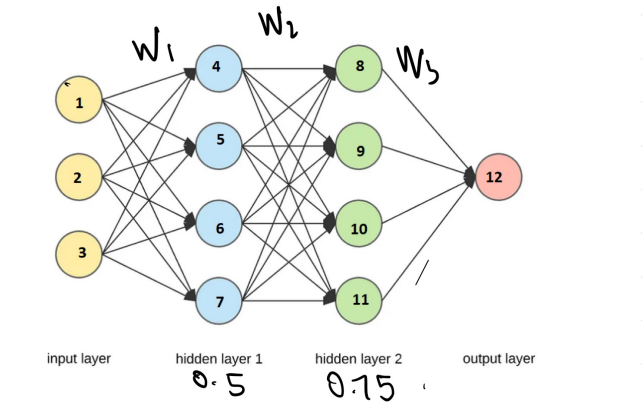
Use the dropout probability for 0.5 for hidden layer 1 and 0.75 for hidden layer 2. All forward and backward propagation formulas should be shown clearly.

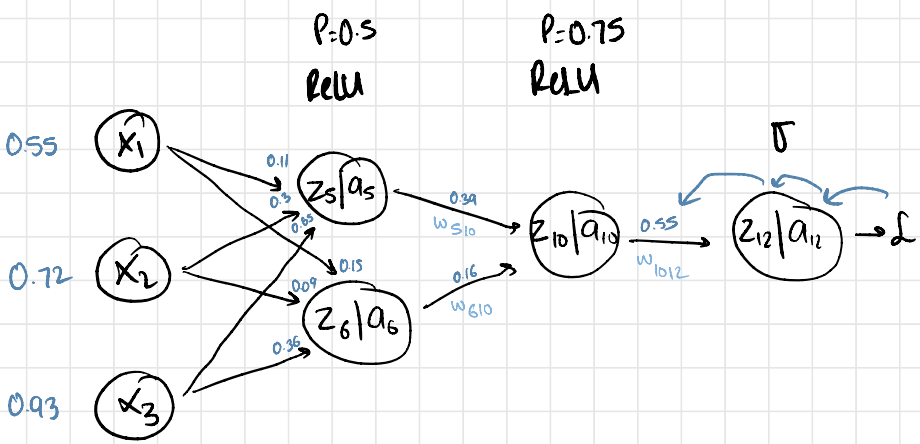
i. [1 Point] Compute forward propagation on the thinned network.

ii. [1 Point] Compute the error using the squared error function if y=0.98.

iii.[3 Points] Perform backpropagation on a thinned network and update the weights.

iv.[1 Point] Test the updated network on new test data if X=[[0.33], [0.45], [0.81]]. Compute the output.





learning rate = 0.7

$L = \frac{1}{2}(y-a)^2$

$\frac{\delta L}{\delta a} = (a-y)$

$y = 0.98$

↳ FF

$z_5 = (0.55)(0.11) + (0.72)(0.3) + (0.93)(0.65) = 0.881$ $\quad a_5 = 0.881$

$z_6 = (0.55)(0.15) + (0.72)(0.09) + (0.93)(0.36) = 0.4821$ $\quad a_6 = 0.4821$

$z_{10} = 0.881(0.39) + 0.4821(0.16) = 0.420726$ $\quad a_{10} = 0.420726$

$z_{12} = 0.420726(0.55) = 0.2313905$ $\quad a_{12} = \frac{1}{1+e^{-z_{12}}} = 0.55559$

(a_5, w_{510}, a_6, w_{610}; $a_{10} \cdot w_{1012}$)

↳ Compute error using sq error function if y = 0.98

$\text{Loss} = \frac{1}{2}(0.98 - 0.55559)^2$

$= 0.089213$

↳ BP

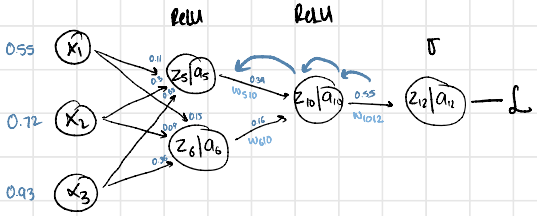
$\frac{\delta L}{\delta a_{12}} = a_{12} - y \qquad \frac{\delta a_{12}}{\delta z_{12}} = a_{12}(1-a_{12}) \qquad \frac{\delta z}{\delta w_{1012}} = a_{10}$

$\frac{\delta L}{\delta w_{1012}} = \frac{\delta L}{\delta a_{12}} \cdot \frac{\delta a_{12}}{\delta z_{12}} \cdot \frac{\delta z_{12}}{\delta w_{1012}} = (a_{12}-y) \cdot (a_{12}(1-a_{12})) \cdot a_{10} = -0.04$

$w^{+}_{1012} = 0.55 - 0.7(-0.044) = \boxed{0.519}$

$\frac{\delta L}{\delta b_{12}} = (a_{12}-y) \cdot (a_{12}(1-a_{12})) = -0.10479$

$b^{+}_{12} = 0 - 0.7(-1.0479) = \boxed{0.07294}$



$$\frac{\delta L}{\delta w_{510}} = \frac{\delta L}{\delta a_{10}} \cdot \frac{\delta a_{10}}{\delta z_{10}} \cdot \frac{\delta z_{10}}{\delta w_{510}}$$

$$\frac{\delta L}{\delta a_{10}} = \frac{\delta L}{\delta a_{12}} \cdot \frac{\delta a_{12}}{\delta z_{12}} \cdot \frac{\delta z_{12}}{\delta a_{10}}$$

$$(a_{12}-y).(a_{12}(1-a_{12})).w_{1012}$$

$$(a_{12}-y).(a_{12}(1-a_{12})).w_{1012}.(0,1)'.a_5$$

$$= -0.0505$$

$$w^+_{510} = 0.39 - (0.1)(-0.0505)$$

$$= 0.42535$$

$$\frac{\delta L}{\delta w_{610}} = (a_{12}-y).(a_{12}(1-a_{12})).w_{1012}.(0,1)'.a_6$$

$$= 0.0276$$

$$w^+_{610} = 0.16 - (0.1)(0.0276)$$

$$= 0.17932$$

$$\frac{\delta L}{\delta b_{10}} = (a_{12}-y).(a_{12}(1-a_{12})).w_{1012}.(0,1)'$$

$$= -0.0573$$

$$b^+_{10} = 0 - (0.1)(-0.0573)$$

$$= 0.0401$$

↳ Repeat of last left

$w^+_{15} = 0.118603$

$w^+_{25} = 0.31127$

$w^+_{35} = 0.664546$

$b^+_5 = 0.015645$

$w^+_{16} = 0.153528$

$w^+_{26} = 0.09462$

$w^+_{36} = 0.36596$

$b_6 = 0.006412$

↳ Test updated Network

$X_1 = 0.33$
$X_2 = 0.45$
$X_3 = 0.81$

updated weights, multiply by P

$$Z_4 = 0.33(0.21) + 0.45(0.36) + 0.81(0.105)$$
$$= 0.31635$$

Updated weights, multiply by P

$W1 = [[0.21, 0.36, 0.105], [0.0593, 0.15568, 0.32273], [0.076764, 0.04731, 0.18278], [0.16, 0.255, 0.06]]$

$W2 = [[0.1275, 0.2175, 0.2625, 0.3825], [0.6825, 0.06, 0.3825, 0.2775], [0.4575, 0.319012.5, 0.13449, 0.0075], [0.285, 0.1575, 0.615, 0.105]]$

$W3 = [0.44, 0.03, 0.581, 0.81]$

$Z_4 = 0.31635$
$a_4 = 0.31635$

$Z_5 = 0.3747588$
$a_5 = 0.3747588$

$Z_6 = 0.20124742$
$a_6 = 0.20124742$

$Z_7 = 0.21615$
$a_7 = 0.21615$

$Z_8 = 0.25734$
$a_8 = 0.25734$

$Z_9 = 0.37535$
$a_9 = 0.37535$

$Z_{10} = 0.293$
$a_{10} = 0.293$

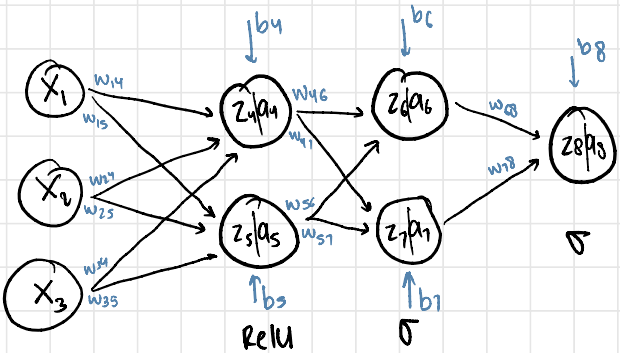
$Z_{11} = 0.295647$
$a_{11} = 0.295647$

$Z_{12} = 0.60713$
$a_{12} = 0.64728$

Be like a neural network. Learn from your mistake

21K4688

1. [20 Points] Draw 3-layer Neural Networks (Input layer is not counted here)with 2 nodes in each hidden layer and 1 output node. Input size is 3. Activation function is relu for 1st hidden layer and sigmoid for 2nd hidden and last layer. Write mathematical equations of all hidden and output layer nodes for a forward pass. Write down mathematical equations of the partial derivative of Loss w.r.t. all weights and bias for a backward pass. Simplify all these equations. ~~Use (i) Squared Loss (ii)~~ Cross Entropy Loss



$\delta\sigma = a(1-a)$

$\delta Relu = (0,1)$

Feed Forward

$z_4 = x_1 \cdot w_{14} + x_2 \cdot w_{24} + x_3 \cdot w_{34} + b_4$ $\quad a_4 = Relu(z_4) = max(0, z_4)$

$z_5 = x_1 w_{15} + x_2 w_{25} + x_3 w_{35} + b_5$ $\quad a_5 = Relu(z_5) = max(0, z_5)$

$z_6 = a_4 \cdot w_{46} + a_5 \cdot w_{56} + b_6$ $\quad a_6 = \sigma(z_6) = 1/1+e^{-z_6}$

$z_7 = a_4 \cdot w_{47} + a_5 \cdot w_{57} + b_7$ $\quad a_7 = \sigma(z_7) = 1/1+e^{-z_7}$

$z_8 = a_6 \cdot w_{68} + a_7 \cdot w_{78} + b_8$ $\quad a_8 = \sigma(z_8) = 1/1+e^{-z_8}$

Backward Propogation

1) Squared loss

$L = \frac{1}{2}(y-\hat{y})^2$

$L = \frac{1}{2}(y-a_8)^2$

$\frac{\delta L}{\delta a_8} = \frac{1}{2}(2)(-1)(y-a_8)$

$\frac{\delta L}{\delta a_8} = a_8 - y$

$\frac{\delta a_8}{\delta z_8} = a_8(1-a_8)$

$\frac{\delta z_8}{\delta w_{68}} = a_6$ $\quad \frac{\delta z_8}{\delta w_{78}} = a_7$

$$\frac{\delta L}{\delta w_{68}} = \frac{\delta L}{\delta a_8} \cdot \frac{\delta a_8}{\delta z_8} \cdot \frac{\delta z_8}{\delta w_{68}}$$

$$\rightarrow \quad (a_8 - y) \cdot a_8(1-a_8) \cdot a_6 \qquad w_{68}^{+} = w_{68} - \alpha \frac{\delta L}{\delta w_{68}}$$

$$\rightarrow \frac{\delta L}{\delta w_{78}} = (a_8 - y) \cdot a_8(1-a_8) \cdot a_7 \qquad w_{78}^{+} = w_{78} - \alpha \frac{\delta L}{\delta w_{78}}$$

$$\rightarrow \frac{\delta L}{\delta b_8} = (a_8 - y) \cdot a_8(1-a_8)$$

H1

$$\rightarrow \frac{\delta L}{\delta w_{46}} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta a_4}$$

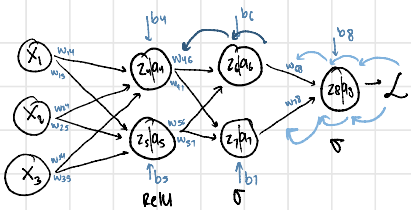
$$= (a_8 - y)(a_8(1-a_8)) \cdot w_{68} (a_6(1-a_6)) a_4$$

$$\frac{\delta L}{\delta a_6} = \frac{\delta L}{\delta a_8} \times \frac{\delta a_8}{\delta z_8} \times \frac{\delta z_8}{\delta a_6}$$

$$(a_8 - y)(a_8(1-a_8)) \cdot w_{68}$$

$$\rightarrow \frac{\delta L}{w_{56}} = (a_8 - y)(a_8(1-a_8)) \cdot w_{68} \ (a_6(1-a_6)) a_5$$

$$\frac{\delta L}{\delta b_6} = (a_8 - y)(a_8(1-a_8)) \cdot w_{68} \ (a_6(1-a_6))$$



$$\frac{\delta L}{\delta w_{47}} = \frac{\delta L}{a_7} \cdot \frac{\delta a_7}{\delta z_7} \cdot \frac{\delta z_7}{\delta w_{47}}$$

