CS5242 Neural Networks and Deep Learning

Lecture 03: From Shallow to Deep Neural Networks

Wei WANG

cs5242@comp.nus.edu.sg

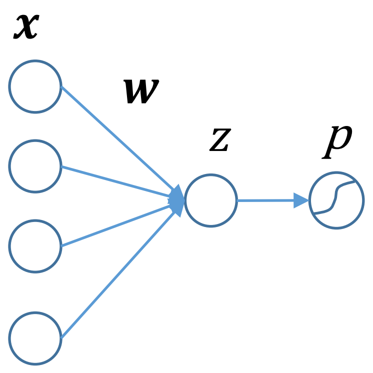
NUS National University of Singapore | School of Computing

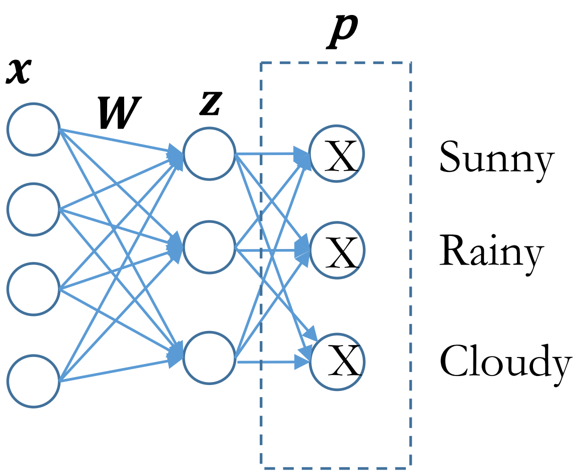
Agenda

- Recap
  - Logistic regression model
  - Softmax regression model
- Multi-layer perceptron model
- Back-propagation algorithm

Recap

- Binary classification model
  - Logistic function → probability
  - Binary cross-entropy loss
- Multi-class classification model
  - Softmax/multinomial regression
    - Softmax function → a vector of probabilities
  - Cross-entropy loss



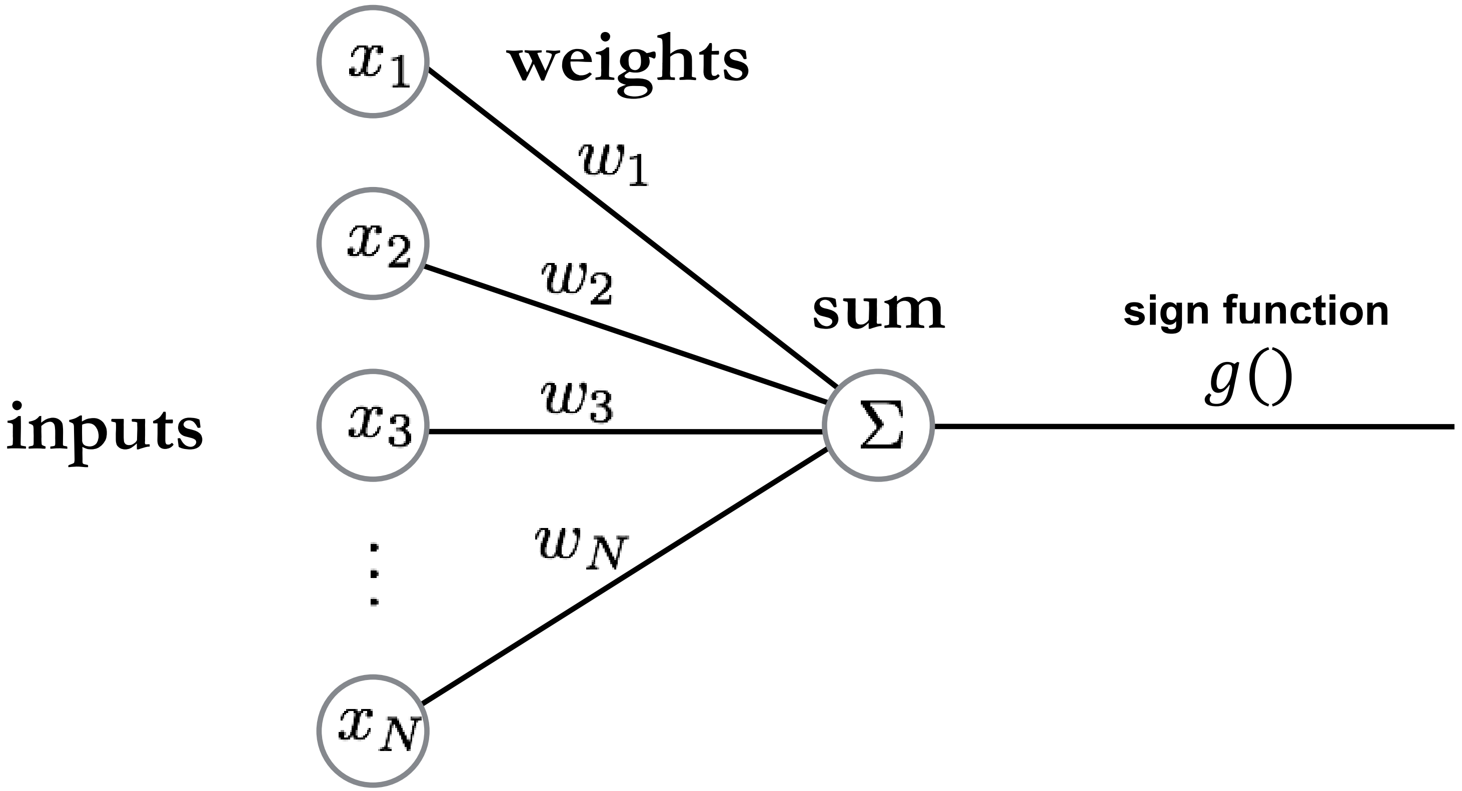


Multilayer Perceptron (MLP)

Non-linear Activation Functions

Definition of MLPs

The Perceptron



$$y = \begin{cases} 1, & \text{if } \boldsymbol{w}^T \boldsymbol{x} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Slide credit: Angela Yao

Linear functions are limited

- Simple binary example:

$$y = \begin{cases} 1, & \text{if } \boldsymbol{w}^T\boldsymbol{x} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Possible weights for AND function:

$$w_1 = 1, w_2 = 1, b = -1.5$$

AND function

x1	x2	y
1	1	1
1	0	0
0	0	0
0	1	0

XOR function

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Find the weights for the XOR function?

