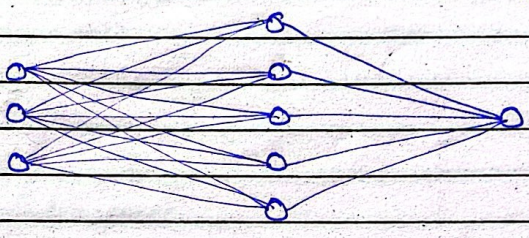
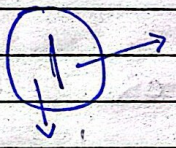
DATE: __/__/____

Artifical Neural Network

An ANN perform many operation in parallel and arranged in layers.



Preceptron :



∴ Each connection from preceptron to other has weight w_j

∴ Bais $X_0 = 1$

- A neuron with no activation are called Linear Unit.

Layers types :

Convolution Deconvolution Fully Connected (FC) LSTM
Recurent [redacted] pooling Concat Batch Norm
Dense layers also called FC.

Activation Functions :

Input/Hidden	Relu, LeakyReLU, tanh, Sigmoid
output	Softmax, Sigmoid
	Multiclass Binary

DATE: __/__/____

Advantage of ANN	Disadvantage of ANN
• The ability to learn Non-linear complex relationship.	• Lack of rules find optimality based on trial and error.
• They are good at organizing, preprocessing and categorizing data.	• It is also blackbox in nature Benge layer How they work is mystery?
• Ability to generalize Data.	• Every data feed should be numerical.

Example 1 :-

Activation Sigmoid $= \frac{1}{1+e^{-y}}$

OR ReLU $\begin{cases} 1 & \text{if } y>0 \\ 0 & \text{else} \end{cases}$

OR

0 0	0
0 1	1
1 0	1
1 1	1

$y = x_0 + x_1 Q_1 + x_2 Q_2$

$y = 1 + 1(0) + 1(0)$

✓ $y = -1 + 1(0) + 1(0) = -1$

$y = -1 + 1(0) + 1(1)$

✓ $y = -1 + 1(0) + 2(1) = 1$

$y = -1 + 1(1) + 2(0)$

✓ $y = -1 + 2(1) + 2(0) = 1$

✓ $y = -1 + 2(1) + 2(1) = 1$

$x_0 = -1$, $x_1 = 2$, $x_2 = 2$

DATE: __/__/____

E2. Preceptron Rule for XOR Gate

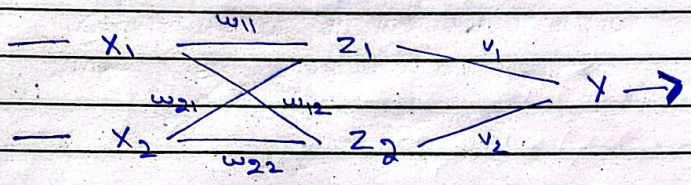
X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$y = x_1\bar{x}_2 + \bar{x}_2 x_2$

$z_1 = x_1 \cdot \bar{x}_2$ ∴ '.' AND

$z_2 = \bar{x}_1 \cdot x_2$

$y = z_1$ OR z_2



$z_1 = x_1\bar{x}_2$ $\alpha = 1.5$

x_1	x_2	$\bar{x}_2$	z_1
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

$AF = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & y < 0 \end{cases}$

$y = w_1x_1 + w_2x_2$

✓ $y = 1.0 + 1.0 = 0$

X $y = 1.0 + 1.1 = 1$

$w_{21} = old + \alpha(T - O)x_i$
$= 1 + 1.5(0 - 1)1$
$= -0.5$

$y = 1x_1 + (-0.5)x_2$

✓ $y = 1.0 + (-0.5)1 = -0.5$

✓ $y = 1.1 + (-0.5)(1) = 1$

✓ $y = 1.1 + (-0.5)(1) = 0.5$

DATE : __/__/____

$z_2 = \bar{x}_1 x_2$

x_1	x_2	$\bar{x}_1$	z_2
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

$y = 1*0 + 1*0 = 0$

$y = 1*0 + 1*1 = 1$

$y = 1*1 + 1*0 = 1$

$w_{12} = 1 + 1.5(0-1)*1 = 0.5$

$w_{22} = 1 + 1.5(0-1)*0 = 1$

$y = (0.5)*0 + 1*0 = -0.5$

$y = (-0.5)*1 + 1*1 = 0.5$

$y = z_1$ OR z_2

x	y	z_1	z_2	Y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

$\alpha = 1.5 \qquad v_1 = v_2 = 1$

$y_{in} = 1\times0 + 1\times0 = 0$

$y_{in} = 1\times0 + 1\times1 = 1$

$y_{in} = 1\times1 + 1\times0 = 1$

$y_{in} = 1\times0 + 1\times0 = 0$

$w_{11} = 1 \qquad w_{21} = -0.5$

$w_{12} = -0.5 \qquad w_{22} = 1$

$v_1 = 1 \qquad v_2 = 1$

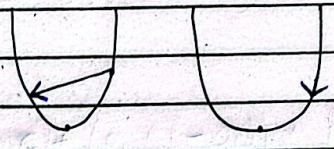
DATE: __/__/____

1.

(i) Hyper parameters

Learning Rate (α)

This show how much longer jump is to be made to reach min loss.



Large learning Rate

Small learning Rate.

★ If not set properly take huge time to optimize.

Minibatch Convert data to small than pass for each epoch.

Learning depends on data in one epoch. If Batch size is large more memory is require.

⇒ If batch size is too small with noisey data the convergence will be slow

(ii) Optimiser

Adam

Combine Root mean Square and Adaptive Gradient Algorithm.

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\Delta J(\theta_t)$$

$$v_{t+2} = \beta_2 v_t + (1-\beta_2)\Delta J(\theta_t)$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^{t+1}} \qquad \hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$$

$$\theta_{t+1} = \theta - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} m_{t+1}$$

SGD

Updates parameter (θ) opposite to gradient

$$\theta_{t+1} = \theta_t - \alpha \Delta J(\theta_t)$$

- Gradient direction is opposite to min loss or optimization.

DATE: __/__/____

★ Beta 1, Beta 2 Parameters for Adam. and Epsilon.

★ Adam maintain seprate learning rate for each parameter and update dynamically based on past gradient.

★ Adam works for wide range of Deep learning tasks.

(iii) Regularization

Dropout — Easystopping — Batch Normalization

⇒ Turn off some connection to learn less

⇒ Learn less to avoid overfitting

⇒ Adjust data to have mean of 0 and std of 1

⇒ In learn help avoid noise which can result in more error is test.

⇒ The mode keep trace of error and which its not improving it stop training.

⇒ avoid overfitting.