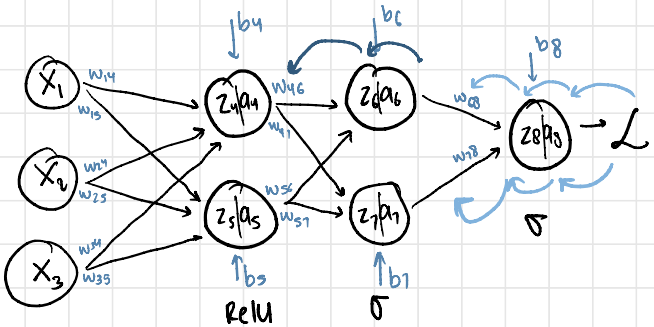
$$= (a_8 - y) \cdot a_8(1-a_8) \cdot w_{78} \cdot a_7(1-a_7) \cdot a_4$$

$$\frac{\delta L}{\delta w_{57}} = (a_8 - y) \cdot a_8(1-a_8) \cdot w_{78} \cdot a_7(1-a_7) \cdot a_5$$

$$\frac{\delta L}{\delta b_7} (a_8 - y) \cdot a_8(1-a_8) \cdot w_{78} \cdot a_7(1-a_7)$$

$$\frac{\delta L}{\delta a_7} = \frac{\delta L}{\delta a_8} \cdot \frac{\delta a_8}{\delta z_8} \cdot \frac{\delta z_8}{\delta a_7}$$

$$= (a_8 - y) \cdot a_8(1-a_8) \cdot w_{78}$$



H2

$$\frac{\delta L}{\delta w_{14}} = \frac{\delta L}{\delta a_4}\cdot\frac{\delta a_4}{\delta z_4}\cdot\frac{\delta z_4}{\delta w_{14}}$$

$$\frac{\delta L}{\delta a_4} = \frac{\delta L}{a_6}\cdot\frac{\delta a_6}{\delta z_6}\cdot\frac{\delta z_6}{a_4} + \frac{\delta L}{a_7}\cdot\frac{\delta a_7}{\delta z_7}\cdot\frac{\delta z_7}{\delta a_4}$$

$$= (a_8 - y)(a_8(1-a_8))\cdot w_{68}\cdot a_6(1-a_6)\cdot w_{46} + (a_8-y)\cdot a_8(1-a_8)\cdot w_{78}\cdot a_7(1-a_7)\cdot w_{47}$$

$$\frac{\delta}{\delta w_{14}} = \left[(a_8 - y)(a_8(1-a_8))\cdot w_{68}\cdot a_6(1-a_6)\cdot w_{46} + (a_8-y)\cdot a_8(1-a_8)\cdot w_{78}\cdot a_7(1-a_7)\cdot w_{47}\right]\cdot_{(0,1)} x_1$$

$$\frac{\delta}{\delta w_{24}} = \left[(a_8 - y)(a_8(1-a_8))\cdot w_{68}\cdot a_6(1-a_6)\cdot w_{46} + (a_8-y)\cdot a_8(1-a_8)\cdot w_{78}\cdot a_7(1-a_7)\cdot w_{47}\right]\cdot_{(0,1)} x_2$$

$$\frac{\delta}{\delta w_{34}} = \left[(a_8 - y)(a_8(1-a_8))\cdot w_{68}\cdot a_6(1-a_6)\cdot w_{46} + (a_8-y)\cdot a_8(1-a_8)\cdot w_{78}\cdot a_7(1-a_7)\cdot w_{47}\right]\cdot_{(0,1)} x_3$$

$$\frac{\delta}{\delta w_{15}} = \left[(a_8 - y)(a_8(1-a_8))\cdot w_{68}\cdot a_6(1-a_6)\cdot w_{56} + (a_8-y)\cdot a_8(1-a_8)\cdot w_{78}\cdot a_7(1-a_7)\cdot w_{57}\right]\cdot_{(0,1)} x_1$$

$$\frac{\delta}{\delta w_{25}} = \left[(a_8 - y)(a_8(1-a_8))\cdot w_{68}\cdot a_6(1-a_6)\cdot w_{56} + (a_8-y)\cdot a_8(1-a_8)\cdot w_{78}\cdot a_7(1-a_7)\cdot w_{57}\right]\cdot_{(0,1)} x_2$$

$$\frac{\delta}{\delta w_{35}} = \left[(a_8 - y)(a_8(1-a_8))\cdot w_{68}\cdot a_6(1-a_6)\cdot w_{56} + (a_8-y)\cdot a_8(1-a_8)\cdot w_{78}\cdot a_7(1-a_7)\cdot w_{57}\right]\cdot_{(0,1)} x_3$$

ii) Cross entropy

O2

$$L = -[y \log a_8 + (1-y) \log(1-a_8)]$$

$$\frac{\delta L}{\delta w_{68}} = \frac{\delta L}{\delta a_8} \times \frac{\delta a_8}{\delta z_8} \cdot \frac{\delta z_8}{\delta w_{68}}$$

$$\left(\frac{-y}{a_8} + \frac{1-y}{1-a_8}\right) \cdot a_8(1-a_8) \cdot a_6$$

$$\frac{\delta L}{\delta w_{68}} = (a_8 - y)\, a_6$$

$$\frac{\delta L}{\delta w_{78}} = (a_8 - y)\, a_7$$

$$\frac{\delta L}{\delta b_8} = (a_8 - y)$$

H2

$$\frac{\delta L}{\delta w_{46}} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta w_{46}}$$

$$\frac{\delta L}{\delta w_{46}} = (a_8 - y) \cdot w_{68} \cdot a_6(1-a_6)\, a_4$$

$$\frac{\delta L}{\delta w_{56}} = (a_8 - y) \cdot w_{68} \cdot a_6(1-a_6)\, a_5$$

$$\frac{\delta L}{\delta b_6} = (a_8 - y) \cdot w_{68} \cdot a_6(1-a_6)$$

$$\frac{\delta L}{\delta w_{47}} = (a_8 - y) \cdot w_{78} \cdot a_7(1-a_7) \cdot a_4$$

$$\frac{\delta L}{\delta w_{57}} = (a_8 - y) \cdot w_{78} \cdot a\,(1-a_7) \cdot a_5$$

$$\frac{\delta L}{\delta b_7} = (a_8 - y) \cdot w_{78} \cdot a\,(1-a_7)$$

$$\frac{\delta L}{\delta a_6} = \frac{\delta L}{\delta a_8} \cdot \frac{\delta a_8}{\delta z_8} \cdot \frac{\delta z_8}{\delta a_6}$$

$$= (a_8 - y) \cdot w_{68}$$

#2

$$\frac{\delta L}{\delta w_{14}} = \frac{\delta L}{\delta a_{14}} \cdot \frac{\delta a_{14}}{\delta z_{14}} \cdot \frac{\delta z_{14}}{\delta w_{14}}$$

$$\frac{\delta L}{\delta a_{14}} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta a_{14}} + \frac{\delta L}{\delta a_7} \cdot \frac{\delta a_7}{\delta z_7} \cdot \frac{\delta z_7}{\delta a_{14}}$$

$$= (a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{46} + [a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{47}$$

$$\frac{\delta L}{\delta w_{14}} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{46} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{47} \end{bmatrix} \times (0,1) \cdot n_1$$

$$\frac{\delta L}{\delta w_{24}} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{46} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{47} \end{bmatrix} \times (0,1) \cdot n_2$$

$$\frac{\delta L}{\delta w_{34}} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{46} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{47} \end{bmatrix} \times (0,1) \cdot n_3$$

$$\frac{\delta L}{\delta b_4} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{46} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{47} \end{bmatrix} \times (0,1)$$

$$\frac{\delta L}{\delta a_5} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{56} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{57} \end{bmatrix} \times (0,1) \cdot n_1$$

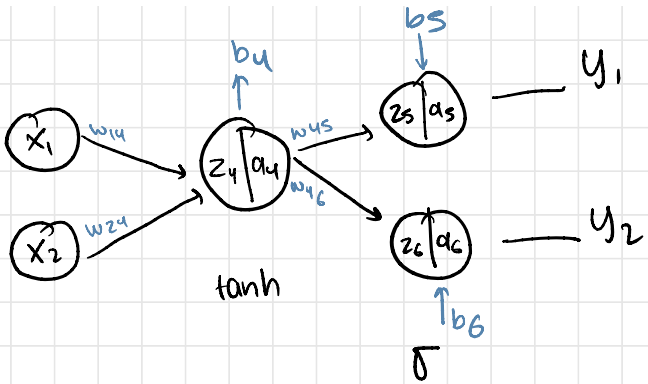
$$\frac{\delta L}{\delta_{15}} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{56} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{57} \end{bmatrix} \times (0,1) \cdot n_1$$

$$\frac{\delta L}{\delta_{25}} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{56} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{57} \end{bmatrix} \times (0,1) \cdot n_2$$

$$\frac{\delta L}{\delta_{35}} = \begin{bmatrix} a_8 - y) \cdot w_{68} \cdot a_6(1-a_6) \cdot w_{56} + \\ a_8 - y) \cdot w_{78} \cdot (a_7(1-a_7)) \cdot w_{57} \end{bmatrix} \times (0,1) \cdot n_3$$

H O

2. [10 Points] Draw 2-layer Neural Networks with 1 node in the hidden layer and 2 output nodes. Input size is 2. The activation function in the hidden layer is tanh while in the output layer, sigmoid is used. Write mathematical equations of all hidden and output layer nodes for a forward pass. Write down mathematical equations of the partial derivative of Loss w.r.t. all weights and bias for a backward pass. Simplify all these equations. Use Cross Entropy Loss.



Feed Forward

$z_4 = x_1 w_{14} + x_2 w_{24} + b_4 \rightarrow a_4 = \tanh(z_4)$

$z_5 = a_4 \cdot w_{45} + b_5 \rightarrow a_5 = \sigma(z_5) = 1/(1+e^{-z_5})$

$z_6 = a_4 \cdot w_{46} + b_6 \rightarrow a_6 = \sigma(z_6) = 1/(1+e^{-z_6})$

Backward

↳ Cross Entropy loss

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

$L = -[y \log a_5 + (1-y)\log(1-a_5)] - [y \log a_6 + (1-y)\log(1-a_6)]$

$\frac{\delta L}{a_5} = \frac{-y_1}{a_5} + \frac{1-y_1}{1-a_5}$ $\quad$ $\frac{\delta L}{a_6} = \frac{-y_2}{a_6} + \frac{1-y_2}{1-a_6}$

$\frac{\delta L}{\delta w_{45}} = \frac{\delta L}{\delta a_5} \cdot \frac{\delta a_5}{\delta z_5} \cdot \frac{\delta z_5}{\delta w_{45}}$

$= \left[-\frac{y}{a_5} + \frac{1-y}{1-a_5}\right] \cdot \left[a_5(1-a_5)\right] \cdot a_4$

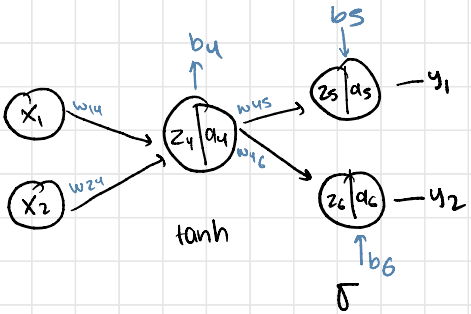
$= -\frac{(y)(1-a_5) + a_5(1-y)}{a_5(1-a_5)} \cdot \left[a_5(1-a_5)\right] \cdot a_4$

$= -y + a_5 y + a_5 - a_5 y$

$= (a_5 - y_1) a_4$

$\frac{\delta L}{\delta w_{45}} = (a_5 - y_1)\, a_4$ $\quad$ $\frac{\delta L}{\delta w_{46}} = (a_6 - y_2)\, a_4$

$\frac{\delta L}{\delta b_5} = a_5 - y_1$ $\quad$ $\frac{\delta L}{\delta b_6} = a_6 - y_2$



H1

$$\frac{\delta L}{a_4} = \frac{\delta L}{\delta a_5} \cdot \frac{\delta a_5}{\delta z_5} \cdot \frac{\delta z_5}{\delta a_4} + \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta a_4}$$

$$\frac{\delta L}{\delta a_4} = \frac{\delta L}{\delta z_5} w_{45} + \frac{\delta L}{\delta z_6} w_{46}$$

$$= (a_5 - y_1) w_{45} + (a_6 - y_2) w_{46}$$

$$\frac{\delta L}{\delta w_{14}} = \frac{\delta L}{\delta a_4} \cdot \frac{\delta a_4}{\delta z_4} \cdot \frac{\delta z_4}{\delta w_{14}}$$

$$= \left[(a_5 - y_1) w_{45} + (a_6 - y_2) w_{46}\right] \cdot (1 - a_4^2) \cdot x_1$$

$$\frac{\delta L}{\delta w_{24}} = \left[(a_5 - y_1) w_{45} + (a_6 - y_2) w_{46}\right] \cdot (1 - a_4^2) \cdot x_2$$

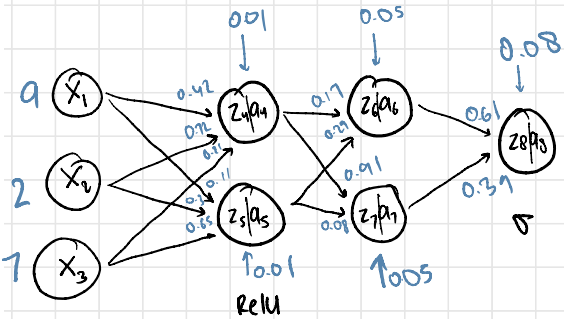
$$\frac{\delta L}{\delta b_4} = \left[(a_5 - y_1) w_{45} + (a_6 - y_2) w_{46}\right] \cdot (1 - a_4^2) \cdot$$

3. [20 Points] Complete the following exercise using formulas computed in Question #1 for
weights of L1
W1 = [[0.42, 0.72, 0.21], [0.11, 0.3, 0.65]]
B1 =0.01 (for all neurons)
weights of L2
W2 = [[0.17, 0.29], [0.91, 0.08]]
B1 =0.05 (for all neurons)
weights of L3
W3 = [[0.61, 0.39]] B1 =0.08
Learning rate = 0.7
(i) Squared Loss Function
(ii) Cross Entropy Loss Function

21k-4658
927A

F1	F2	F3	Class
4th digit of student ID + 1	2	2nd digit of student ID + 1	A
5	8	3rd digit of student ID + 1	B

Be like a neural network. Learn from your mistakes :)