HWQ: can logistic regression model fit the XOR data?

Non-Linear feature transformations

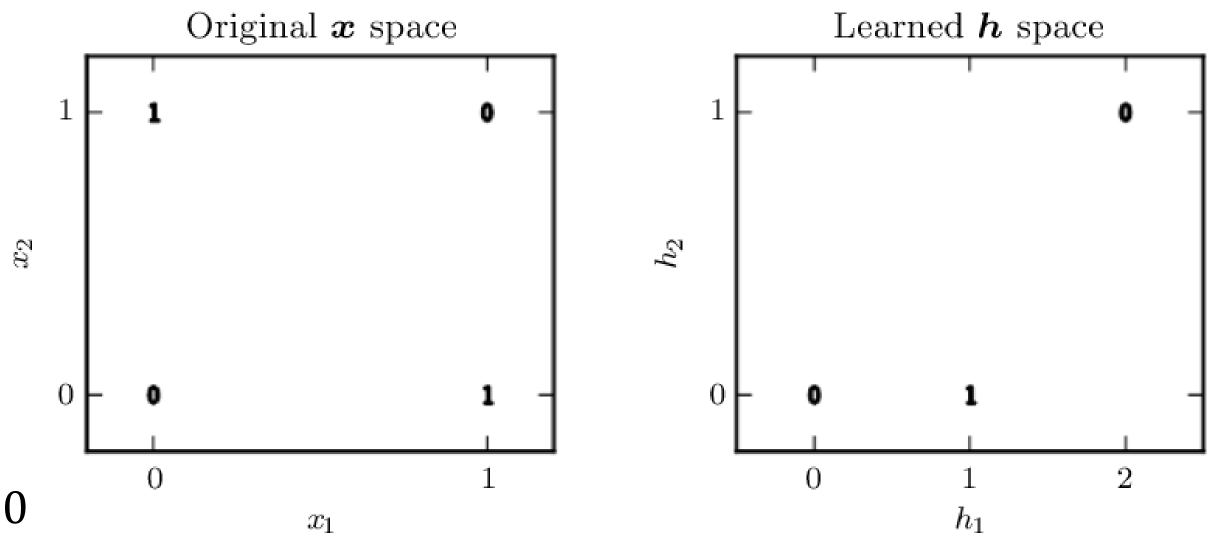
- $\boldsymbol{h} = \max(0, W\boldsymbol{x} + \boldsymbol{c}), W \in R^{2\times 2}, c \in R^2$
- $\boldsymbol{h}^{\boldsymbol{T}}\boldsymbol{w} + b, \boldsymbol{w} \in R^2, b \in R$

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad b = 0$$

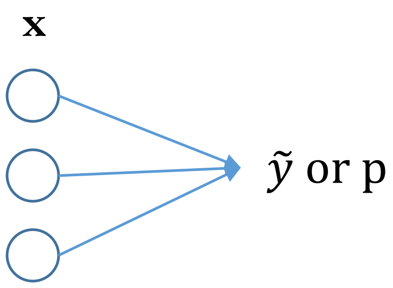


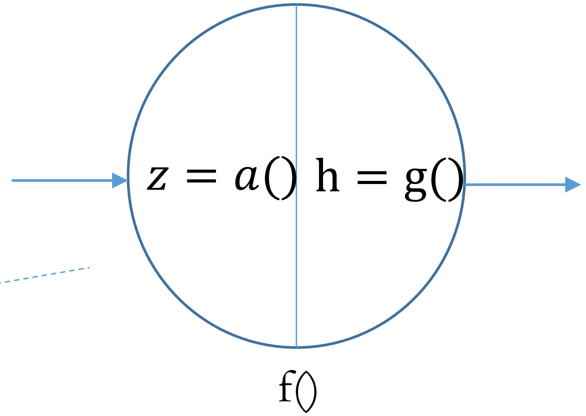
Try it! Compute h and $\tilde{y}$ to see the classification results

More examples at google playground

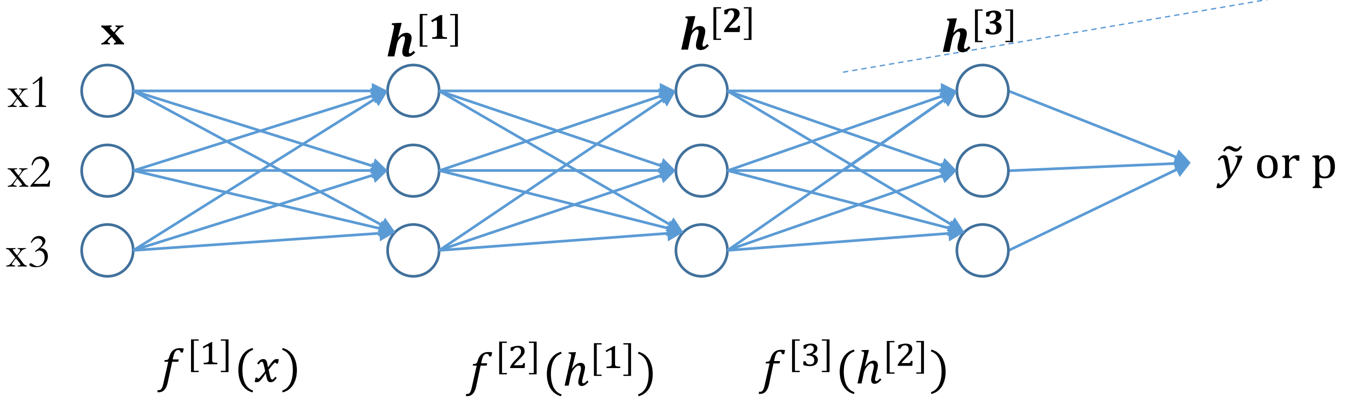
Multilayer Perceptron

Single layer perceptron (no hidden layers)





Multi-layer perceptron



Multilayer Perceptron

A net with multiple layers that transform input features into hidden features and then make predictions

- At least one **(non-linear) hidden layer**
- i-th layer consists of a **linear/affine** transformation function