Example solved for Forward propogation and Backward propogation

b1 = 0.25 b2 = 0.25

(1) Forward

∴ $w_i = 0.i$

x_1 w_1 x_2 w_3 b_1

$H_1 = 0.1 \times 0.1 + 0.5 \times 0.3 + 0.25 = 0.41 \Rightarrow \frac{1}{1+e^{-(0.41)}} = 0.60$

$H_2 = 0.1 \times 0.2 + 0.5 \times 0.4 + 0.25 = 0.47 \Rightarrow \frac{1}{1+e^{-(0.47)}} = 0.61$

w_2 w_4 b_2 +0.25

h_1 w_5 h_2 w_6

$O_1 = 0.60 * 0.5 + 0.61 * 0.6 + 1.01 \Rightarrow \frac{1}{1+e^{-(1.01)}} = 0.73$

$O_2 = 0.60 * 0.7 + 0.61 * 0.8 + 0.25 = 1.26 \Rightarrow \frac{1}{1+e^{-(1.26)}} = 0.77$

DATE: __/__/____

(2) Error y_org y_Pred

$E_1 = \frac{1}{2}(0.05 - 0.73492)^2 = 0.23456$

n=2 $E_2 = \frac{1}{2}(0.95 - 0.77955)^2 = 0.01452$

$E_t = 0.24908$

(3) Backpropogation $\alpha = 0.6$

$dw_5^B = (o_1\ y\_Pred - y\_true) * (o_1(1-o_1)) * H_1$

$= (0.73 - 0.05)(0.73(1-0.73) * 0.60$

$= 0.08020$ H_2

$dw_6^B = (0.68492)(0.19880)(0.61) = 0.08211$

$dw_7^B = (0.77955 - 0.95)(0.9(1-0.9) * 0.60$

$= -0.01760$ H_1

$dw_8^B = (-0.17044) * (0.17184) * 0.60108 = -0.01802$

$w_5' = w_5 - \alpha(\partial w_5)$

$= 0.5 - 0.6(0.08020) = 0.45187$ ↓

$w_6' = 0.6 - 0.6(0.08211) = 0.55073$ ↓

$w_7' = 0.7 - 0.6(-0.01760) = 0.71056$ ↑

$w_8' = 0.6 + 0.6(-0.081) = 0.81081$ ↑

$dw_1^{E_1} = (o_1 - t_1) \times w_5 \times [h_1 \times (1-h_1) \times X_1]$ $\times (o_1(1-o_1))$

$= 0.68492 \times 0.198 \times 0.5 \times 0.2397 * 0.1 = 0.00159$

$\partial w_1^{E_2} = (-0.17044) \times (0.17184) * 0.7 \times 0.23978 \times 0.1 = (-0.0049)$

$\partial w_1 = 0.00110$ (sume E_1 and E_2)

$= 0.3$
0.6

$w_1' = w_1 - \alpha(\partial w_1)$

$= 0.1 - 0.6(0.00110) = 0.09933$ ↓

$= 0.0105$
83

$w_2' = 0.19919$ ↓

$w_3 = 0.29667$ ↓ $(0.00812 + 0.000245)$

$w_4 = 0.39547$ ↓

$= 0.2936$

This is for one sample movement forward and updating weights backwards.

DATE: __/__/____

Convolution Operations

Normal Convolution

$6\times6\times3$ (Image) $\quad 3\times3\times3$ (filter) $\quad \begin{pmatrix} \text{no of filters} \\ 5 \end{pmatrix} = 4\times4\times5$

$S=1 \quad p=0$

$n=1 \quad f=3$

$\left[\frac{n+2p-f}{p}+1\right] = 4$

Computation Each $3\times3\times3$ is Multi to each $6\times6\times3$

Compute cost = # Filter para x # filter position x No of filters

$2160 = 3\times3\times3 \times 4\times4\times5$

Depth-Wise Seprable Convolution

Depth wise → Point wise

- Seprate Each channel apply filter.

$6\times6 * 3\times3 = 4\times4$

$6\times6 * 3\times3 = 4\times4$

$6\times6 * 3\times3 = 4\times4$

stack them

$CS = 3\times3 \times 4\times4 \times 3$

$= 432$

$4\times4\times3 * \underbrace{1\times1\times3}_{5} = 4\times4\times5$

$CS = 1\times1\times3 \times 4\times4\times5$

$= 240$

$\Sigma CS = 432+240 = 672$

But Not unlike Normal thes 4×4 are seprate has no interlink so we use Pointwise conv after Depth wise.

Convolutionary Neural Networks

Formulas :

1. Convolution layer

No of param = $[(fw \times fh \times depth)+1] \times$ No of filter

2. Fully Connected (Dense)

No of param = $(Input + 1) \times$ output size

Note :: for Input (Polling) layer their is no learning parameters but the activation is length x width x height.

3. Activation Size

AS = height x width x channels

Solution :

for conv 1 filter size (5,5) stride (1)

so, No of param = $[(5 \times 5 \times 3)+1] \times 8 = 608$

3 for No of channels of Input Image

8 for No of filters to be applied to Image.

DATE: __/__/____

For Conv2 filter (5,5) stride (1)

So, No of param = [(5×5×8)+1] × 16 = 3216

8 for previous layer depth

16 for filter applied onto them.

For Conv2

Activation size = 10×10×16 = 1600

For Conv1

Activation size = 28×28×8 = 6272

For Pooling 1

Activation size = 14×14×8 = 1568

For FC

Activation size = 120×1 = 120

For [redacted] FC

No of param = (Input + 1) × output

= (400+1)×120 = 48120

For output (Softmax)

No of param = (84+1) × 10

= 850

Day / Date ANN

Less Training Data → Transfer Learning
→ Data Augmentation (generate more data)
GANs Generative Adversarial Neural Network.

Auto Anotation
↳ (Media pipe) (Yolo v8)

Small data → handcrafted features
↓
complex Methods
SHIFT, HOG

More Data
↓
General feature
Deep learning
RNN, CNN, LSTM.

Image classification
1 Image 1 Label

Classification localization.
1 Image 1 object

Detection
1 Image Multipe object Multiple classes

Probility
y = [Foreground/background, x, y, height, width, class1, class2]

How to compute loss?
[ ] − [ ] = [ ]
y_true − y_pred

Find all or any one landmark then uses formulas to generate any missing to create 3D mesh.

visibility Probability
[p, (x1, y1), visibility, landmarks] → landmark coordinates, landmarker coordinate,
p, (x2, y2)
Sliding windows (Mediapipe)

Prince

Day / Date

Sliding Windows

Fixed — To find initial spot of space where change of object

Random. — Then introduced random size to get exact size of y true and y pred. — object are not of same size can be one small big by camera view.

Mediapipe

Background removal.

Any change in background / Masking will also not work. Intensity.

FC to Convolution ? So we adopted pointwise Convolution.
These convolution output won't go into linear equation
No linearity equation.