$\alpha = 0.7$

Q3) 1. Squared Loss Function

$z_4 = 0.42(a) + 2(0.72) + (7)(0.21) = 6.7$ $\quad a_4 = 6.7$

$z_5 = 6.15$ $\quad a_5 = 6.15$

$z_6 = 2.9725$ $\quad a_6 = 0.9513$

$z_7 = 6.639$ $\quad a_7 = 0.9987$

$z_8 = 1.0498$ $\quad a_8 = 0.7407$

$\delta w_{14} = -0.0123$ $\quad w_{14} = 0.4286$

$\delta w_{24} = -0.0027$ $\quad w_{24} = 0.7219$

$\delta w_{34} = -0.0096$ $\quad w_{34} = 0.2167$

$\delta b_4 = -0.0014$ $\quad b_4 = 0.011$

$\delta b_5 = -0.0021$ $\quad b_5 = 0.0115$

$\delta b_6 = -0.0073$ $\quad b_6 = 0.0551$

$\delta b_7 = -0.0001$ $\quad b_7 = 0.0561$

$\delta b_8 = -0.2593$ $\quad b_8 = 0.2615$

$\delta w_{15} = -0.0197$ $\quad w_{15} = 0.1234$

$\delta w_{25} = -0.0043$ $\quad w_{25} = 0.303$

$\delta w_{35} = -0.0149$ $\quad w_{35} = 0.6605$

$\delta w_{16} = -0.0491$ $\quad w_{16} = 0.2044$

$\delta w_{26} = -0.045$ $\quad w_{26} = 0.3245$

$\delta w_{17} = -0.0009$ $\quad w_{17} = 0.9106$

$\delta w_{27} = -0.0003$ $\quad w_{27} = 0.0806$

$\delta w_{68} = -0.2466$ $\quad w_{68} = 0.7826$

$\delta w_{78} = -0.2589$ $\quad w_{78} = 0.5712$

2) Cross Entropy

chr

$z_4 = 0.42(9) + 2(0.72) + (7)(0.21) = 6.7$ $\quad a_4 = 6.7$

$z_5 = 6.15$ $\quad a_5 = 6.15$

$z_6 = 2.9725$ $\quad a_6 = 0.9513$

$z_7 = 6.639$ $\quad a_7 = 0.9987$

$z_8 = 0.0336$ $\quad a_8 = 0.7407$

$\delta w_{14} = -0.0024$ $\quad w_{14} = 0.4217$

$\delta w_{24} = -0.0005$ $\quad w_{24} = 0.7201$

$\delta w_{34} = -0.0019$ $\quad w_{34} = 0.2113$

$\delta b_4 = -0.0003$ $\quad b_4 = 0.0102$

$\delta b_5 = -0.0004$ $\quad b_5 = 0.0103$

$\delta b_6 = -1.4067$ $\quad b_6 = 0.051$

$\delta b_7 = -2.533$ $\quad b_7 = 0.05$

$\delta b_8 = -0.0498$ $\quad b_8 = 0.1199$

$\delta w_{15} = -0.0057$ $\quad w_{15} = 0.1126$

$\delta w_{25} = -0.0008$ $\quad w_{25} = 0.3006$

$\delta w_{35} = -0.0024$ $\quad w_{35} = 0.652$

$\delta w_{46} = -0.0094$ $\quad w_{46} = 0.1766$

$\delta w_{56} = -0.0057$ $\quad w_{56} = 0.2961$

$\delta w_{47} = -0.0002$ $\quad w_{47} = 0.9101$

$\delta w_{57} = -0.0002$ $\quad w_{57} = 0.0881$

$\delta w_{68} = -0.074$ $\quad w_{68} = 0.6432$

$\delta w_{78} = -0.0471$ $\quad w_{78} = 0.4248$

CONVOLUTION NEURAL NETWORKS 6/

CNN

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes? Uh.. not really.

1	1	0
4	2	1
0	2	1

1
1
0
4
2
1
0
2
1

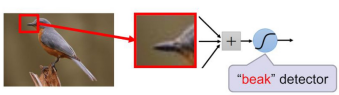
To capture Spatial/temporal dependencies.

- Flattening ignores spatial dependencies.
- CNNs preserve spatial and temporal relationships by using convolutions to detect local patterns.

NN with images

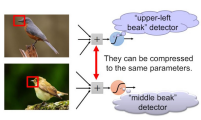
Some patterns are much smaller than the whole image.

Can represent a small region with fewer parameters



Same pattern appears in different places: They can be compressed!

What about training a lot of such "small" detectors and each detector must "move around".



Edge Detection

Edge is Where Change Occurs

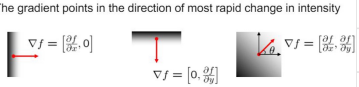
Change is measured by derivative in 1D
Biggest change, derivative has maximum magnitude
Or 2nd derivative is zero.

image gradient

The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid change in intensity



The gradient direction is given by:

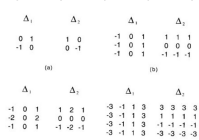
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- how does this relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Gradient operators



(a): Roberts' cross operator (b): 3x3 Prewitt operator
(c): Sobel operator (d) 4x4 Prewitt operator

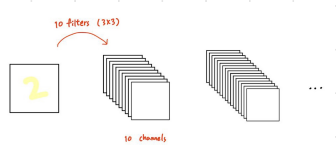
Convolution

Convolution mathematically combines two signals (an input image and a filter/kernel) to produce a new signal (feature map).

CNNs

A CNN is a neural network with some convolutional layers (and some other layers).
A convolutional layer has a number of filters that does convolutional operation.

Result array from a convolution operation using a filter is called a feature map.
If you use 10 filters, then there will be 10 output channels

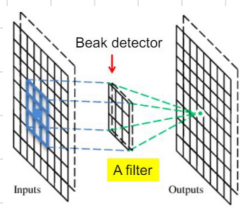


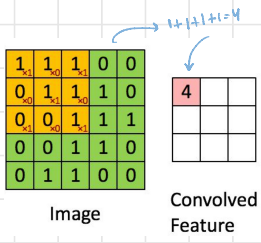
How Convolution Works (Diagram explained):

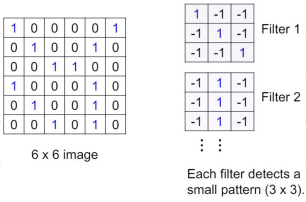
- A filter (small matrix) slides across the input image (larger matrix), performing element-wise multiplication followed by summation.
- The resulting smaller matrix is a feature map, capturing local features such as edges and textures.

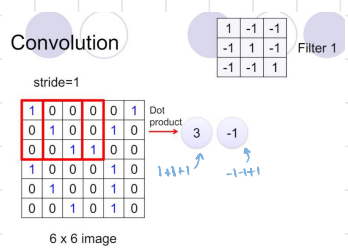
Filters/Kernels:

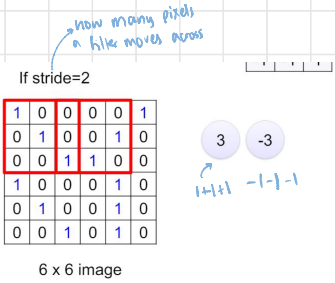
- Filters identify various features (edges, textures, patterns).
- Initially random, filters are trained through backpropagation to recognize specific meaningful patterns.

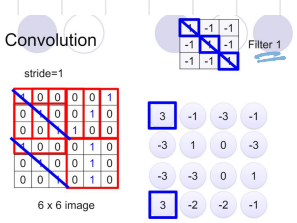


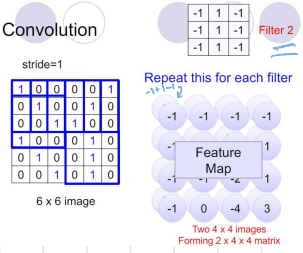




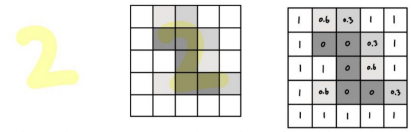


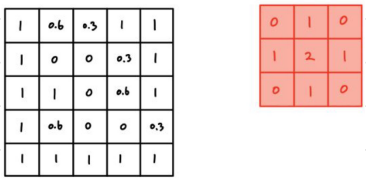


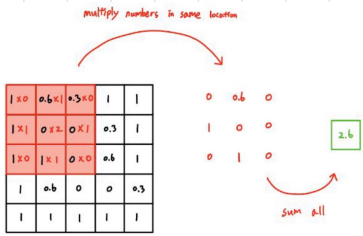


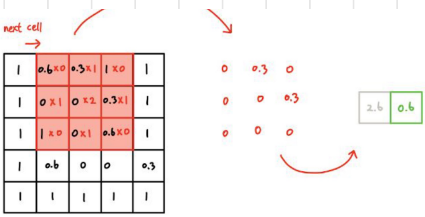


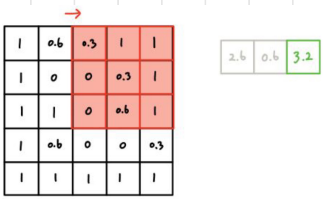
For Grey Images → 1 filter



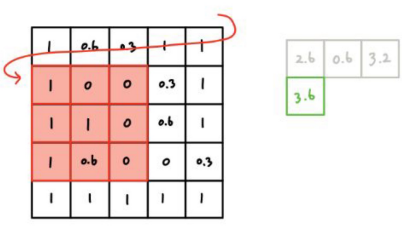


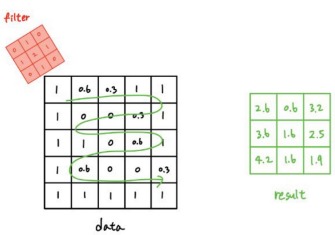




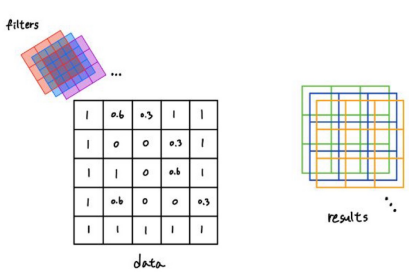


This sliding operation is called convolution

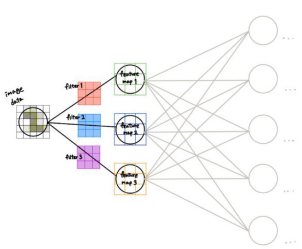




For Grey Images

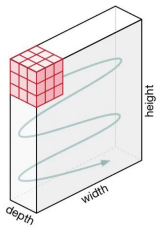


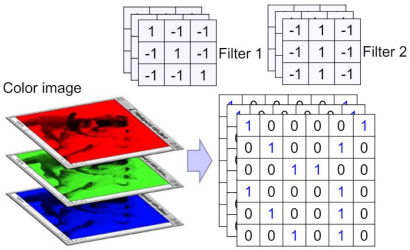
For Grey Images

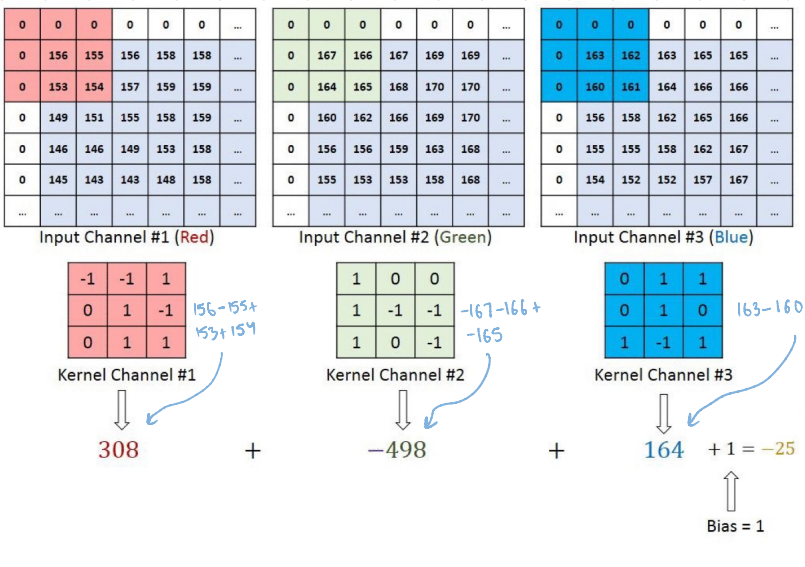


For RGB: Color images → 3 filters (R G B)

- The filter moves to the right with a certain Stride Value till it parses the complete width.
- Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.



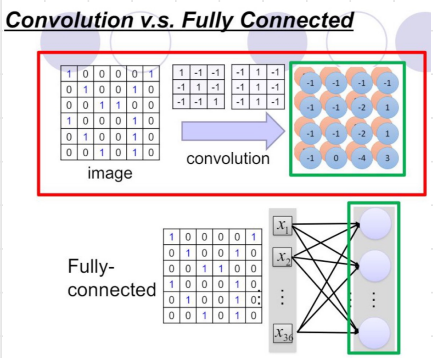




Input × Filter R + Input × Filter G + Input × Filter B + Bias (1) = Output

Fully Connected vs. CNN:

- Fully Connected Networks: Every neuron is connected to every other neuron in the next layer. This requires an immense number of parameters for images, leading to complexity and inefficient learning.
- CNNs: Introduce layers that recognize local patterns, significantly reducing complexity by focusing only on small areas of the input at a time.



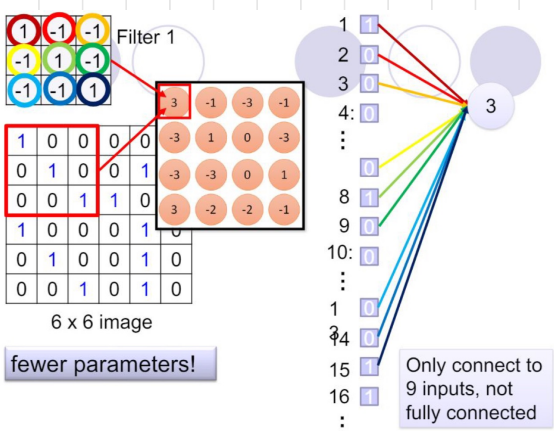
So what kind of values should be in the filters?