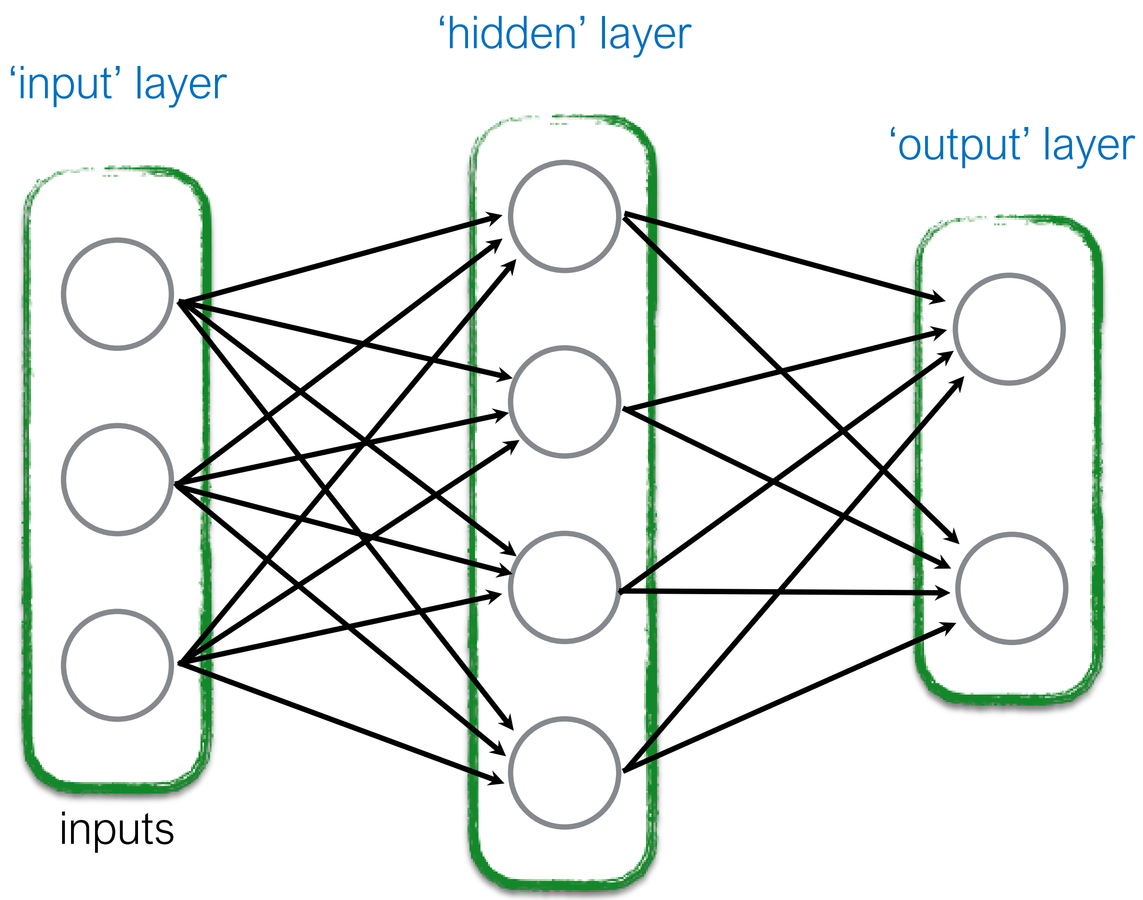
$$\boldsymbol{z}^{[i]} = a^{[i]}\left(\boldsymbol{h}^{[i-1]}\right) = W^{[i]}\boldsymbol{h}^{[\boldsymbol{i-1}]} \quad + \boldsymbol{b}^{[i]}$$

$$W^{[i]} \in R^{n_i \times n_{i-1}}, \boldsymbol{b}^{i} \in R^{n_i}$$

- m_i is the number of hidden units at the i-th layer and is a hyper-parameter
- followed by a **non-linear activation** function

