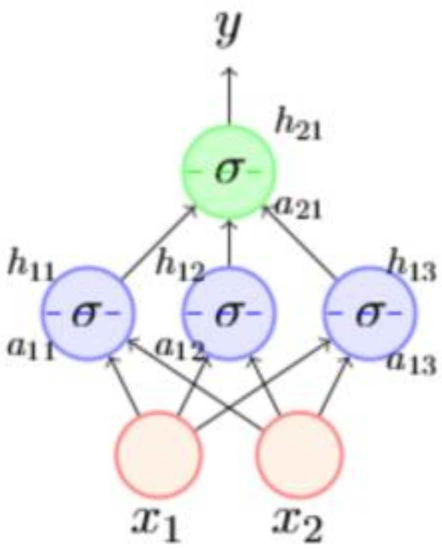
Weight initialization

Assume the scenario of equal weight initialization



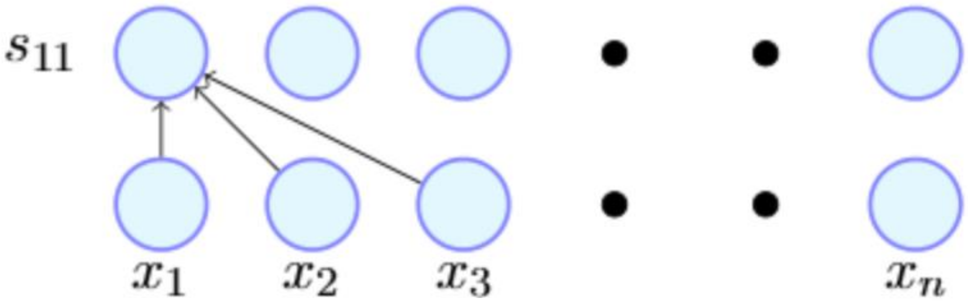
$$a_{11} = w_{11}x_1 + w_{12}x_2$$
$$a_{12} = w_{21}x_1 + w_{22}x_2$$

$$\therefore a_{11} = a_{12}$$
$$\therefore h_{11} = h_{12}$$

All neurons in layer 1 will get the same activation

Now what will happen during back propagation?

$$\nabla w_{11} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} . \frac{\partial y}{\partial h_{11}} . \frac{\partial h_{11}}{\partial a_{11}} .x_1$$
$$\nabla w_{21} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} . \frac{\partial y}{\partial h_{12}} . \frac{\partial h_{12}}{\partial a_{12}} .x_1$$
$$but \quad h_{11} = h_{12}$$
$$and \quad a_{12} = a_{12}$$
$$\therefore \nabla w_{11} = \nabla w_{21}$$

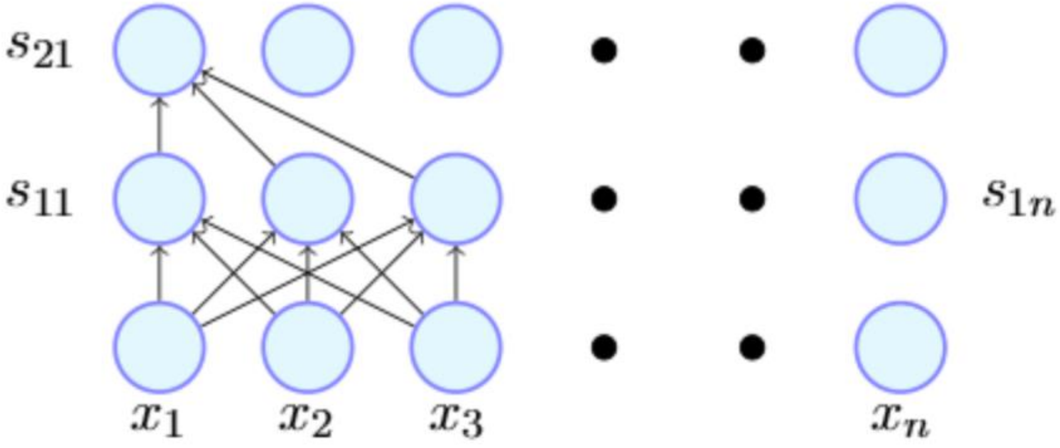


- [Assuming 0 Mean inputs and weights]
- [Assuming $Var(x_i) = Var(x) \forall i$]
- [Assuming $Var(w_{1i}) = Var(w) \forall i$]