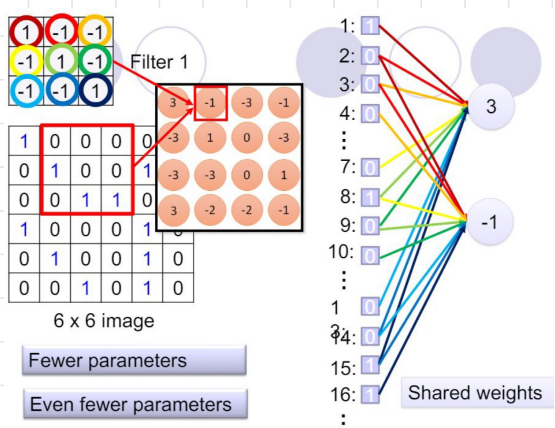
Before the era of deep learning, the values in the filter were fixed. Have you ever used "sharpen filter" in a photo editor?

The values in a convolution filter are the weights of a Convolutional Neural Network.

CNN PROS

↳ reduces complexity → few parameters, shared weights, not fully connected





CNN

Filters: Values and Training

- Initially, filters have random values.
- Through training (gradient descent), these values become optimized to detect meaningful patterns (e.g., edges, textures, shapes).

CNN Layers and Feature Extraction (Diagram explained):

- First Layer: Captures basic features (edges, colors).
- Middle Layers: Identify intermediate features (shapes, complex patterns).
- Final Layers: Capture high-level semantic features (objects, scenes).

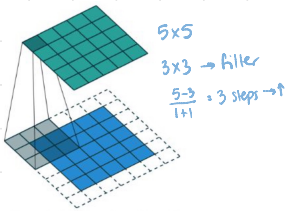


Figure: In this example, 5x5 input is convolved with 3x3 kernel with stride=padding=1 to produce an output of size 5x5.

$O = \frac{5+2(1)-3}{1} + 1$

$= 5 \times 5$

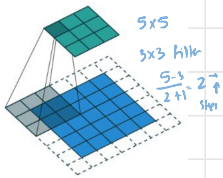


Figure: In this example, 5x5 input is convolved with 3x3 kernel with **stride=2** and padding=1 to produce an output of size 3x3.

$O = \frac{5+2(1)-3}{2} + 1$

$= \frac{4}{2} + 1$

$= 3 \times 3$

- Convolution layer takes an input feature map of dimension $W \times H \times N$ and produces an output feature map of dimension $\hat{W} \times \hat{H} \times M$
- Each layer is defined using following parameters:
  - # Input channels (N)
  - # Output channels (M)
  - Kernel size
  - Padding
  - Stride → how many pixels a filter moves across

↳ Stride 1: large feature map
↳ more detailed

↳ Stride 2: smaller feature map
↳ more faster → as jumps 2 pixels
↳ reduces complexity
↳ lower dim of data

$$\text{dim of output feature map} = \frac{\text{input size} + 2p - \text{kernel size}}{\text{stride}} + 1$$

Q) s:1, p:0, K:3, input: 28

$\text{output} = \frac{28+2(0)-3}{1} + 1$

$= 26$

$= 26 \times 26$

Problem: In AlexNet, the input image is of size 227x227x3. The first convolutional layer has 96 kernels of size 11x11x3. The stride is 4 and padding is 0. Therefore, the size of the output image right after the first bank of convolutional layers is

$$O = \frac{227 - 11 + 2 \times 0}{4} + 1 = 55$$

$O = \frac{227+2(0)-11}{4} + 1$

$= 55$

So, the output image is of size **55x55x96** (one channel for each kernel)

Parameters

num. Parameters: Weights + biases

$i \times f \times f \times O + O$

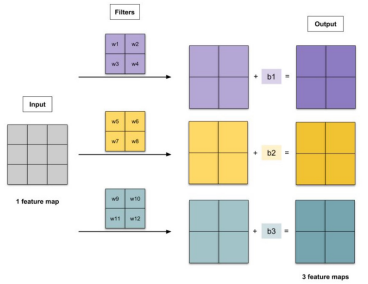
- i: no of input maps / channels
- f: filter size only length
- O: no of output maps
- $+O$: 1 filter is applied to ever output map

grey scale image - 1 channel

RGB image - 3 channels

Greyscale image with 2×2 filter, output 3 channels

- $i = 1$ (greyscale has only 1 channel)
- $f = 2$
- $o = 3$
- *num_params*
  $= [i \times (f \times f) \times o] + o$
  $= [1 \times (2 \times 2) \times 3] + 3$
  $= 15$



RGB image with 2×2 filter, output of 1 channel

- $i = 3$ (3 channels)
- $f = 2$
- $o = 1$
- *num_params*
  $= [i \times (f \times f) \times o] + o$
  $= [3 \times (2 \times 2) \times 1] + 1$
  $= 13$

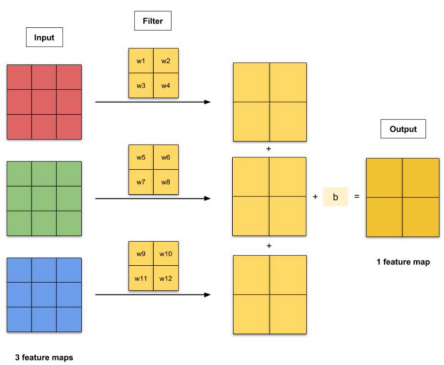
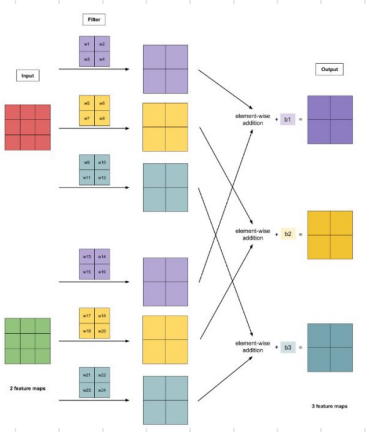


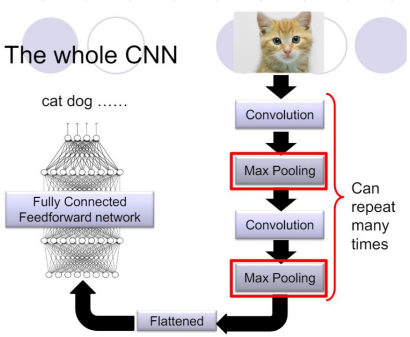
Image with 2 channels, with 2×2 filter, and output of 3 channels

- $i = 2$ (2 channels)
- $f = 2$
- $o = 3$
- *num_params*
  $= [i \times (f \times f) \times 3] + 3$
  $= [2 \times (2 \times 2) \times 3] + 3$
  $= 27$



Pooling Layers

Similar to the Convolutional Layer, the Pooling layer is responsible for
- Reduce dimensionality (height and width) of feature maps.
- Lower computational complexity and control overfitting.



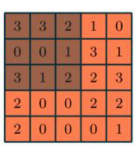
Types of Pooling

- Max Pooling: Retains only the maximum value in a pooling window, suppressing noise.
- Min Pooling: Retains the minimum value (rarely used).
- Average Pooling: Retains the average value, smoothing feature maps.

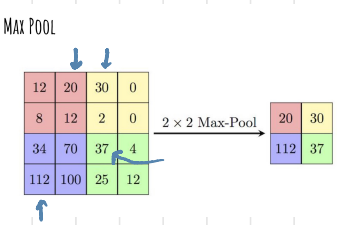
Max Pool

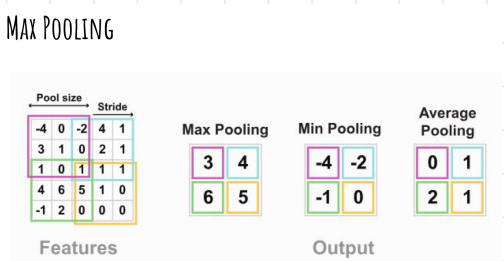
Max Pooling also performs as a Noise Suppressant.

It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.



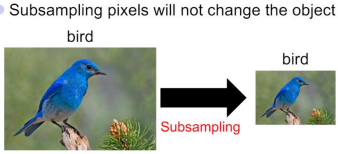
- A pooling window (e.g., 2x2) slides across the feature map, taking the maximum value in each region.
- Result: reduced spatial dimension, preserving the strongest activations.





Why Pooling

- Subsampling pixels will not change the object



We can subsample the pixels to make image smaller

fewer parameters to characterize the image

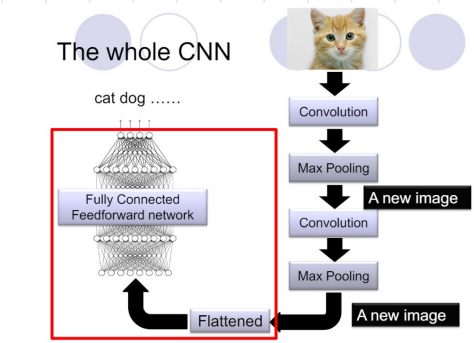
CNN

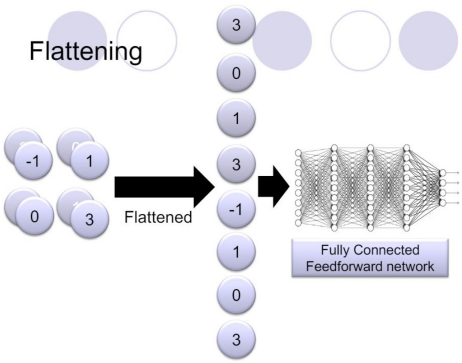
A CNN compresses a fully connected network in following ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

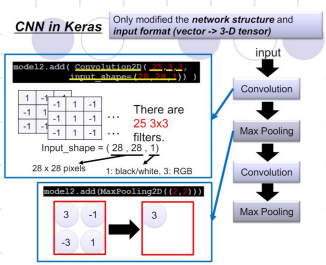
Three Fundamental CNN Concepts:

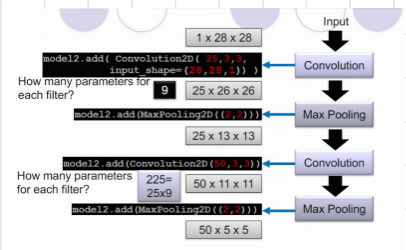
- Local receptive fields: Individual neurons respond to specific areas, capturing local dependencies.
- Shared weights: Reduces the number of parameters by reusing filter weights across the image.
- Spatial/Temporal sub-sampling: Reduces the feature map size through pooling, maintaining important information while reducing complexity.

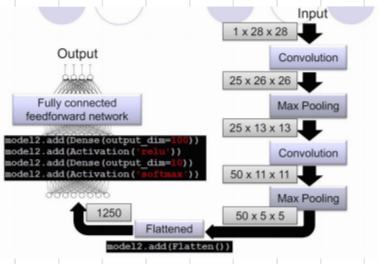




CNN in keras



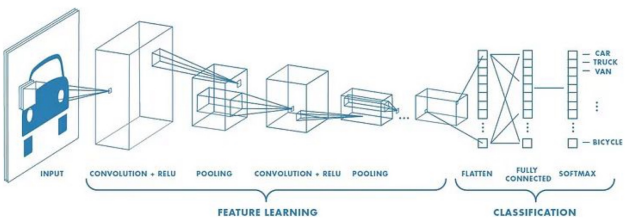




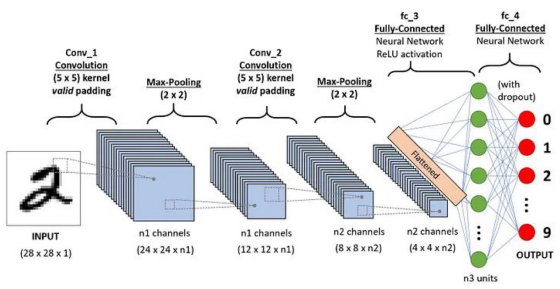
Practical Summary (What Happens in CNN training):

- Initialize filters randomly.
- Apply convolution to create initial feature maps.
- Apply activation functions (e.g., ReLU).
- Use pooling layers to reduce feature dimensions.
- Connect resulting features to fully connected layers for final classification.
- Optimize filter weights through backpropagation to improve feature detection capabilities and predictions.

CNN



CNN



Conclusion:

- CNNs are a powerful type of neural network optimized for handling images by exploiting spatial relationships, significantly outperforming traditional fully connected neural networks in visual recognition tasks.

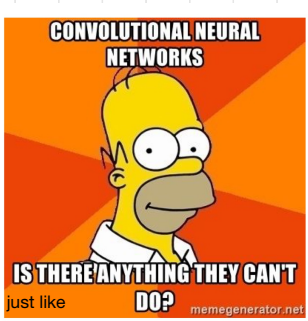
what are spacial features?

to the way pixels are arranged and how they relate to one another in terms of position.

e.g

- An edge is a transition between light and dark pixels in close proximity.
- A face in a photo has eyes above the nose, and the nose above the mouth — that's a spatial relationship between facial features.
- In a photo of a cat, the ears are usually at the top of the head — their position relative to other features matters.

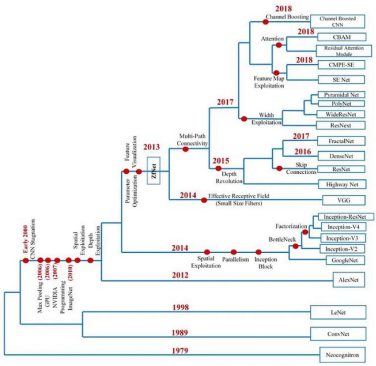
CNNs use this spatial information to understand shapes, edges, textures, and ultimately objects — just like our brains do.