$$\boldsymbol{h}^{[i]} = g^{[i]}(\boldsymbol{z}^{[i]}), \in R^{n_i}$$

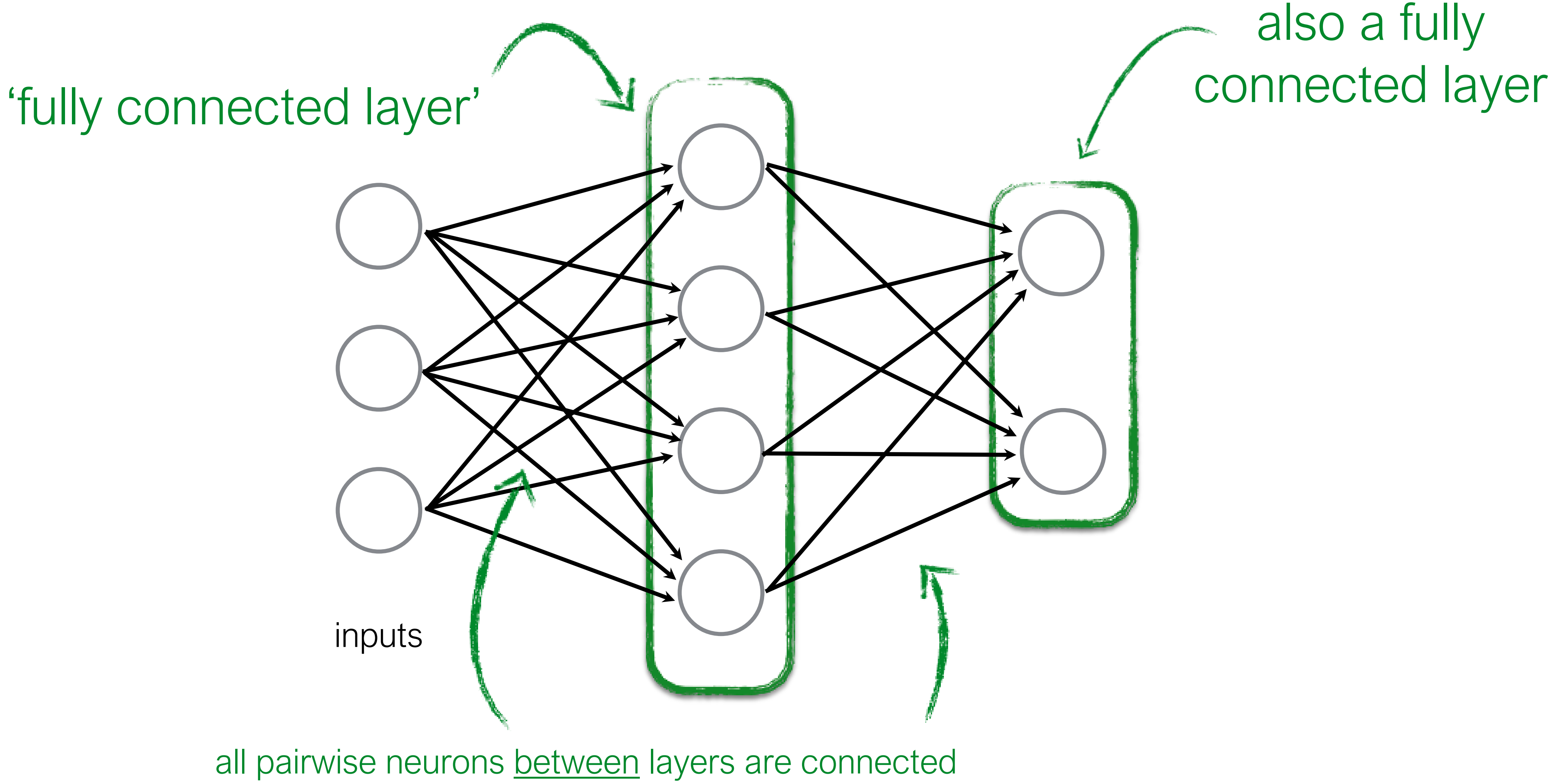
Multilayer Perceptron



...also called a **Multi-layer Perceptron** (MLP)

Slide Credit: K. Kitani

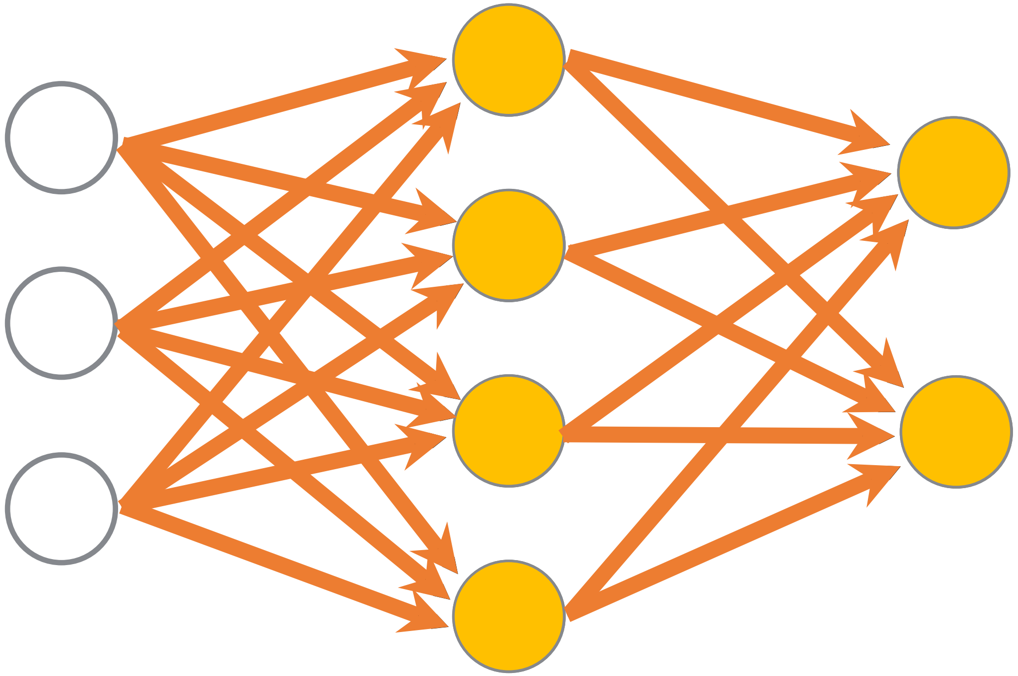
Multilayer Perceptron



Slide Credit: K. Kitani

How many neurons (perceptrons)? $4 + 2 = 6$

How many weights (edges)? $(3 \times 4) + (4 \times 2) = 20$



How many learnable parameters total? $20 + (4 + 2) = 26$

6 bias terms
1 per perceptrons

Slide Credit: K. Kitani

Why non-linear activation?

This result says that if all activation functions were linear, we can always define an equivalent collapsed network.

Because successive linear transformations together form yet another linear transformation, a multi-layered linear network is not so interesting. Therefore, we need non-linear activations.

Slide credit: Angela Yao

Activation function $g()$

- Logistic (Sigmoid, σ) VS ReLU

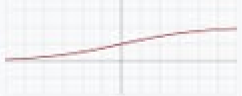
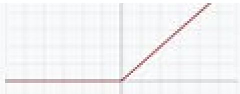
Logistic activation

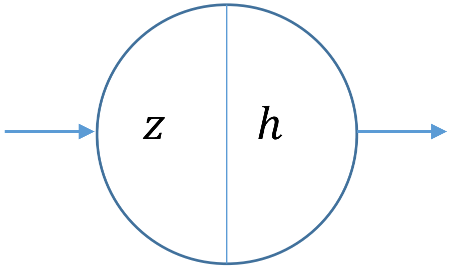
If $z_k$is large, e.g. >10, then h_k is near 1

If z_k is small, e.g. <-10, Then h is near 0

For both cases, $\frac{\partial h_k}{\partial z_k} \approx 0$ → gradient vanishing

→ gradient vanishing

Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$



$h^{[i]} = g^{[i]}(z^{[i]})$

ReLU activation

If z_k is positive,$\frac{\partial h_k}{\partial z_k} = 1$, no gradient vanishing

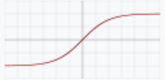
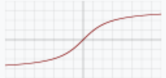
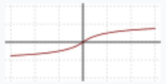
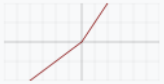
If z_k is negative, $\frac{\partial h_k}{\partial z_k} = 0$ no gradients ☹

better than sigmoid, since z_k has a larger working domain

Leaky ReLU (see previous slides) resolves the gradient vanishing problem for negative z_k

