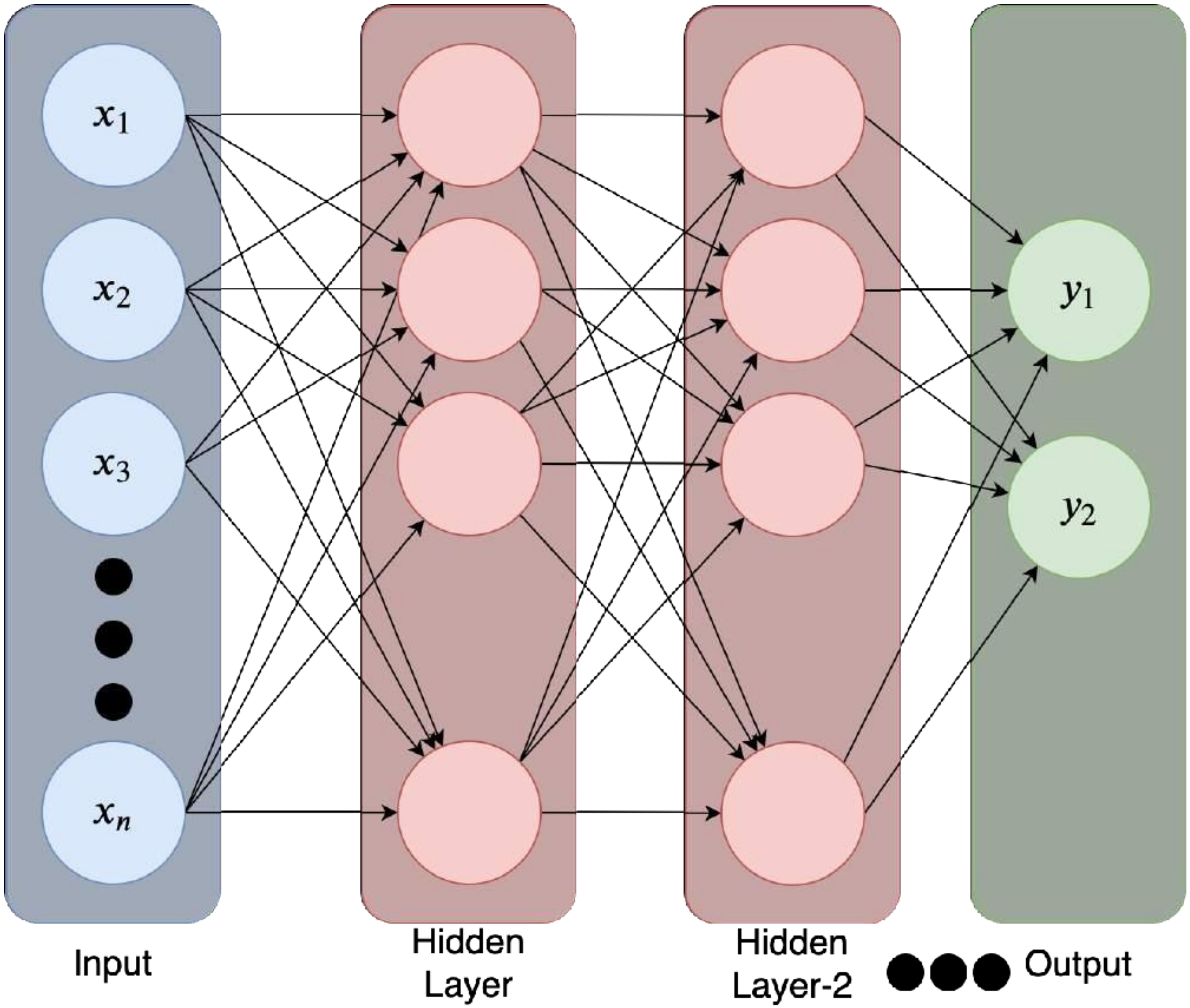
Part 2: Weights

Artificial Neural Networks



Vectorizing Forward Computation

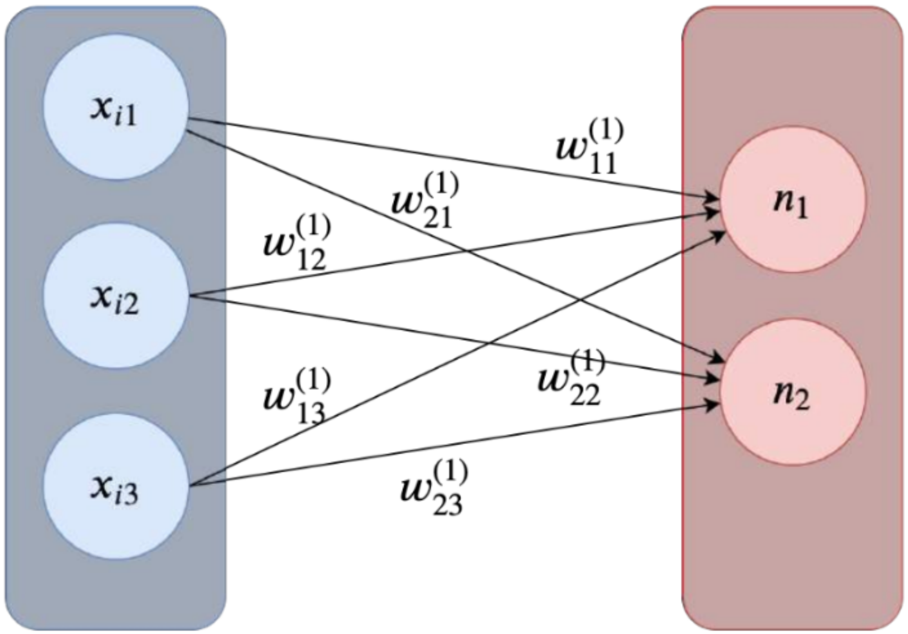
For now let's ignore bias

Inputs in layer $l-1$

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}$$

Neurons in layer l

$$w_{ij}^{(l)}: weight\ from\ j^{th}\ neuron\ on\ layer\ l\ to$$
$$the\ i^{th} neuron\ on\ layer\ l+1$$



