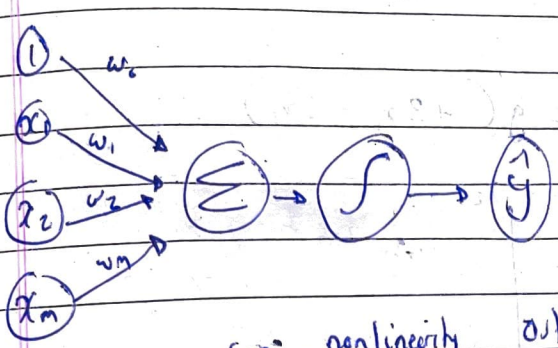
MIT 6.S191 D2

The perceptron: Forward Prop



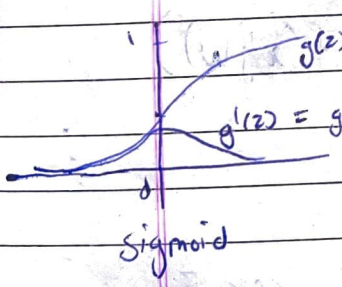
inputs weights sum non-linearity output

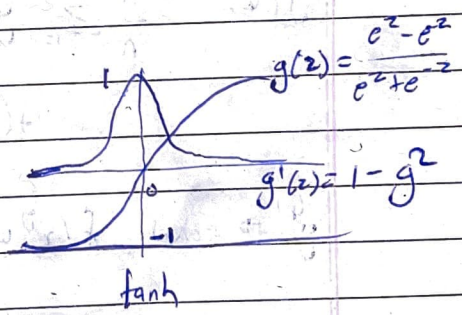
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

$$= g(w_0 + X^T W)$$

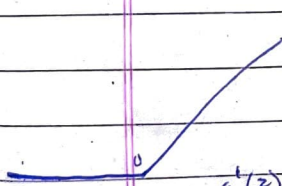
$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \qquad W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

Activation functions



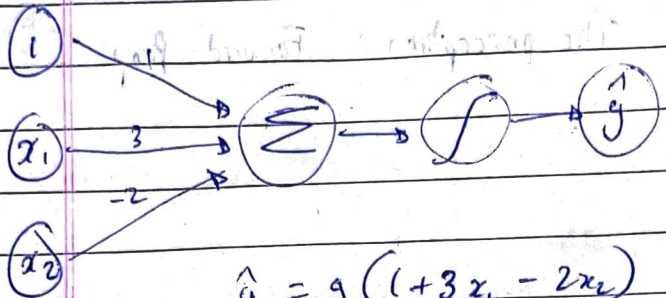


→ these activation func° are used to introduce non-linearity to the network

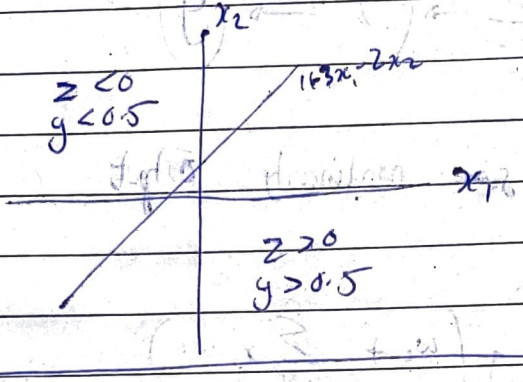


$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

- ReLU



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



Loss functions

→ Cross entropy loss for models that have probability output b/w 0 & 1

$$J(w) = \frac{-1}{n} \sum_{i=1}^{n} y^{(i)} \log\left(f(x^{(i)}; w)\right) + (1 - y^{(i)}) \log\left(1 - f(x^{(i)}; w)\right)$$

$y^{(i)}$ → actual $f(x^{(i)}, w)$ → predicted

~~In TF,~~ In TF, search softmax cross EL ~~loss = tf. reduce_mean (ff nn.~~ so

loss optimisation

$W^* \rightarrow W$ that minimises $J(W)$

Gradient descent

1. Initialise weights randomly
2. loop till ~~~~ convergence!
3. Compute gradient
4. Update weights $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

Back prop

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

→ how small change in one wt affects the loss

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

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

→ Stable learning rates converge smoothly and avoid local min

Adaptive learning rates

Gradient descent algo

- SGD
- Adam
- Adadelta
- Adagrad
- RMS Prop

Regularisation

During training

① randomly set some activ^n to 0
↓ → Forces network to not rely on any 1 node
Dropout → Drops different every iteration

② Early stopping