Activation function $g()$

- Activation functions perform non-linear feature transformations

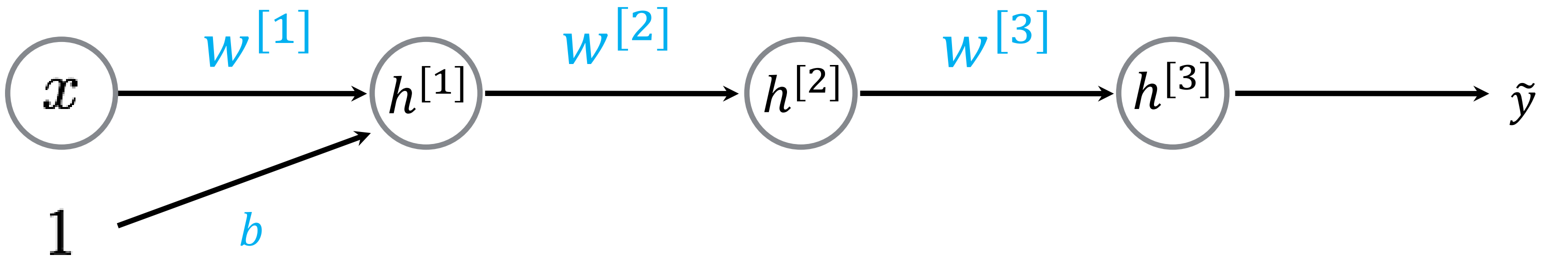
name	plot	equation	derivative	range
Binary step		$f(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x\neq 0\\ ? & \text{for } x=0\end{cases}$	$\{0,1\}$
Logistic (a.k.a. Soft step)		$f(x)=\frac{1}{1+e^{-x}}$	$f'(x)=f(x)(1-f(x))$	$(0,1)$
TanH		$f(x)=\tanh(x)=\frac{2}{1+e^{-2x}}-1$	$f'(x)=1-f(x)^2$	$(-1,1)$
ArcTan		$f(x)=\tan^{-1}(x)$	$f'(x)=\frac{1}{x^2+1}$	$\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$
Softsign [7][8]		$f(x)=\frac{x}{1+\lvert x\rvert}$	$f'(x)=\frac{1}{(1+\lvert x\rvert)^2}$	$(-1,1)$
Rectified linear unit (ReLU)[9]		$f(x)=\begin{cases}0 & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$[0,\infty)$
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x)=\begin{cases}0.01x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0.01 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$
Parameteric rectified linear unit (PReLU)[11]		$f(\alpha,x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(\alpha,x)=\begin{cases}\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$
Randomized leaky rectified linear unit (RReLU)[12]		$f(\alpha,x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$ [1]	$f'(\alpha,x)=\begin{cases}\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$

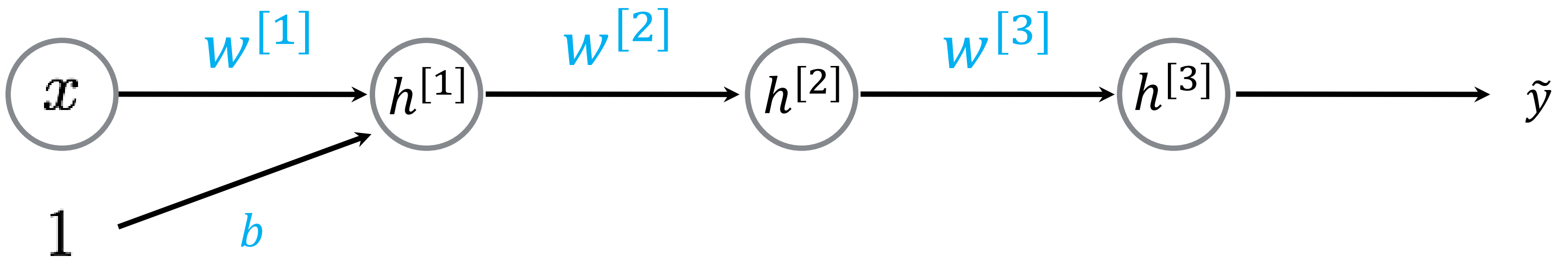
Source from: https://en.wikipedia.org/wiki/Activation_function

Training MLP

- Cross-entropy loss for classification
- Squared Euclidean distance for regression
- GD algorithm
  - Compute $J(\boldsymbol{X}, \boldsymbol{Y})$
  - Compute gradient : $\frac{\partial J}{\partial \boldsymbol{W}^{[1]}}, \frac{\partial J}{\partial \boldsymbol{b}^{[1]}}, \frac{\partial J}{\partial \boldsymbol{W}^{[2]}}, \frac{\partial J}{\partial \boldsymbol{b}^{[2]}} \cdots$
  - Update: $\boldsymbol{W}^{[\boldsymbol{k}]} = \boldsymbol{W}^{[\boldsymbol{k}]} - \alpha \frac{\partial J}{\partial \boldsymbol{W}^{[k]}}, \boldsymbol{b}^{[\boldsymbol{k}]} = \boldsymbol{b}^{[\boldsymbol{k}]} - \alpha \frac{\partial J}{\partial \boldsymbol{b}^{[k]}}$

A simple MLP example





$$h^{[1]} = g^{[1]}(w^{[1]}x + b)$$

$$h^{[2]} = g^{[2]}\left(w^{[2]}g^{[2]}\right)$$

$$h^{[3]} = g^{[3]}(w^{3}h^{[2]})$$

$$\tilde{y} = h^{[3]}$$

Entire network can be written out as one long equation

known

$$y$$

$$\tilde{y} = h^{[3]} = g^{[3]}(w^{[3]}g^{[2]}(w^{[2]}g^{[1]}(w^{[1]}x + b)))$$

unknown

We need to train the network:

What is known? What is unknown?

Slide Credit: K. Kitani

Gradient descent algorithm

```
Random initialize
```
$w^{[1]}, w^{[2]}$ …

```
Repeat
      J = 0
      For each sample
```
$\boldsymbol{x}^{(i)}, y^{(i)}$

```
         a. Forward pass
         b. Accumulate Loss
         c. Accumulate partial gradient (derivative)
```
how to compute?

