Lecture 9:
Deep Learning
Learning Theory
Fall 2022

Kai-Wei Chang
CS @ UCLA
kw+cm146@kwchang.net

The instructor gratefully acknowledges Dan Roth, Vivek Srikuar, Sriram Sankararaman, Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

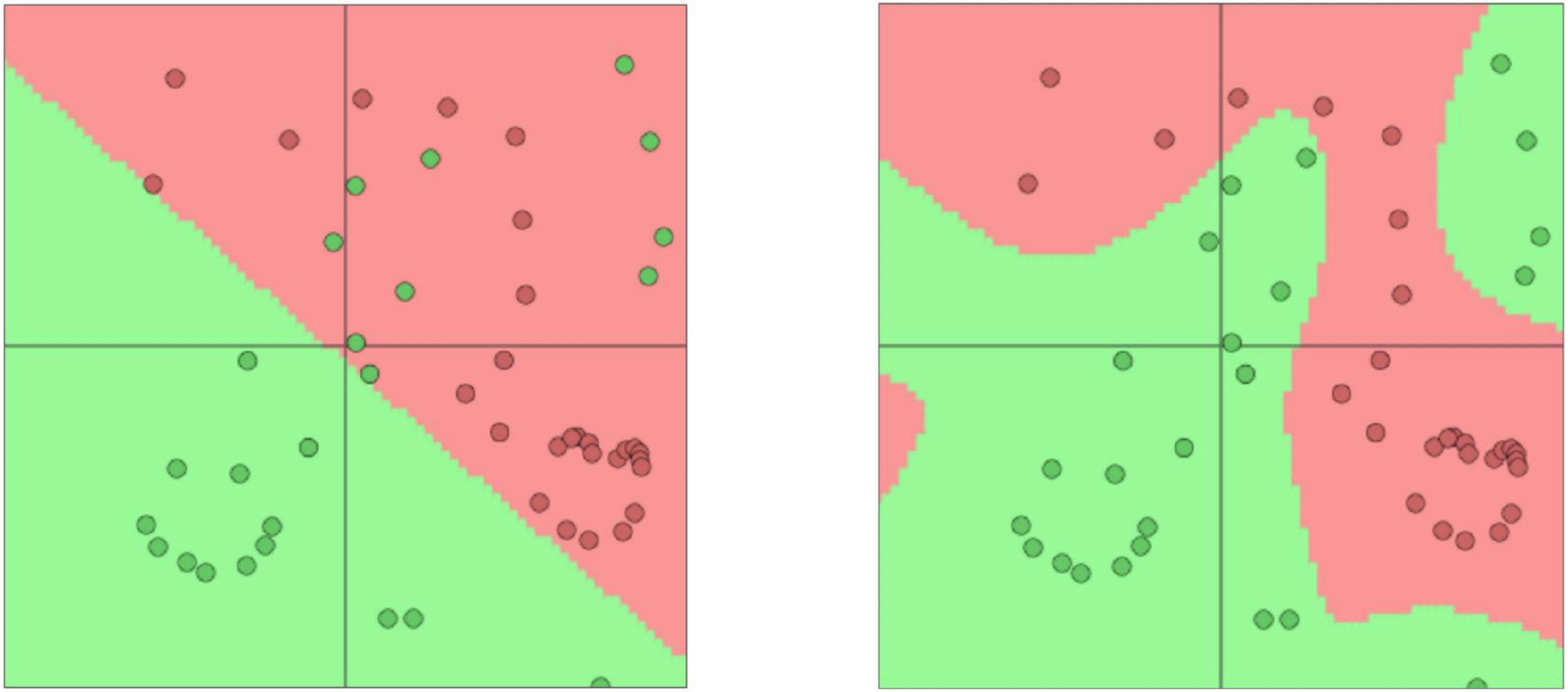
Announcements

- Midterm postpones to 11/3
  - The practice exam is posted
- Hw1 is due next Tue!
- Hw 2 & Quiz 3 will be released tomorrow

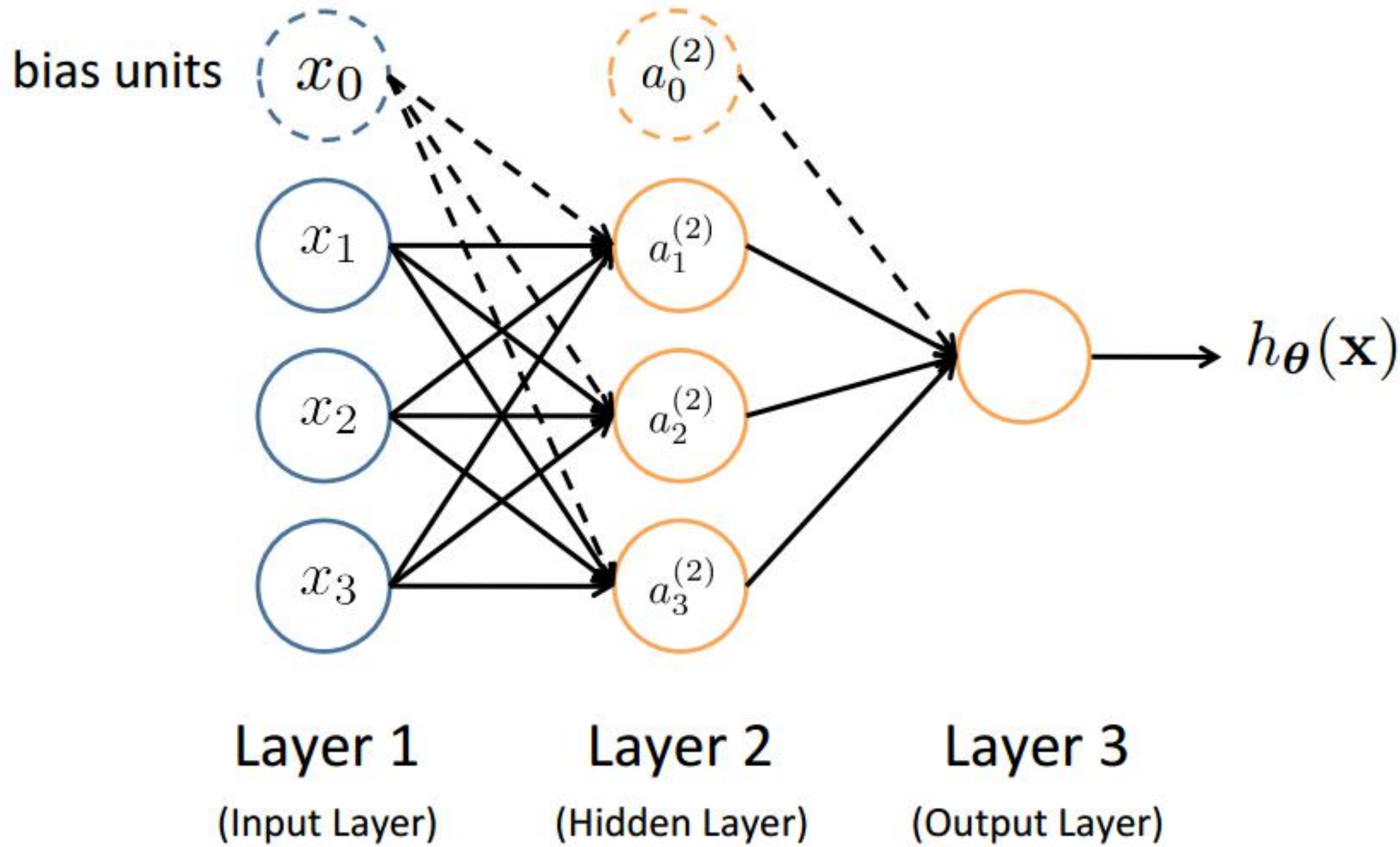
What you will learn today

- Neural network / Deep learning
  - Non-linear classifier
  - Feed-forward neural network
  - Back Propagation
  - Deep learning architecture

Non-Linear Decision Boundary

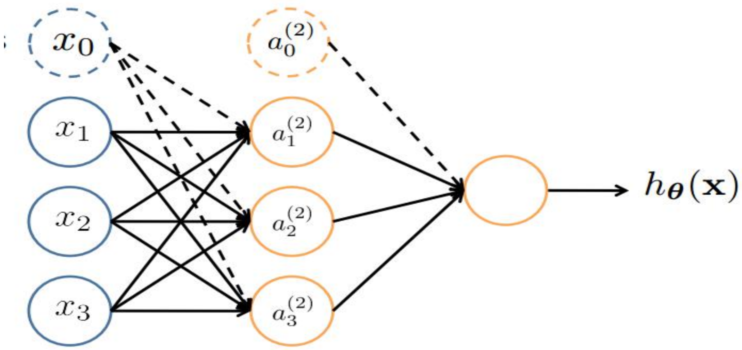


Neural Network



Slide by Andrew Ng

Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

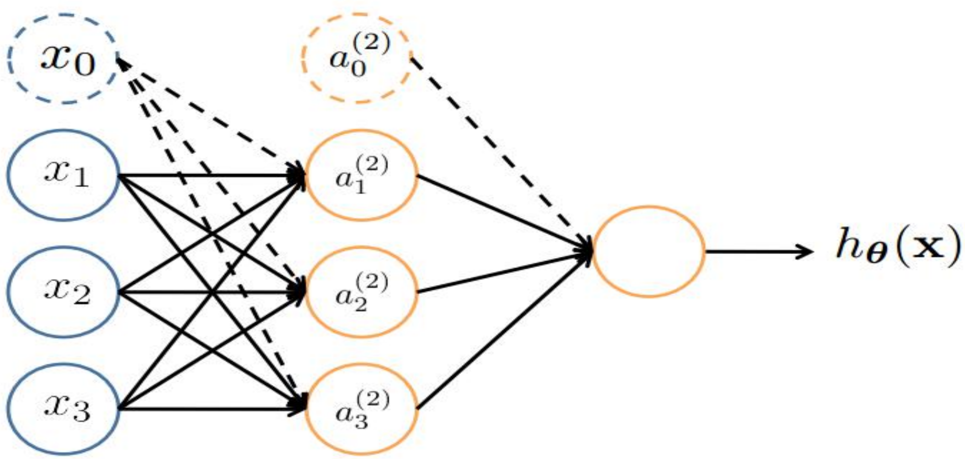
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

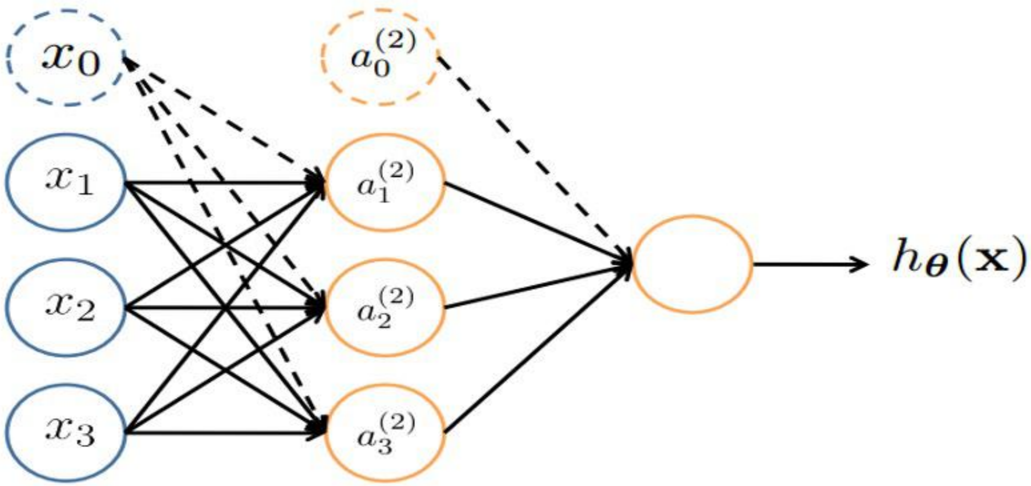
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Example



Feed-Forward Steps:

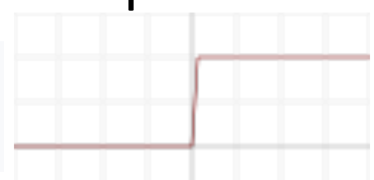
$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$
$$\text{Add } a_0^{(2)} = 1$$
$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Let $\Theta^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \\ 2 & 0 & 1 & 1 \end{bmatrix}$

$\Theta^{(2)} = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$

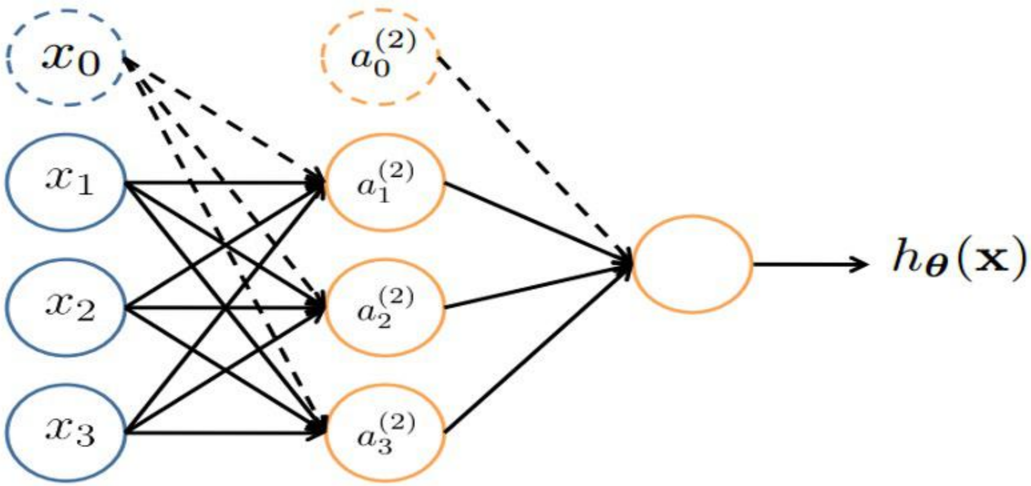
$g(z)$ is a step function

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



What is the output of $x = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix}$?