Conv → y=mx+c

conv → conv → Predict (pixels)
conv

⇒ Convolution is better in feature sharing more good.

Yolo Algorithm

⇒ convert each image into grid.
⇒ Make y for each grid. (as before)
⇒ Pass whole image you only look once
⇒ output (n x n x 8) → y length
grid numbers

Loss compilation

Prediction — bound box

y_True − y_pred = loss — Intersection over Union

Prince

1.

In You Only Look Once there are several anchor boxes also known defined as prior boxes or default boxes.

V1 — 2 anchor boxes, No grid formation (single scale)

V2 — 5 anchor boxes, 13×13 grid (at each scale)

V3 — 3 anchor boxes (at 3 different scales) 13×13, 26×26, 52×52 grids (at each scale)

V4 — Same as V3 Mosaic Training combine multiple grids

V5 — same as V4 and improved anchor box initilization.

V6 — Introduced "anchor free" approch uses key point Detection and Non maximum seprsation

V7 — Introduced "Auto Assign Labels" "dual-teaching" In dual-teacher model V7 learn from 2 teacher one is rigid and other is ease in such manner knowledge distilled and learning is consistance.

DATE : __/__/__

V8

Same as previous models

- Anchor free
- 16x16, 32x32, 64x64 grid (at each scale)
- Mosaic Training (combine multiple grid)
- Improved auto assign label (IOU)

Spatial attention

1. Apply Average Polling and Max polling along channel axis and concat them.
2. On concat feature descriptor you apply Conv layer. which encodes where to emphasize or supress.
3. Then aggaregate chanel info into 2D map by using two polling operations.

$$Ms(F) = \sigma(f_{7\times7}([AvgPool(F), MaxPool(F)]))$$
$$Ms(F) = \sigma(f_{7\times7}([f_{savg}, f_{smax}])$$

σ => Sigmoid function $f_{7\times7}$ => Conv operation with filter 7x7

New use cases :- Instance segmentation.
Keypoint Detection.

2.

Region Based Conv Neural Network family

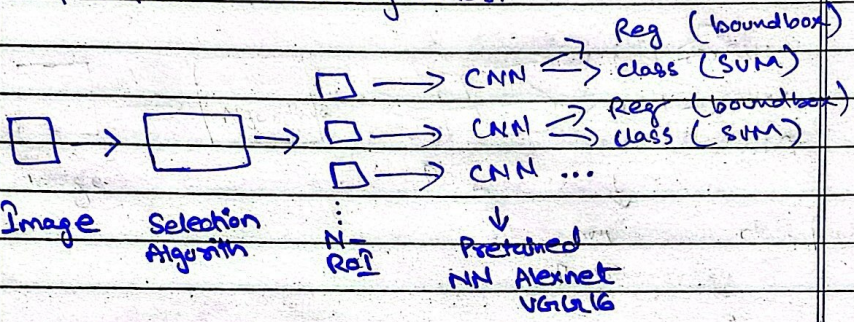
RCNN uses selective search algorithm to propose Region of Interest (RoI) and then uses CNN to classify each RoI.

DATE: __/__/____

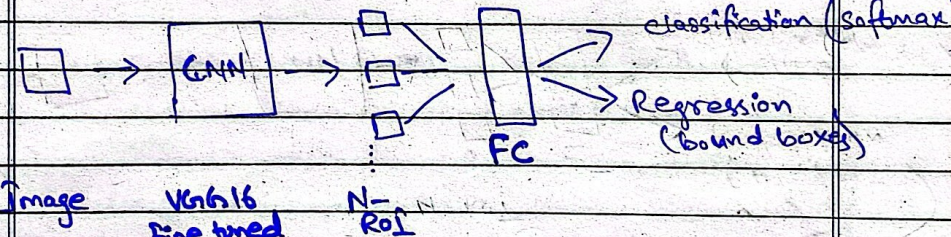
Fast RCNN uses shared convolution layer to whole image and all RoI rather then dealing with each seperately (Reduces computational time)

Faster RCNN uses Region Proposed Networks (RPN) to generate RoI which is faster than selective search algorithm. These networks also uses feature map to predict bounding boxes.

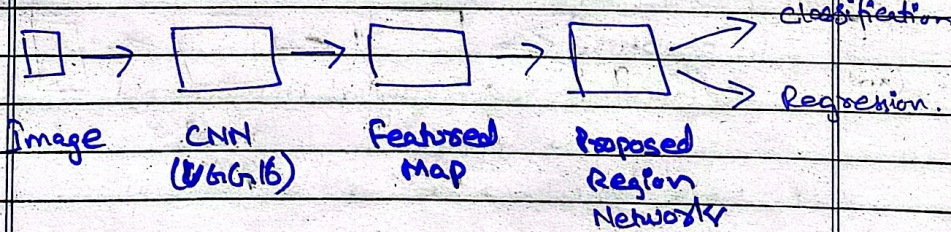
RCNN



Fast RCNN



Faster RCNN



Sliding Windows in CNN:

9 Anchors of different scales 1/4, 1/8, 1/16 are applied to featured map. Then fed to proposed Region Networks.

DATE: __/__/____

Non-Maximum Suppression:

On ouput of RPN it is applied to remove duplication of detection.

Note:: Bounded Box with max confidence is choosen.

3.

Key Differences:

1. RCNN uses selective search Algorithm while Yolo uses single neural network to predict bounding boxes.
2. RCNN has two stage approach (Region proposal and classification). YOLO is single stage.
3. RCNN prioritize accuracy over speed while YOLO prioritized speed over accuracy.
4. YOLO is used on live video stream while RCNN benefits for high accurate decision making on regions.

In structure

5. RCNN has Region Proposal Network and RoI pooling extra layer.
   While Yolo uses one grid cell generation.

Handle any size of token length as Input?
we used chunking (simple trim)

- Attention Mechanisms.
- Retrieval Augumented Generation.
- Recurrent Neural Network.

Embedding Layer(input_dim, output_dim, initilizer='uniform', regularization, Constraints)

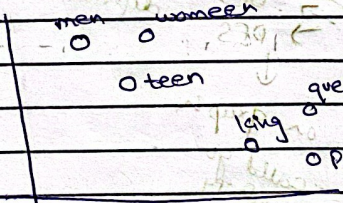
⇒ Categorical Data can not be passed to neural network direct.

⇒ Encoding scheme fails when number of words so vector size of each word become very long and it does not caputure semantic in sentences.

⇒ Embedding layer is a mapping b/w categorical data and its vector sepresentation.

⇒ The goal of embedding layer is to arrange embedding in a vector space such that similar words are together.

⇒ Not just that if apply operation on these vectors you will get another vector in dimension.