$\frac{\partial J}{\partial w^{[1]}}, \frac{\partial J}{\partial w^{[2]}}$ …

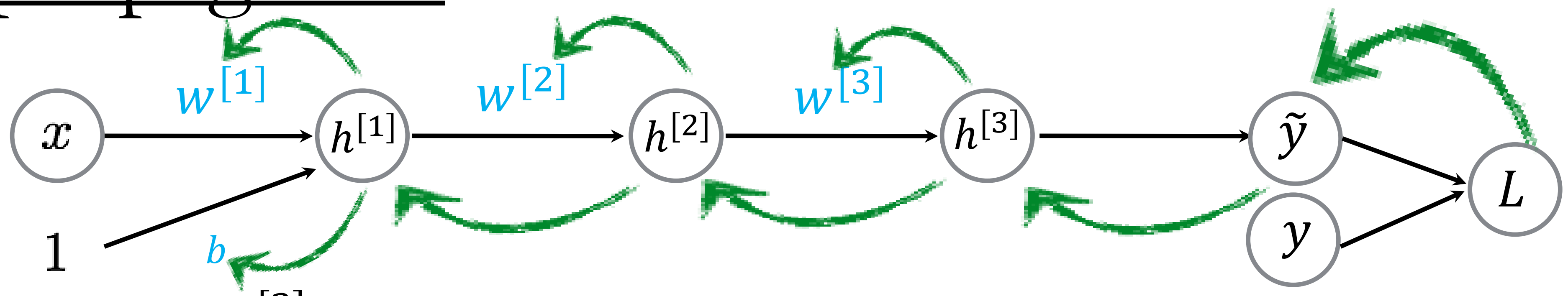
```
      Update
```

$$w^{[1]} = w^{[1]} - \alpha \frac{\partial J}{\partial w^{[1]}}$$

$$w^{[2]} = w^{[2]} - \alpha \frac{\partial J}{\partial w^{[2]}}$$

move in opposite direction of partial derivatives

Back-propagation



$loss = L(\tilde{y}, y) \qquad \tilde{y} = h^{[3]}$

$h^{[3]} = g^{[3]}(w^3 h^{[2]})$

$h^{[2]} = g^{[2]}\left(w^{[2]} g^{[2]}\right)$

$h^{[1]} = g^{[1]}(w^{[1]} x + b)$

$$\frac{\partial L}{\partial w^{[3]}} = \frac{\partial L}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial h^{[3]}} \frac{\partial h^{[3]}}{\partial w^{[3]}}$$

$$\frac{\partial L}{\partial w^{[2]}} = \frac{\partial L}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial h^{[3]}} \frac{\partial h^{[3]}}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial w^{[2]}}$$

$$\frac{\partial L}{\partial w^{[1]}} = \frac{\partial L}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial h^{[3]}} \frac{\partial h^{[3]}}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial w^{[1]}}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial h^{[3]}} \frac{\partial h^{[3]}}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial b}$$

The term back-propagation comes from the application of the chain rule, in which the gradients or partial derivatives from down-stream are used upstream.

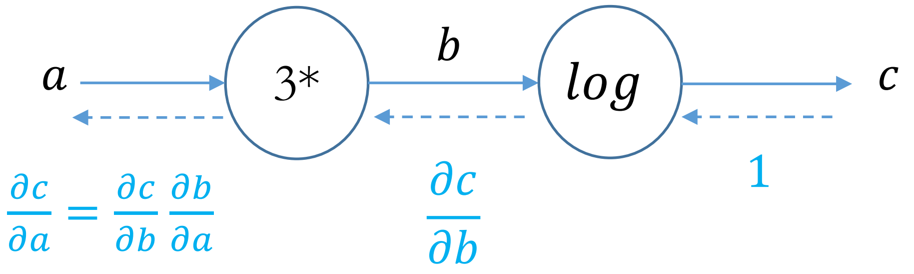
Backpropagation

How to compute the gradient?

- GD algorithm needs the gradient of the loss w.r.t each parameter
- Linear regression model
  - $\frac{\partial L}{\partial \boldsymbol{w}} = (\tilde{y} - y)\boldsymbol{x}$
- Logistic regression model
  - $\frac{\partial L}{\partial \boldsymbol{w}} = (p - y)\boldsymbol{x}$
- Softmax regression model
  - $\frac{\partial L}{\partial W} = (\boldsymbol{p} - \boldsymbol{y})\boldsymbol{x}^T$
- How about MLP model with 10 hidden layers?
  - We need a **modular** way to compute the gradients

Chain rule I

- $c = \log 3a$ Given $a = 1$, what is $\dfrac{\partial c}{\partial a}$?



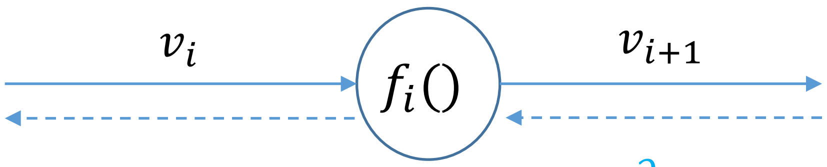
- $b = f_1(a) = 3a, \quad c = f_2(b) = \log b \;\rightarrow c = \log 3a$
- $\dfrac{\partial c}{\partial a} = \dfrac{\partial c}{\partial b}\dfrac{\partial b}{\partial a} = f_2'(b)f_1'(a) = \dfrac{1}{b} * 3 = \dfrac{1}{3a} * 3 = \dfrac{3}{3} = 1$

Chain rule I

- Basic chain rule
  - $v_2 = f_1(v_1), v_3 = f_2(v_2) \quad \dots \quad v_k = f_{k-1}(v_{k-1})$
  - v_i could be a scalar, vector, matrix, tensor
  - $\frac{\partial v_k}{\partial v_i} = mul(\frac{\partial v_k}{\partial v_{i+1}}, \frac{\partial v_{i+1}}{\partial v_i})$
    - Reorder and transpose
    - $\frac{\partial v_k}{\partial v_{i+1}} \frac{\partial v_{i+1}}{\partial v_i}$ if all are scalars
    - $\frac{\partial v_{i+1}}{\partial v_i} \frac{\partial v_k}{\partial v_{i+1}}$ if v_k is a scalar, v_{i+1}, v_i are vectors

General multiplication operation; to avoid the scalar/matrix/tensor transpose/ordering messiness …

Some special cases; shape-check to confirm!