Example



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

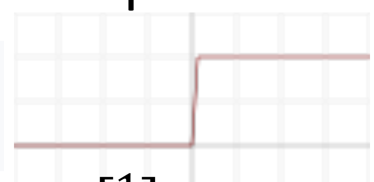
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Let $\Theta^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \\ 2 & 0 & 1 & 1 \end{bmatrix}$

$\Theta^{(2)} = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$

$g(z)$ is a step function

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



What is the output of $x = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix}$?

$z^{(2)} = \begin{bmatrix} 2 \\ 2 \\ 5 \end{bmatrix}$ $a^{(2)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ $z^{(3)} = 2$ $h_\Theta(x) = 1$

Exercise

- Why do we need non-linear activation functions?
  - What happen if $g(z) = z$

Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

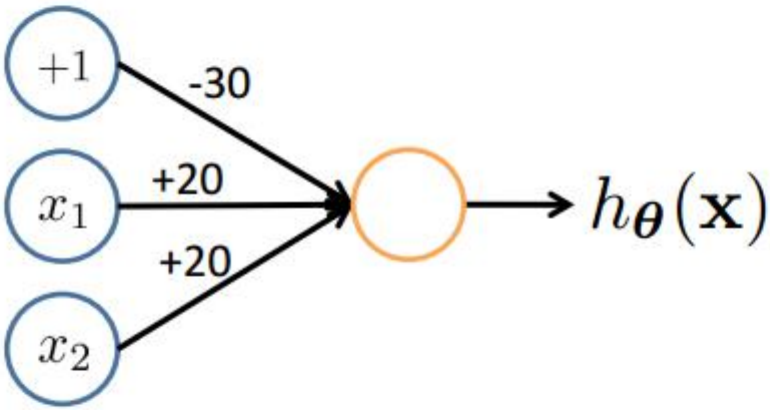
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Non-Linear Representations

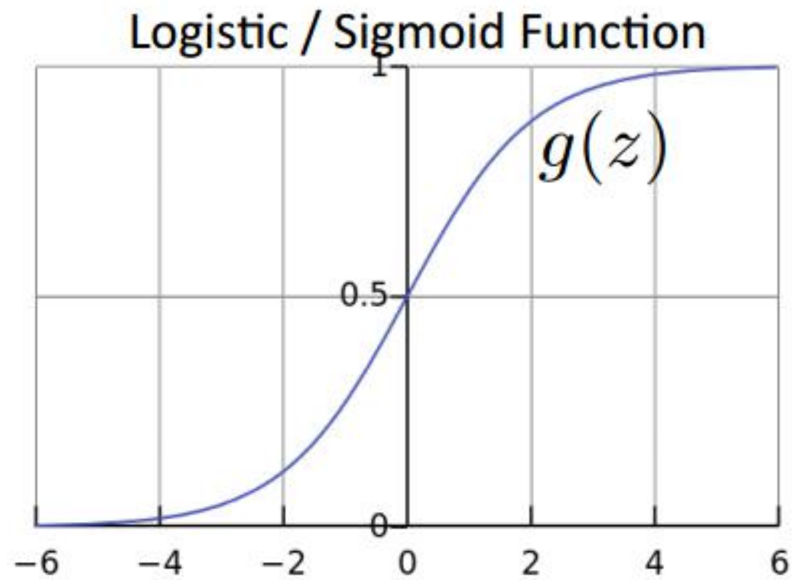
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$

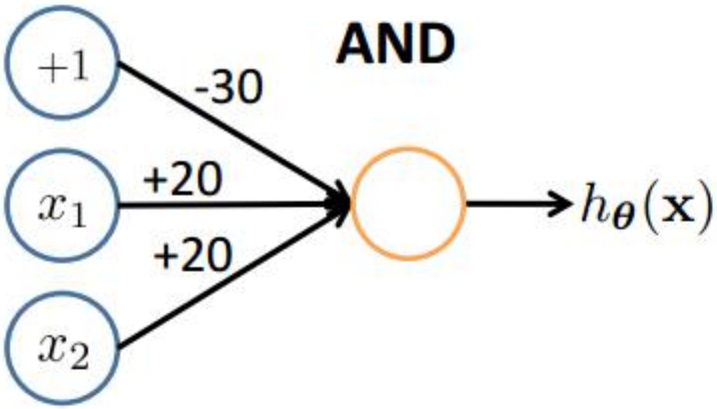


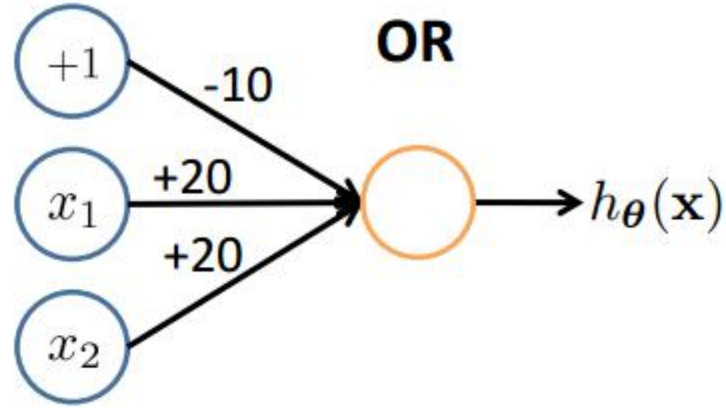
$$h_{\Theta}(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

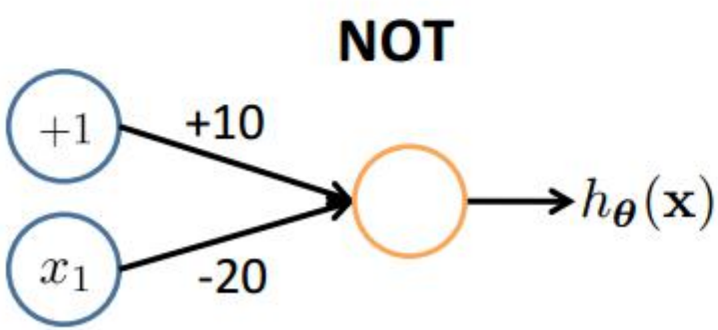


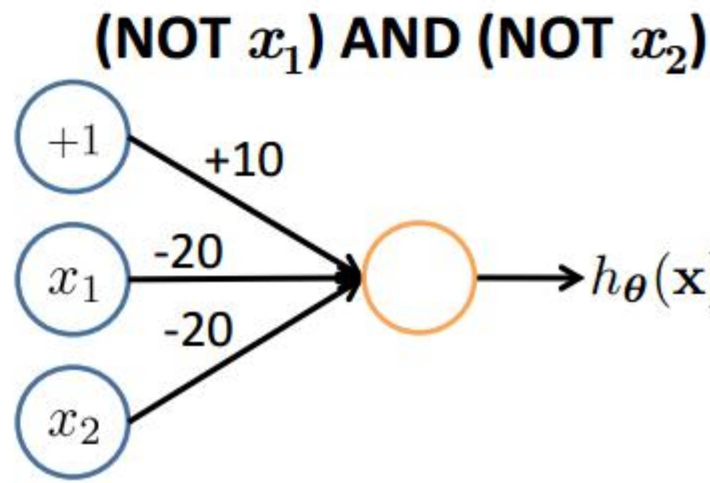
x_1	x_2	$h_{\Theta}(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

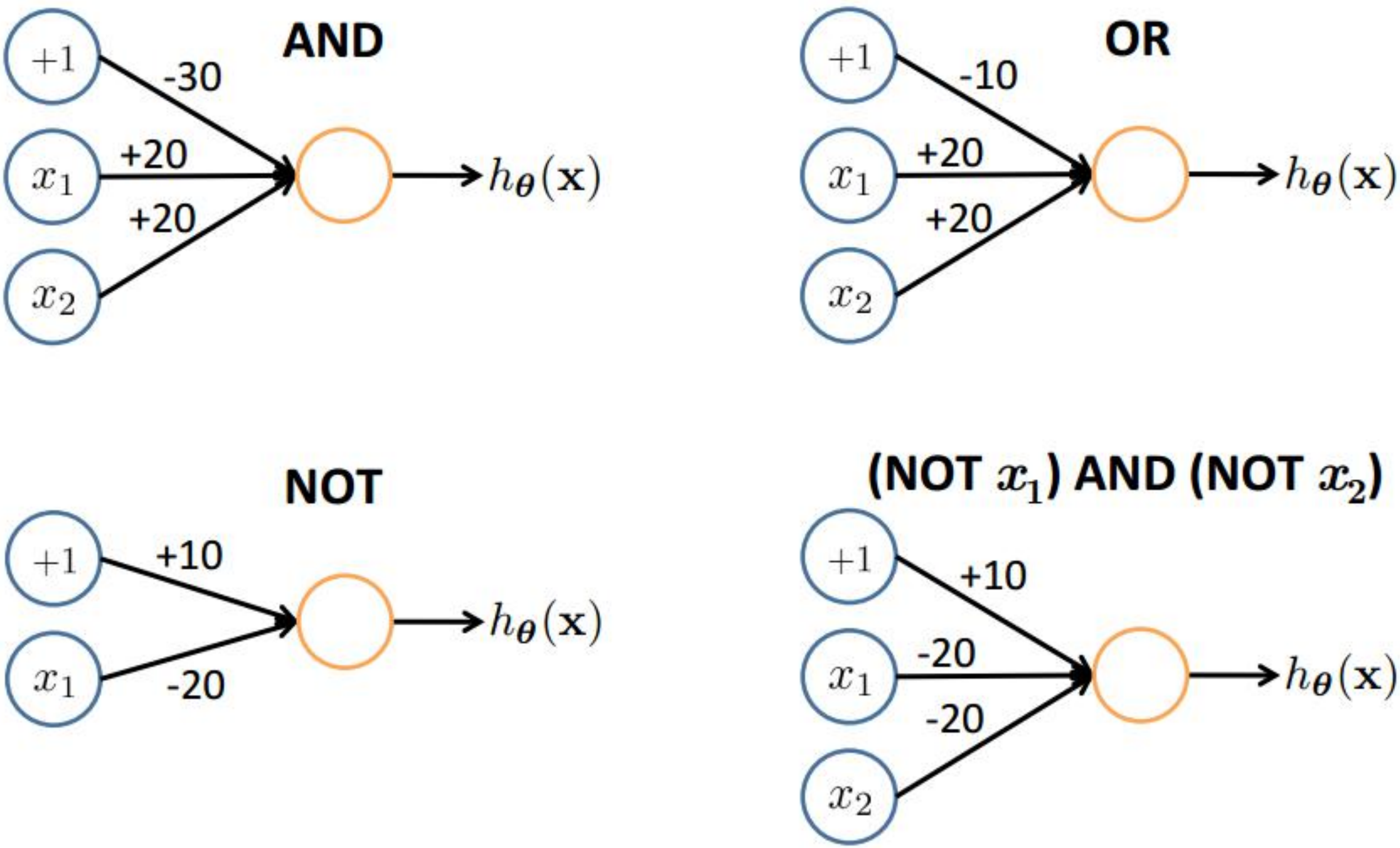
Based on slide and example by Andrew Ng


AND
$+1$
-30
x_1
+20
x_2
+20
$h_{\theta}(\mathbf{x})$

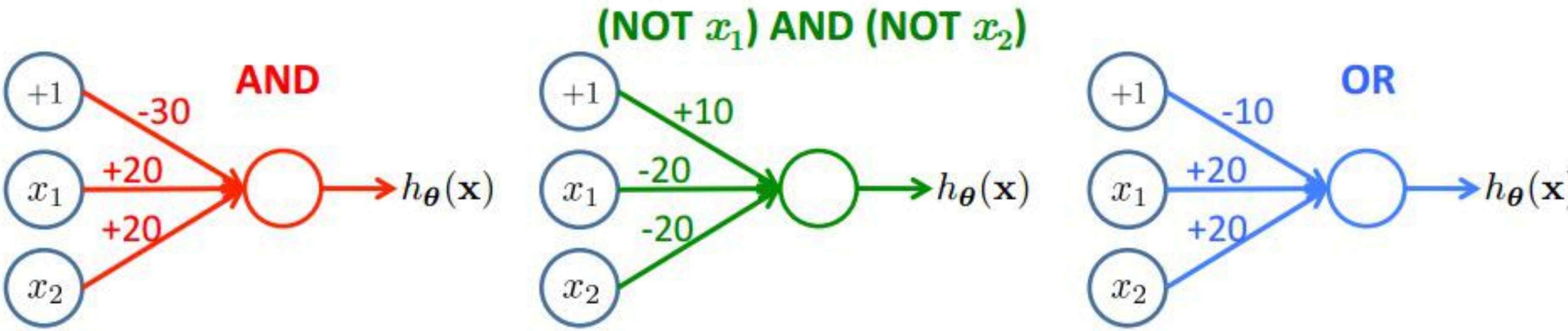

OR
$+1$
-10
x_1
+20
x_2
+20
$h_{\theta}(\mathbf{x})$


NOT
$+1$
+10
x_1
-20
$h_{\theta}(\mathbf{x})$

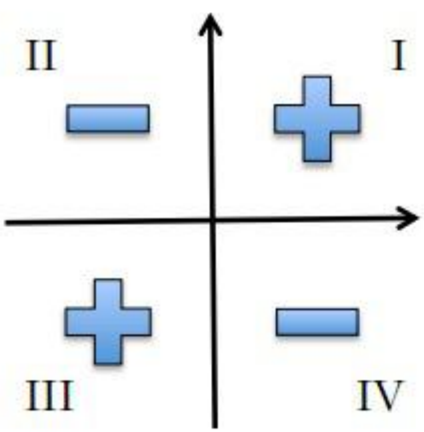

(NOT x_1) AND (NOT x_2)
$+1$
+10
x_1
-20
x_2
-20
$h_{\theta}(\mathbf{x})$


AND
+1
-30
x1
+20
+20
x2
hθ(x)
OR
+1
-10
x1
+20
+20
x2
hθ(x)
NOT
+1
+10
x1
-20
hθ(x)
(NOT x1) AND (NOT x2)
+1
+10
x1
-20
-20
x2
hθ(x)