CNN ARCHITECTURES

CNN Architectures
CNN Architectures refer to various structured designs of convolutional neural networks tailored to perform specific tasks efficiently (classification, detection, segmentation).
Key architectures discussed in the slides are:

CNN ARCHITECTURES

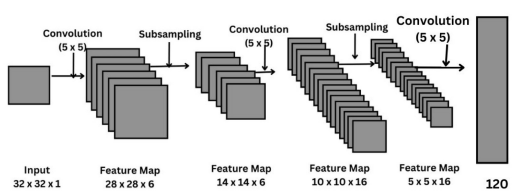


CNN ARCHITECTURES

DenseNet
Xception
MobileNet
ResNext
ZFNet

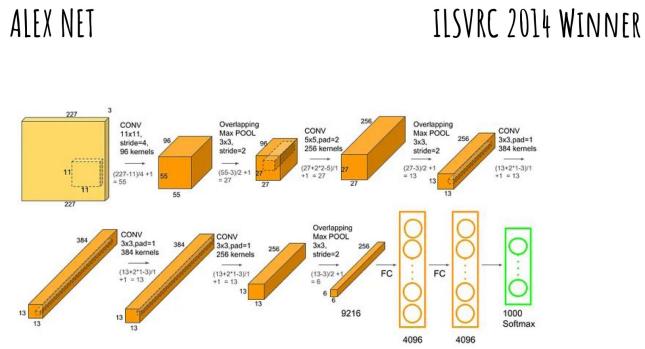
1. LeNet
- Early CNN architecture developed by Yann LeCun.
- Initially created for recognizing handwritten digits.
- Composed of convolutional layers, pooling layers, and fully connected layers.

LENET



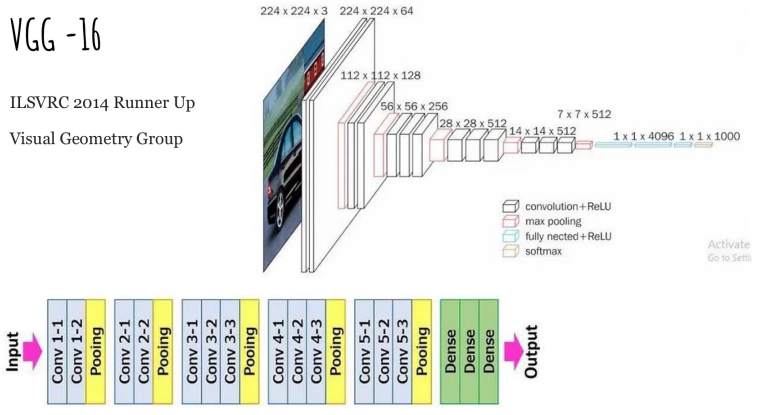
2. AlexNet (ILSVRC 2012 Winner)

- Significantly deeper and larger CNN than LeNet.
- Introduced ReLU activation function to tackle vanishing gradients.
- First CNN to leverage GPU acceleration.
- Achieved breakthrough performance on ImageNet classification.



3. VGG-16 (ILSVRC 2014 Runner-up)

- Characterized by its simplicity and uniform structure.
- Uses small (3x3) convolution filters stacked to increase depth.
- 16 layers (13 convolutional, 3 fully connected).

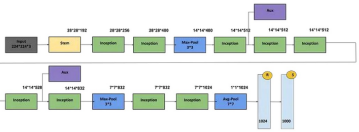


4. GoogleNet (ILSVRC 2014 Winner)

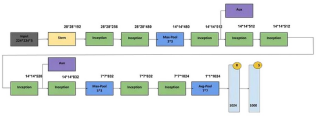
- Introduced the Inception module, significantly improving efficiency.
- Features parallel convolutions of different sizes (1x1, 3x3, 5x5), concatenating outputs.
- Reduces dimensionality with Stem Block (initial layer sequence reducing size and complexity) and multiple stacked Inception Blocks.

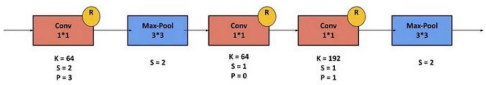
GoogleNet

ILSVRC 2014 Winner



GoogleNet - Stem Block

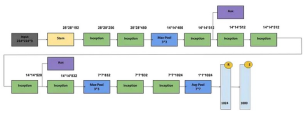


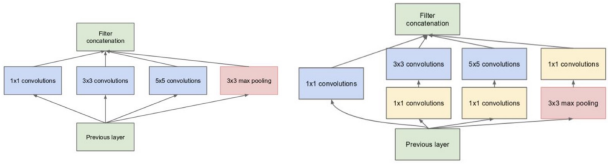


GoogleNet - Inception Block

Inception Block Diagram:

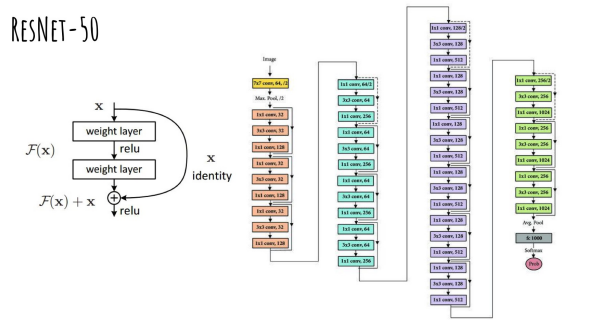
- Parallel branches process input differently (1x1, 3x3 convolutions, pooling).
- Outputs concatenated into a deeper, more powerful representation.





5. ResNet-50

- Introduced the concept of Residual Connections (skip connections).
- Addresses the vanishing gradient problem, enabling extremely deep networks (50+ layers).
- Skip connections allow layers to learn residual functions, significantly improving accuracy.



6. Inception-ResNet

- Hybrid architecture combining GoogleNet's inception modules with ResNet's skip connections.
- Exploits both width (parallel convolutions) and depth (residual connections).
- Enhanced feature extraction capabilities.

Inception - Resnet

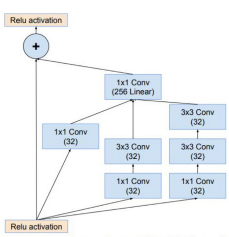


Figure 10. The schema for 35×35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.

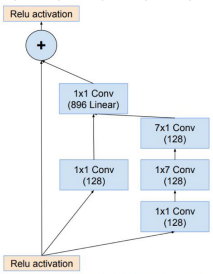


Figure 11. The schema for 17×17 grid (Inception-ResNet-B)

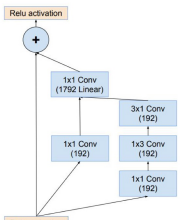


Figure 13. The schema for 8×8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network.

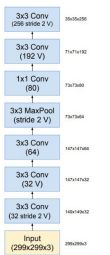


Figure 14. The stem of the Inception-ResNet-v1 ne

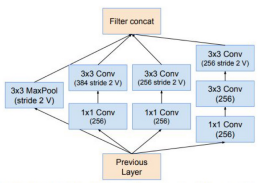


Figure 12. "Reduction-B" 17×17 to 8×8 grid-reduction module. This module used by the smaller Inception-ResNet-v1 network in Figure 15.

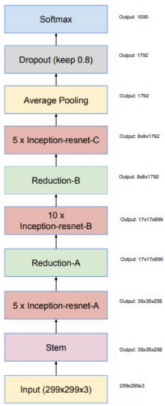
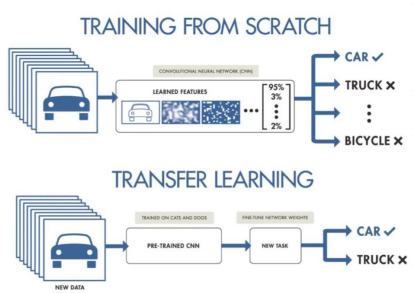


Figure 15. Schema for Inception-ResNet-v1 and Inception-ResNet-v2 networks. This schema applies to both networks but the underlying components differ. Inception-ResNet-v1 uses the blocks as described in Figures 14, 10, 7, 11, 12 and 13. Inception-ResNet-v2 uses the blocks as described in Figures 3, 16, 7,17, 18 and 19. The output sizes in the diagram refer to the activation vector tensor shapes of Inception-ResNet-v1.

Transfer Learning

Applying knowledge learned from solving one task to a different but related task.

Examples:
Using knowledge of riding a bicycle to ride a motorcycle.



Approaches in Transfer Learning

Feature Extracation:
Freeze convolutional layers of a pretrained model.
Retrain only the fully connected layers on new dataset.
Useful when datasets are similar.

Fine Tuning:
Unfreeze the last convolutional layers and fully connected laver + output layer.
Retrain these layers alongside fully connected layers.
Useful when datasets or labels are significantly different.

For deep learning models, the data needs to be labelled, and if the dataset is not publicly available, then we have to manually label the data which is time consuming.

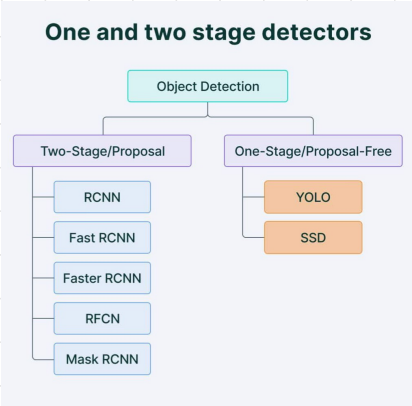
Deep learning models takes a lot of time to get trained.

Pre-trained models: VGG16, ResNet, MobileNet, EfficientNet

Object Detection

Detecting and locating objects within images or videos.

Applications:
Surveillance, self-driving cars, robotics



Types of Object Detection

1. Single Shot
2. Two Shot Detection

1. Single-Shot Detection:

- Processes images in one pass.
- Faster and efficient, suitable for real-time
- applications.
  Example: YOLO (You Only Look Once).

2. Two-Shot Detection:

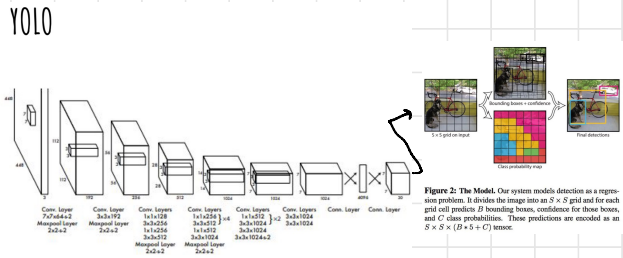
- Two-step process:
  - ↳ First pass: Generate candidate regions (proposals).
  - ↳ Second pass: Classify and refine these proposals.
- More accurate but computationally heavier.
- Example: R-CNN, Faster R-CNN.

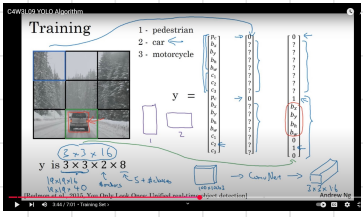
Potential object locations

However, single-shot object detection is generally less accurate than other methods, and it's less effective in detecting small objects. Such algorithms can be used to detect objects in real time in resource-constrained environments.

YOLO Diagram Explained:

Input image passed once through the network.
Outputs bounding boxes with class probabilities.





IoU (Intersection over Union):

- Metric measuring overlap between predicted bounding box and ground-truth.
- Ranges between
  - ↳ 0 (no overlap) and
  - ↳ 1 (perfect overlap).
- Higher IoU = better detection accuracy.

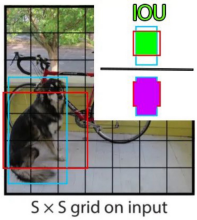
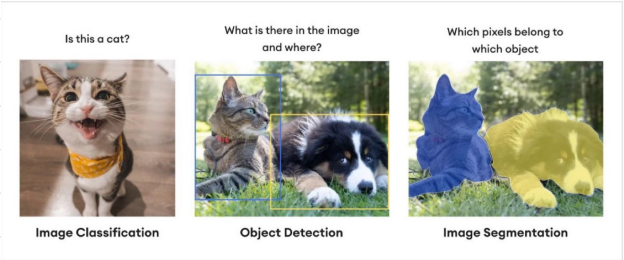


Image Segmentation

- Assigning labels to every pixel of an image.
- Pixels with the same label share certain characteristics.



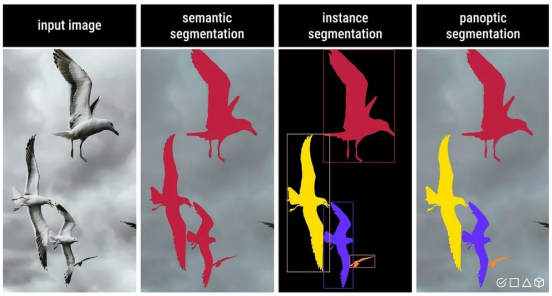
Types of image segmentation

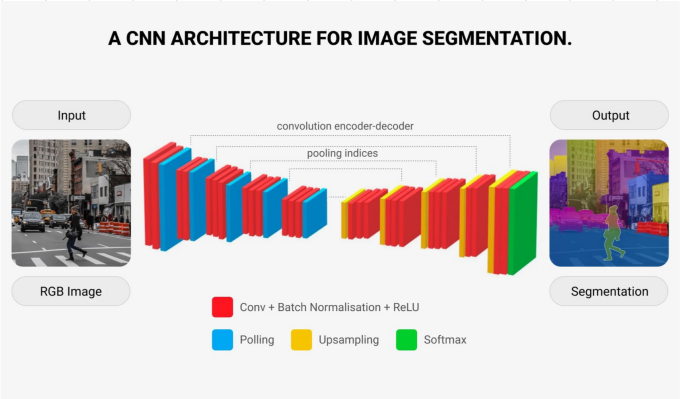
Semantic Segmentation:
Labels each pixel according to its class
(all birds belong to the same class).

Instance Segmentation:
Assigns a unique label to each instance of an object
(each bird belongs to a different class).