Features/Inputs

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \end{bmatrix}$$

Observations

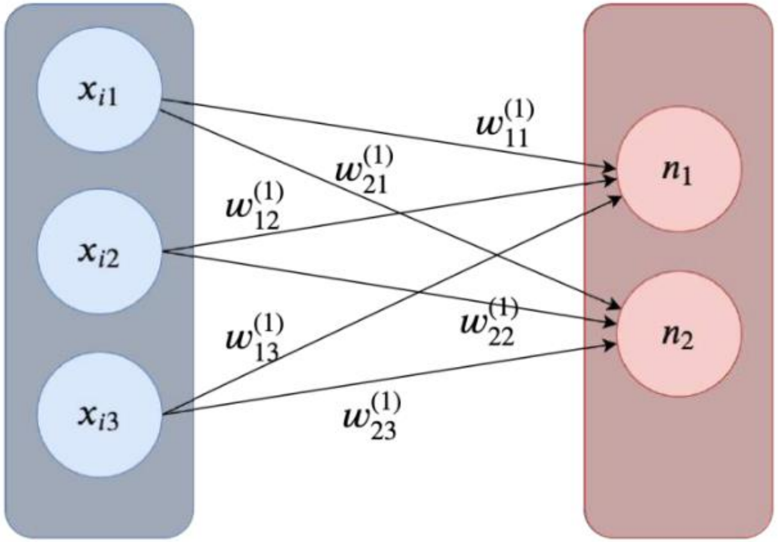
$$x_{ij}: i^{th}\ sample\ for\ feature\ j$$

Vectorizing Forward Computation

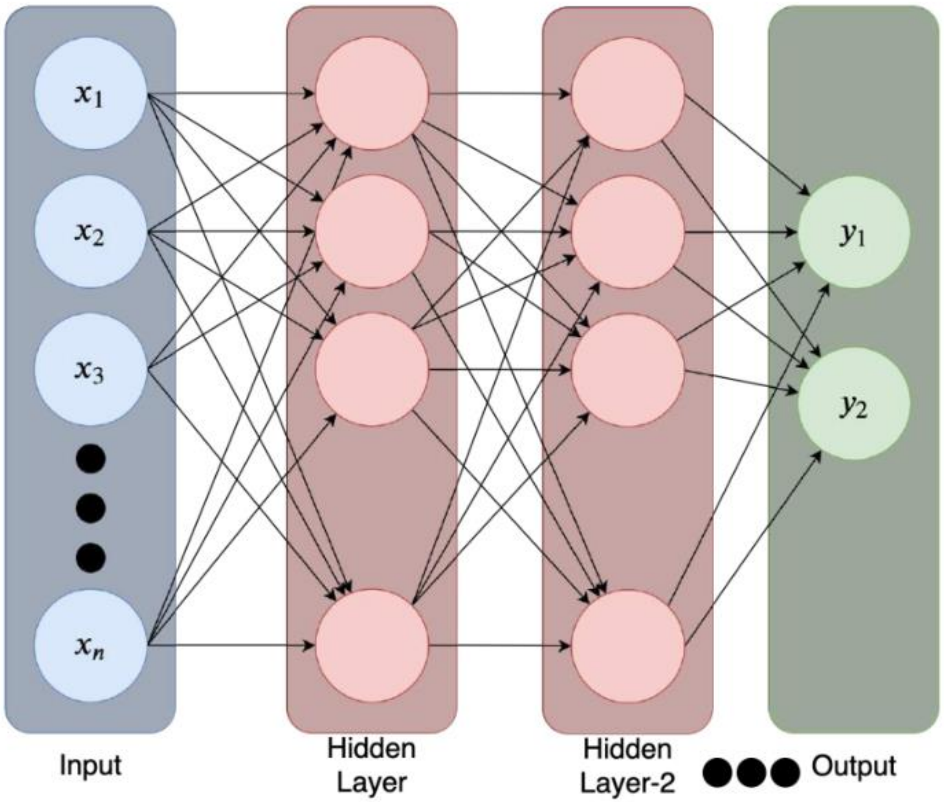
For now let’s ignore bias

$$W^{(1)} \cdot X^T = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix} \cdot \begin{array}{cc} \text{Observation} \longrightarrow & \\ \begin{bmatrix} x_{11} & x_{21} & x_{31} & x_{41} \\ x_{12} & x_{22} & x_{32} & x_{42} \\ x_{13} & x_{23} & x_{33} & x_{43} \end{bmatrix} & \downarrow \text{Features/Inputs} \end{array}$$

$$= \begin{bmatrix} w_{11}^{(1)} x_{11} + w_{12}^{(1)} x_{12} + w_{13}^{(1)} x_{13} & \dots & w_{11}^{(1)} x_{41} + w_{12}^{(1)} x_{42} + w_{13}^{(1)} x_{43} \\ w_{21}^{(1)} x_{11} + w_{22}^{(1)} x_{12} + w_{23}^{(1)} x_{13} & \dots & w_{21}^{(1)} x_{41} + w_{22}^{(1)} x_{42} + w_{23}^{(1)} x_{43} \end{bmatrix}$$