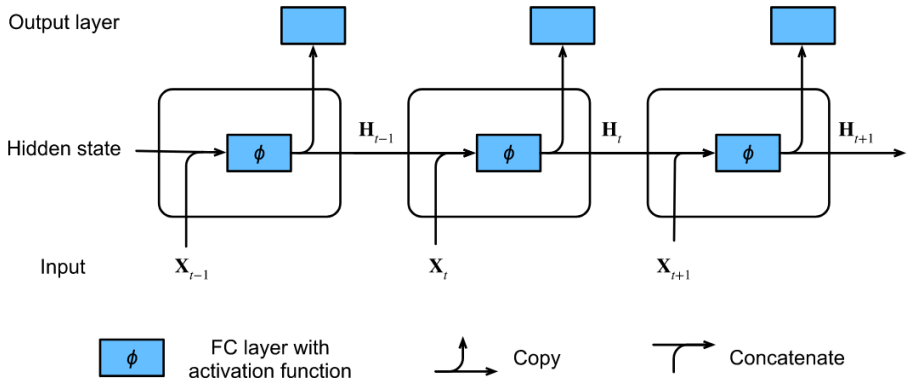


men = [.....]
king = [.....]
same size

King − men + women = queen

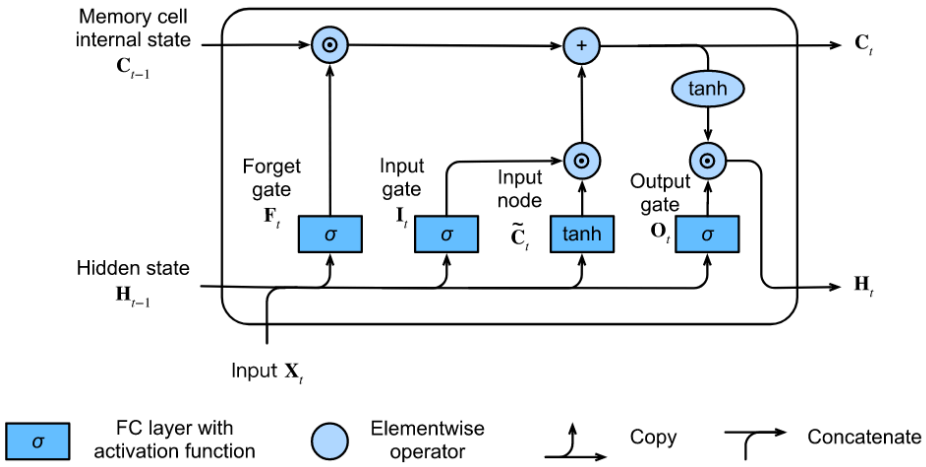
- This is same as game we play in child-hood king is to men and women is to ____? queen Relationship

- All LLMs now a days use this Word embedding as base.



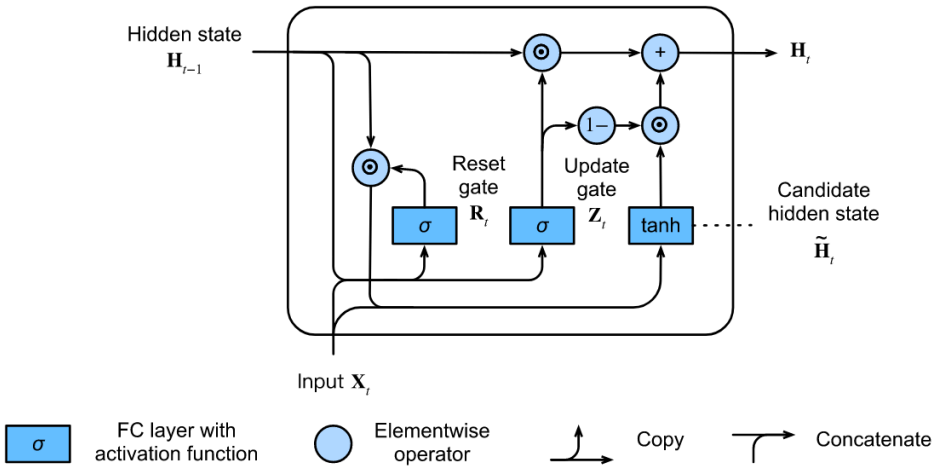
RNN (Recurrent Neural Network)

- A type of neural network that processes sequential data
- Uses recurrent connections to capture temporal relationships
- Suffers from vanishing gradient problem, making training difficult



LSTM (Long Short-Term Memory)

- A type of RNN that addresses vanishing gradient problem
- Uses memory cells to store information for long periods
- Gates (input, output, forget) control information flow
- More effective for long-term dependencies



GRU (Gated Recurrent Unit)

- A simpler alternative to LSTM
- Uses gates (reset, update) to control information flow
- Faster training and inference compared to LSTM
- Less effective for very long sequences

In summary:

- RNN: basic recurrent network, struggles with long-term dependencies.
- LSTM: more effective for long-term dependencies, but slower.
- GRU: balances performance and speed, suitable for most tasks.

Backpropogating an LSTM: A Numerical Example

Let us begin by defining out internal weights:

$$W_a = \begin{bmatrix} 0.45 \\ 0.25 \end{bmatrix}, U_a = \begin{bmatrix} 0.15 \end{bmatrix}, b_a = \begin{bmatrix} 0.2 \end{bmatrix}$$

$$W_i = \begin{bmatrix} 0.95 \\ 0.8 \end{bmatrix}, U_i = \begin{bmatrix} 0.8 \end{bmatrix}, b_i = \begin{bmatrix} 0.65 \end{bmatrix}$$

$$W_f = \begin{bmatrix} 0.7 \\ 0.45 \end{bmatrix}, U_f = \begin{bmatrix} 0.1 \end{bmatrix}, b_f = \begin{bmatrix} 0.15 \end{bmatrix}$$

$$W_o = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}, U_o = \begin{bmatrix} 0.25 \end{bmatrix}, b_o = \begin{bmatrix} 0.1 \end{bmatrix}$$