Panoptic Segmentation:
Combines semantic and instance segmentation.
Each instance gets a unique label, grouped under semantic categories.
(each bird has its own class, but they're all identified as a bird)

Image segmentation

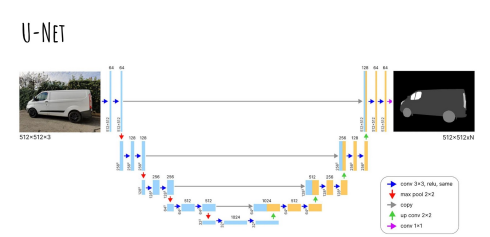




Key Segmentation Architectures

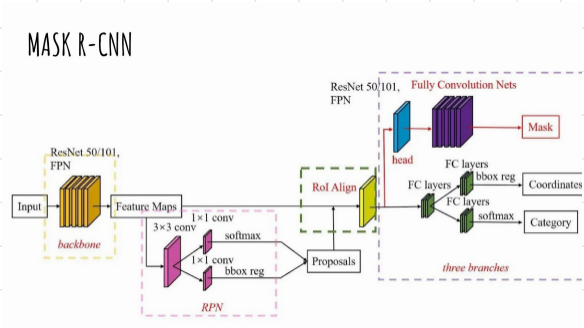
U-Net:

- Primarily used in biomedical image segmentation.
- Encoder-decoder structure.
- Symmetrical, with skip connections copying feature maps directly from encoder to decoder.



Mask R-CNN:

- Extends Faster R-CNN for instance segmentation.
- Adds a parallel branch predicting segmentation masks for each detected instance.
- Highly accurate but computationally demanding.

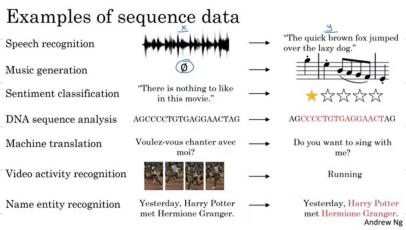


RNNs

8/

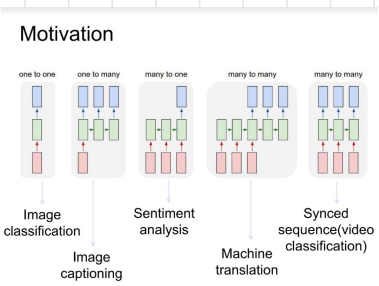
Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are designed specifically for handling sequential data, such as text, speech, or time series data.
Unlike standard neural networks, RNNs have loops that allow information to persist across sequential inputs.



Common applications of RNNs include:

- Generating Text: Predict next words or characters in a sentence.
- Machine Translation: Translating text from one language to another.
- Speech Recognition: Transcribing spoken language into text.
- Generating Image Descriptions: Describing images with meaningful captions.
- Chatbots: Conversational AI capable of generating relevant responses.

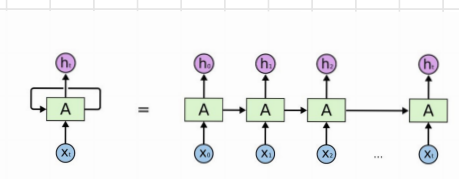


Core Idea of RNNs

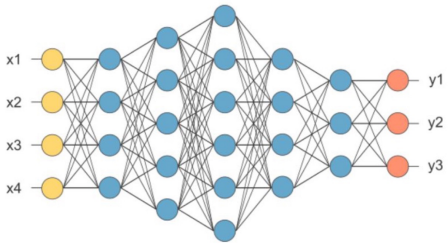
- Standard feed-forward neural networks process inputs independently.
- RNNs maintain hidden states that carry forward information from previous inputs, allowing them to remember context.

Key Components:

- Input Layer: Receives input at each time step.
- Hidden Layer(s): Maintain states that capture temporal dependencies.
- Output Layer: Produces output at each step or at the end of the sequence.

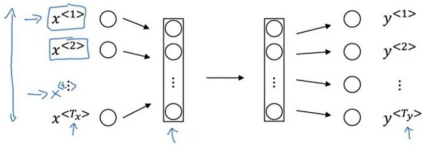


Fully Connected Network



This is our fully connected network. If x_1 ... x_n, n is very large and growing, this network would become too large. We now will input one x_i at a time, and re-use the same edge weights.

Why not a standard network?



Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

Andrew

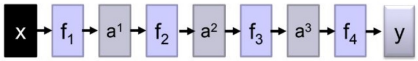
Feedforward Neural Network (FNN):
Processes inputs independently without any memory.
Each input flows one way: input → hidden layer(s) → output.
No awareness of sequence or context.
Example use: Image classification.

Recurrent Neural Network (RNN):
Designed for sequential data.
Has loops—each output depends not just on the current input but also on past hidden states.
Retains memory across time steps via hidden states (h_t depends on h_{t-1}).
Example use: Language modeling, time series prediction.

Feed-forward vs Recurrent Network

1. Feedforward network does not have input at each step
2. Feedforward network has different parameters for each layer

FF

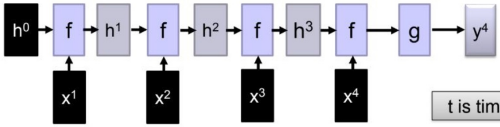


Straight flow from input → layers → output

$a^t = f_t(a^{t-1}) = \sigma(W^t a^{t-1} + b^t)$

t is layer

RNN



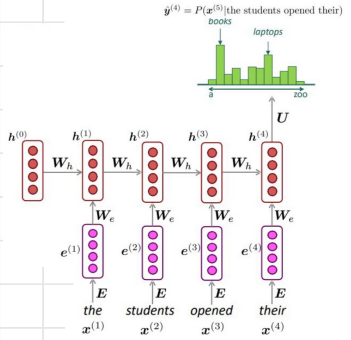
Looped architecture where output and hidden state from previous time step feed into the next

t is time step

$h^t = f(h^{t-1}, x^t) = \sigma(W^h h^{t-1} + W^i x^t + b^i)$

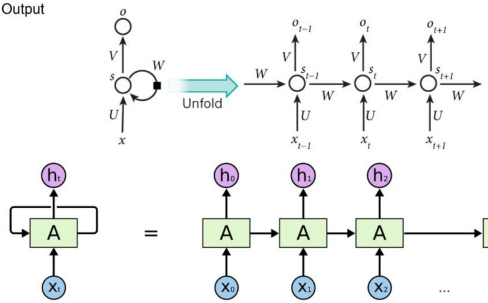
RMN Advantages:
• Can process any length input
• Computation for step t can (in theory) use information from many steps back
• Model size doesn't increase for longer input context
• Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:
• Recurrent computation is slow
• In practice, difficult to access information from many steps back



Recurrent Neural Network

Input → Hidden → Output
it becomes
Input + Previous Hidden → Hidden → Output

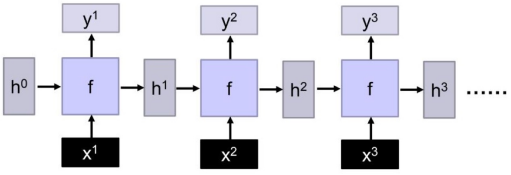


The recurrent loop is formed because hidden state from one step (h_{t-1}) is passed to the next step (h_t).

How does RNN reduce complexity?

- Given function f: h',y=f(h,x)

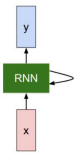
h and h' are vectors with the same dimension



No matter how long the input/output sequence is, we only need one function f. If f's are different, then it becomes a feedforward NN. This may be treated as another compression from fully connected network.

Simple RNN

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

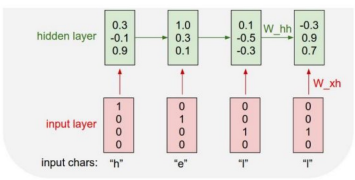
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

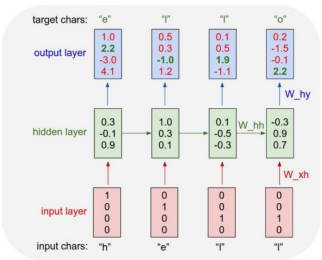
Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**

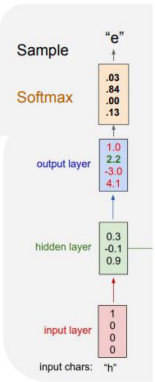
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

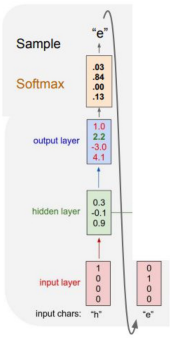


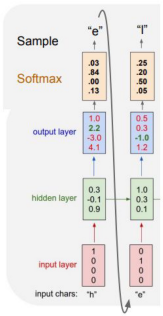


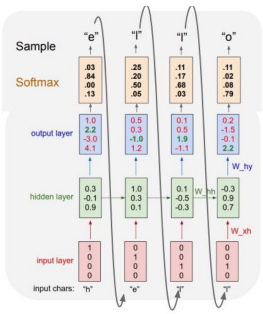
?

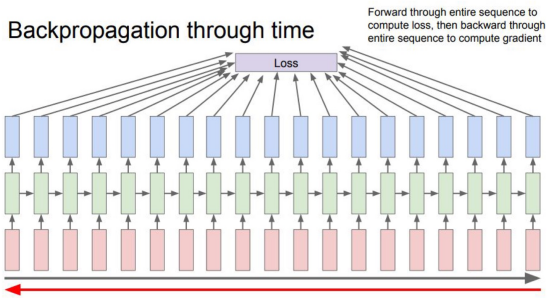
At test-time sample characters one at a time, feed back to model

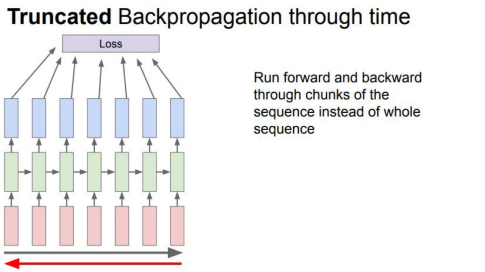


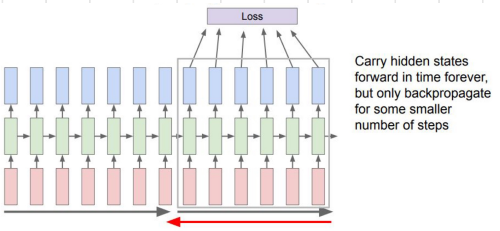


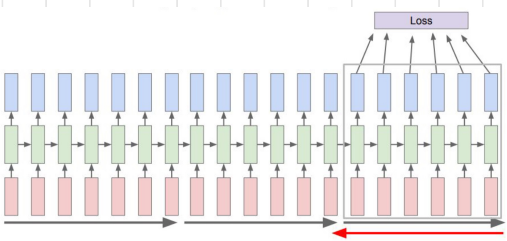












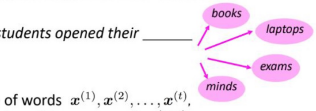
Limitations of Simple RNNs

Vanishing Gradients:
Difficulty in learning long-range dependencies due to gradients becoming very small (close to zero) over many time steps.
Exploding Gradients:
Gradients can become excessively large, causing unstable learning.

slow training
difficulty in handling very long seq

Language Modeling

- **Language Modeling** is the task of predicting what word comes next



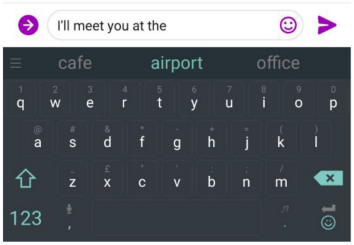
- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$

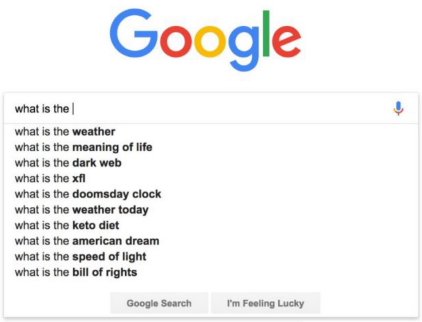
$$P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**

You use Language Models every day!





A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

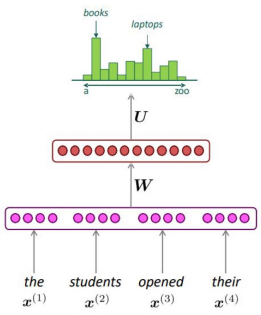
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$

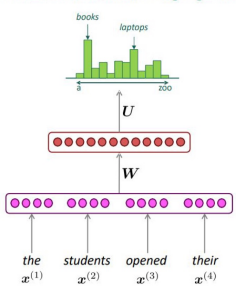


Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

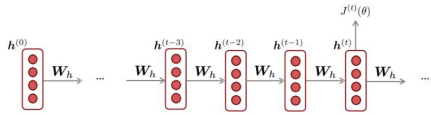
Improvements over *n*-gram LM:
- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:
- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- $x^{(1)}$ and $x^{(})$ are multiplied by completely different weights in $\boldsymbol{W}$. No symmetry in how the inputs are processed.

Need a neural architecture that can process *any length input*



Backpropagation for RNNs



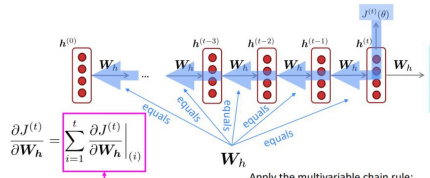
Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix W_h ?

Answer: $\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$

"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Why?

Training the parameters of RNNs: Backpropagation for RNNs



In practice, often "truncated" after ~20 timesteps for training efficiency reasons

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$$

Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)} \frac{\partial W_h|_{(i)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$$

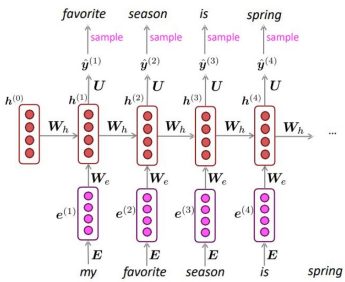
($\frac{\partial W_h|_{(i)}}{\partial W_h} = 1$)

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i = t, \ldots, 0$, summing gradients as you go. This algorithm is called **"backpropagation through time"** [Werbos, P.G., 1988, *Neural Networks* **1**, and others]

Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output becomes next step's input.