Combining Representations to Create Non-Linear Functions



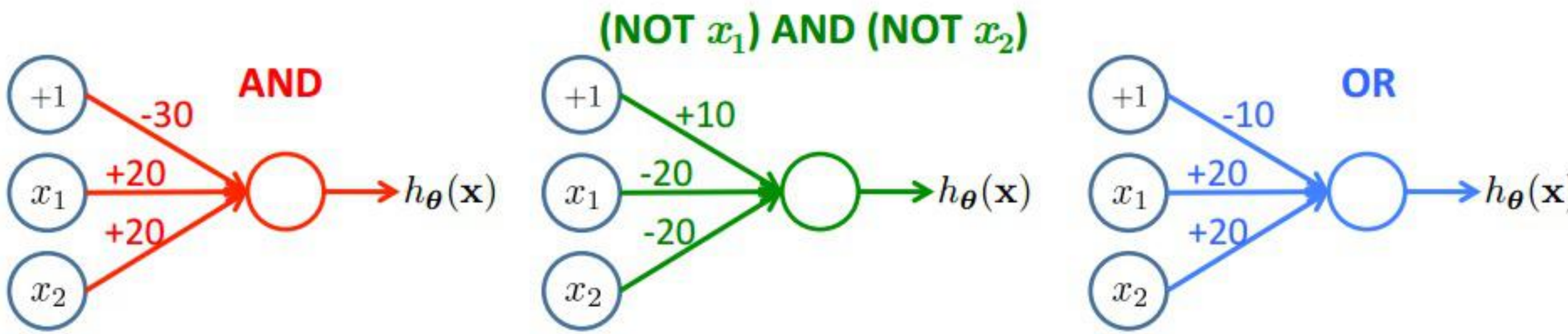
XNOR



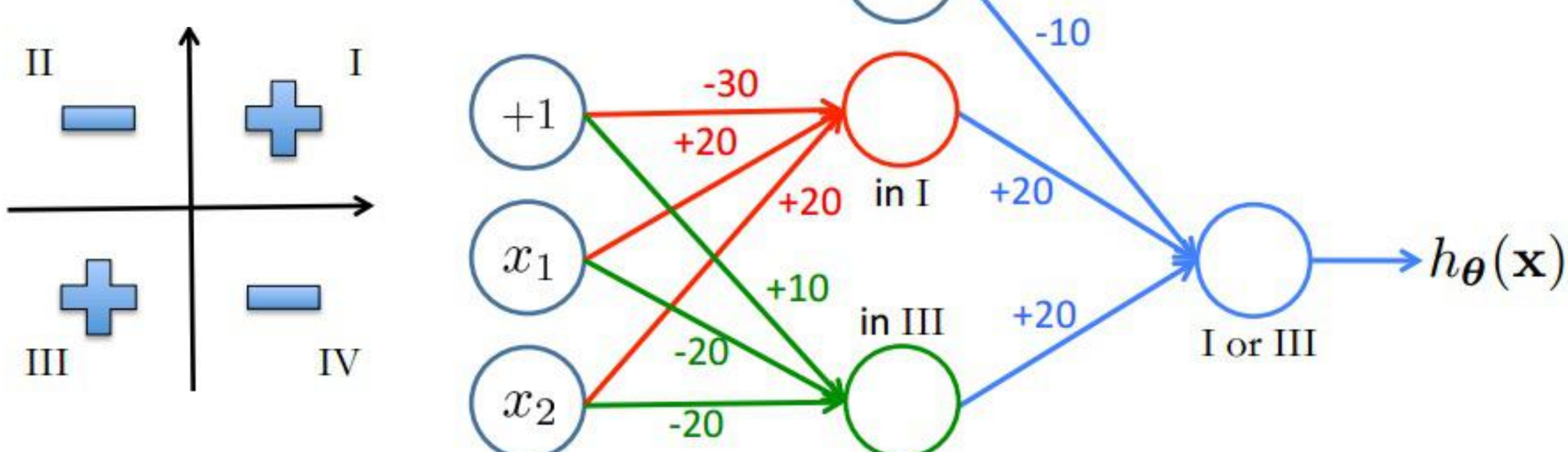
XNOR

Based on example by Andrew Ng

Combining Representations to Create Non-Linear Functions

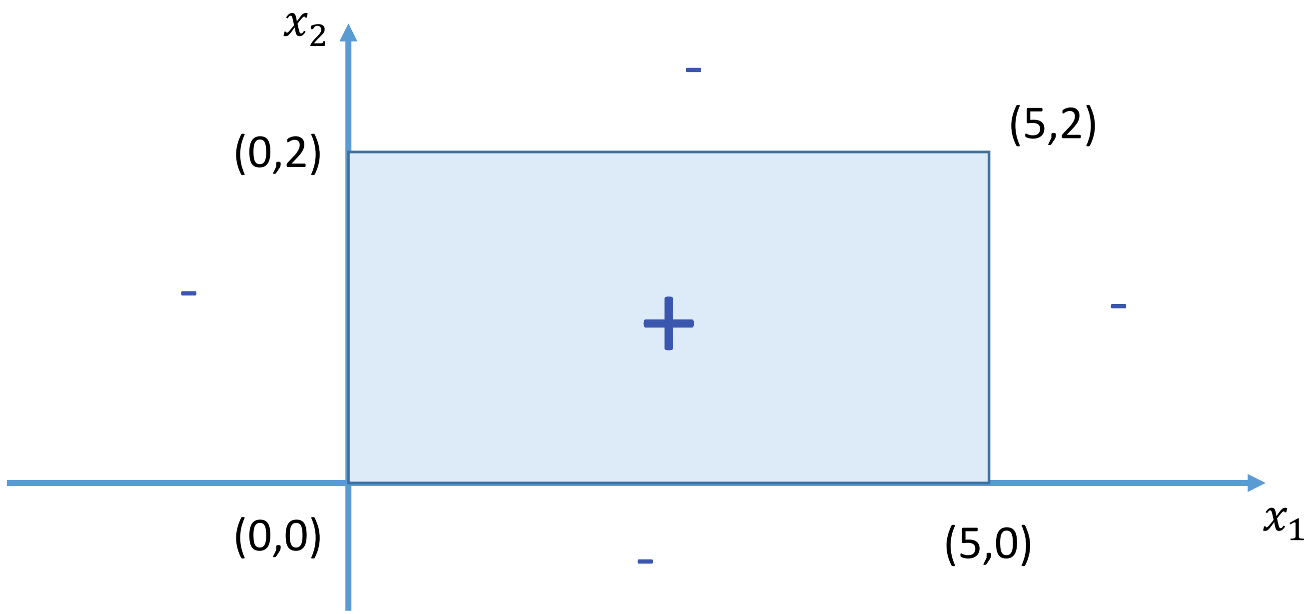


XNOR



Based on example by Andrew Ng

Exercise



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

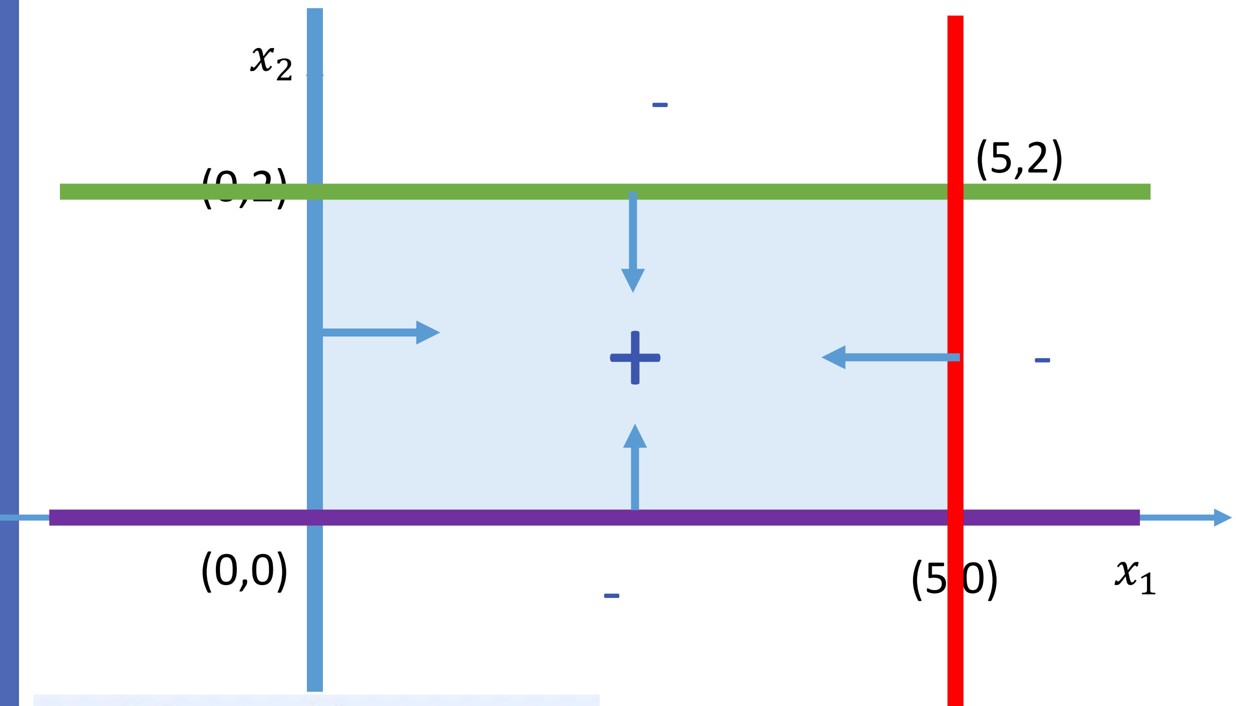
If all the samples inside the rectangle are positive; otherwise are negative

Show a feedforward NN can classify all the samples correctly

For simplicity, we assume g(z) is a step function.

What are $\Theta^{(1)}$ and $\Theta^{(2)}$

Exercise



$$\begin{cases} x_1 > 0 \\ 5 - x_1 > 0 \\ x_2 > 0 \\ 2 - x_2 > 0 \end{cases}$$

$$\Rightarrow \ \Theta^{(1)} = \begin{bmatrix} 0 & 1 & 0 \\ 5 & -1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & -1 \end{bmatrix}$$

$$\Theta_0^{(1)} \ \Theta_1^{(1)} \ \Theta_2^{(1)}$$

$$-3.5 + a_1 + a_2 + a_3 + a_4 > 0$$

$$\Theta^{(2)} = [-3.5 \quad 1 \quad 1 \quad 1 \quad 1]$$

$$\Theta_0^{(2)} \ \Theta_1^{(2)} \ \Theta_2^{(2)} \Theta_3^{(2)} \ \Theta_4^{(2)}$$

Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

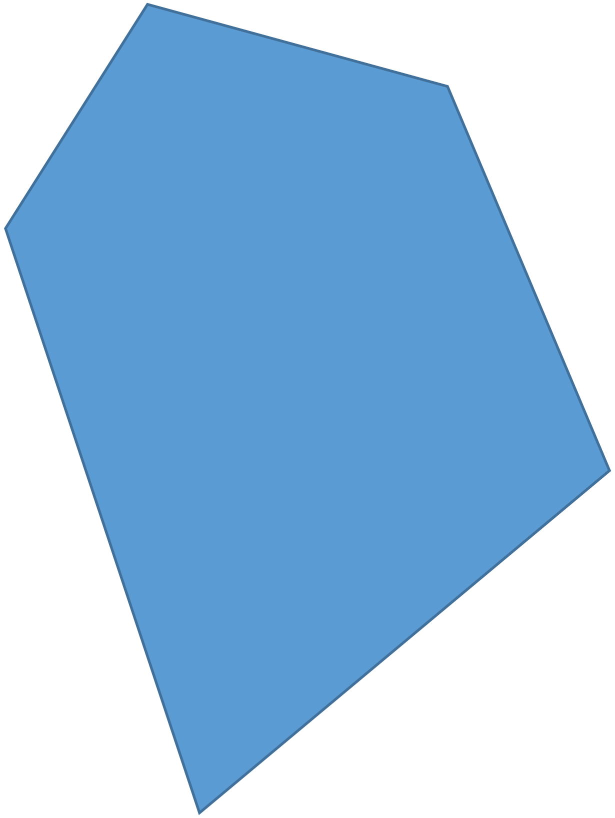
$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

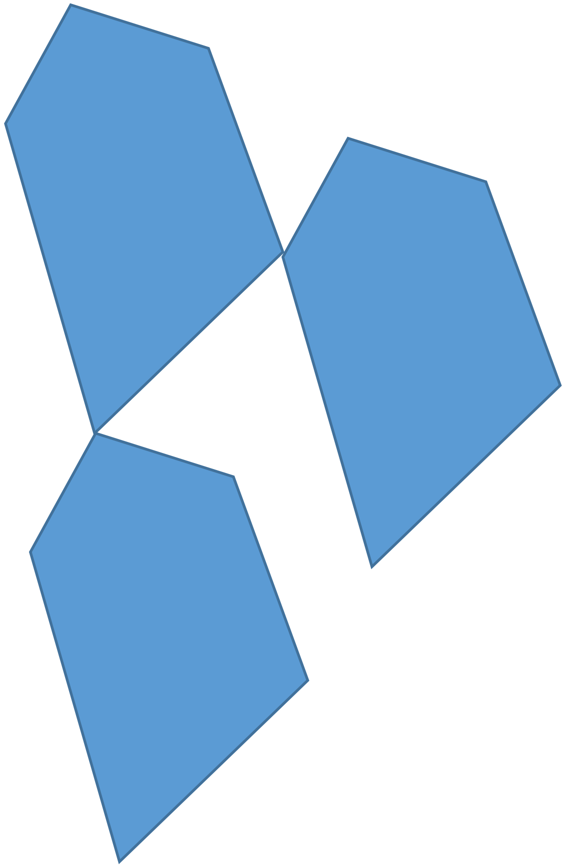
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

