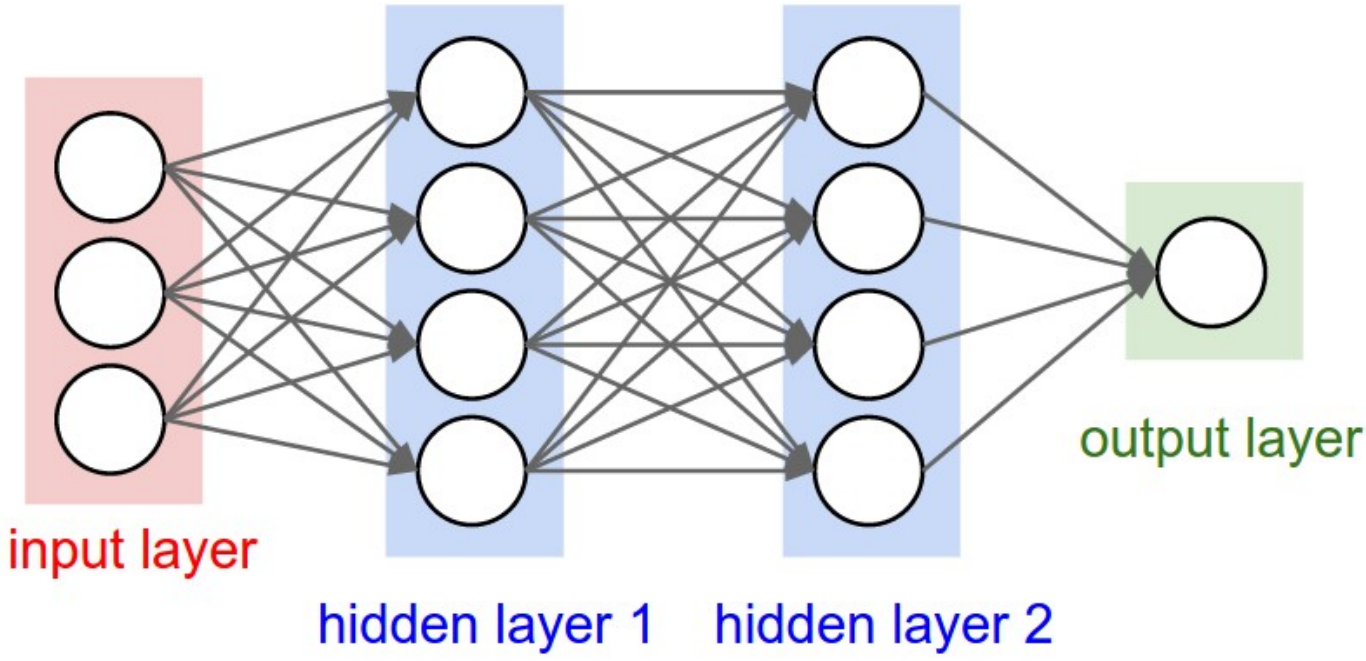
Neural Network :

- learns by taking inputs, mixing them with learned importance weights, and producing an output that improves with experience.
- many neurons connected together in layers.
- consists of nodes and connections between the nodes.



Activation Function :

- decides whether a neuron should activate

Without activation function

$$y = w_1 x_1 + w_2 x_2$$

no matter how many layers you add, the result is still linear

An activation function:

- takes the weighted sum from a neuron
- transforms it in a nonlinear way
- controls signal flowing forward

$$\text{output} = f(w \cdot x + b)$$

Weights decide how imp. each input is

input value

Bias decides how easy or hard it is to activate (handled internally)

Activation Functions

ReLU

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

- -ve → ignored
- +ve → pass through

Simple & fast

reduces vanishing gradients

→ A gradient tells you how much model should change to reduce error.