$$Var(s_{11}) = Var(\sum_{i=1}^{n} w_{1i}x_i) = \sum_{i=1}^{n} Var(w_{1i}x_i)$$

$$= \sum_{i=1}^{n} Var(x_i)Var(w_{1i})$$

$$= (nVar(w))(Var(x))$$



$$Var(s_{21}) = \sum_{i=1}^{n} Var(s_{1i})Var(w_{2i})$$

$$= nVar(s_{1i})Var(w_2)$$

$$Var(s_{21}) \propto [nVar(w_2)][nVar(w_1)]Var(x)$$

$$\propto [nVar(w)]^2 Var(x)$$

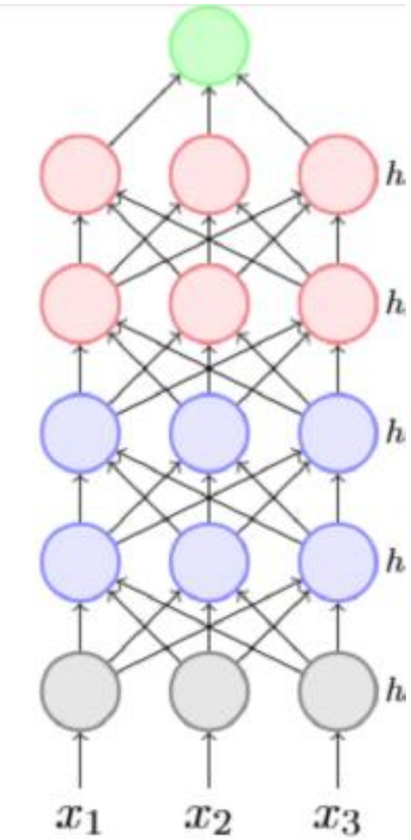
Typically, we draw the weights from a unit Gaussian and scale them by $\frac{1}{\sqrt{n}}$

- Reading Assignment-

Xavier ‘s initialization

He’s initialization

Batch normalization



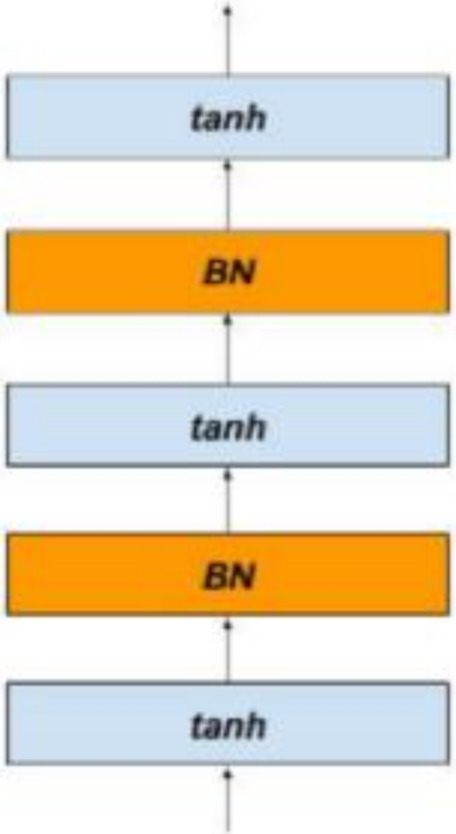
E$[s_{ik}]$ and Var$[s_{ik}]$

$$\hat{s_{ik}} = \frac{s_{ik} - E[s_{ik}]}{\sqrt{var(s_{ik})}}$$

- To understand the intuition behind Batch Normalization let us consider a deep network
- Let us focus on the learning process for the weights between these two layers
- Typically we use mini-batch algorithms
- What would happen if there is a constant change in the distribution of h_3
- In other words what would happen if across mini-batches the distribution of h_3 keeps changing

- It would help if the pre-activations at each layer were unit gaussians

- We compute it from a mini-batch
- Thus we are explicitly ensuring that the distribution of the inputs at different layers does not change across batches



- Catch: Do we necessarily want to force a unit gaussian input to the $tanh$ layer?
- Why not let the network learn what is best for it?
- After the Batch Normalization step add the following step:

$$y^{(k)} = \gamma^k \hat{s_{ik}} + \beta^{(k)}$$

- What happens if the network learns:

$$\gamma^k = \sqrt{var(x^k)}$$

$$\beta^k = E[x^k]$$

- We will recover s_{ik}
- In other words by adjusting these additional parameters the network can learn to recover s_{ik} if that is more favourable