If all the samples inside the rectangle are positive;
otherwise are negative
Show a feedforward NN can classify all the samples correctly
For simplicity, we assume g(z) is a step function.
What are $\Theta^{(1)}$ and $\Theta^{(2)}$

Arbitrary Decision Boundary

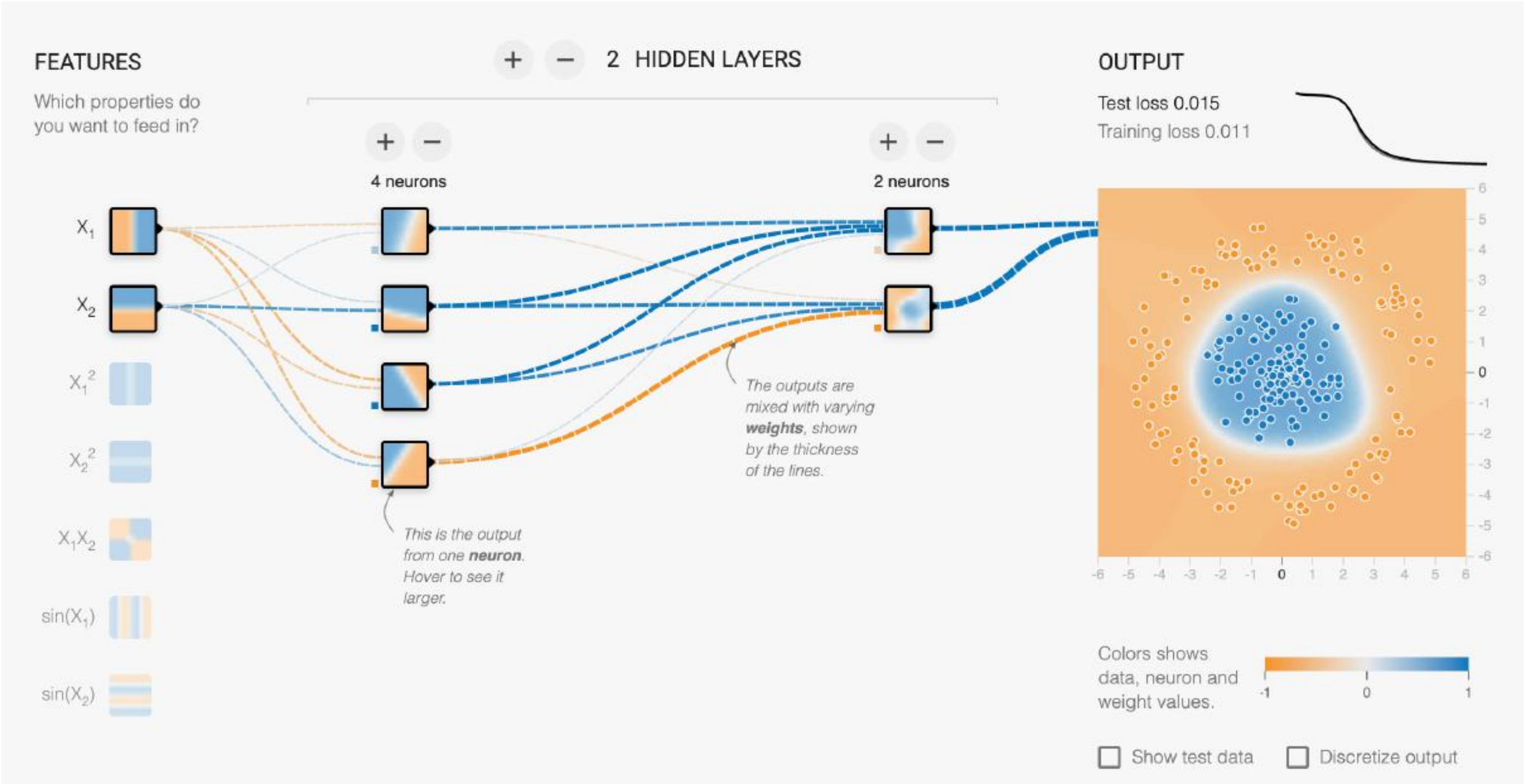


Arbitrary Decision Boundary



Neural Network Training Animation

- https://playground.tensorflow.org/



Neural Network Learning

Maximum Likelihood

- Training data: $S = \{(x_i, y_i)\}$, m examples
  $y_i = \{0,1\}$
- Consider a NN $h_\Theta(x) \in [0,1]$ modeling $P(y=1|x)$

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$
$$h_\Theta(\mathbf{x}) = \underbrace{g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)} = g\left(z_1^{(3)}\right)$$

Remember in logistic regression, $h_{w,b}(x) = \sigma(w^T x + b)$
If we choose the activation function g as a sigmoid function,
this part is equivalent to a logistic regression with input a

Based on slide by Andrew Ng

Maximum Likelihood

- Training data: $S = \{(x_i, y_i)\}$, m examples
  $y_i = \{0,1\}$
- Consider a NN $h_\Theta(x) \in [0,1]$ modeling $P(y = 1|x)$
- Maximum Likelihood estimator $\Theta^* = \arg\max_\Theta L(\Theta; S)$
  $L(\Theta; S) = \prod_{i=1}^{m} P_\Theta(y_i|x_i)$
  $$P_\Theta(y_i|x_i) = \begin{cases} h_\Theta(x_i), & y_i = 1 \\ 1 - h_\Theta(x_i), & y_i = -1 \end{cases}$$
- We can rewrite $L(\Theta; S)$ as
  $$L(\Theta; S) = \prod_{i=1}^{m} h_\Theta(x_i)^{y_i} (1 - h_\Theta(x_i))^{1-y_i}$$

Based on slide by Andrew Ng

Minimum Negative Log-Likelihood & Cross-Entropy Loss

- Training data: $S = \{(x_i, y_i)\}$, m examples
  $y_i = \{0,1\}$
- Consider a NN $h_\Theta(x) \in [0,1]$ modeling $P(y = 1|x)$
- Likelihood $L(\Theta; S)$ is
  $$L(\Theta; S) = \prod_{i=1}^{m} h_\Theta(x_i)^{y_i} (1 - h_\Theta(x_i))^{1-y_i}$$
- Log-Likelihood:
  $$\log L(\Theta; S) = \sum_i^m [y_i \log h_\Theta(x_i) + (1 - y_i) \log\big(1 - h_\Theta(x_i)\big)]$$
- Optimal Θ can be obtained by solving $\arg\min_\Theta J(\Theta)$
  $$\mathrm{J}(\Theta) = -\sum_i^m \underbrace{[y_i \log h_\Theta(x_i) + (1 - y_i) \log\big(1 - h_\Theta(x_i)\big)]}_{\text{Cross-entropy between prediction } h_\Theta(x) \text{ and true label } y_i}$$

Based on slide by Andrew Ng

Stochastic gradient Descent

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

1. Initialize $\Theta \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For epoch $1 \ldots T$:
3. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
4. Update $w \leftarrow w - \eta \boxed{\nabla J(\Theta)}$
5. Return Θ

(Similar to logistic regression)

$$J(\Theta) = -\sum_i^m [y_i \log h_\Theta(x_i) + (1 - y_i) \log\big(1 - h_\Theta(x_i)\big)]$$

Optimizing the Neural Network

$$\mathrm{J}(\Theta) = -\sum_i^m [y_i \log h_\Theta(x_i) \ + (1 - y_i) \log\big(1 - h_\Theta(x_i)\big)]$$

- Need to compute $\nabla J(\Theta)$

Based on slide by Andrew Ng

Chain Rule

- Given a function

$$f(x) = A(B(C(x)))$$

- The derivative is

$$f'(x) = A'(B) \cdot B'(C) \cdot C'(x)$$

Backpropagation through Computation Graphs

Computation Graphs and Backpropagation

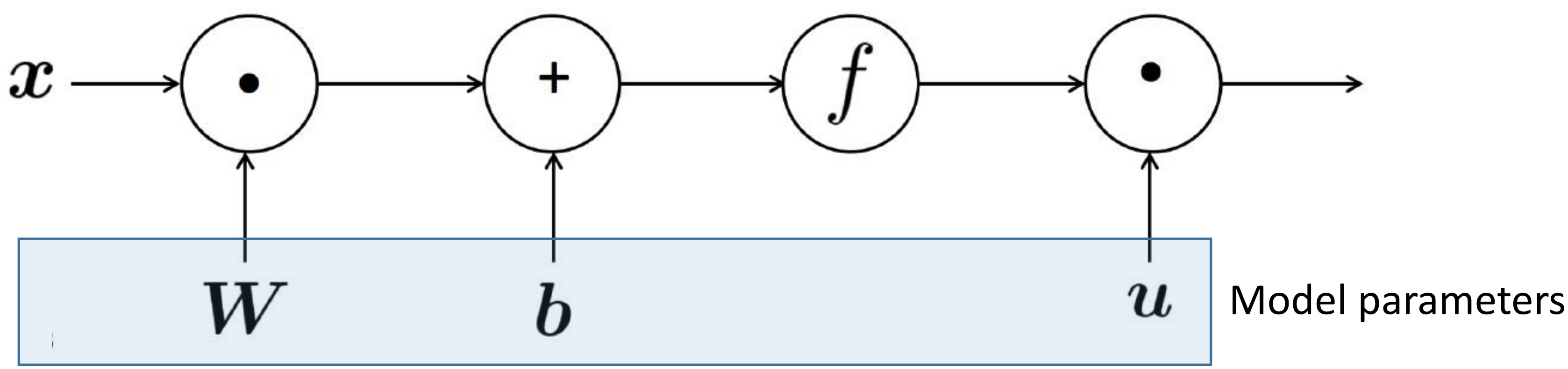
- Consider the NN on the right
- We represent NN as a graph

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

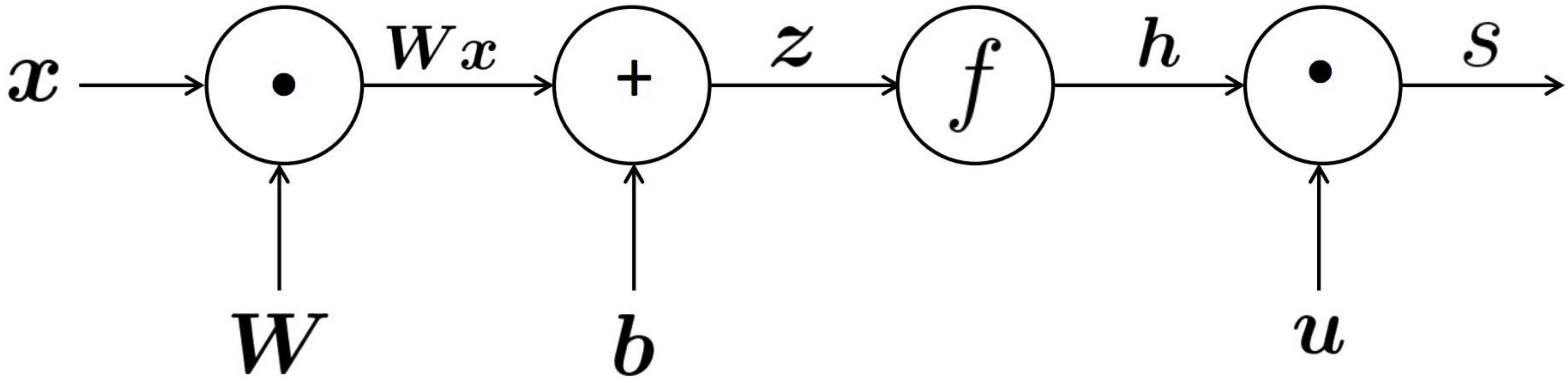
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$



Forward Propagation

- Edges pass along result of the operation



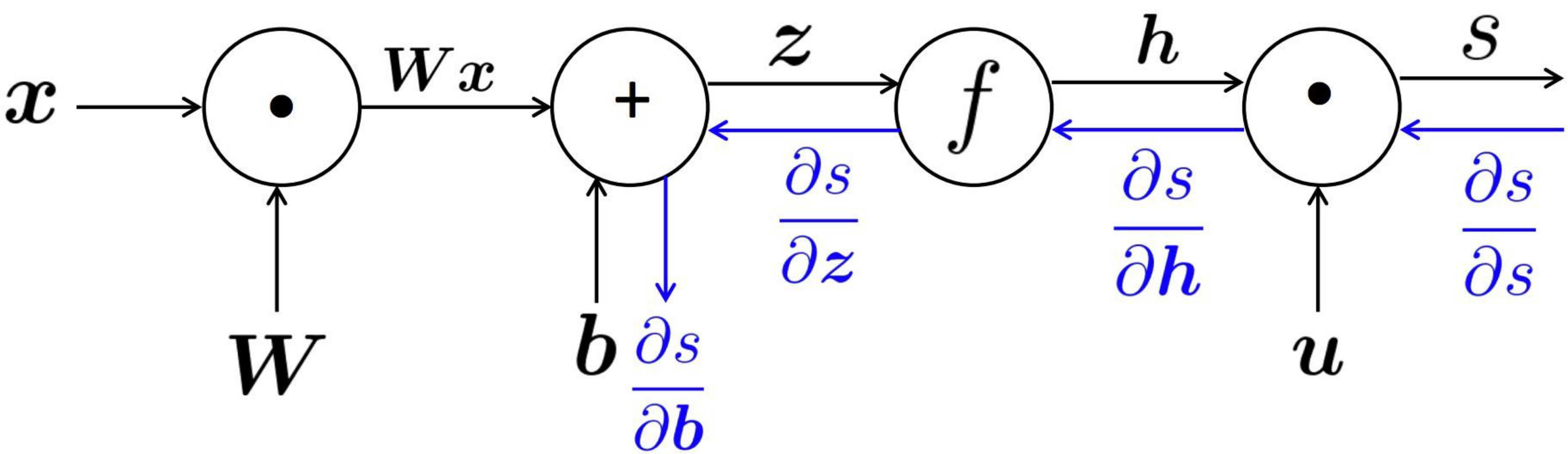
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