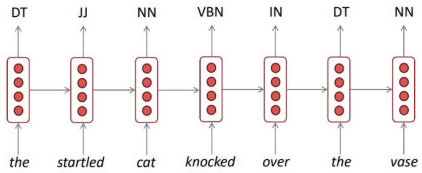


Recap

- **Language Model**: A system that predicts the next word
- **Recurrent Neural Network**: A family of neural networks that:
  - Take sequential input of any length
  - Apply the same weights on each step
  - Can optionally produce output on each step
- Recurrent Neural Network ≠ Language Model
- We've shown that RNNs are a great way to build a LM.
- But RNNs are useful for much more!

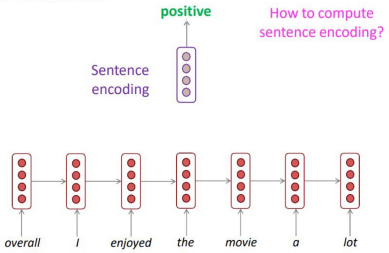
RNNs can be used for tagging

e.g., **part-of-speech tagging**, named entity recognition



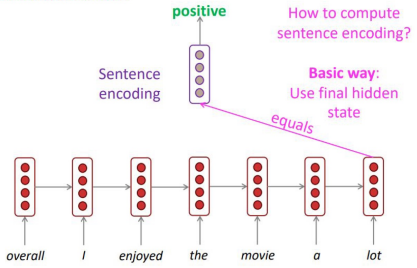
RNNs can be used for sentence classification

e.g., **sentiment classification**



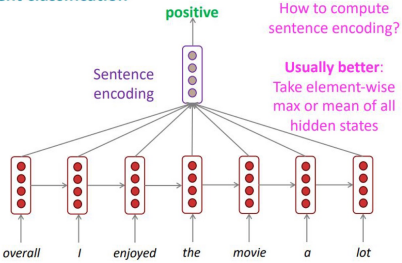
RNNs can be used for sentence classification

e.g., **sentiment classification**



RNNs can be used for sentence classification

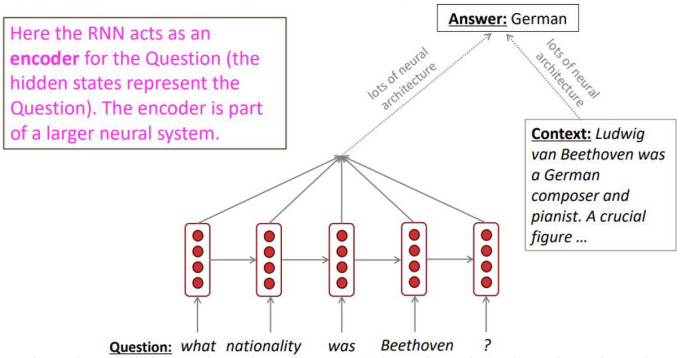
e.g., **sentiment classification**



RNNs can be used as an encoder module

e.g., **question answering**, machine translation, *many other tasks!*

Here the RNN acts as an **encoder** for the Question (the hidden states represent the Question). The encoder is part of a larger neural system.



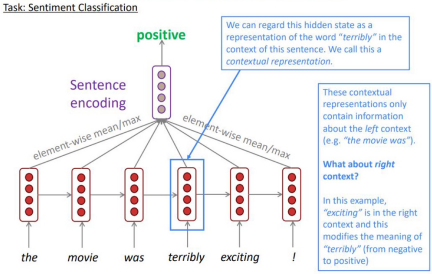
Bidirectional RNNs

Standard RNNs only consider past information (left-to-right).
Bidirectional RNNs (BiRNNs) use:
A forward RNN (left-to-right)
A backward RNN (right-to-left)
Then concatenate their outputs.

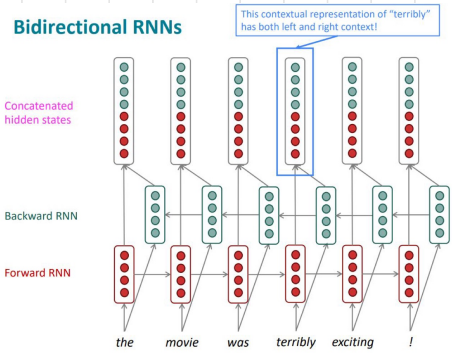
Why?
Some tasks require future context as well as past.
Example: In the word "read," context matters for pronunciation —"I will read" vs. "I have read".

Bidirectional and Multi-layer RNNs: motivation

Task: Sentiment Classification



Bidirectional RNNs



Bidirectional RNNs

On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a simple RNN or LSTM computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

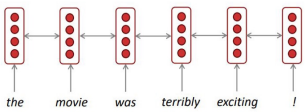
Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

Bidirectional RNNs: simplified diagram



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

Bidirectional RNNs

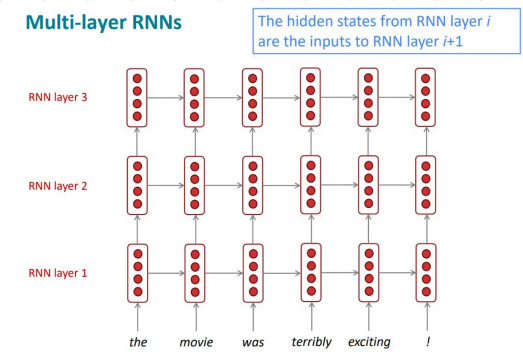
- Note: bidirectional RNNs are only applicable if you have access to the entire input sequence
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.
- If you do have entire input sequence (e.g., any kind of encoding), bidirectionality is powerful (you should use it by default).
- For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.
  - You will learn more about transformers, including BERT, in a couple of weeks!

Multi-layer RNNs (a.k.a. Deep RNNs)

A Multi-layer RNN is simply an RNN where the output from one RNN layer becomes input to another RNN layer.
This enables the network to learn more complex representations.

Pros

Better feature extraction from complex sequential data.
Used in advanced language models and speech recognition systems.



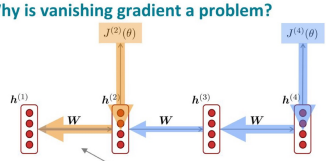
Multi-layer RNNs in practice

- Multi-layer or stacked RNNs allow a network to compute more complex representations – they work better than just have one layer of high-dimensional encodings!
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- High-performing RNNs are usually multi-layer (but aren't as deep as convolutional or feed-forward networks)
- For example: In a 2017 paper, Britz et al. find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - Often 2 layers is a lot better than 1, and 3 might be a little better than 2
  - Usually, skip-connections/dense-connections are needed to train deeper RNNs (e.g., 8 layers)
- Transformer-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers.
  - You will learn about Transformers later; they have a lot of skipping-like connections

LSTM

9

Why is vanishing gradient a problem?



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are basically updated only with respect to near effects, not long-term effects.

How to fix the vanishing gradient problem?

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps.*
- In a vanilla RNN, the hidden state is constantly being rewritten

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b\right)$$

- Could we design an RNN with separate memory which is added to?

Long Short-Term Memory RNNs (LSTMs)

- On step *t*, there is a hidden state $h^{(t)}$ and a cell state $c^{(t)}$
  - Both are vectors length *n*
  - The cell stores long-term information
  - The LSTM can read, erase, and write information from the cell
    - The cell becomes conceptually rather like RAM in a computer
- The selection of which information is erased/written/read is controlled by three corresponding gates
  - The gates are also vectors of length *n*
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between
  - The gates are dynamic: their value is computed based on the current context

Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep *t*:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

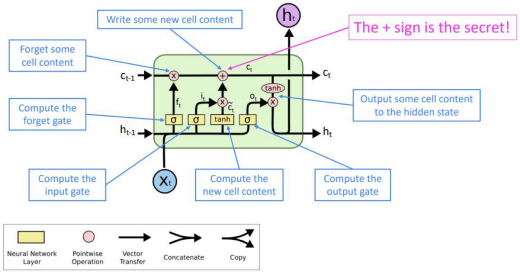
$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$
$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$
$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$
$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$
$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

All these are vectors of same length *n*

Gates are applied using element-wise (or Hadamard) product: $\circ$

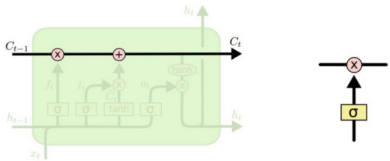
Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

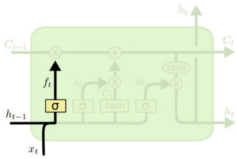


Conveyor Belt

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged



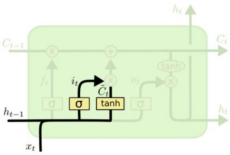
Forget Gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

Input Gate Layer

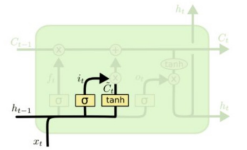


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

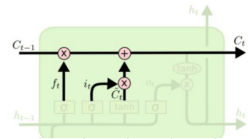
The next step is to decide what new information we're going to store in the cell state
This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update

Next, a tanh layer creates a vector of new candidate values, , that could be added to the state. In the next step, we'll combine these two to create an update to the state

Update Gate Layer + Memory Cell

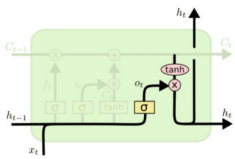


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

How does LSTM solve vanishing gradients?

- The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
  - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
  - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
  - In practice, you get about 100 timesteps rather than about 7

Is vanishing/exploding gradient just an RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (i.e., are hard to train)
- Another solution: lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

For example:

- Residual connections aka "ResNet"
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train

ReLU

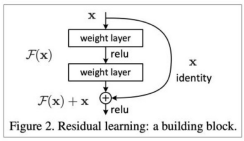


Figure 2. Residual learning: a building block.

LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks

- Now (2019–2023), Transformers have become dominant for all tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
    - In WMT 2014, there were 0 neural machine translation systems (!)
    - In WMT 2016, the summary report contains "RNN" 44 times (and these systems won)
    - In WMT 2019: "RNN" 7 times, "Transformer" 105 times

MACHINE TRANSLATION

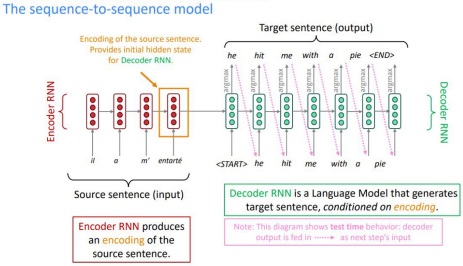
What is Neural Machine Translation?

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single end-to-end neural network*
- The neural network architecture is called a sequence-to-sequence model (aka seq2seq) and it involves *two* RNNs

Machine Translation (MT) is the process of automatically converting text from one language to another using neural networks.

Neural Machine Translation (NMT)

The sequence-to-sequence model



Sequence-to-sequence is versatile!

- The general notion here is an encoder-decoder model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq model
- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

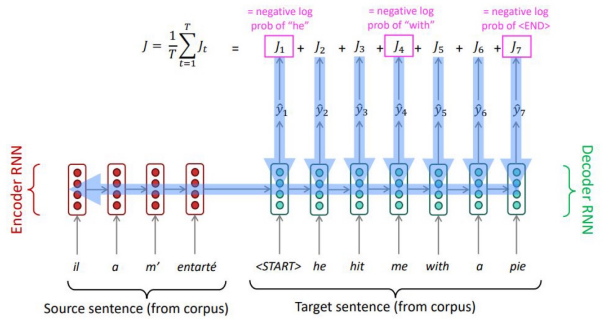
Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*
- NMT directly calculates $P(y|x)$:

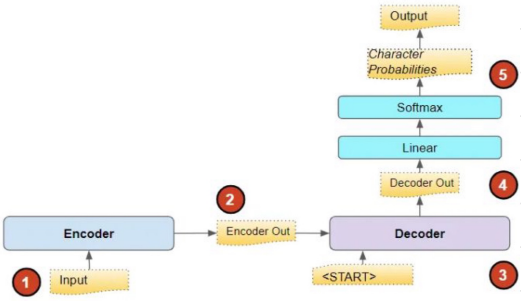
$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}_{\text{Probability of next target word, given target words so far and source sentence } x}$$

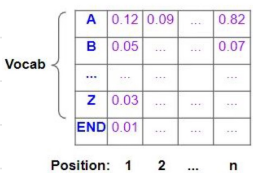
- **Question:** How to train an NMT system?
- **(Easy) Answer:** Get a big parallel corpus…
  - But there is now exciting work on "unsupervised NMT", data augmentation, etc.

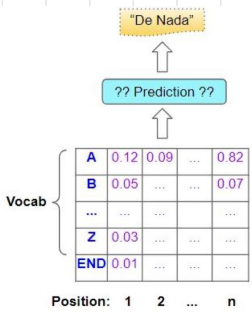
Training a Neural Machine Translation system



Seq2seq is optimized as a **single system.** Backpropagation operates "*end-to-end*".

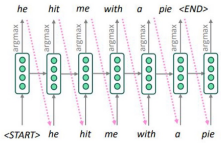






Greedy decoding

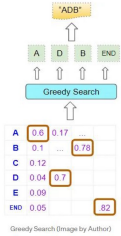
- We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder



- This is greedy decoding (take most probable word on each step)
- **Problems with this method?**

Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - Input: *il a m'entarté* *(he hit me with a pie)*
  - → *he* ____
  - → *he hit* ____
  - → *he hit a* ____ *(whoops! no going back now...)*



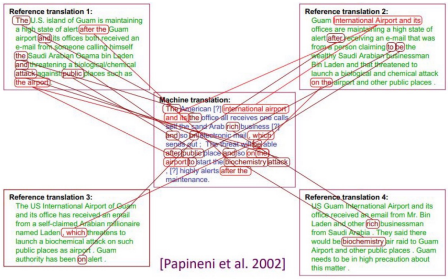
Greedy Search (Image by Author)

How do we evaluate Machine Translation?