Forward Computation

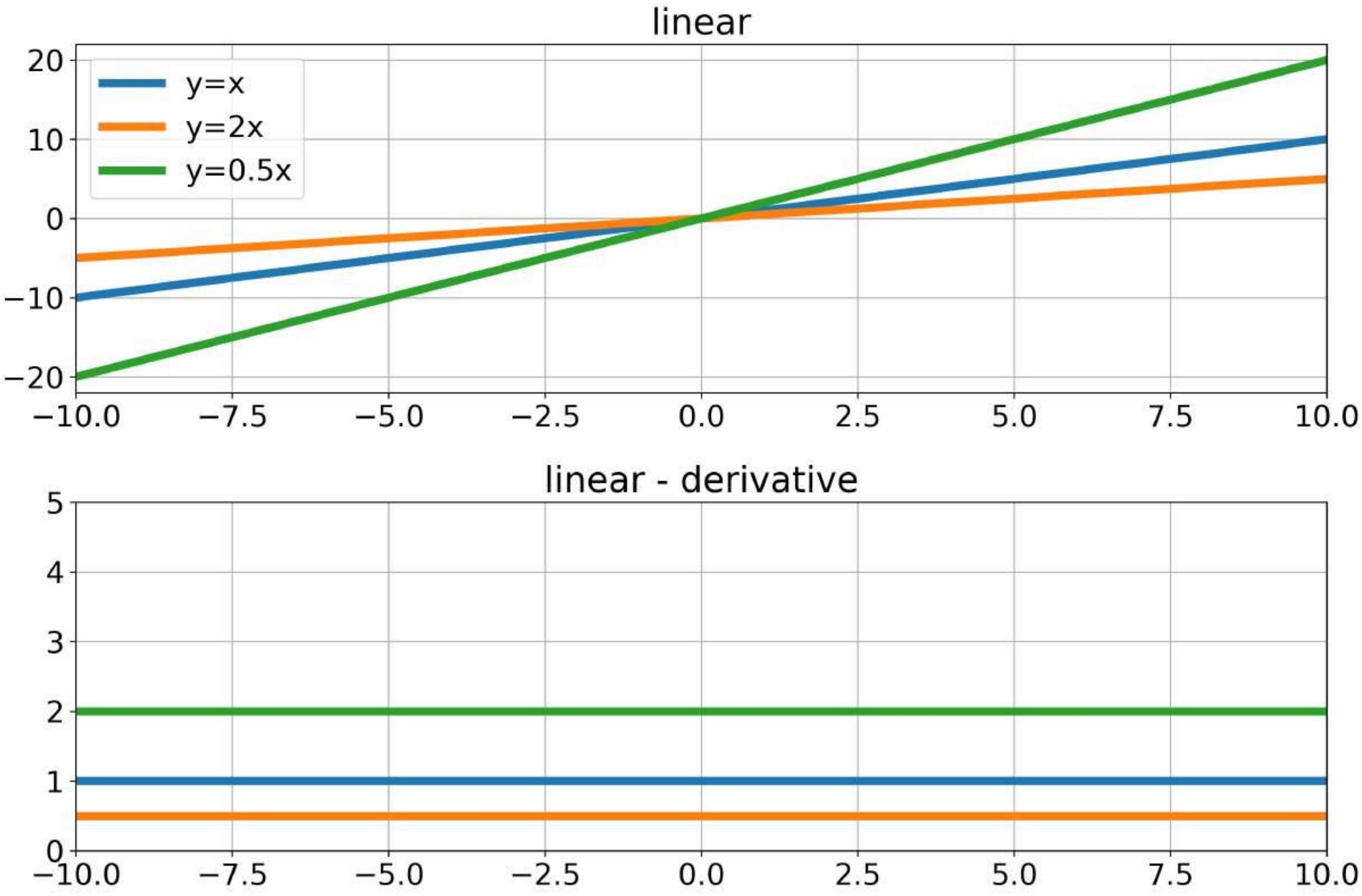


On every layer we multiply the output of the previous layer with the weights and apply activation function $\varphi(\cdot)$:

$$z^{(l)} = W^{(l)} X^T$$
$$a^{(l)} = \varphi(z^{(l)})$$

Note the $W^{(l)}$ contains also the bias

Linear Activation

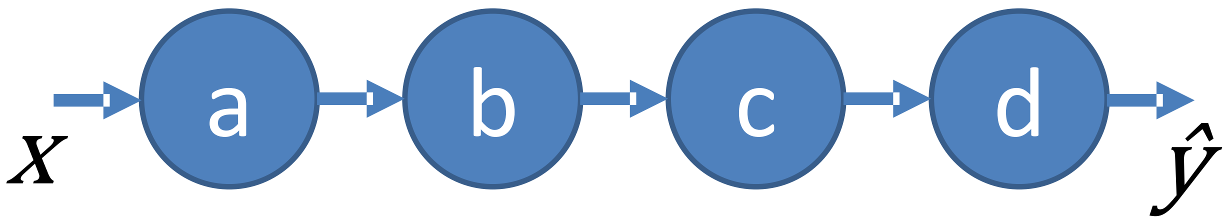


Vanishing/Exploding Gradients

Reminder: Backpropagation uses the chain rule to compute gradients.

Let's examine gradient of the first layer:

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial d} \frac{\partial d}{\partial c} \frac{\partial c}{\partial b} \frac{\partial b}{\partial a}$$

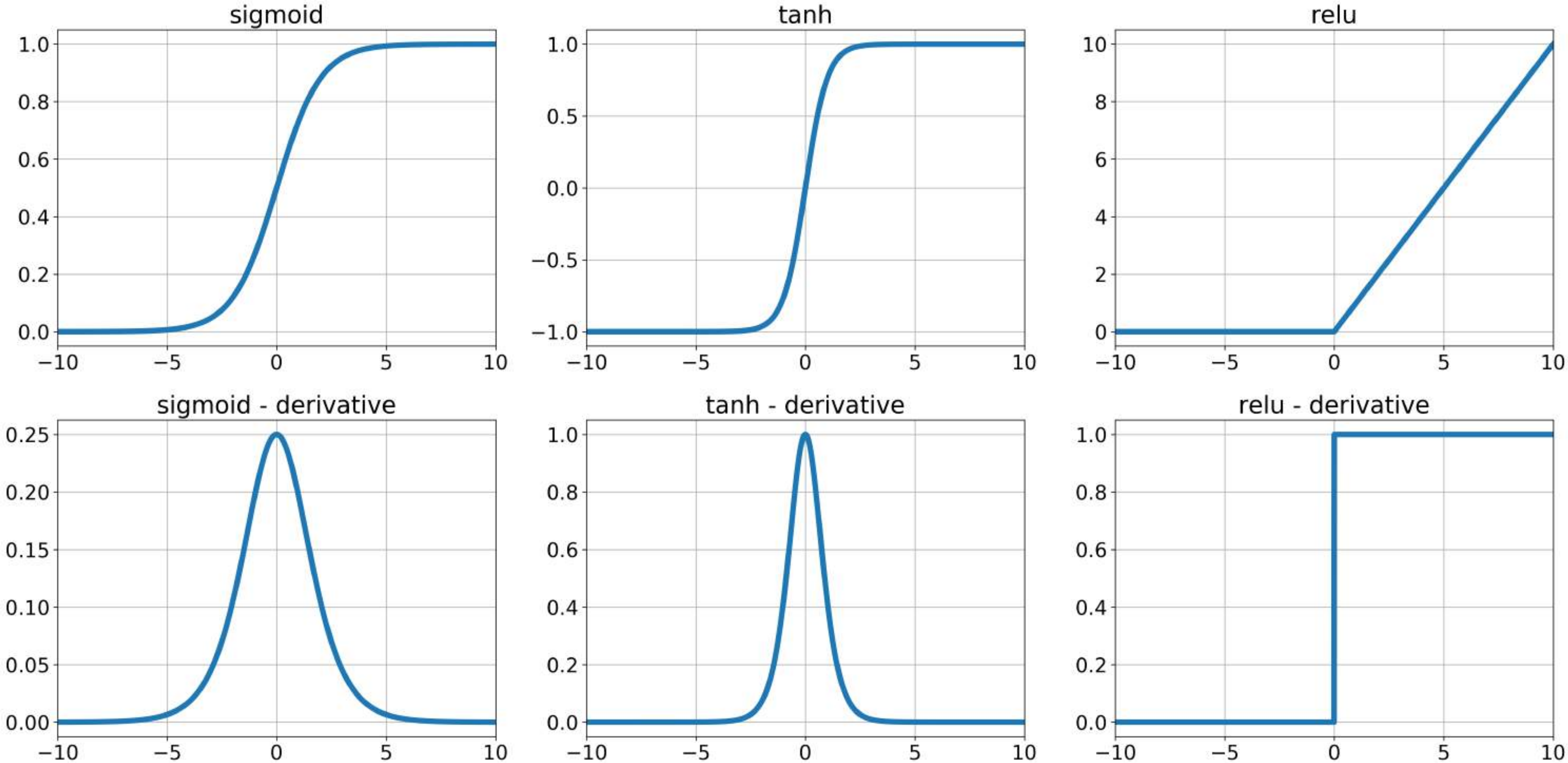


Activation Functions

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

$$\varphi(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$\varphi(x) = \max(0, x)$$