Back Propagation

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

- Compute $\frac{\partial s}{\partial \boldsymbol{b}}$

Chain Rule: $\frac{\partial s}{\partial b} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial b} = \cdots$

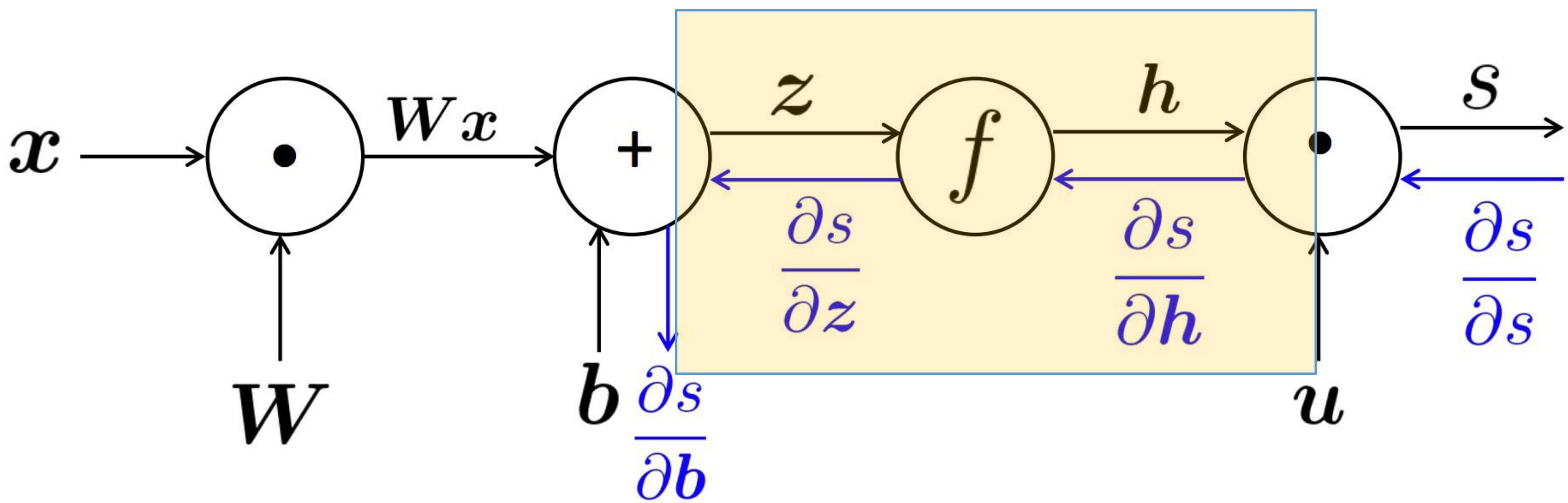


Back Propagation

- Compute $\frac{\partial s}{\partial \boldsymbol{b}}$

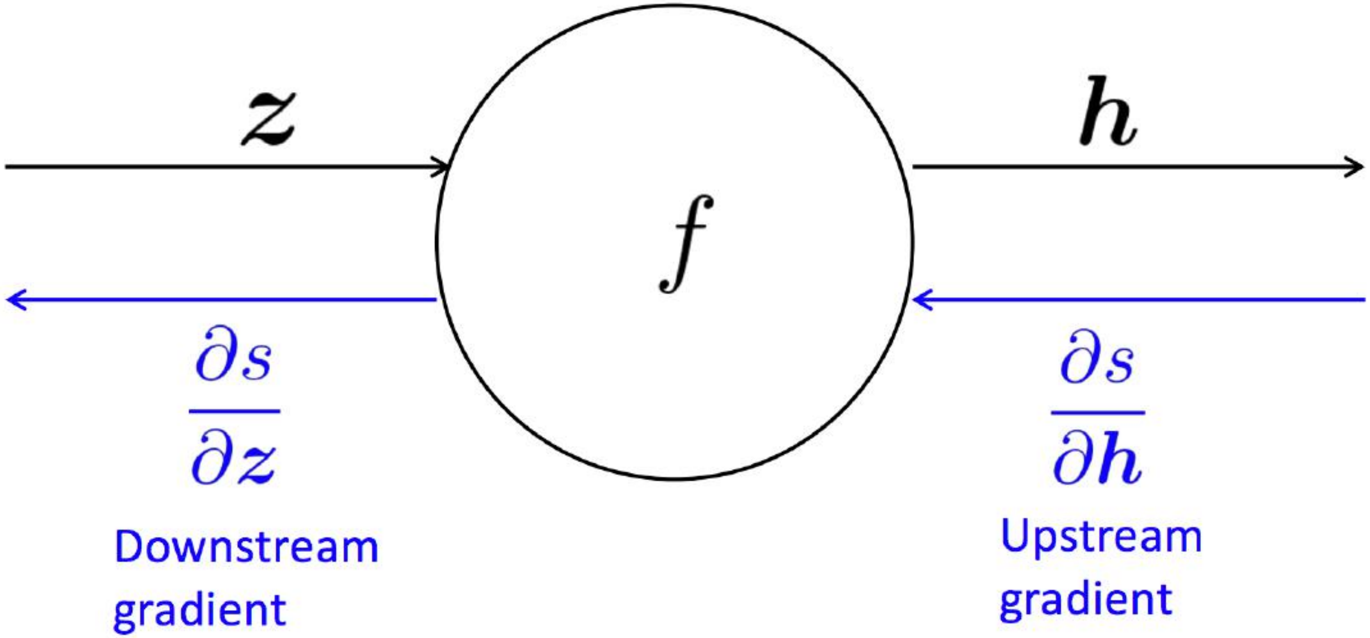
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

Chain Rule: $\frac{\partial s}{\partial b} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial b}$



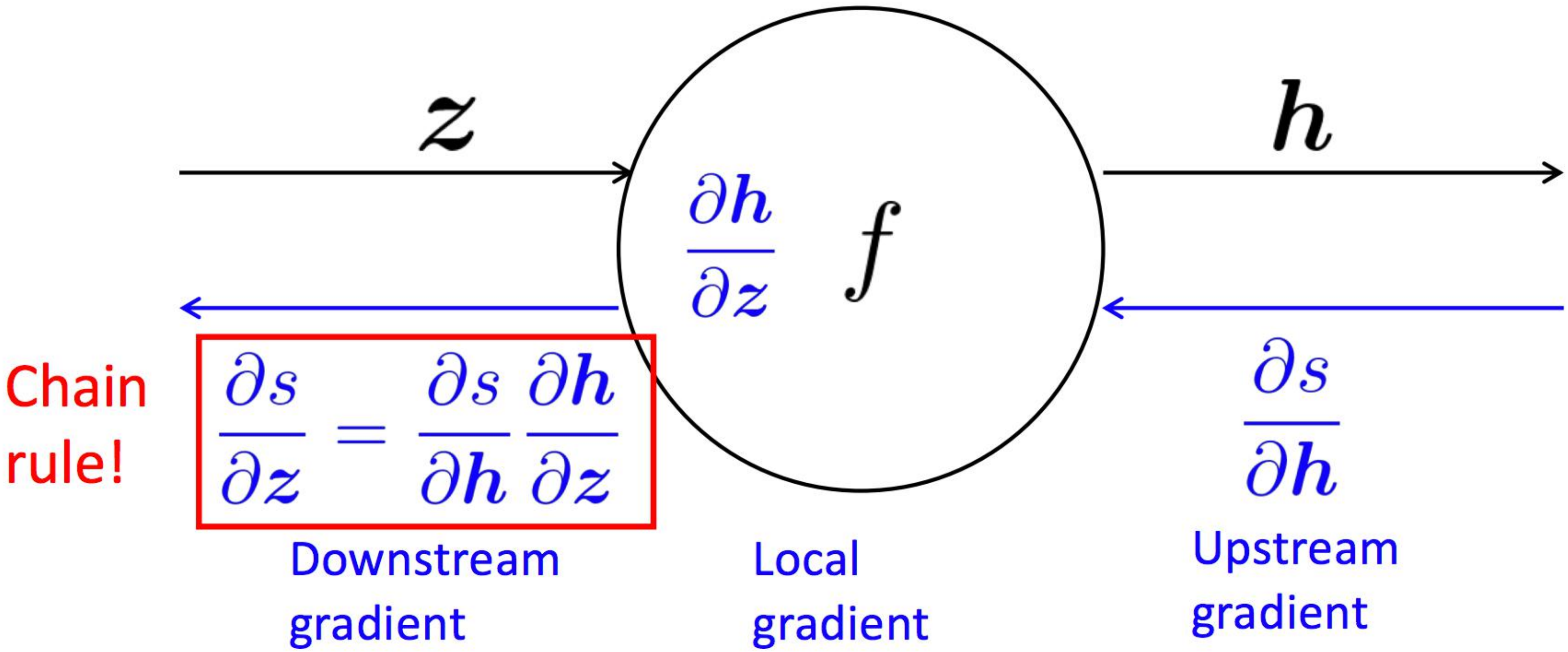
Backpropagation: Single Node

$$\boxed{\boldsymbol{h} = f(\boldsymbol{z})}$$



Chain Rule

$$\boxed{\boldsymbol{h} = f(\boldsymbol{z})}$$

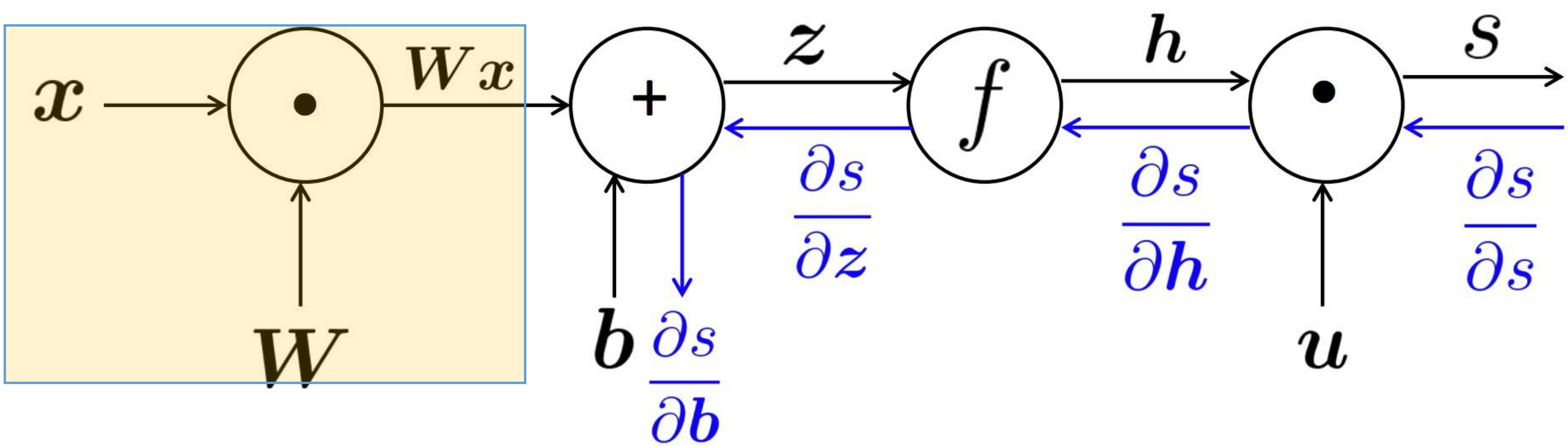


Back Propagation

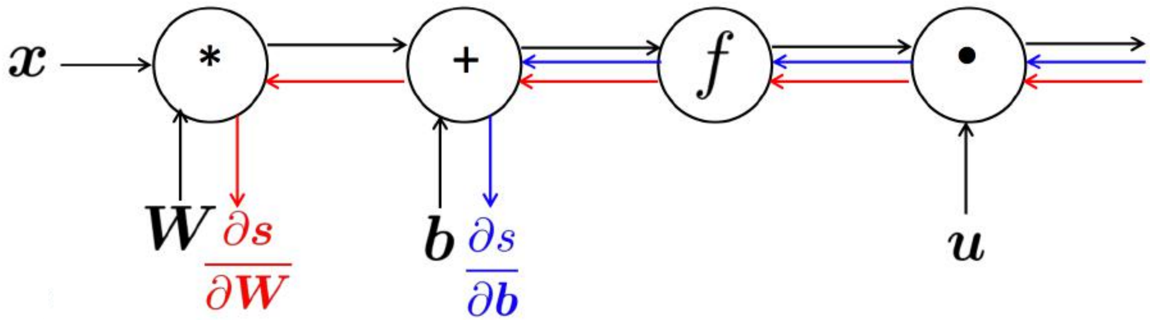
$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

- Compute $\frac{\partial s}{\partial \boldsymbol{b}}$

Chain Rule: $\frac{\partial s}{\partial b} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial b} = \cdots$