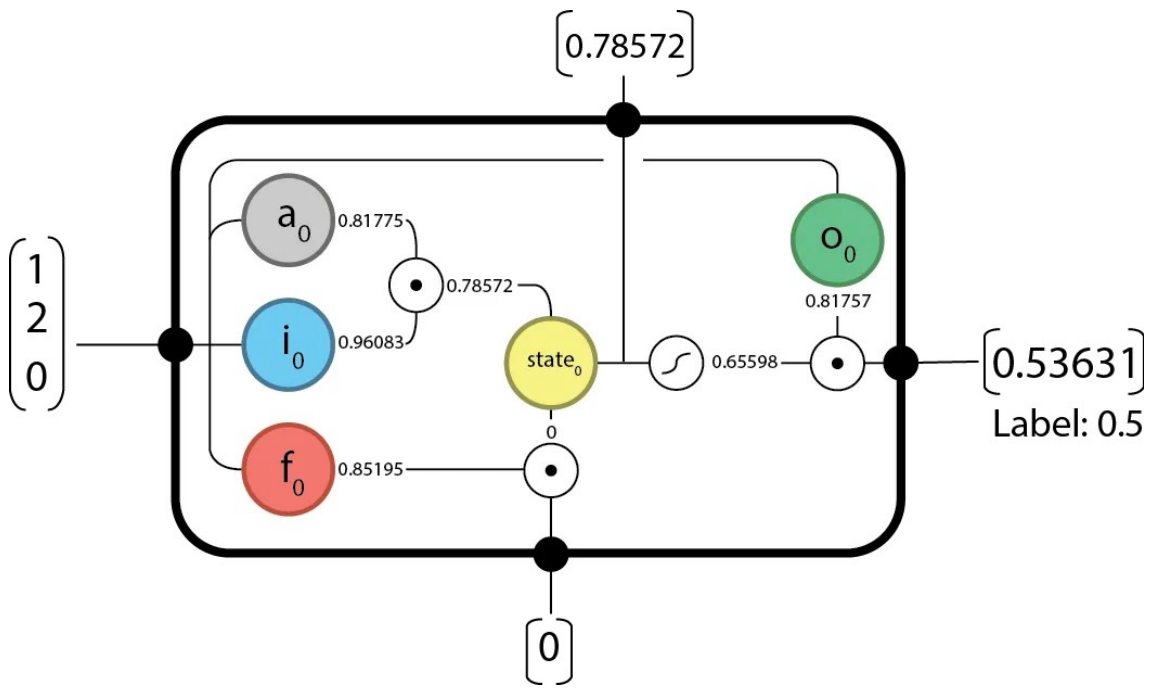
And now input data:

$$x_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{ with label: } 0.5$$

$$x_1 = \begin{bmatrix} 0.5 \\ 3 \end{bmatrix} \text{ with label: } 1.25$$

I'm using a sequence length of two here to demonstrate the unrolling over time of RNNs.

Forward @ t=0



$$a_0 = \tanh(W_a \cdot x_0 + U_a \cdot out_{-1} + b_a) = \tanh(\begin{bmatrix} 0.45 & 0.25 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.15 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.2 \end{bmatrix}) = 0.81775$$

$$i_0 = \sigma(W_i \cdot x_0 + U_i \cdot out_{-1} + b_i) = \sigma(\begin{bmatrix} 0.95 & 0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.8 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.65 \end{bmatrix}) = 0.96083$$

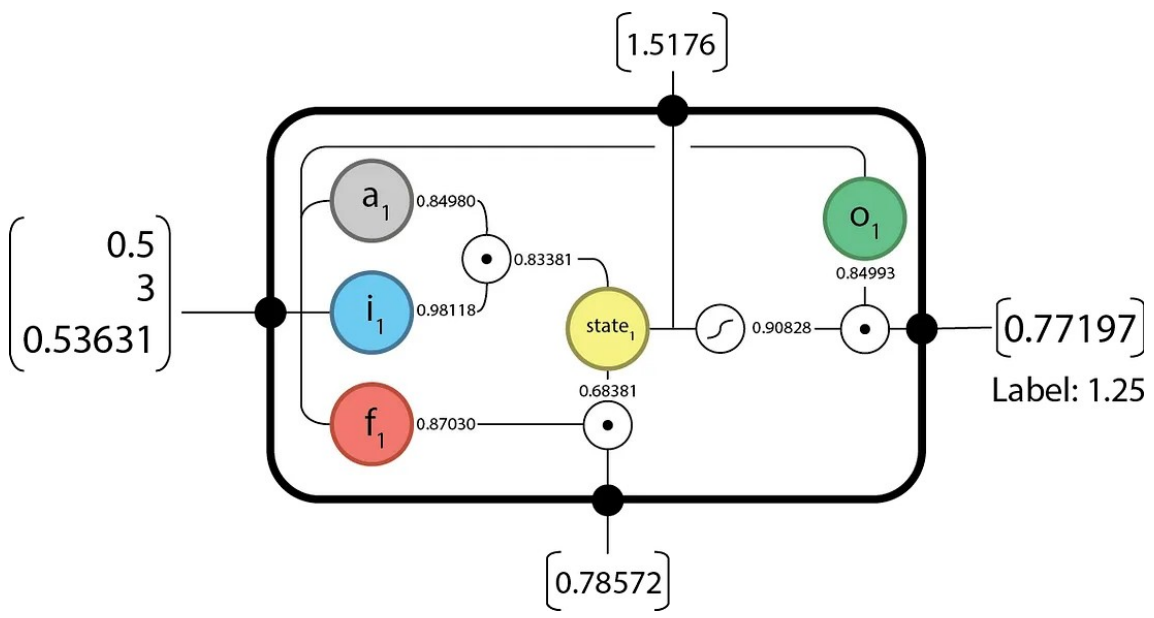
$$f_0 = \sigma(W_f \cdot x_0 + U_f \cdot out_{-1} + b_f) = \sigma(\begin{bmatrix} 0.7 & 0.45 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.1 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.15 \end{bmatrix}) = 0.85195$$

$$o_0 = \sigma(W_o \cdot x_0 + U_o \cdot out_{-1} + b_o) = \sigma(\begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.25 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.1 \end{bmatrix}) = 0.81757$$

$$state_0 = a_0 \odot i_0 + f_0 \odot state_{-1} = 0.81775 \times 0.96083 + 0.85195 \times 0 = 0.78572$$
$$out_0 = \tanh(state_0) \odot o_0 = \tanh(0.78572) \times 0.81757 = 0.53631$$

From here, we can pass forward our state and output and begin the next time-step.

Forward @ t=1



$$a_1 = \tanh(W_a \cdot x_1 + U_a \cdot out_0 + b_a) = \tanh(\begin{bmatrix} 0.45 & 0.25 \end{bmatrix} \begin{bmatrix} 0.5 \\ 3 \end{bmatrix} + \begin{bmatrix} 0.15 \end{bmatrix} \begin{bmatrix} 0.53631 \end{bmatrix} + \begin{bmatrix} 0.2 \end{bmatrix}) = 0.84980$$

$$i_1 = \sigma(W_i \cdot x_1 + U_i \cdot out_0 + b_i) = \sigma(\begin{bmatrix} 0.95 & 0.8 \end{bmatrix} \begin{bmatrix} 0.5 \\ 3 \end{bmatrix} + \begin{bmatrix} 0.8 \end{bmatrix} \begin{bmatrix} 0.53631 \end{bmatrix} + \begin{bmatrix} 0.65 \end{bmatrix}) = 0.98118$$