$$\text{gradient} = \frac{\partial \text{loss}}{\partial \text{weight}}$$

Sigmoid

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

Output : 0 - 1

Usecase: probability & binary classification

Gradients vanish for large

Tanh

$f(x) = \tanh(x)$

Output : -1 & 1

Chain Rule:

raw weighted sum, $z = wx + b$ → linear

activation function, $a = f(z)$ → non-linear

loss associated, $L = loss(a)$

Chain, $w \rightarrow z \rightarrow a \rightarrow L$

Gradient, $$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Question: Consider a single neuron NN defined as follows:

$$z = wx + b$$

$$a = ReLU(z)$$

where the loss for a single data point is given by the Mean Squared Error (MSE)

$$L = (a - y)^2$$

a) Derive the expression for the loss L in terms of w, b and x

$$z = wx + b$$

$$a = ReLU(z)$$

$$L = (a - y)^2$$

$$L = (ReLU(wx + b) - y)^2$$

(b) Compare the gradients w.r.t to weight and bias

$$\frac{\partial L}{\partial a} = 2(a-y) \qquad \frac{\partial a}{\partial z} = 1 \qquad \frac{\partial z}{\partial w} = x \qquad \frac{\partial z}{\partial b} = 1$$

Gradient w.r.t weight, $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}$

$= 2(a-y) \cdot x$

Gradient w.r.t bias, $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial b}$

$= 2(a-y)$

(c) Explain how these gradients are used to update the parameters w and b during gradient descent.

$$w_{new} = w - \eta \frac{\partial L}{\partial w}$$
$$b_{new} = b - \eta \frac{\partial L}{\partial b}$$

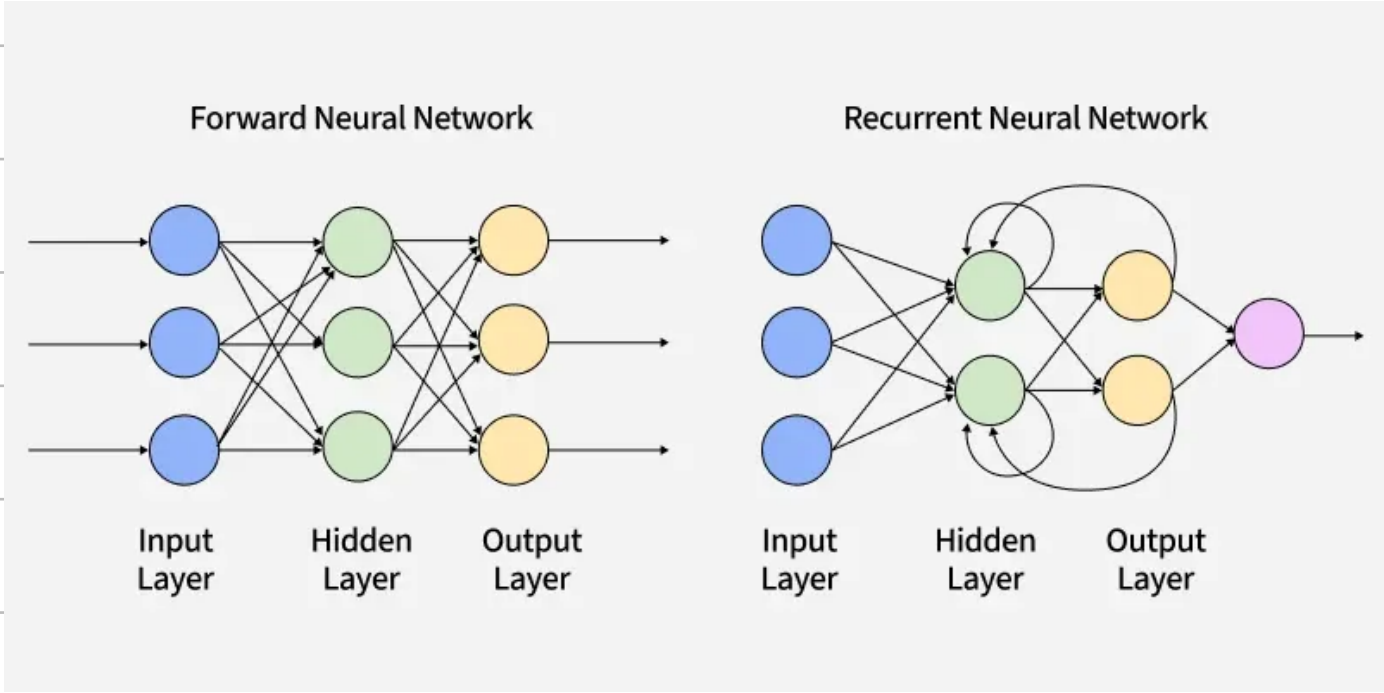
where η is the learning rate

Interpretation of a Gradient Value:

Consider a parameter θ with gradient:

$$\frac{\partial L}{\partial \theta}$$

Difference between Feed-Forward NN and Recurrent NN:

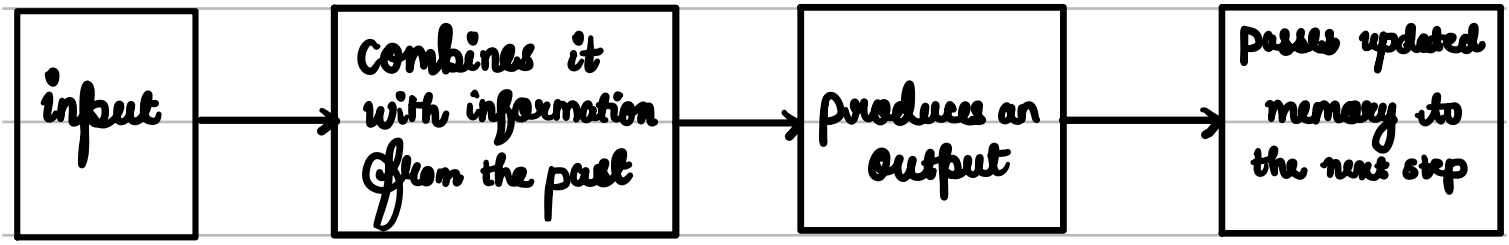


Sparse vs Dense Matrix:

Sparse	Dense
→ Mostly zeroes	→ Packed with meaningful values
→ One hot encoded vectors	→ NN weights
→ Low memory usage	→ High memory usage
→ DL inputs	→ DL weights
→ no semantic meaning	→ carries semantic meaning

Recurrent Neural Network :

- type of NN designed to work with sequences
- unlike standard neural network, it has memory.



Where RNNs are good at

→ Capturing temporal patterns

→ Learning trends & momentum

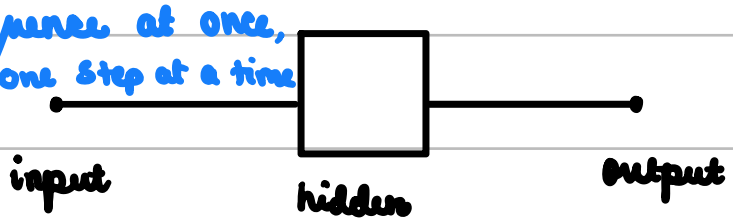
→ Modeling sequences with short-term dependencies

Where RNNs struggle

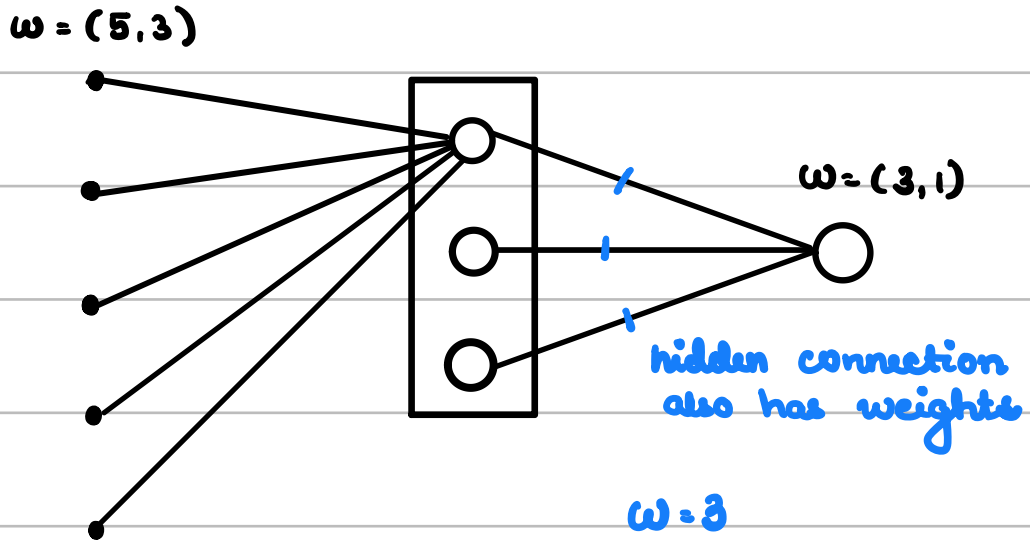
→ Vanishing gradient (they forget long term dependencies)