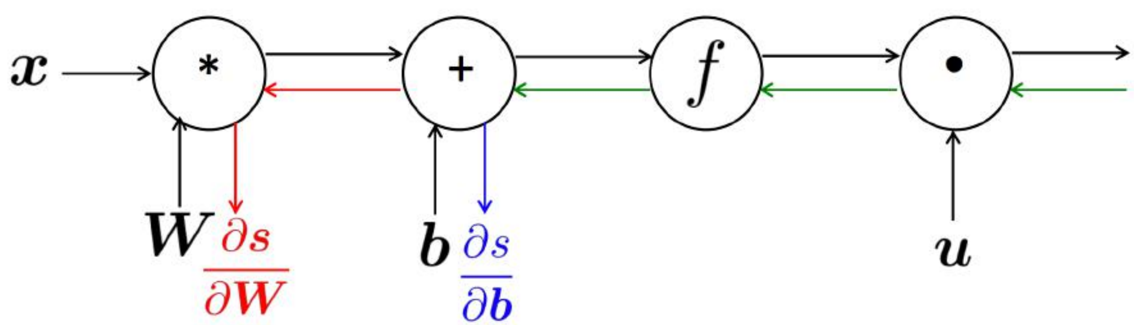
Compute all gradients at once



Naïve way to compute gradients:

Compute each component separately

⇒ Redundant computation



Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Draw the computation graph and calculate $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Example

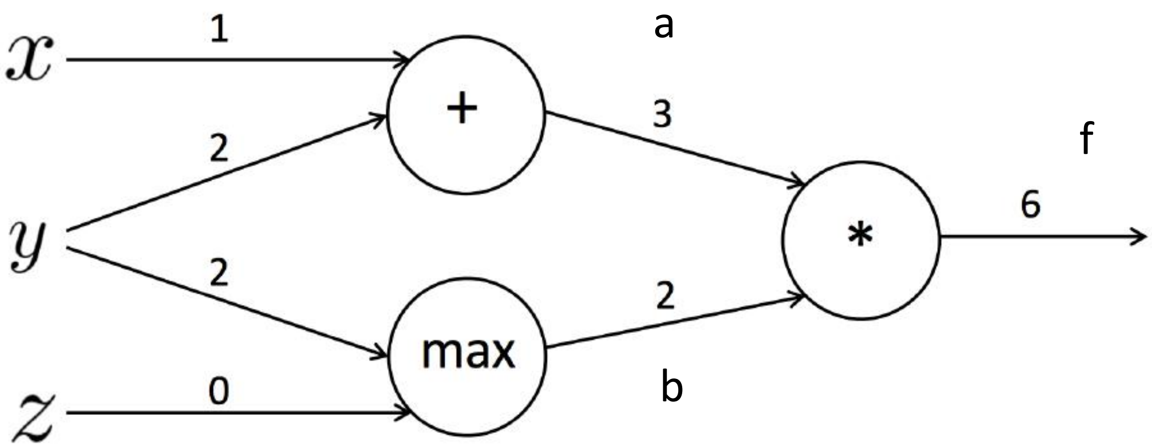
$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$

$$b = \max(y, z)$$

$$f = ab$$



Example

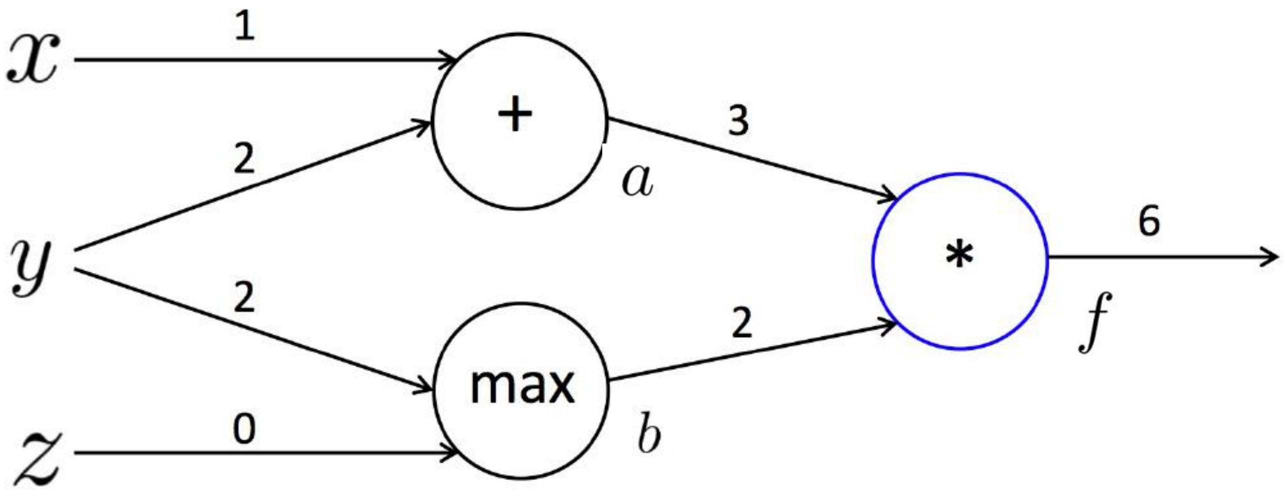
$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



Example

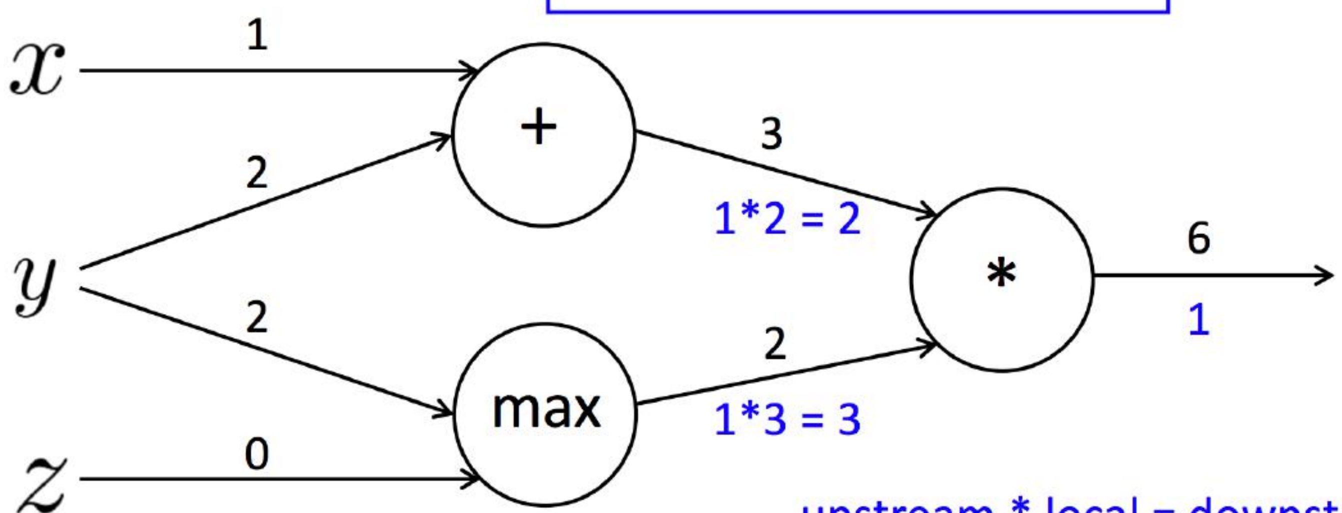
$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradie

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\boxed{\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3}$$



upstream * local = downstream

Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

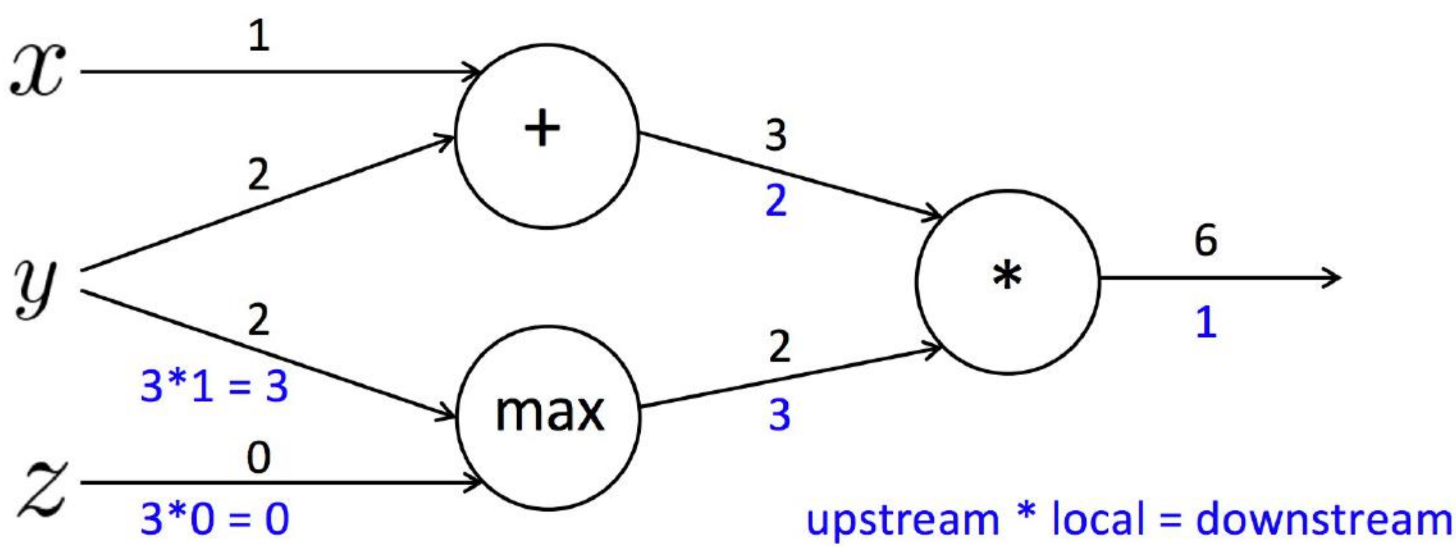
$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$

$$\boxed{\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0}$$

$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



Example

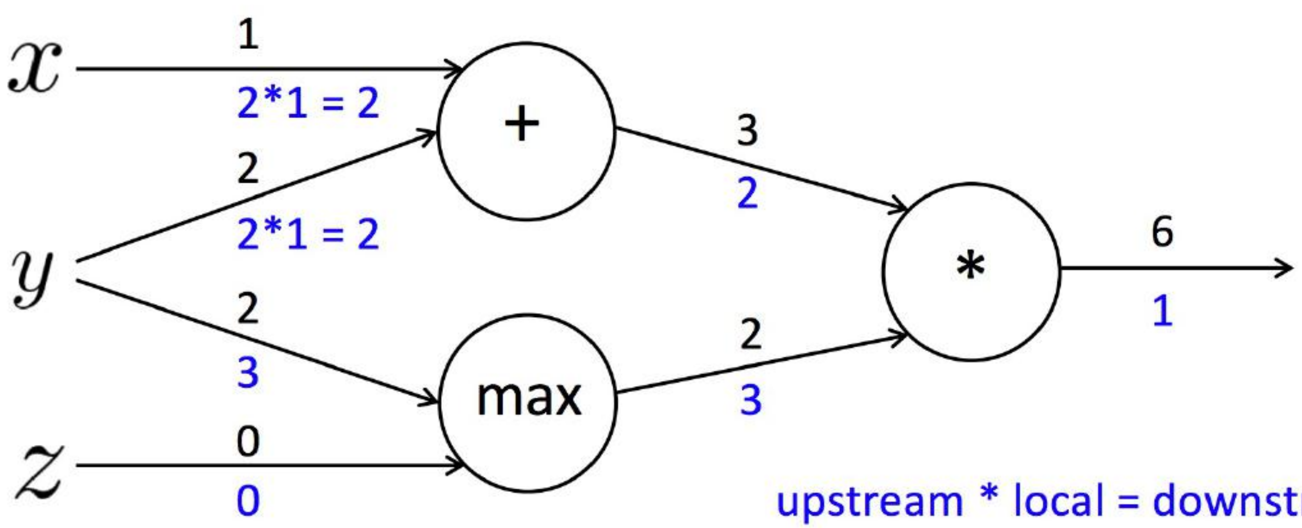
$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\boxed{\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1}$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$



upstream * local = downstream

Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

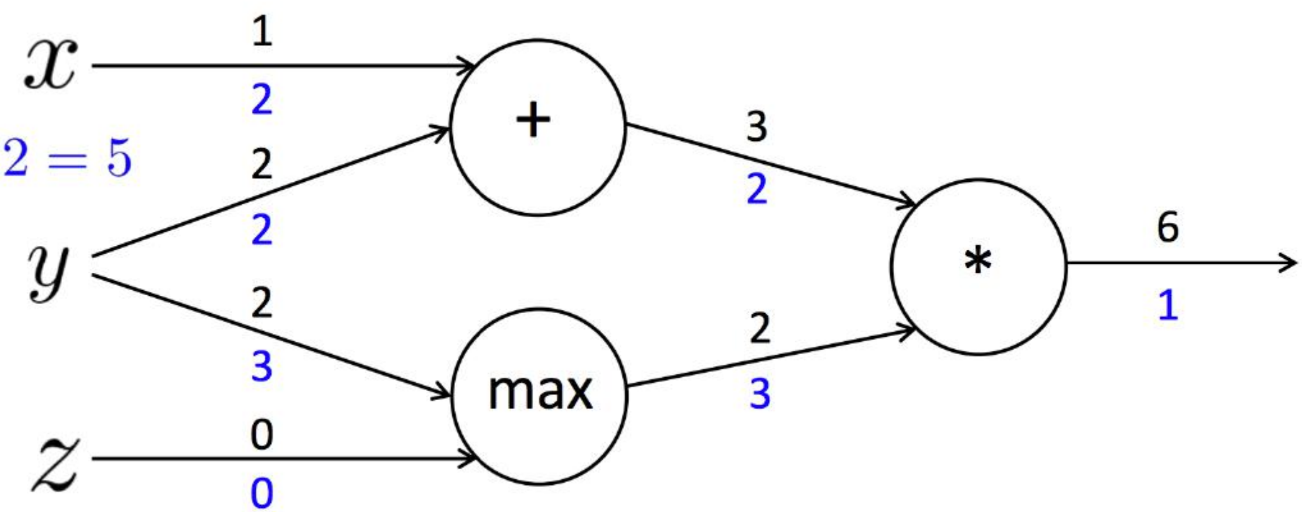
Forward prop steps

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Local gradients

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$
$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1 \quad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$
$$\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$$

$$\frac{\partial f}{\partial x} = 2$$
$$\frac{\partial f}{\partial y} = 3 + 2 = 5$$
$$\frac{\partial f}{\partial z} = 0$$