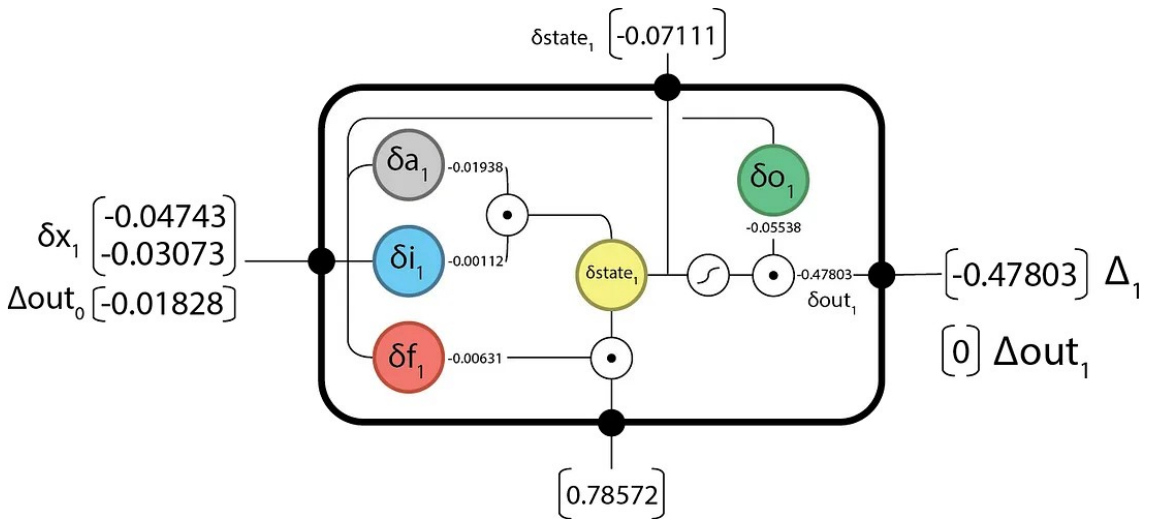
$$f_1 = \sigma(W_f \cdot x_1 + U_f \cdot out_0 + b_f) = \sigma(\begin{bmatrix} 0.7 & 0.45 \end{bmatrix} \begin{bmatrix} 0.5 \\ 3 \end{bmatrix} + \begin{bmatrix} 0.1 \end{bmatrix} \begin{bmatrix} 0.53631 \end{bmatrix} + \begin{bmatrix} 0.15 \end{bmatrix}) = 0.87030$$

$$o_1 = \sigma(W_o \cdot x_1 + U_o \cdot out_0 + b_o) = \sigma(\begin{bmatrix}0.6 & 0.4\end{bmatrix}\begin{bmatrix}0.5\\3\end{bmatrix} + \begin{bmatrix}0.25\end{bmatrix}\begin{bmatrix}0.53631\end{bmatrix} + \begin{bmatrix}0.1\end{bmatrix}) = 0.84993$$

$$state_1 = a_1 \odot i_1 + f_1 \odot state_0 = 0.84980 \times 0.98118 + 0.87030 \times 0.78572 = 1.5176$$
$$out_1 = \tanh(state_1) \odot o_1 = \tanh(1.5176) \times 0.84993 = 0.77197$$

And since we're done our sequence we have everything we need to begin backpropogating.

Backward @ t=1



First we'll need to compute the difference in output from the expected (label).

Note for this we'll be using L2 Loss:

$$E(x, \hat{x}) = \frac{(x - \hat{x})^2}{2}$$

The derivate w.r.t. x is:

$$\partial_x E(x, \hat{x}) = x - \hat{x}$$

So,

$$\Delta_1 = \partial_x E = 0.77197 - 1.25 = -0.47803$$

$\Delta out_1 = 0$ because there are no future time-steps.

$$\delta out_1 = \Delta_1 + \Delta out_1 = -0.47803 + 0 = -0.47803$$
$$\delta state_1 = \delta out_1 \odot o_1 \odot (1 - \tanh^2(state_1)) + \delta state_2 \odot f_2 = -0.47803 \times 0.84993 \times (1 - \tanh^2(1.5176)) + 0 \times 0 = -0.07111$$

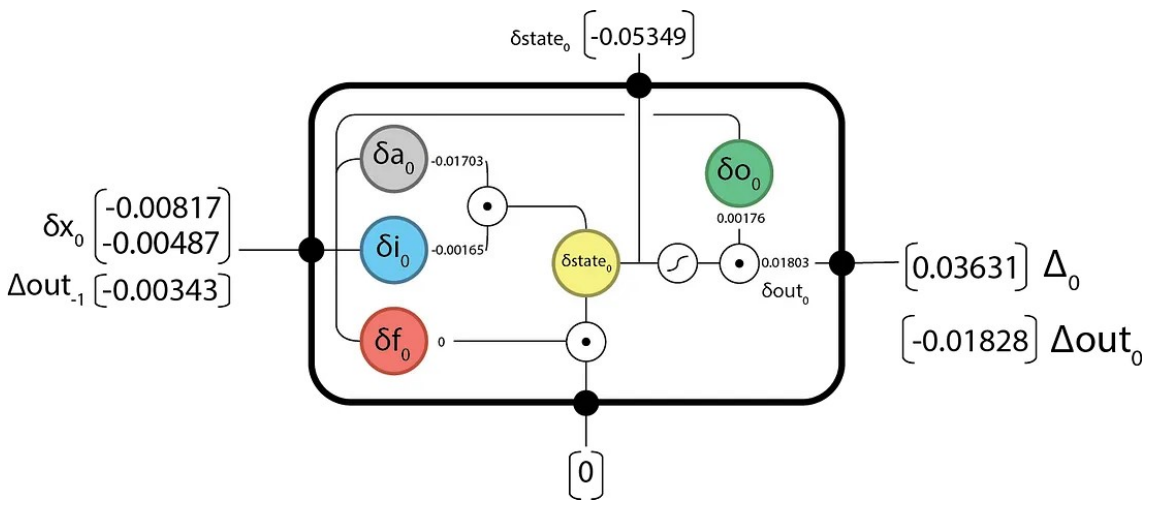
$$\delta a_1 = \delta state_1 \odot i_1 \odot (1 - a_1^2) = -0.07111 \times 0.98118 \times (1 - 0.84980^2) = -0.01938$$
$$\delta i_1 = \delta state_1 \odot a_1 \odot i_1 \odot (1 - i_1) = -0.07111 \times 0.84980 \times 0.98118 \times (1 - 0.98118) = -0.00112$$
$$\delta f_1 = \delta state_1 \odot state_0 \odot f_1 \odot (1 - f_1) = -0.07111 \times 0.78572 \times 0.87030 \times (1 - 0.87030) = -0.00631$$
$$\delta o_1 = \delta out_1 \odot \tanh(state_1) \odot o_1 \odot (1 - o_1) = -0.47803 \times \tanh(1.5176) \times 0.84993 \times (1 - 0.84993) = -0.05538$$

$$\delta x_1 = W^T \cdot \delta gates_1$$
$$= \begin{bmatrix}0.45 & 0.95 & 0.70 & 0.60\\0.25 & 0.80 & 0.45 & 0.40\end{bmatrix}\begin{bmatrix}-0.01938\\-0.00112\\-0.00631\\-0.05538\end{bmatrix} = \begin{bmatrix}-0.04743\\-0.03073\end{bmatrix}$$

$$\Delta out_0 = U^T \cdot \delta gates_1$$
$$= \begin{bmatrix}0.15 & 0.80 & 0.10 & 0.25\end{bmatrix}\begin{bmatrix}-0.01938\\-0.00112\\-0.00631\\-0.05538\end{bmatrix} = -0.01828$$