→ Over-smoothing (Predictions collapse toward averages)

RNN doesn't take the entire sequence at once, it takes one step at a time



Mechanism of vanilla RNN:



weights	$w = 5 \times 3$	$w = 3 \times 3$	$w = 3 \times 1$
biases		$b = 3$	$b = 1$

Forward Propagation :

- the step where inputs flow through the neural network to generate output using current weights and biases

Suppose you have :

- input vector x
- weights w
- bias b
- activation function : $f(\cdot)$

For one layer, $z = Wx + b$ (linear transformation)

Output, $O_1 = f(z)$

Output a becomes the input for next

Input			Output
x_{11}	x_{12}	x_{13}	1
x_{21}	x_{22}	x_{23}	0
x_{31}	x_{32}	x_{33}	0

Let x_{11} be 5 dimensional vector $[1\ 0\ 0\ 0\ 0]$ by One hot encoding

(n features)

Suppose you want 3 neurons. (m neurons)

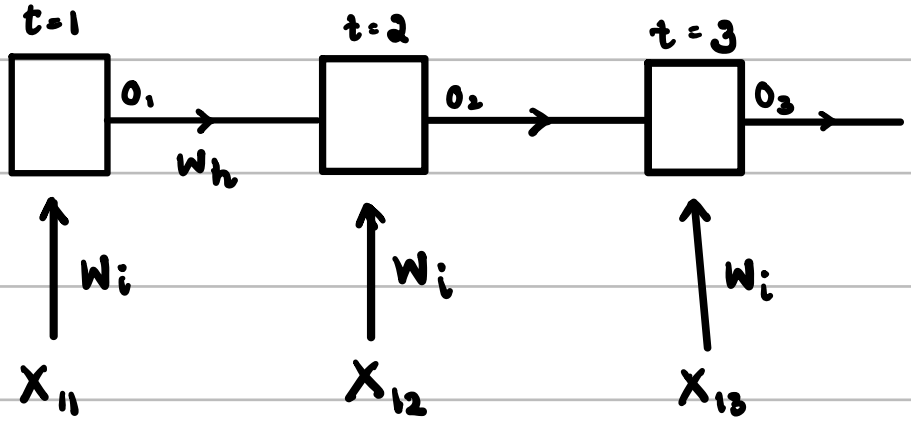
Two weight matrices:

(1) W_i for input → hidden

(2) W_h for hidden → hidden

} Weight dimensions are chosen to match your architectural choice

Activation function $f(x_1 w_h)$ $f(x_{12} w_i + o_1 w_h)$ $f(x_{13} w_i + o_2 w_h)$



$x_{11} \times w_i$ $\quad$ $x_{12} \times w_i + o_1 \times w_h$

(1,5) (5x3) $\quad$ (1,5) (5,3) $\quad$ (1,3) (3x3)

$o_1 = (1 \times 3)$ $\quad$ $(1 \times 3) \quad (1 \times 3)$

$+$

$o_2 = (1 \times 3)$

To make the activation function consistent even at timestep 1, we need:

$$o_1 = f(x_1 w_i + o_0 w_h)$$

Common choices for o_0:

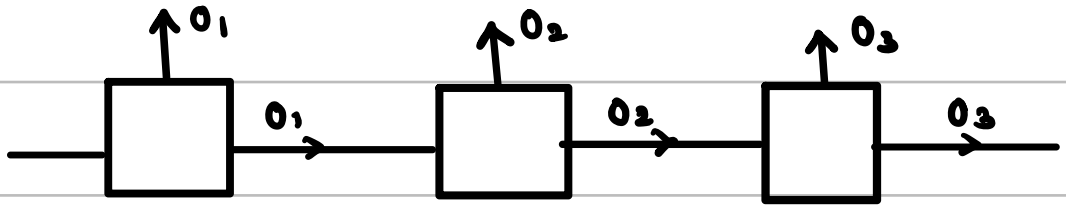
- Zero initialization (most common)
- Random (rare)
- Learned initial state

NOTE :

We use same weight matrix for each step. So, at every step we are updating w_i and w_h

Q) Is there a way to check output at each node?

If you set `return_sequence = True`, you can also get output at each node.



Q) What if input size is not consistent ?