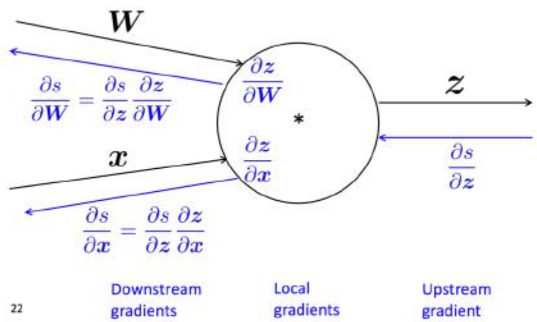


Why you should understand Backprop

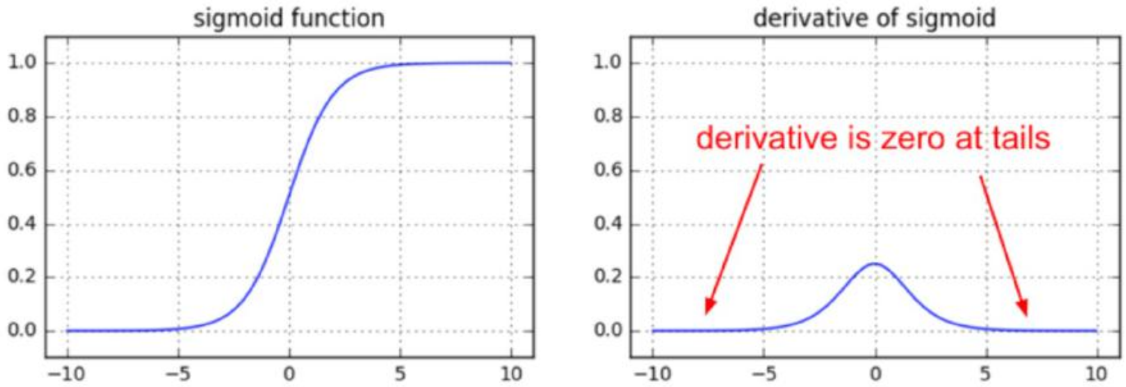
- Modern deep learning library implements backprop as a black-box for you
  - You can take a plane without knowing why it flies
  - but you're designing aircraft…
- Backpropagation doesn't always work perfectly.
  - Understanding why is crucial for debugging and improving models

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

Example: Gradient of sigmoid




```
z = 1/(1 + np.exp(-np.dot(W, x))) # forward pass
dx = np.dot(W.T, z*(1-z)) # backward pass: local gradient for x
dW = np.outer(z*(1-z), x) # backward pass: local gradient for W
```



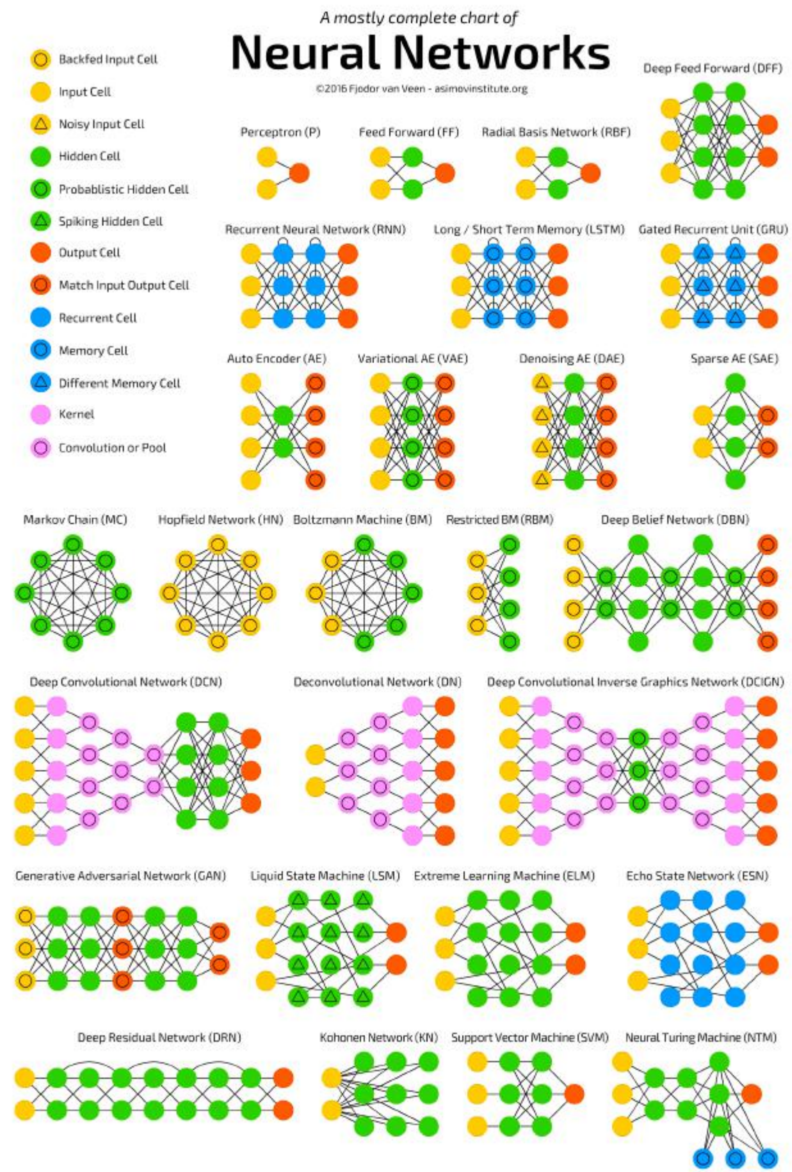
vanish gradient issue

More Details

- Parameter Initialization
  - Normally initialize weights to small random values; various designs
- Optimizer
  - Usually SGD works
  - Several SGD variants (e.g,. ADAM) automatically adjust learning rate based on an accumulated gradient

A neural network zoo

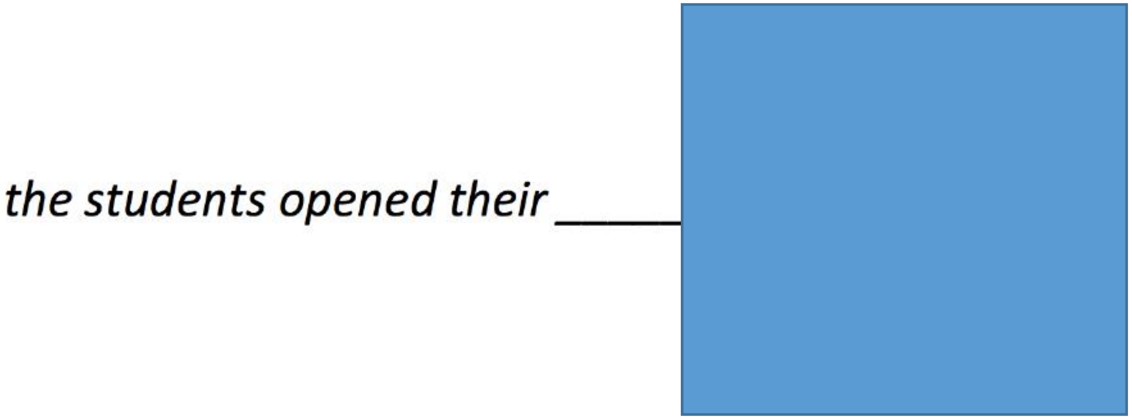
- The flexibility of NN allows us to try out different ideas
- However,
  there is no magic



Modeling with Neural Networks
(Advanced Topic/Not Included in Final)

Example – Language Model

- Predict next word



Idea 1:
A fixed-window neural Language Model

output distribution

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

hidden layer

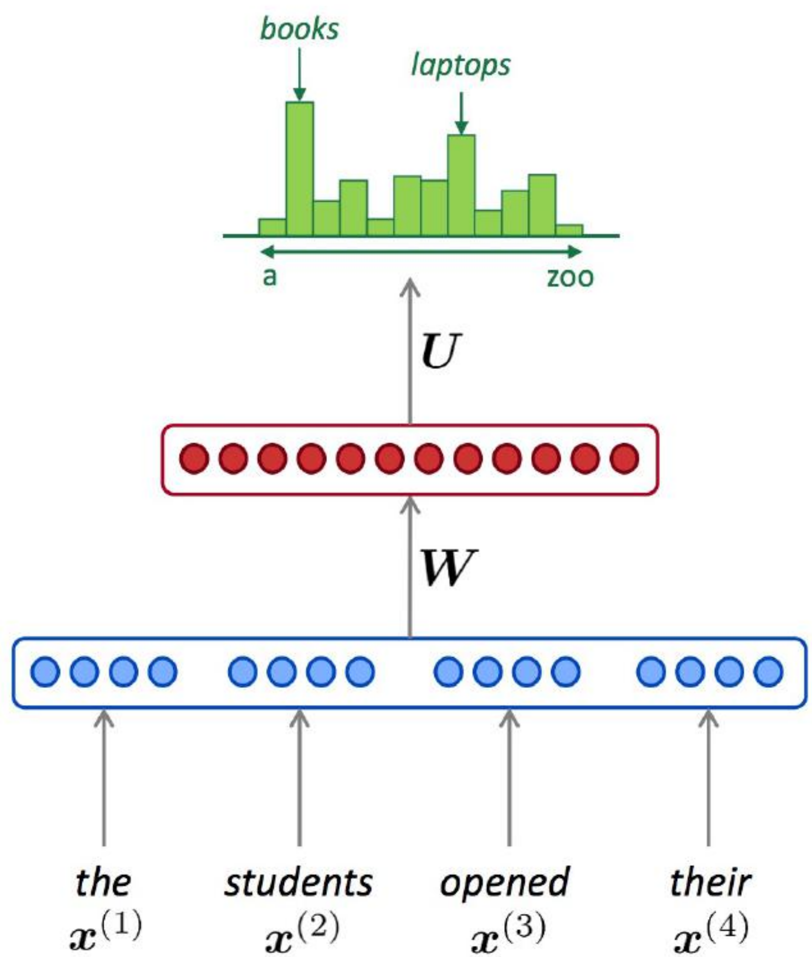
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

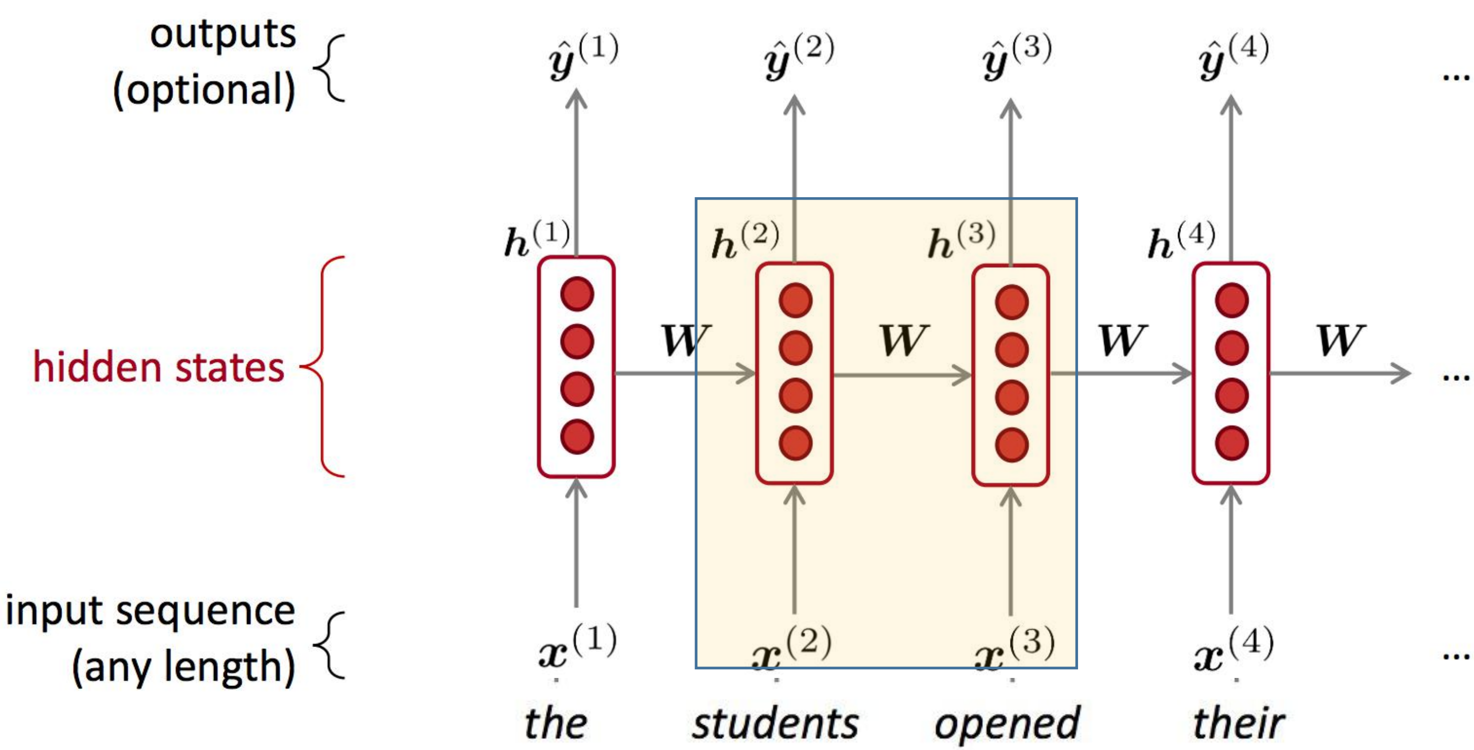
$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