Vanishing/Exploding Activations

To simplify assume linear activation function $\varphi(x) = x$ and 0 bias. Then the NN output will be:

$$\hat{y} = W^{(n)} W^{(n-1)} \ldots \overbrace{W^{(2)} \underbrace{W^{(1)} X^T}_{first\ layer}}^{second\ layer}$$

If we initialize weights randomly:

- $W >> 1$

What will happen to the output of the NN?

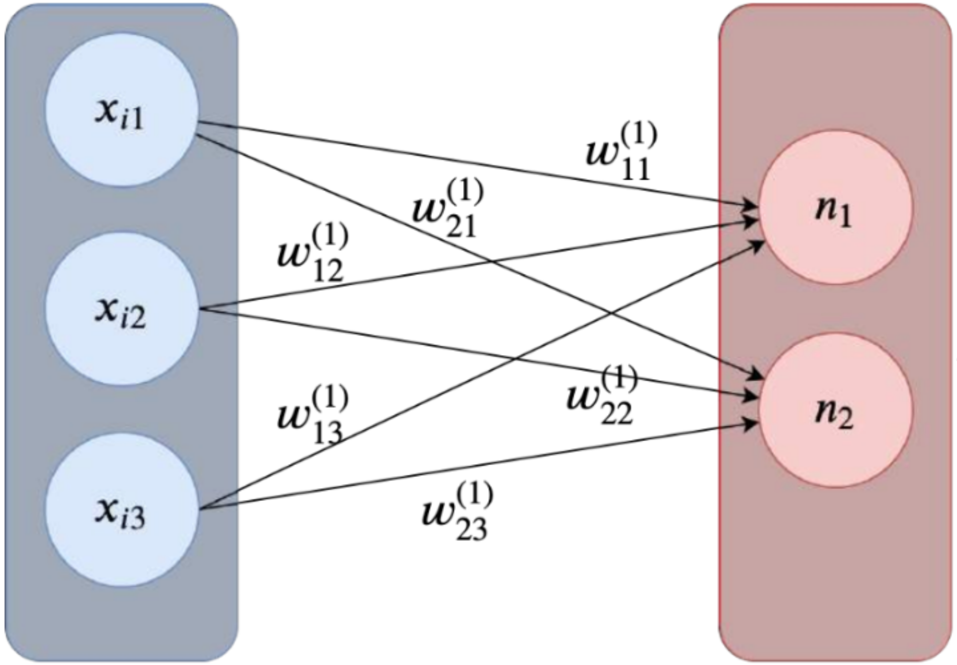
Vectorizing Forward Computation

Bias — Inputs in layer $l-1$

$$W^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{10}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}$$

Neurons in layer l

$$w_{ij}^{(l)}: weight\ from\ j^{th}\ neuron\ on\ layer\ l\ to\ the\ i^{th} neuron\ on\ layer\ l+1$$



For Bias — Features/Inputs

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ 1 & x_{31} & x_{32} & x_{33} \\ 1 & x_{41} & x_{42} & x_{43} \end{bmatrix}$$

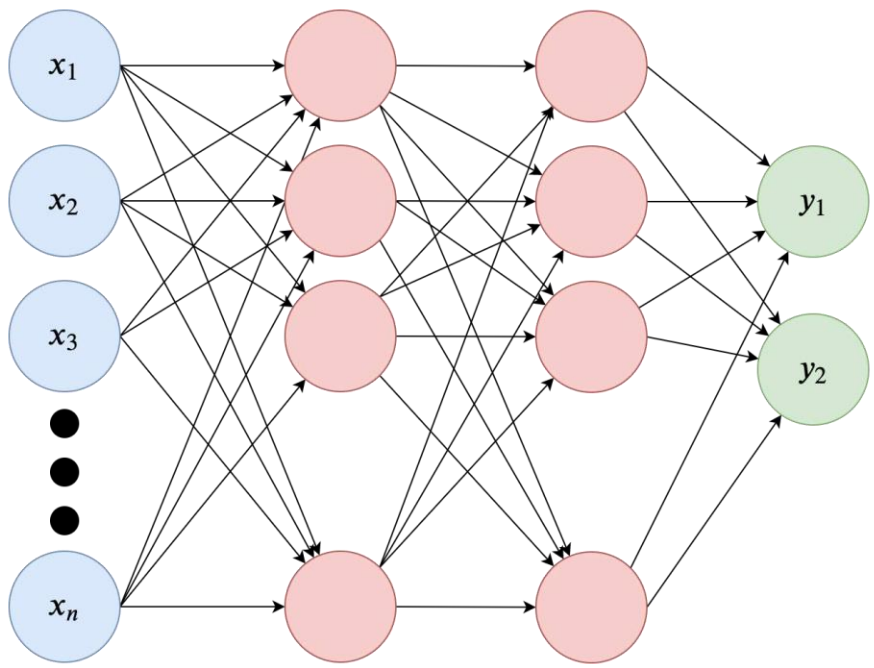
Observations

$$x_{ij}: i^{th}\ sample\ for\ feature\ j$$

Weight Initialisation

Q: Why not initialize all the weights to 0?