We add extra values to data so everything has the same size. This technique is called

Padding

Loss Function :

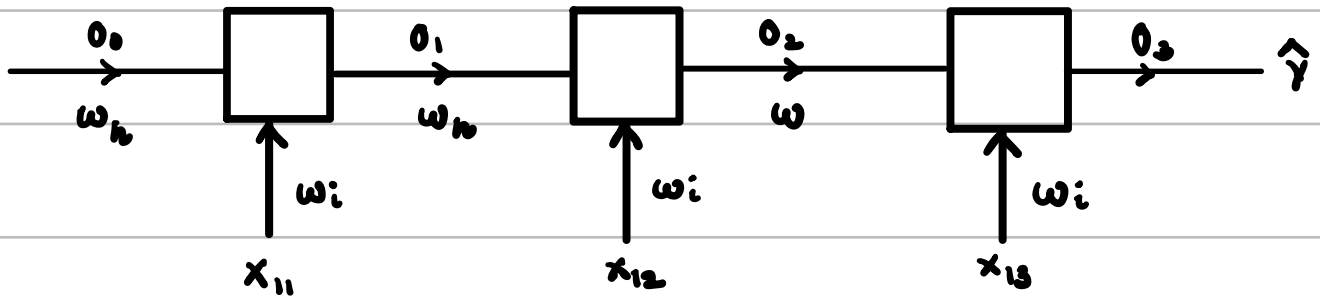
a mathematical function that measures how wrong a model's prediction is compared to the true value.

Prediction → Loss → Gradient → Weight update

Name	Simple Definition	Formula	Where It Is Used
Mean Squared Error (MSE)	Average squared difference between actual and predicted values.	$L = \frac{1}{n}\sum(y - \hat{y})^2$	Regression
Mean Absolute Error (MAE)	Average absolute difference between actual and predicted values.	$L = \frac{1}{n}\sum\lvert y - \hat{y}\rvert$	Robust regression
Huber Loss	Combines MSE and MAE: Quadratic for small errors, linear for large errors.	$L = \frac{1}{2}(y - \hat{y})^2 \text{ if } \lvert y - \hat{y}\rvert \leq \delta \text{ else } \delta(\lvert y - \hat{y}\rvert - \frac{1}{2}\delta)$	Robust regression
Binary Cross Entropy (BCE)	Negative log likelihood for binary outcomes.	$L = -[y\log(\hat{y}) + (1-y)\log(1-\hat{y})]$	Binary classification
Categorical Cross Entropy	Negative log likelihood for multi-class outcomes.	$L = -\sum_k y_k \log(\hat{y}_k)$	Multi-class classification
Hinge Loss	Margin-based classification loss.	$L = \max(0, 1 - y\hat{y})$	Support Vector Machines
Negative Log Likelihood (NLL)	Log-based probabilistic loss.	$L = -\log P(y \mid x)$	Probabilistic models

Backpropagation :

- A systematic way to compute gradients of the loss with respect to all weights using the chain rule



$$o_1 = f(x_{11}\omega_i + o_0\omega_h)$$

$$o_2 = f(x_{12}\omega_i + o_1\omega_h)$$

$$o_3 = f(x_{13}\omega_i + o_2\omega_h)$$

$$\hat{y} = \sim(o_3\omega_0)$$ → Prediction using sigmoid function

$$\text{loss} = y - \hat{y}$$

$$= -\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$$

Weights are calculated to minimize loss, so we use gradient descent.

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

$$w_n = w_n - \eta \frac{\partial L}{\partial w_n}$$

$$w_0 = w_0 - \eta \frac{\partial L}{\partial w_0}$$

Since we are doing backpropagation, we start with $\hat{y}$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_0}$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{\partial}{\partial \hat{y}} \left[y \log(\hat{y}) + (1-y) \log(1-\hat{y}) \right]$$

$$= -\left[\frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})} \right]$$

$$= -\frac{y - y\hat{y} - \hat{y} + y\hat{y}}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial w_0} = \frac{\partial}{\partial w_0} \sigma(o_3 w_0)$$

$$= \frac{\partial}{\partial w_0} \left(\frac{1}{1 + e^{-(o_3 w_0)}} \right)$$

$$= \frac{-1}{(1+e^{-o_3 w_0})^2} \frac{\partial}{\partial w_0} (1 + e^{-o_3 w_0})$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$= \frac{-1}{(1+e^{-o_3 w_0})^2} \frac{\partial}{\partial w_0}(e^{-o_3 w_0})$$

