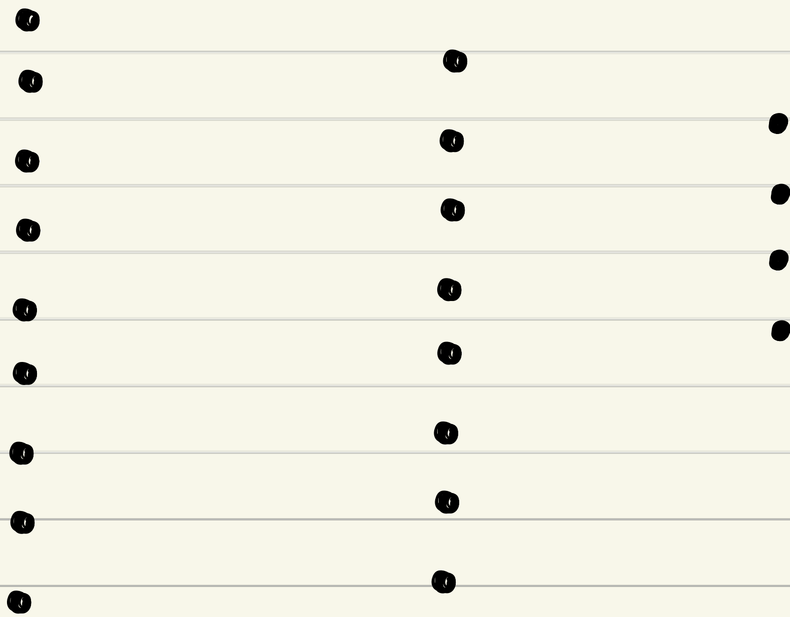
#1 Initialize the neural network

so we have → 9 input features
4 classification stages

for shallow neural network lets consider 8 hidden units



input = 9 hidden = 8 output = 4

weight matrix (W) = [9 x 8]

bias (b) = [1 x 8]

weight matrix (W) = [8 x 4]

bias (b) = [1 x 4]

- Understanding matrix multiplication (input to hidden layer)

weight matrix (W) = [9 x 8]

input features (X) = [620 x 9]

number of samples (n)

number of unique features

so $F(x) = \underbrace{X \cdot W}_{\text{shape}} + b$

620 x 8 + b[1 x 8]

the bias term will be broadcasted to match the shape

$F(x) = (620, 8)$ → output shape of input to hidden layer

- Understanding matrix multiplication (hidden to output layer)

$F(x) = (620, 8)$ → output shape of input to hidden layer (calculated previously)

↳ Z_{hidden}

weight matrix $(W) = [8 \times 4]$

bias $(b) = [1 \times 4]$

$$F(x) = Z_{hidden} \cdot W + b$$

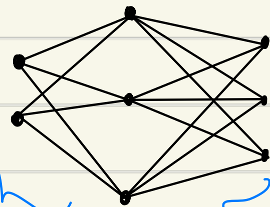
shape $(620, 8) \times (8 \times 4)$

$620 \times 4 + b[1 \times 4]$

$$F(x) = (620, 4)$$

= number of classification

#2 Forward Propogation



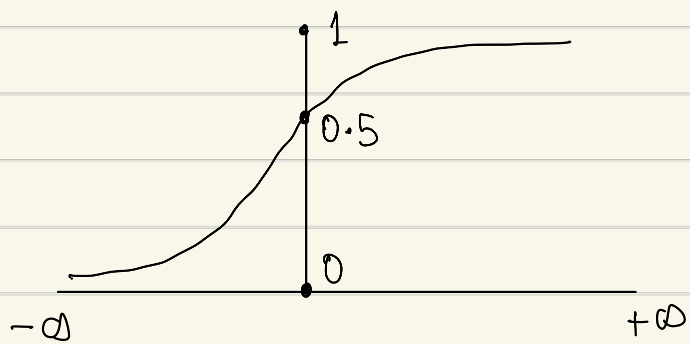
Input to Hidden layer	Hidden to Output layer
calculate	using activation A from previous layer
$Z = X \cdot W + b$	$Z = A \cdot W + b$
pass it to activation function	
$A = \text{ReLU}(0, Z)$	apply softmax function to convert logits to probablities
$\max(0, Z)$	basically converting numbers between the range of 0 and 1

- More about softmax

Softmax is an activation function applied to the last layer of neural network

$$\boxed{\sigma(z) = \frac{e^{z_i}}{\sum_{i}^{k} e^{z_j}}}$$

k: number of classes



- Cross Entropy Loss Function

Now our network predicted classes for our input data
What Next ?

check how correct is our neural network

∴ we use Cross Entropy Loss

number of classes

$$\boxed{H = - \sum_{i=k}^{n} \underbrace{p(x)}_{\text{the correct prediction in our dataset}} \underbrace{\log p(x)}_{\text{prediction our network made}}}$$

#3. Backpropogation

Previously we calculated errors using Cross Entropy Loss.
Now we need to backpropagate these errors to update weight and biases

- what is our goal in backpropogation?

→ calculate gradients of the loss wrt
1) weights in the hidden and output layers
2) biases in the hidden and output layers

→ take these gradients and update the weights and biases

1) output layer gradients

in here we calculate $\Delta output = (y\_pred - y\_true)$

→ shows us the direction in which correction is needed

2) hidden layer gradients

take output layer's error ($\Delta output$) and run it back through the network to understand each hidden neuron's contribution to error

in here we calculate Δ_{hidden}

$$\Delta_{hidden} = \left(\Delta_{output} \cdot W_{hidden\_output}\right) \times \left(z_{hidden} > 0\right)$$

only active neurons contribute to error → ∴ checks if ReLU activation is 0

3) calculate weight and bias gradients

a) Gradient for weight and bias updates in hidden to output layer

$$\nabla W_{hidden\_to\_output} = \underbrace{a_{hidden}}_{\text{activations in hidden layer}} \cdot \underbrace{\Delta_{output}}_{\text{output error}}$$

$$\nabla b_{hidden\_to\_output} = \sum \Delta_{output}$$

b) Gradient for weight and bias updates in hidden to input layer

$$\nabla W_{input\_to\_hidden} = \underbrace{X}_{\text{input features}} \cdot \underbrace{\Delta_{hidden}}_{\text{hidden neurons error}}$$

$$\nabla b_{hidden\_to\_output} = \sum \Delta_{hidden}$$

4) We use these gradients to update our trainable parameters

a) hidden to output layer

$$\text{new } W = \text{old } W - (\text{learning rate}) \times \nabla W_{\text{hidden\_to\_output}}$$

$$\text{new } b = \text{old } b - (\text{learning rate}) \times \nabla b_{\text{hidden\_to\_output}}$$

b) input to hidden layer

$$\text{new } W = \text{old } W - (\text{learning rate}) \times \nabla W_{\text{input\_to\_hidden}}$$

$$\text{new } b = \text{old } b - (\text{learning rate}) \times \nabla b_{\text{hidden\_to\_output}}$$

4. Stochastic Gradient Descent

1) Data shuffling

shuffle the dataset but why?

→ prevent model from learning order-specific patterns
→ now each batch represents the entire dataset randomly
→ mainly reduces bias and improves generalization

2) Mini batch processing

→ breaks large datasets into smaller and manageable chunks

3) Forward Propagation

→ input data flows through the network and it predicts labels

4) Loss Calculation

→ measure how far are the predictions from true labels

5) Compute Gradients

→ calculate gradients with the help of loss function for each trainable parameter

6) Update Parameters

→ use the calculated gradients and learning rate to update weights and biases