W=0

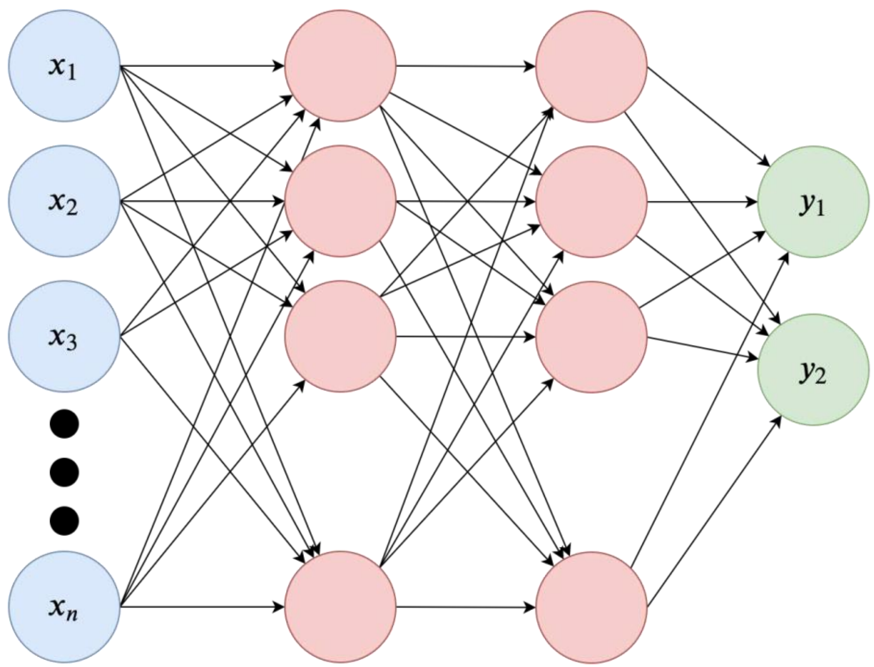


Weight Initialisation (LeCun)

Initialise with small random number $N(0, \alpha)$

$\alpha = 1\text{e-}2$

Works for small networks but leads to skewed activations for deeper networks

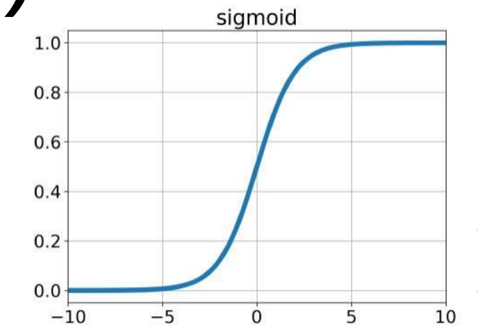


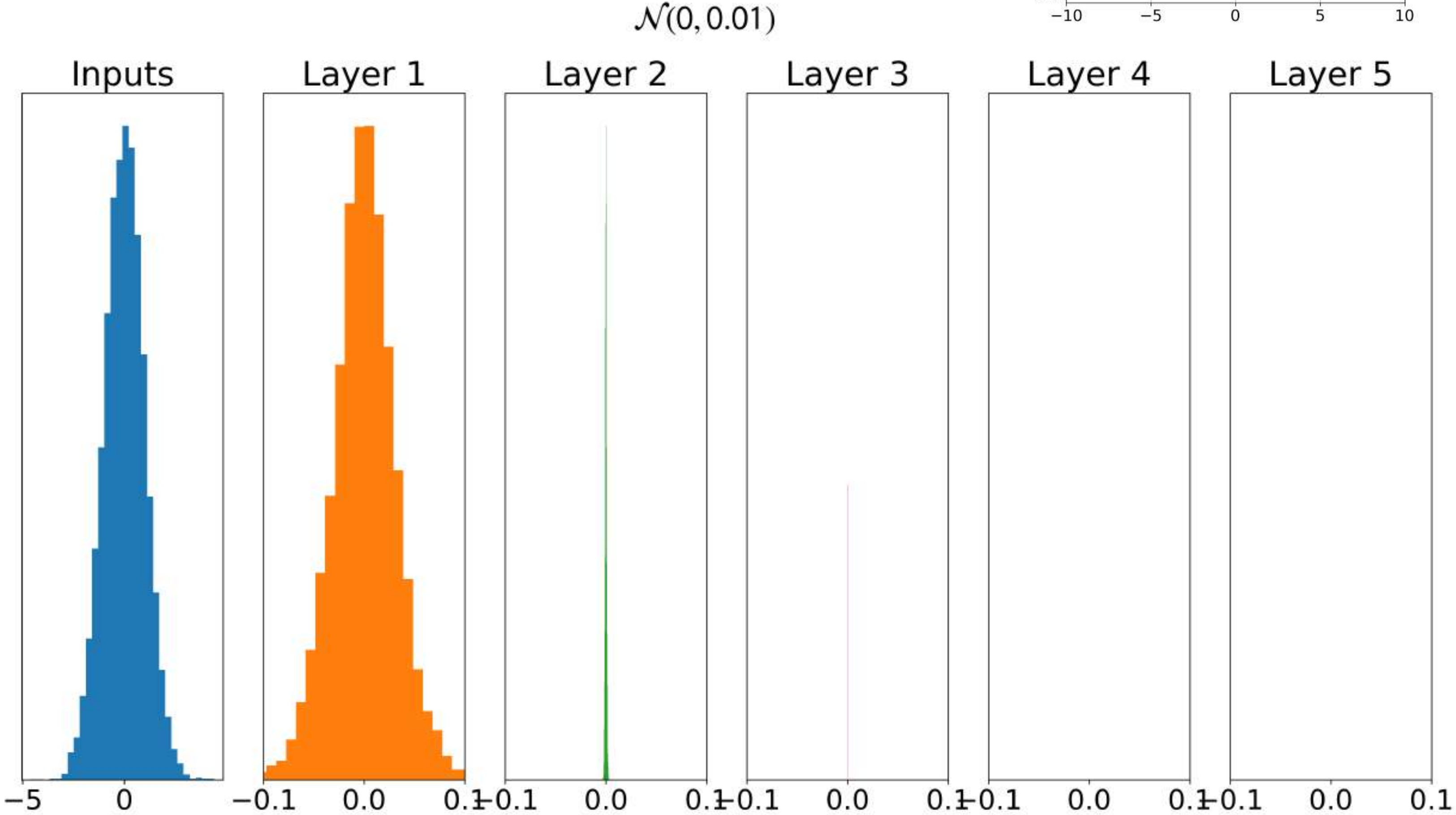
Weight Initialisation (LeCun)

- Let’s assume that we have 10-D input data x data normally distributed in $N(0, 1)$
- Let’s create 5 layers with 10 neurons on each layer
- Let’s initialise weights from using a normal distribution $N(0, 1\text{e -}2)$

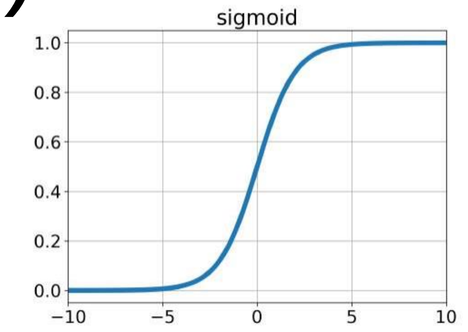
What is the expected distribution of activations after a forward pass?