$$= \frac{-1}{(1+e^{-o_3 w_0})^2} (e^{-o_3 w_0}) \frac{\partial}{\partial w_0}(-o_3 w_0)$$

$$= \frac{e^{o_3 w_0}}{(1+e^{-o_3 w_0})^2} \times (-o_3)$$

$$= \frac{-o_3 e^{o_3 w_0}}{(1+e^{-o_3 w_0})^2}$$

$$= -o_3 \frac{1}{1+e^{-o_3 w_0}} \times \frac{e^{o_3 w_0}}{(1+e^{-o_3 w_0})^2}$$

$$= -o_3 \hat{y}(1-\hat{y})$$

If $\hat{y} = \frac{1}{1+e^{-o_3 w_0}}$

$$1 - \hat{y} = 1 - \frac{1}{1+e^{-o_3 w_0}}$$

$$= \frac{1 + e^{-o_3 w_0} - 1}{1+e^{-o_3 w_0}}$$

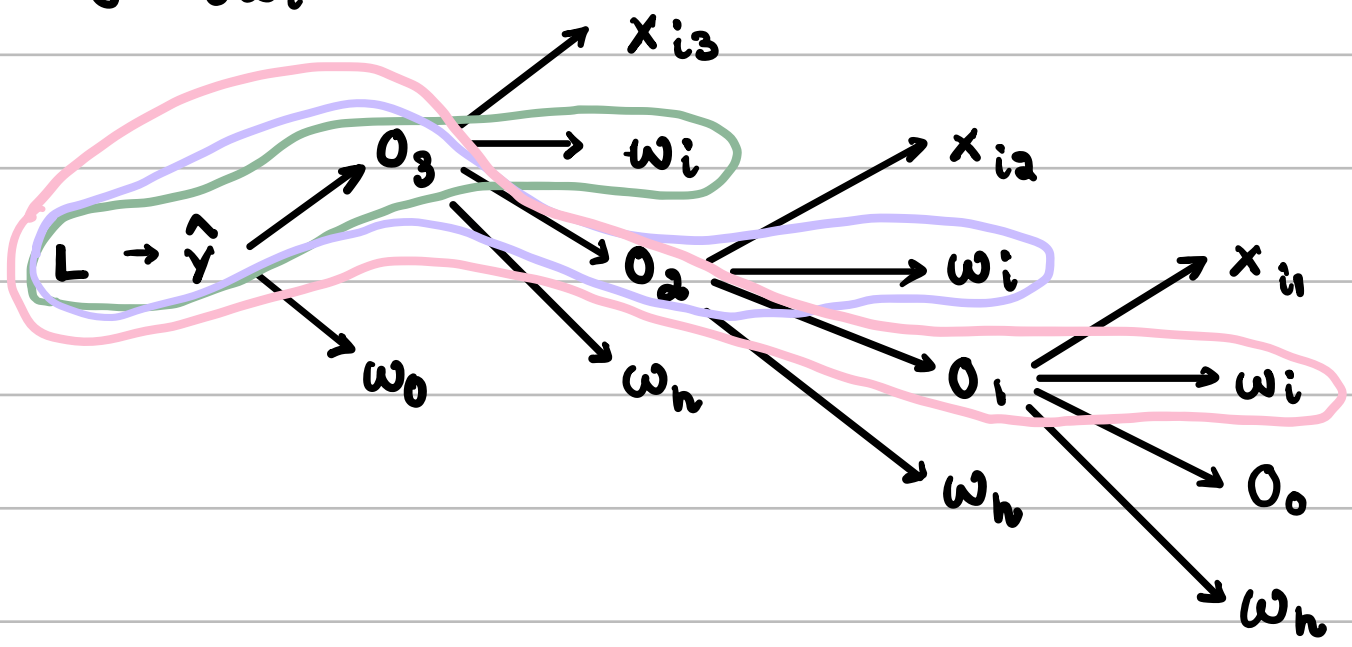
$$= \frac{e^{-o_3 w_0}}{1+e^{-o_3 w_0}}$$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_0}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \times -o_3 \hat{y}(1-\hat{y})$$