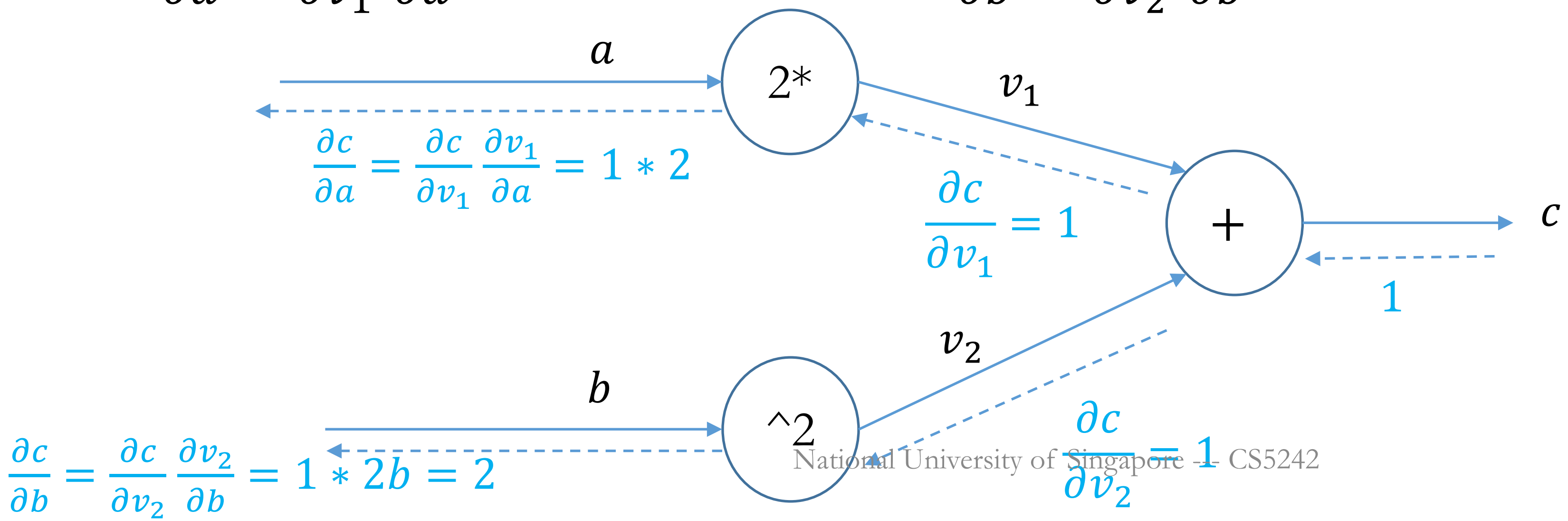


$$\frac{\partial v_k}{\partial v_i} = mul(\frac{\partial v_k}{\partial v_{i+1}}, \frac{\partial v_{i+1}}{\partial v_i}) \qquad \frac{\partial v_k}{\partial v_{i+1}}$$

$\frac{\partial v_{i+1}}{\partial v_i}$: $f_i'(\mathrm{v_i})$

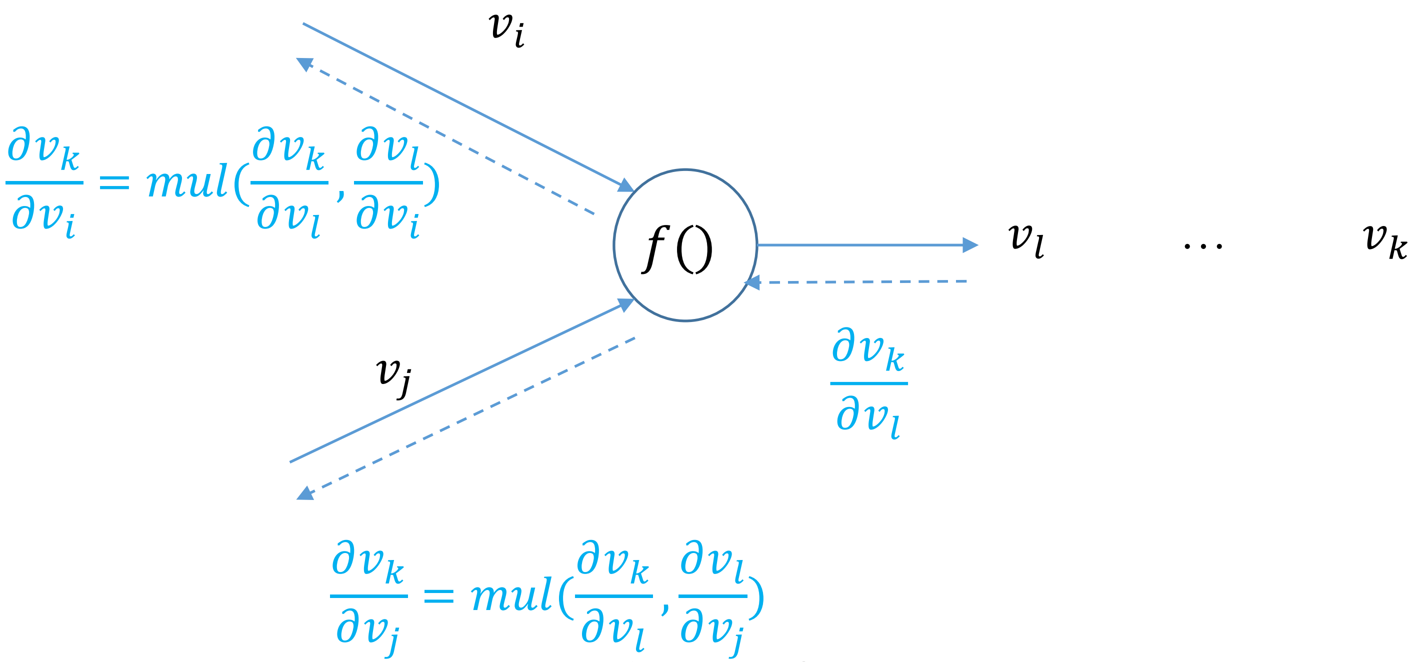
Chain rule II

- $c = f(a,b) = 2a + b^2$
  - Given, $a = 1, b = 1$
  - $\frac{\partial c}{\partial a} = 2, \quad \frac{\partial c}{\partial b} = 2b = 2$
- $v_1 = f_1(a) = 2a, v_2 = f_2(b) = b^2, c = f_3(v_1, v_2) = v_1 + v_2$
  - $\frac{\partial c}{\partial a} = \frac{\partial c}{\partial v_1}\frac{\partial v_1}{\partial a} = 1 * 2 = 2 \qquad \frac{\partial c}{\partial b} = \frac{\partial c}{\partial v_2}\frac{\partial v_2}{\partial b} = 1 * 2b = 2$