Now we can pass back our Δout and continue on computing…

Backward @ t=0



$$\Delta_0 = \partial_x E = 0.53631 - 0.5 = 0.03631$$
$$\Delta out_0 = -0.01828, \text{ passed back from T=1}$$

$$\delta out_0 = \Delta_0 + \Delta out_0 = 0.03631 + -0.01828 = 0.01803$$
$$\delta state_0 = \delta out_0 \odot o_0 \odot (1 - \tanh^2(state_0)) + \delta state_1 \odot f_1 = 0.01803 \times 0.81757 \times (1 - \tanh^2(0.78572)) + -0.07111 \times 0.87030 = -0.05349$$

$$\delta a_0 = \delta state_0 \odot i_0 \odot (1 - a_0^2) = -0.05349 \times 0.96083 \times (1 - 0.81775^2) = -0.01703$$
$$\delta i_0 = \delta state_0 \odot a_0 \odot i_0 \odot (1 - i_0) = -0.05349 \times 0.81775 \times 0.96083 \times (1 - 0.96083) = -0.00165$$
$$\delta f_0 = \delta state_0 \odot state_{-1} \odot f_0 \odot (1 - f_0) = -0.05349 \times 0 \times 0.85195 \times (1 - 0.85195) = 0$$
$$\delta o_0 = \delta out_0 \odot \tanh(state_0) \odot o_0 \odot (1 - o_0) = 0.01803 \times \tanh(0.78572) \times 0.81757 \times (1 - 0.81757) = 0.00176$$

$$\delta x_0 = W^T \cdot \delta gates_0$$

$$= \begin{bmatrix} 0.45 & 0.95 & 0.70 & 0.60 \\ 0.25 & 0.80 & 0.45 & 0.40 \end{bmatrix} \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = \begin{bmatrix} -0.00817 \\ -0.00487 \end{bmatrix}$$

$$\Delta out_{-1} = U^T \cdot \delta gates_1$$

$$= \begin{bmatrix} 0.15 & 0.80 & 0.10 & 0.25 \end{bmatrix} \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = -0.00343$$

And we're done the backward step!

Now we'll need to update our internal parameters according to whatever solving algorithm you've chosen. I'm going to use a simple Stochastic Gradient Descent (SGD) update with learning rate: λ=0.1.

We'll need to compute how much our weights are going to change by:

$$\delta W = \sum_{t=0}^{T} \delta gates_t \otimes x_t$$

$$= \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} \begin{bmatrix} 1.0 & 2.0 \end{bmatrix} + \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00631 \\ -0.05538 \end{bmatrix} \begin{bmatrix} 0.5 & 3.0 \end{bmatrix} = \begin{bmatrix} -0.02672 & -0.0922 \\ -0.00221 & -0.00666 \\ -0.00316 & -0.01893 \\ -0.02593 & -0.16262 \end{bmatrix}$$

$$\delta U = \sum_{t=0}^{T-1} \delta gates_{t+1} \otimes out_t$$

$$= \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00631 \\ -0.05538 \end{bmatrix} \begin{bmatrix} 0.53631 \end{bmatrix} = \begin{bmatrix} -0.01039 \\ -0.00060 \\ -0.00338 \\ -0.02970 \end{bmatrix}$$

$$\delta b = \sum_{t=0}^{T} \delta gates_{t+1}$$

$$= \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} + \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00631 \\ -0.05538 \end{bmatrix} = \begin{bmatrix} -0.03641 \\ -0.00277 \\ -0.00631 \\ -0.05362 \end{bmatrix}$$

And updating out parameters based on the SGD update function:

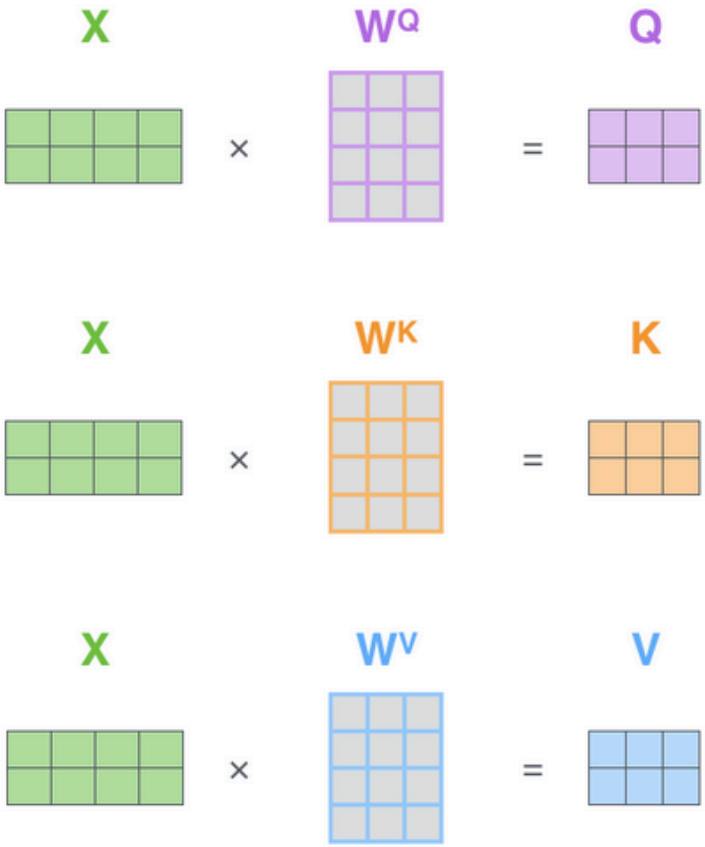
$$W^{new} = W^{old} - \lambda * \delta W^{old}$$

$$W_a = \begin{bmatrix} 0.45267 \\ 0.25922 \end{bmatrix}, U_a = \begin{bmatrix} 0.15104 \end{bmatrix}, b_a = \begin{bmatrix} 0.20364 \end{bmatrix}$$