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

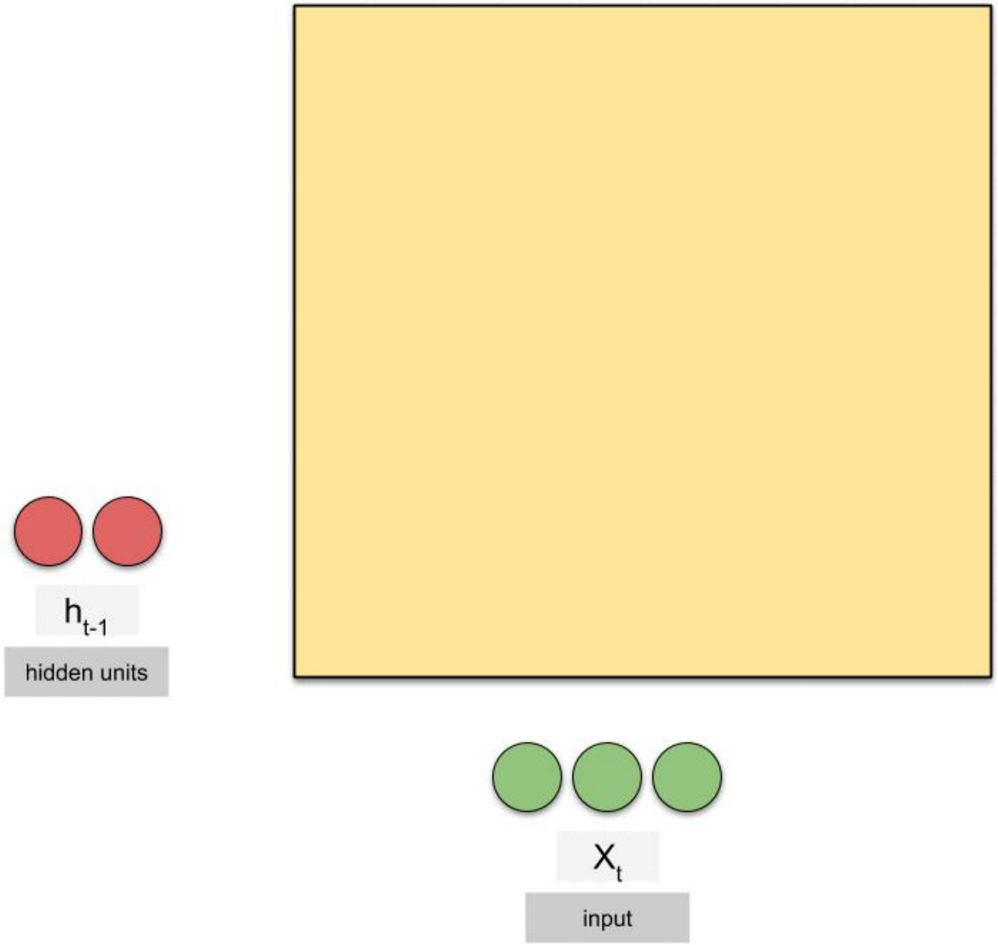


Idea 2: Recurrent Neural Networks (RNN)



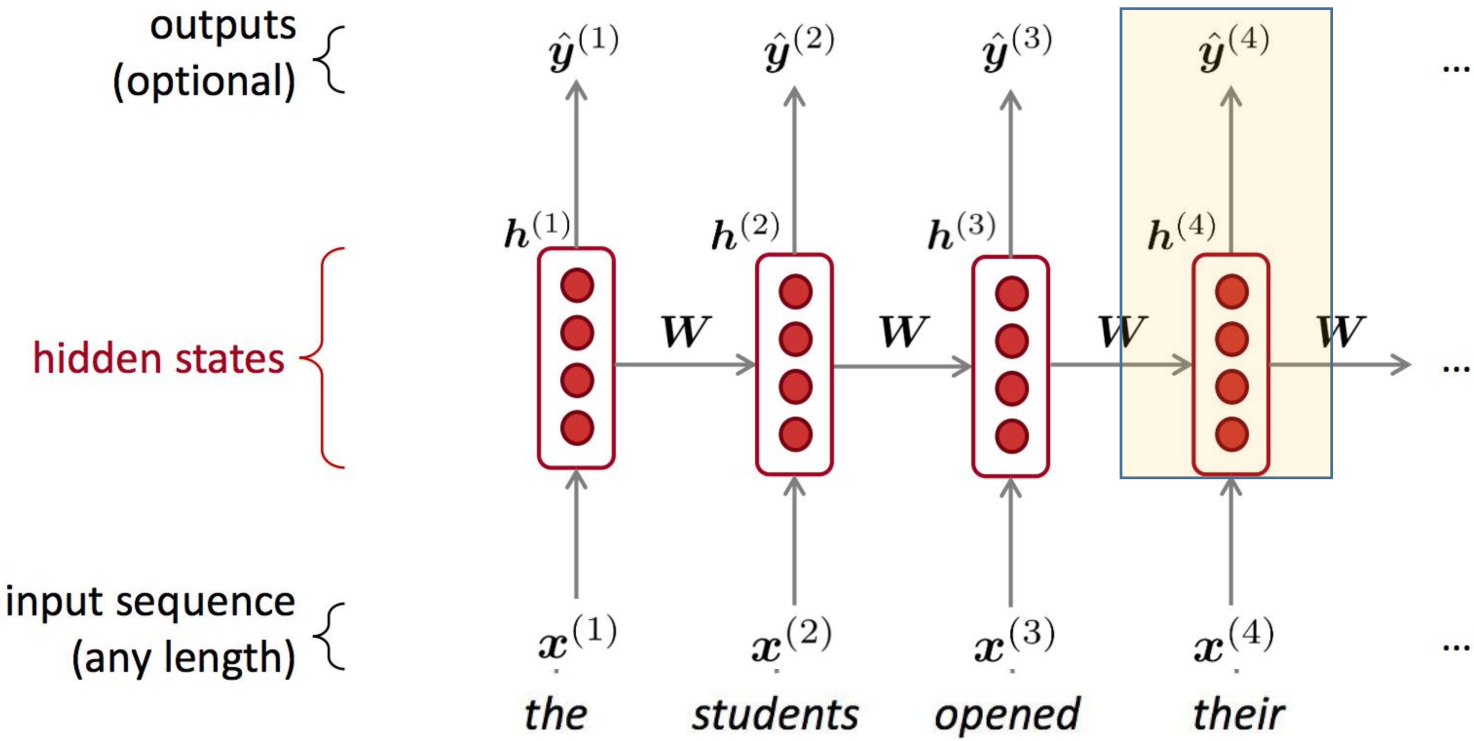
Core idea: Apply the same weights W *repeatedly*

Recurrent Neural Network



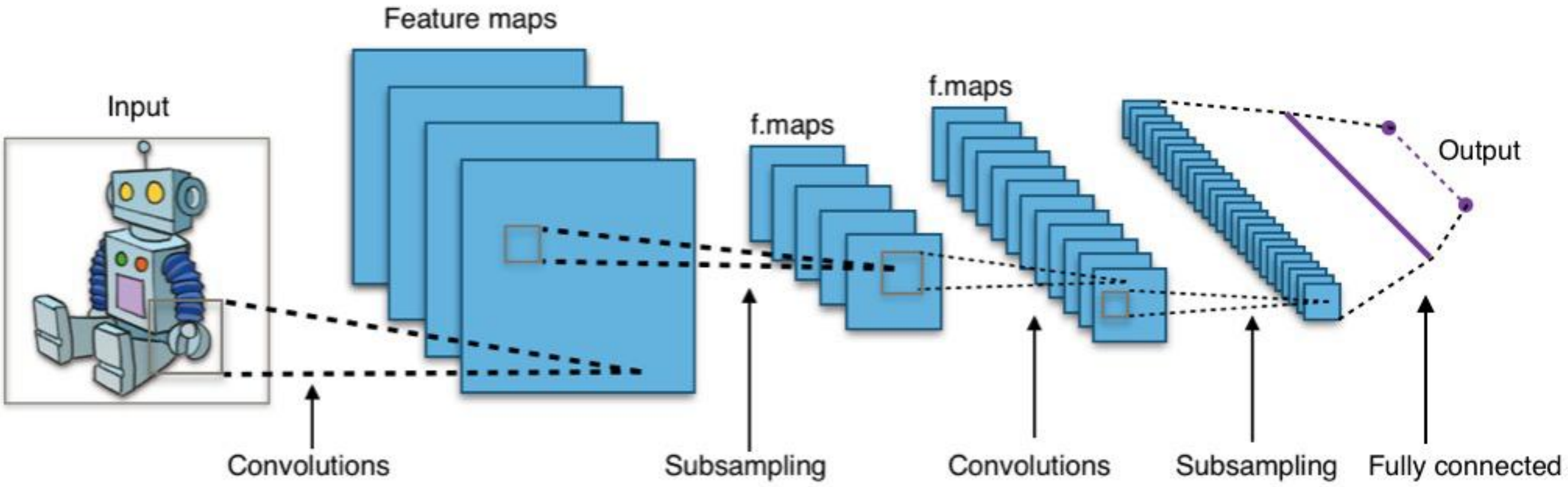
https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

Prediction using Latent State

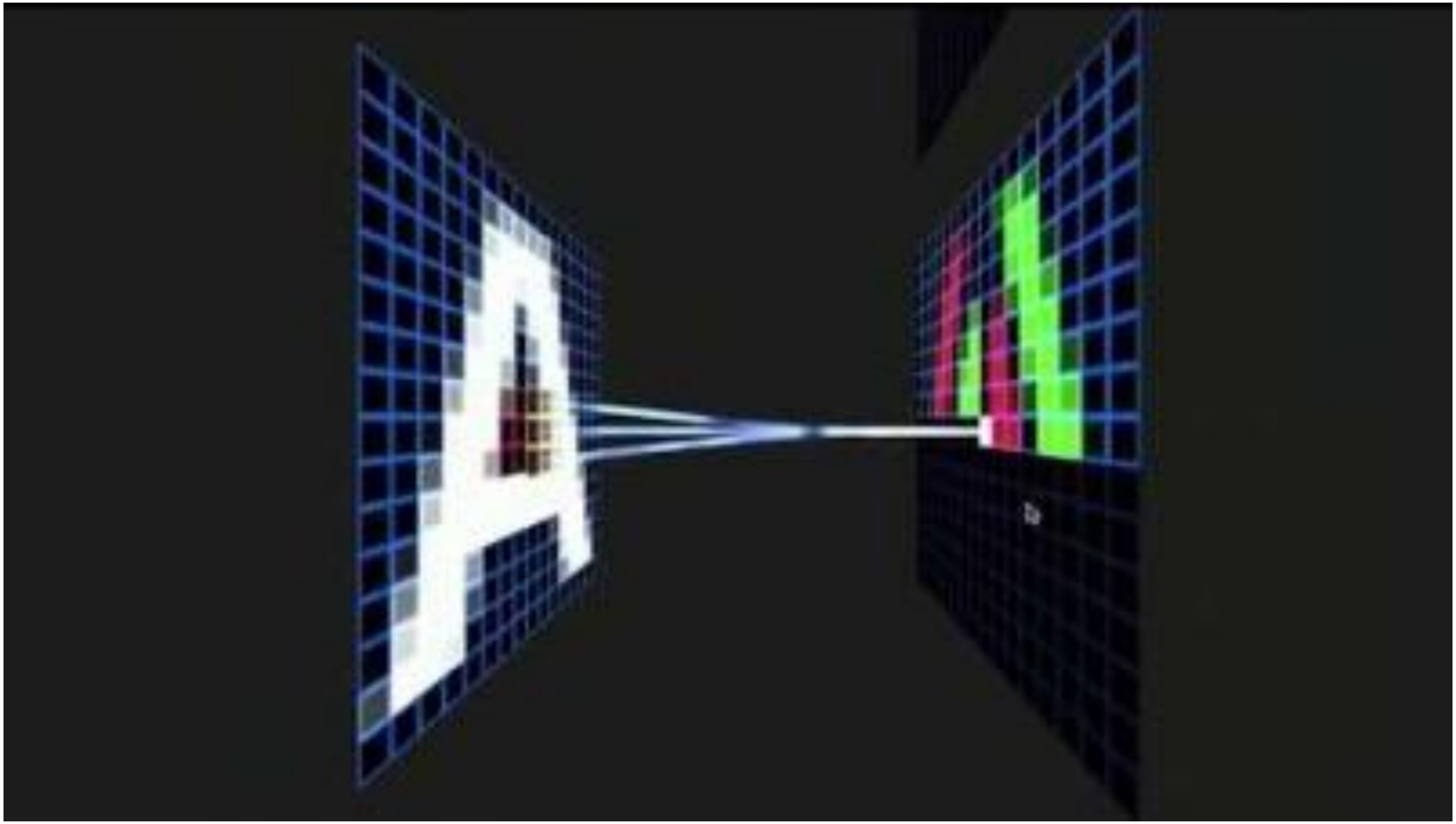


Core idea: Apply the same weights W *repeatedly*

Idea 3: Convolutional NN



Convolutional NN



https://www.youtube.com/watch?v=f0t-OCG79-U