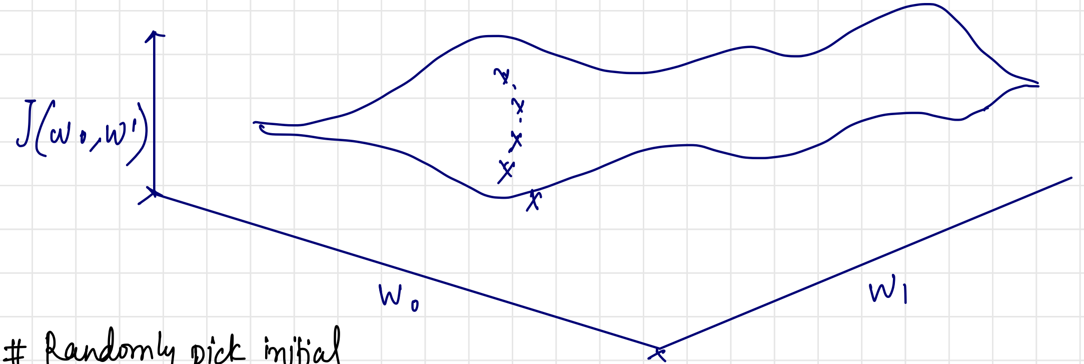
Training Neural Networks $\rightarrow$ minimize error

We want to find the network weights that achieve the lowest loss

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

$$W^* = \underset{W}{\operatorname{argmin}}\ J(W)$$

$$W = \{W^{(0)}, W^{(1)} \ldots \}$$



Randomly pick initial (w_0, w_1)

compute gradient $\frac{\partial J(W)}{\partial(W)} \rightarrow$ local minima

Algorithm for optimization - Gradient Descent

1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence
3. Compute Gradient $\frac{\partial J(W)}{\partial W}$
4. Update weights $W \leftarrow W - \eta \frac{\partial J(W)}{\partial(W)}$
5. Return weights

repeat step 3 & 4 until minimum

Computing gradients: Back propagation

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \longrightarrow J(W)$$

how does a small change in one weight (w_2) affect final loss $J(W)$?

$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_2}$$

using chain rule

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeating this process for every single weight in the network using gradient from later layers

Neural Networks in Practice: Optimization

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

$\uparrow$
learning rate

have to choose medium / average
cannot be too high or too small

Regularization

Technique that constraints our optimization problem to discourage complex model.

Regularization 1 : Drop out

\# Randomly set some activations to 0

Regularization 2 : Early stopping

\# Stop training before we have a chance to Overfit