$$= -o_3 (\hat{y} - y)$$

Finding $\frac{\partial L}{\partial w_i}$ by defining the independent paths



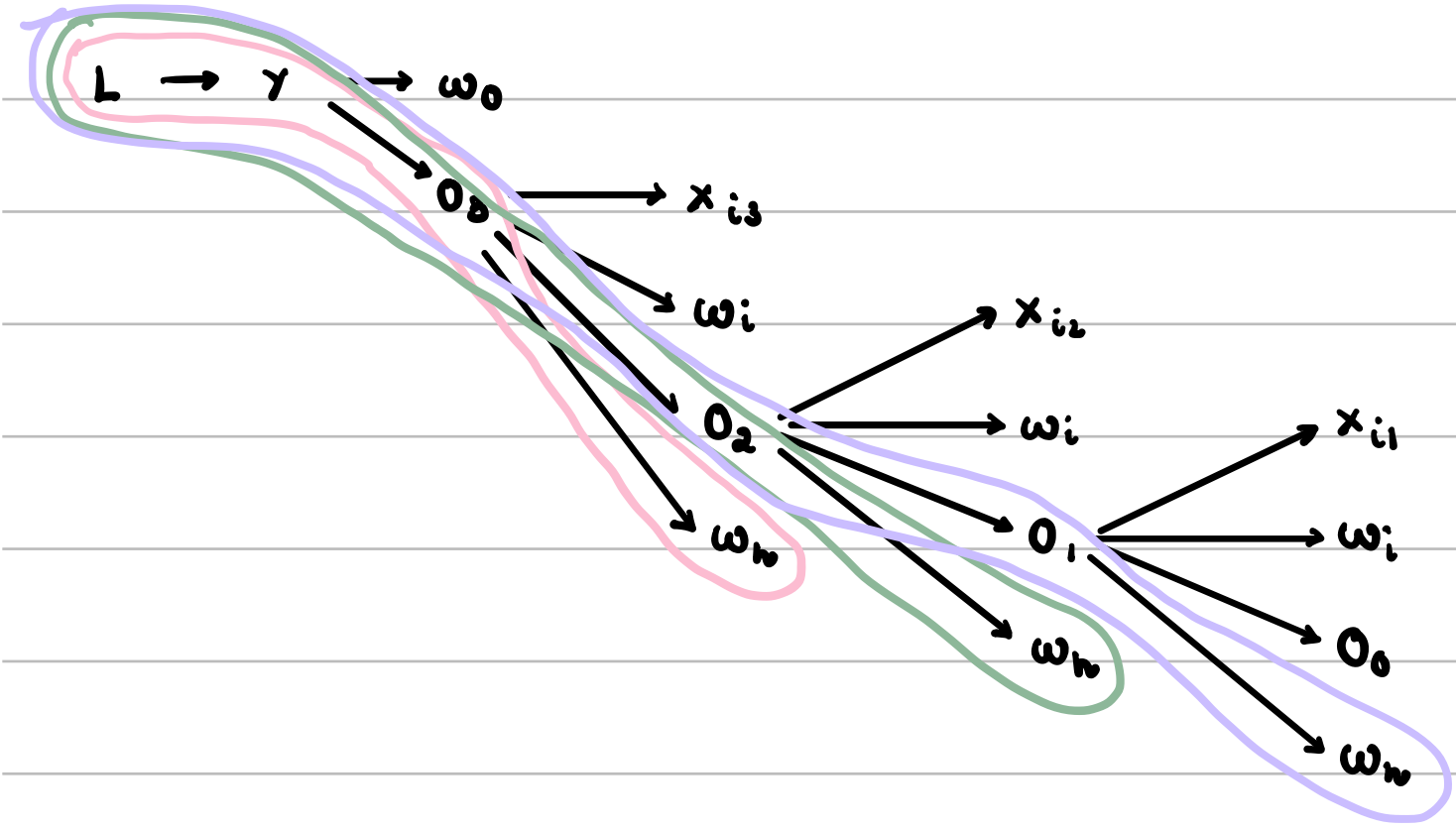
$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial w_i} +$$

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_i}$$

For n timesteps

$$\frac{\partial L}{\partial w_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial o_j} * \frac{\partial o_j}{\partial w_i}$$

Finding $\frac{\partial L}{\partial w_n}$,



$$\frac{\partial L}{\partial w_n} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial w_n} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial w_n} +$$