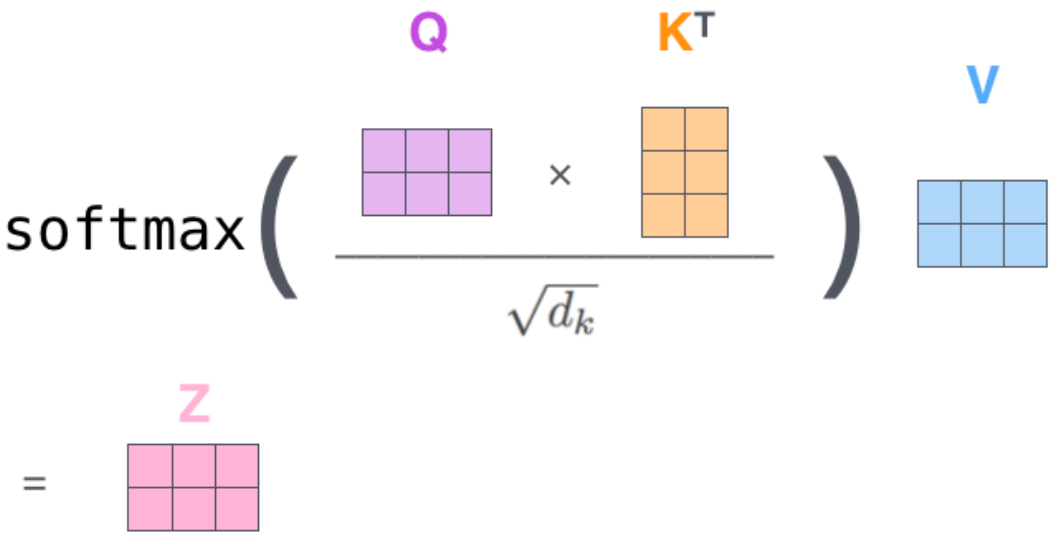
$$W_i = \begin{bmatrix} 0.95022 \\ 0.80067 \end{bmatrix}, U_i = \begin{bmatrix} 0.80006 \end{bmatrix}, b_i = \begin{bmatrix} 0.65028 \end{bmatrix}$$

$$W_f = \begin{bmatrix} 0.70031 \\ 0.45189 \end{bmatrix}, U_f = \begin{bmatrix} 0.10034 \end{bmatrix}, b_f = \begin{bmatrix} 0.15063 \end{bmatrix}$$

$$W_o = \begin{bmatrix} 0.60259 \\ 0.41626 \end{bmatrix}, U_o = \begin{bmatrix} 0.25297 \end{bmatrix}, b_o = \begin{bmatrix} 0.10536 \end{bmatrix}$$



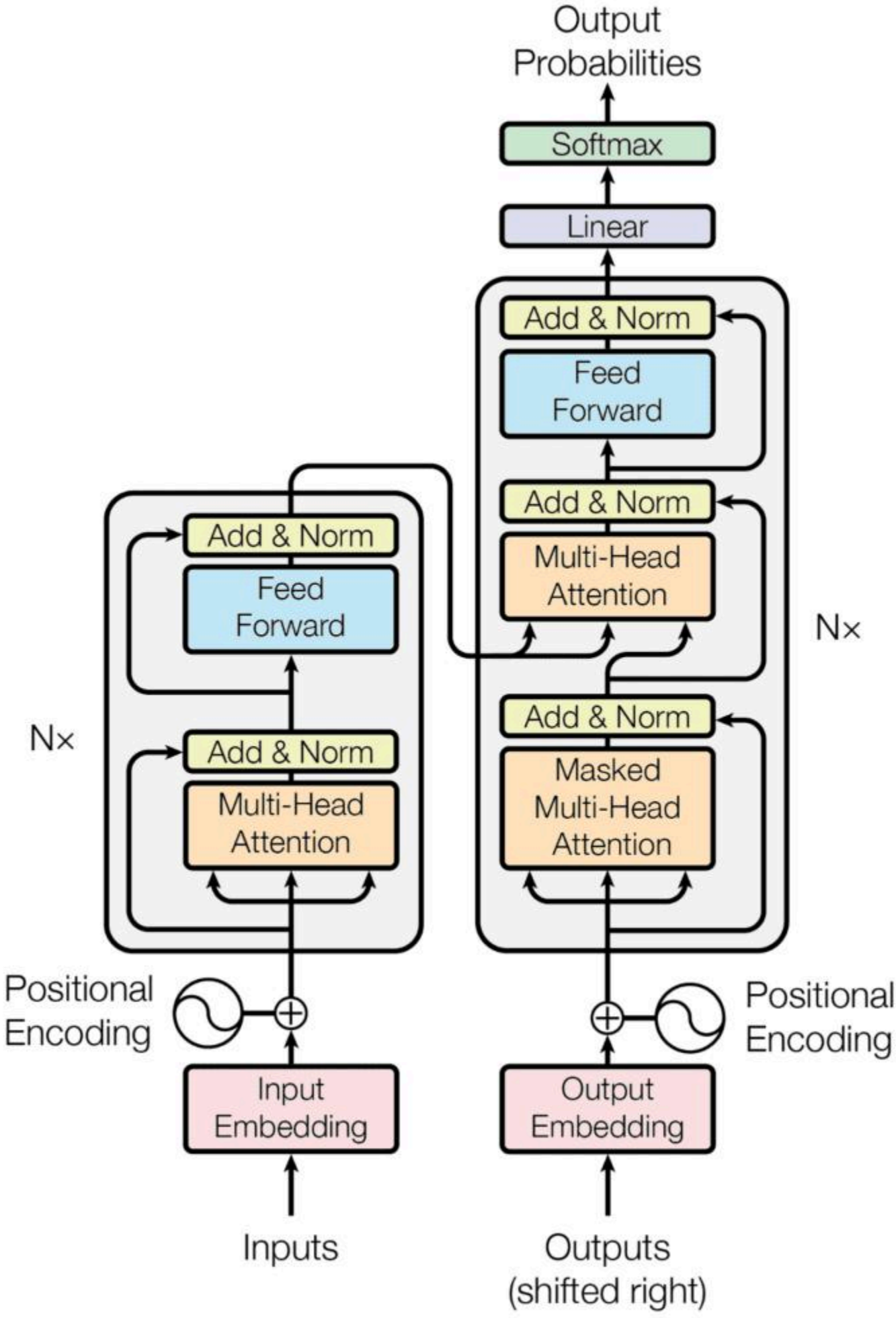
Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)



The Beast With Many Heads

The paper further refined the self-attention layer by adding a mechanism called “multi-headed” attention. This improves the performance of the attention layer in two ways:

1. It expands the model’s ability to focus on different positions. Yes, in the example above, z1 contains a little bit of every other encoding, but it could be dominated by the actual word itself. If we’re translating a sentence like “The animal didn’t cross the street because it was too tired”, it would be useful to know which word “it” refers to.
2. It gives the attention layer multiple “representation subspaces”. As we’ll see next, with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices (the Transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder). Each of these sets is randomly initialized. Then, after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)