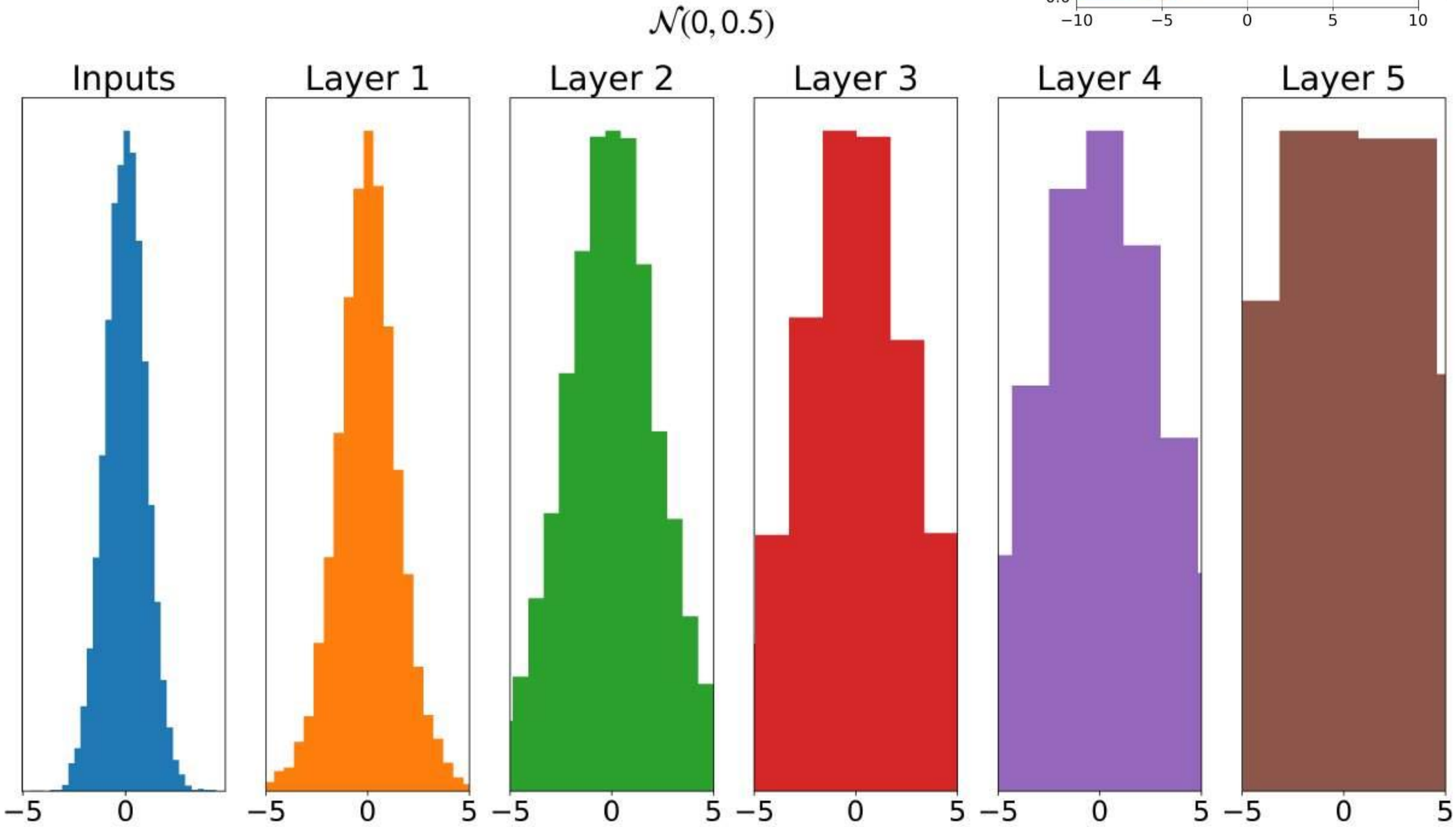
Weight Initialisation (LeCun)





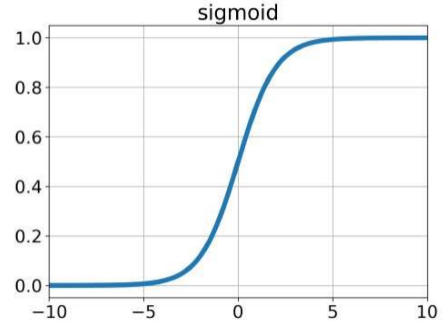
Weight Initialisation (LeCun)