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_n}$$

For n steps,

$$\frac{\partial L}{\partial w_n} = \sum_{i=1}^{n} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_i} \frac{\partial o_i}{\partial w_n}$$

RNN Summary :

Forward Propagation

Compute hidden states

$$O_1 = f(x_1 W_i + O_0 W_h)$$

$$O_2 = f(x_2 W_i + O_1 W_h)$$

$$O_3 = f(x_3 W_i + O_2 W_h)$$

Compute final prediction

$$\hat{y} = \sim (O_3 W_0)$$

Compute loss

$$L = -\left[y \log(\hat{y}) + (1-y) \log(1-\hat{y}) \right]$$

Back Propagation

Output weights : $\frac{\partial L}{\partial W_0}$

Hidden weights : $\frac{\partial L}{\partial W_h}$

Input weights : $\frac{\partial L}{\partial W_i}$

Update Weights

Choose a learning rate η

$$W_0 \leftarrow W_0 - \eta \frac{\partial L}{\partial W_0}$$

$$W_h \leftarrow W_h - \eta \frac{\partial L}{\partial W_h}$$

$$W_i \leftarrow W_i - \eta \frac{\partial L}{\partial W_i}$$

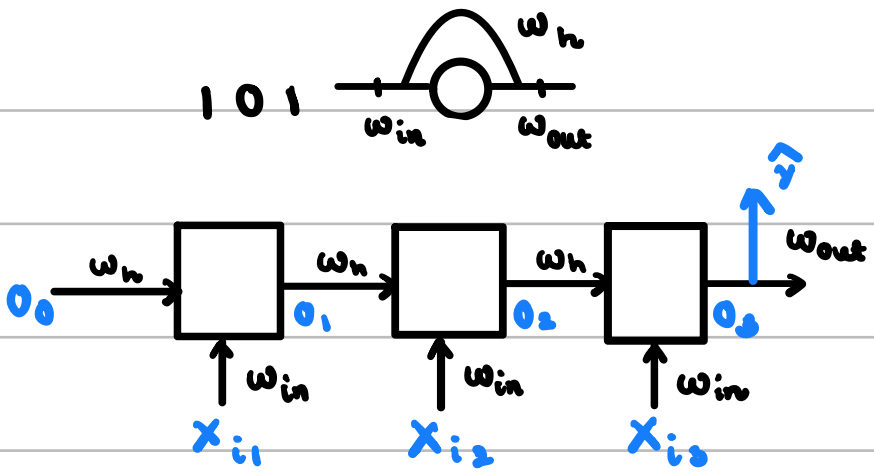
Repeat

- → Do this for many sequences
- → Over many epochs
- → Loss goes down
- → Predictions improve

Drawback of RNNs :

(1) Vanishing and Exploding Gradient :

inp	out
1 0 1	1
0 0 1	0
0 0 0	0
1 1 1	1



Since we are doing back propagation, we update weights to minimize the loss.

For 3 timesteps,

$$\frac{\partial L}{\partial w_{in}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial w_{in}} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial w_{in}} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w}$$

short term memory

long term memory

For n timesteps, long term memory will look like

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_n} \frac{\partial o_n}{\partial o_{n-1}} \cdots\cdots \frac{\partial o_1}{\partial w_{in}}$$

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_n} \prod_{t=2}^{n} \left(\frac{\partial o_t}{\partial o_{t-1}} \right) \frac{\partial o_1}{\partial w_{in}}$$

Here $o_t = \tanh\left(x_{it} w_{in} + o_{t-1} w_h \right)$

$$\frac{\partial o_t}{\partial o_{t-1}} = \tanh'\left(x_{it} w_{in} + o_{t-1} w_h \right) w_h$$

Substituting,

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_n} \prod_{t=2}^{n} \underbrace{\tanh'(x_{it} w_{in} + o_{t-1} w_h)}_{0\text{-}1} \underbrace{w_h}_{0\text{-}1} \frac{\partial o_1}{\partial w_{in}}$$

$$\underbrace{\qquad\qquad}_{\simeq 0}$$

As timesteps increase, repeated multiplication during backdrop causes gradients to decay exponentially, preventing the model from learning long-term dependencies

(2) Sequencial Computation :

$$x_1 \longrightarrow x_2 \longrightarrow x_3 \longrightarrow \ldots\ldots$$

each step depends on the previous one.

So,

- Cannot fully parallize
- Training is slow on GPU