BLEU (**Bi**lingual **E**valuation **U**nderstudy)

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations
- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation ☹

BLEU score against 4 reference translations



[Papineni et al. 2002]

BLEU Score (Bilingual Evaluation Understudy)
A widely used evaluation metric in machine translation for comparing a predicted sentence to a reference sentence.

How BLEU Score Works:
Compares n-grams (e.g., unigrams, bigrams) between the predicted and reference sentences.
Uses clipped precision to prevent overcounting repeated words.

```
Reference Text: It was raining heavily today

Predicted Text: It It It is raining heavily
```

candidate

Unigram Precision:
Total predicted unigrams: 6 → 6 Predicted Words
Matching unigrams from reference: 3
(only one "It", "raining", and "heavily" count)
Clipped precision = 3 / 6 = 0.5

Bigram Precision:
Reference bigrams:
["It was", "was raining", "raining heavily", "heavily today"]
Predicted bigrams:
["It It", "It It", "It is", "is raining", "raining heavily"]
Matching bigrams: only "raining heavily"
Clipped precision = 1 / 5 = 0.2

Final BLEU Score Calculation:
BLEU=Brevity Penalty×Precision Score
Where:
r = reference length = 5
c = candidate length = 6
Brevity Penalty (BP) = 1 (since candidate is longer)
Combined score (from example) ≈ 0.316

Usually we use N=4 and $w_n = 1/4$, but in this case we will use N=2 and $w_n = 1/2$

$$Brevity\ Penalty = \begin{cases} 1, & \text{if c>r} \\ e^{(1-r/c)}, & \text{if c<=r} \end{cases}$$

```
r = number of words in the reference text
c = number of words in the predicted text

Brevity penalty cannot be larger than 1, even if the predicted text
is larger than the reference text.
r = 5 c = 6
Bleu = 0.316
```

NMT: perhaps the biggest success story of NLP Deep Learning?

Neural Machine Translation went from a fringe research attempt in **2014** to the leading standard method in **2016**

- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT – and by 2018 everyone has

Microsoft SYSTRAN Google facebook
Bai du 百度 网易 NetEase 有道 Tencent 腾讯 搜狗搜索

- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a small group of engineers in a few months

NMT research continues

NMT is a **flagship task** for NLP Deep Learning

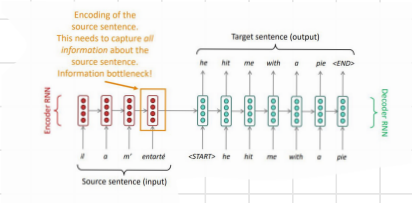
- NMT research has pioneered many of the recent innovations of NLP Deep Learning
- NMT research continues to thrive
  - Researchers have found *many, many* improvements to the "vanilla" seq2seq NMT system we've just presented
  - But we'll present next one improvement so integral that it is the new vanilla…

ATTENTION

ATTENTION MECHANISM

Why?
RNNs process sequences in order and encode the input into a single fixed-length context vector, which is limiting for long sentences.

Solution:
Attention allows the decoder to focus on different parts of the input sequence at each time step.
It helps the model decide "what to look at" while translating each word.



Attention

- **Attention** provides a solution to the bottleneck problem.
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

Attention is a *general* Deep Learning technique

- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

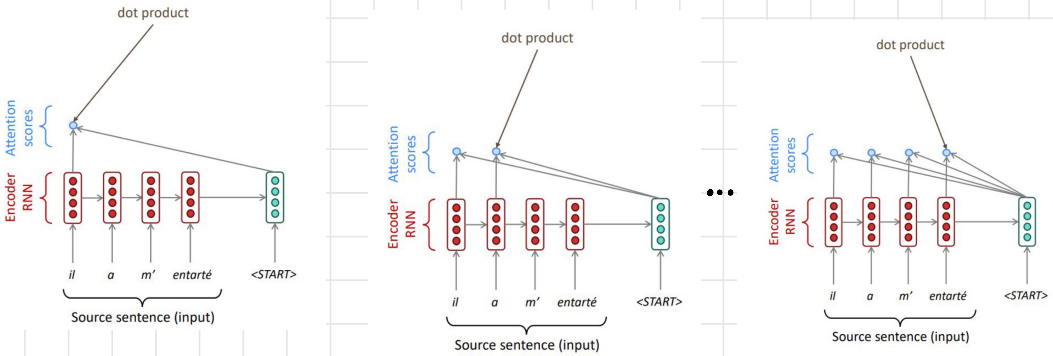
Intuition:

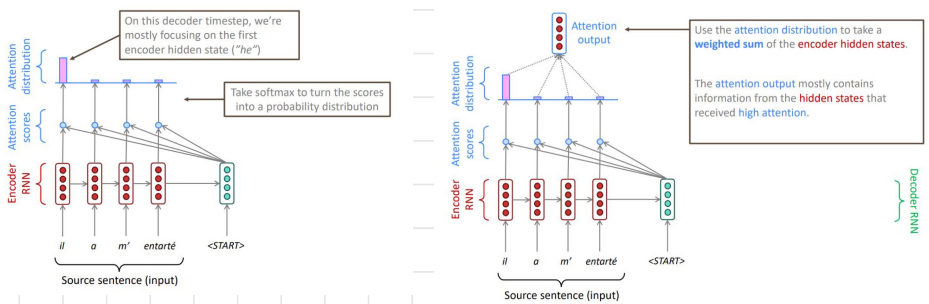
- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

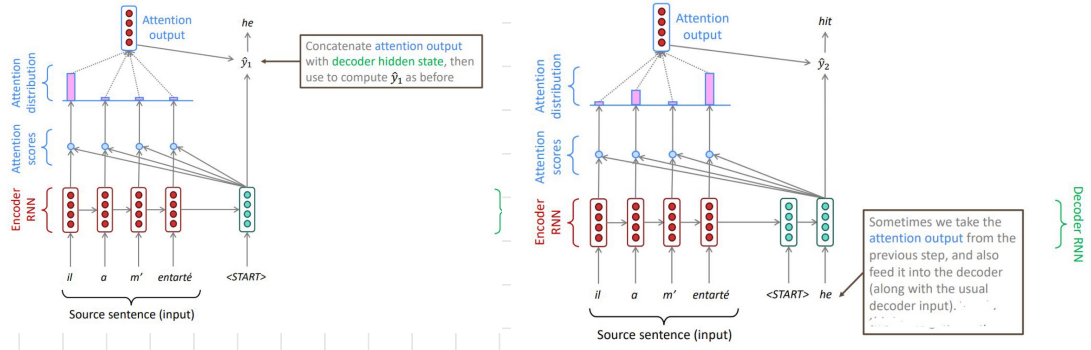
Upshot:

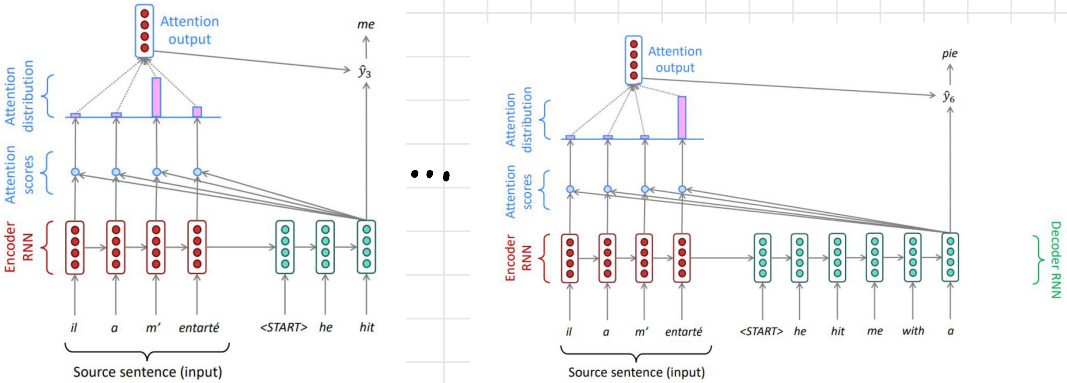
- Attention has become the powerful, flexible, general way pointer and memory manipulation in all deep learning models. A new idea from after 2010! From NMT!

Sequence-to-sequence with attention



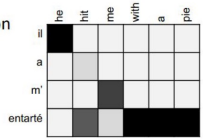


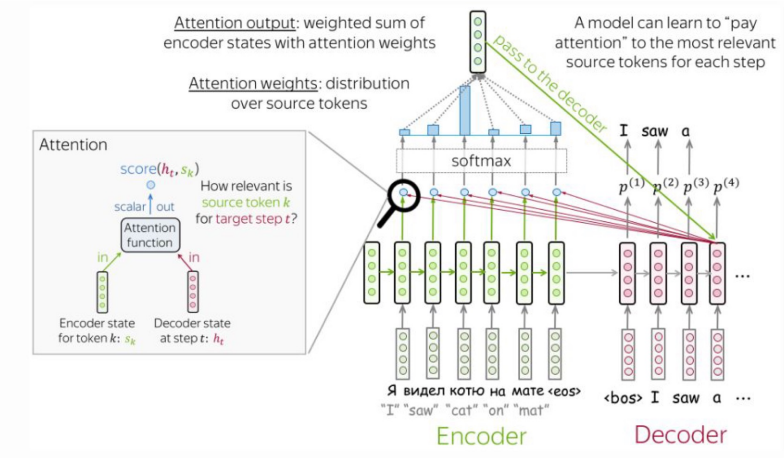




Attention is great!

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides more "human-like" model of the MT process
  - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with the vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself





Attention output $c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \cdots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$

"source context for decoder step t"

(weighted sum)

Attention weights $a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \text{k} = 1..\text{m}$

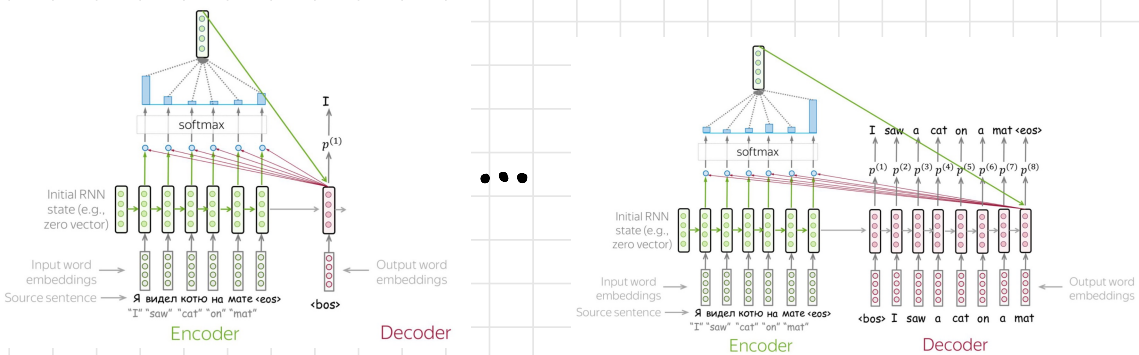
"attention weight for source token k at decoder step t"

(softmax)

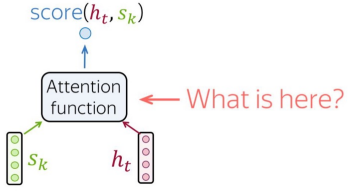
Attention scores $\text{score}(h_t, s_k), \text{k} = 1..\text{m}$

"How relevant is source token k for target step t?"

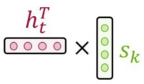
Attention input $s_1, s_2, \ldots, s_m$ (all encoder states) h_t (one decoder state)



How to Compute Attention Score?



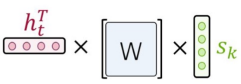
1. Dot-product



score(h_t, s_k) = $h_t^T s_k$

Fast and easy to implement.
Used in basic attention models.

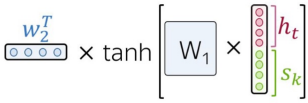
2. Bilinear



score(h_t, s_k) = $h_t^T W s_k$

bilinear function (aka "Luong attention")

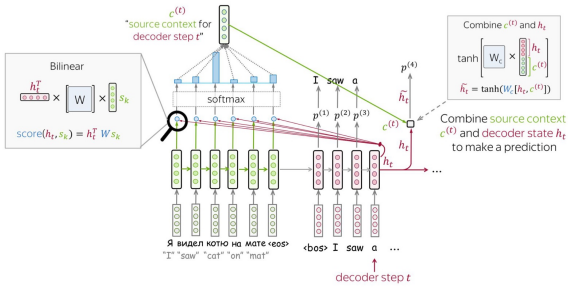
3. Multi-Layer Perceptron



score(h_t, s_k) = $w_2^T \cdot \tanh(W_1[h_t, s_k])$

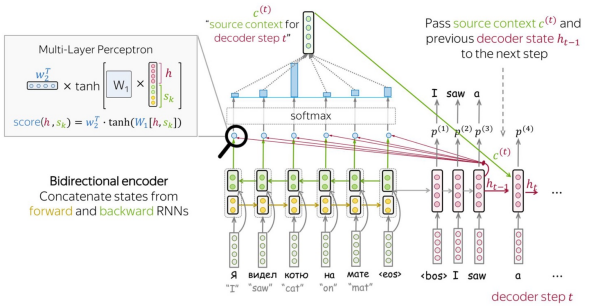
multi-layer perceptron (aka "Bahdanau attention") - the method proposed in the original paper

Luong Model



Luong Model:
Also computes attention, but assumes the decoder already has full context and computes attention post hoc.
Uses dot or bilinear scoring.
More efficient, works well with longer sequences.

Bahdanau Model



Bahdanau Model:
Encoder produces hidden states for each input token.
Attention scores are computed for all encoder states.
Context vector is formed and passed to the decoder.
The decoder predicts the next word using this context + its own state.

Key Point: Alignment is learned jointly with translation.

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ ← There are multiple ways to do this
  2. Taking softmax to get *attention distribution* α:

$$\alpha = \text{softmax}(\boldsymbol{e}) \in \mathbb{R}^N$$

  3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

thus obtaining the *attention output* $\boldsymbol{a}$ (sometimes called the *context vector*)