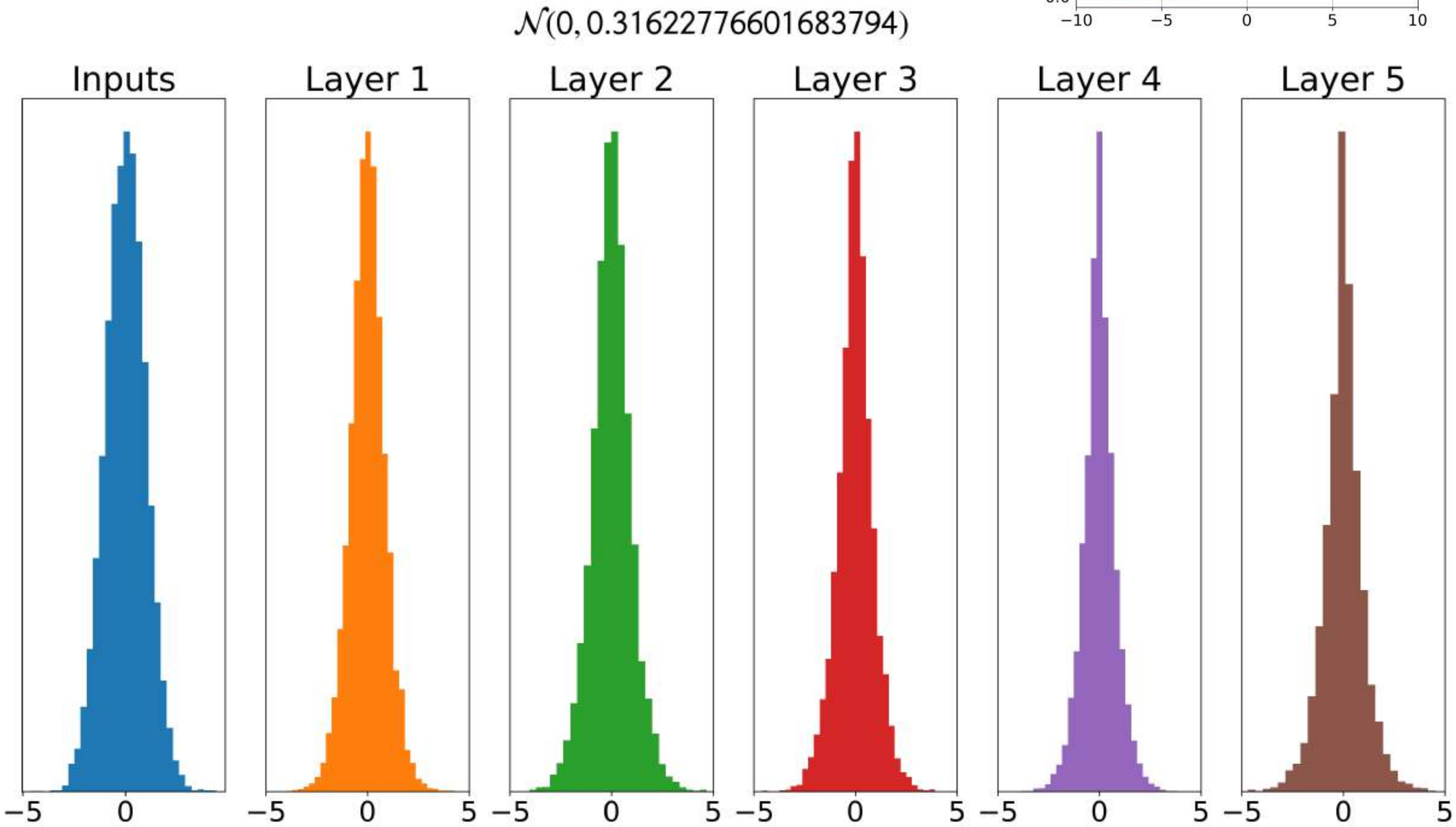




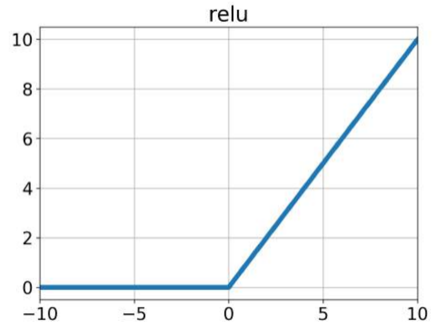
Xavier Initialisation

$$\sqrt{\frac{1}{size^{(l-1)}}}$$



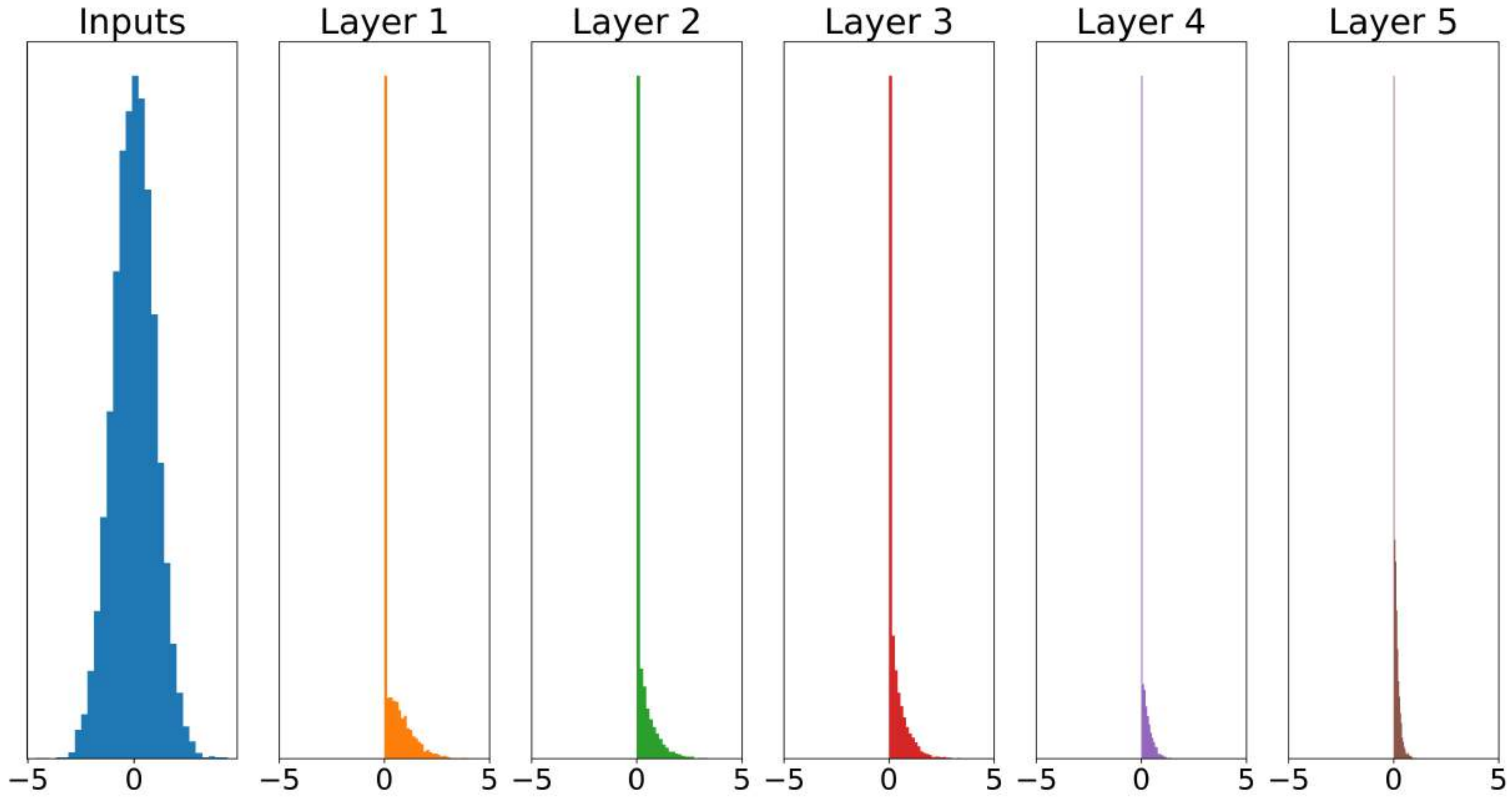


Xavier with ReLU



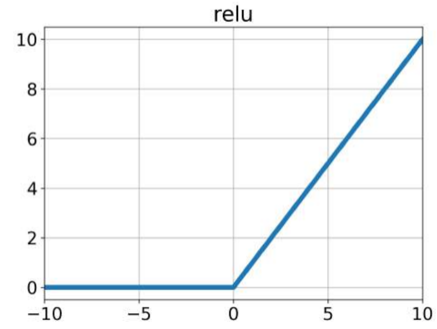
$$\sqrt{\frac{1}{size^{(l-1)}}}$$

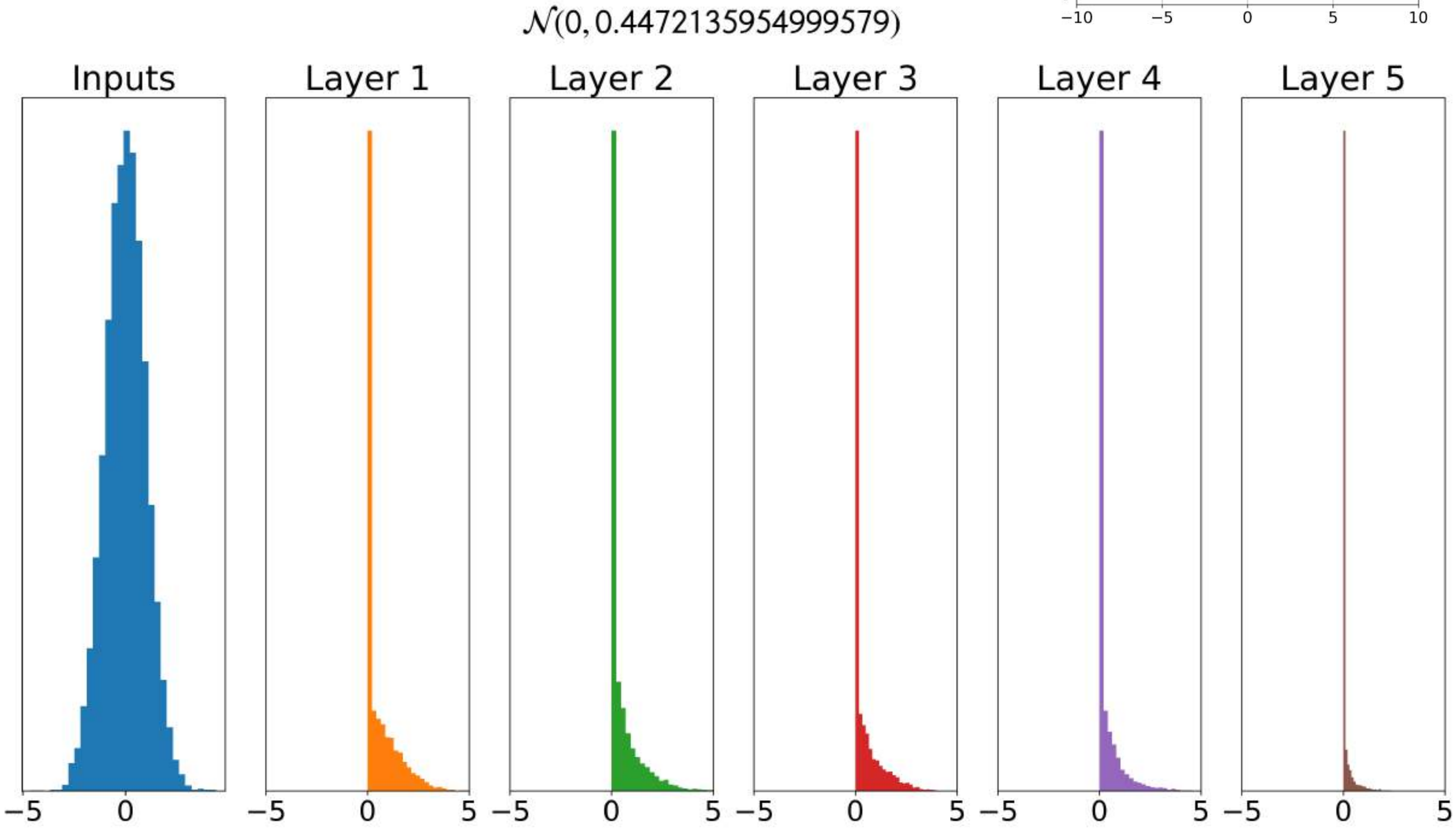
$\mathcal{N}(0, 0.31622776601683794)$



Kaiming Initialisation

$$\sqrt{\frac{2}{size^{(l-1)} + size^{(l)}}}$$





Batch Normalisation

If we need zero mean, unit variance in every layer why not normalize at every layer for every mini-batch?
$\mathcal{B} = \{x_1 \ldots m\}$

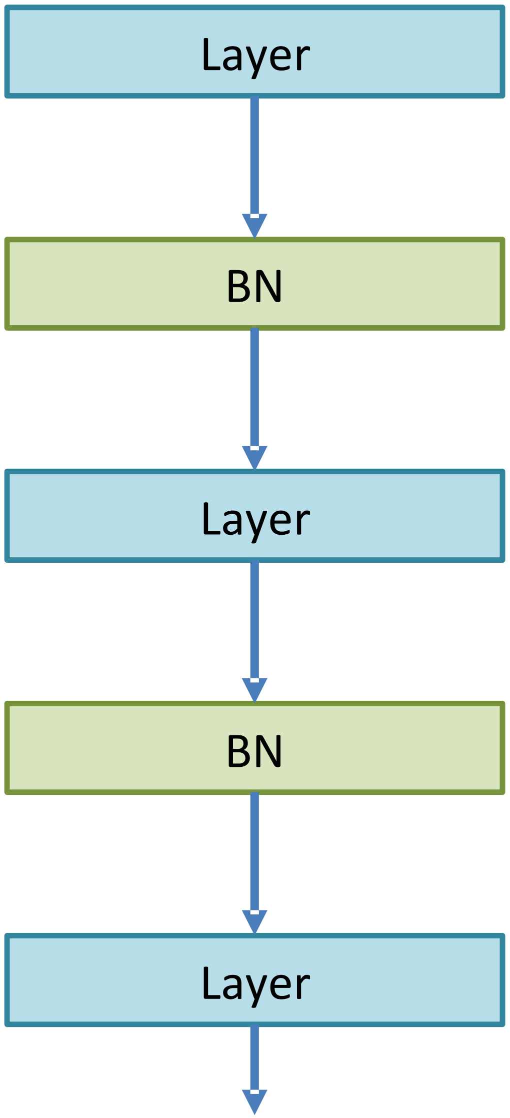
$$\widehat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$$

Where $\mu_{\mathcal{B}}$ and ${\sigma_{\mathcal{B}}}^2$ the mean and variance of the minibatch

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

Batch Normalisation



This way we constrain the layers and non-linear activations to operate in zero mean, unit variance input and this may not be desirable.

$$y_i = \gamma \widehat{x}_i + \beta$$
$$BN_{\gamma,\beta}(x_i)$$

$$\widehat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}} \qquad \gamma = \sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon} \qquad \beta = \mu_{\mathcal{B}}$$