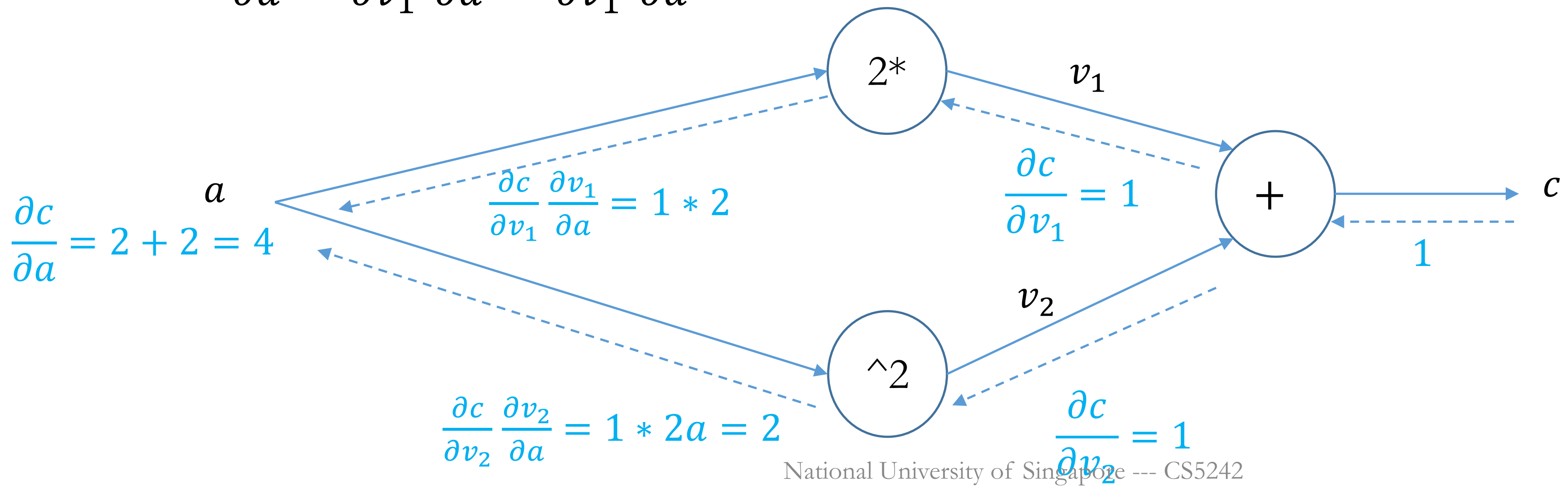
Chain rule II

- $v_l = f(v_i, v_j)$,
- v_k is the final output



Chain rule III

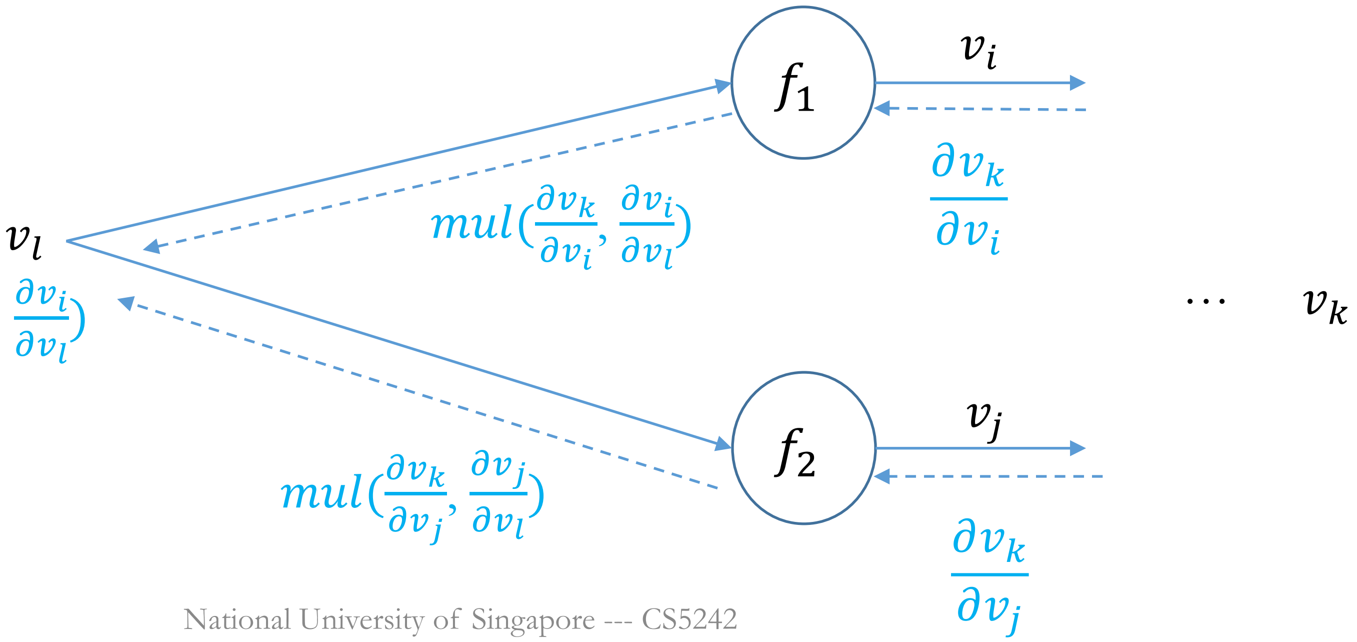
- $c = 2a + a^2$
  - Given $a = 1$, $\frac{\partial c}{\partial a} = 2 + 2a = 4$
- $v_1 = f_1(a) = 2a, v_2 = f_2(a) = a^2, \mathrm{c} = f_3(v_1, v_2) = v_1 + v_2$
  - $\frac{\partial c}{\partial a} = \frac{\partial c}{\partial v_1}\frac{\partial v_1}{\partial a} + \frac{\partial c}{\partial v_1}\frac{\partial v_1}{\partial a} = 1 * 2 + 1 * 2a = 4$



Chain rule III

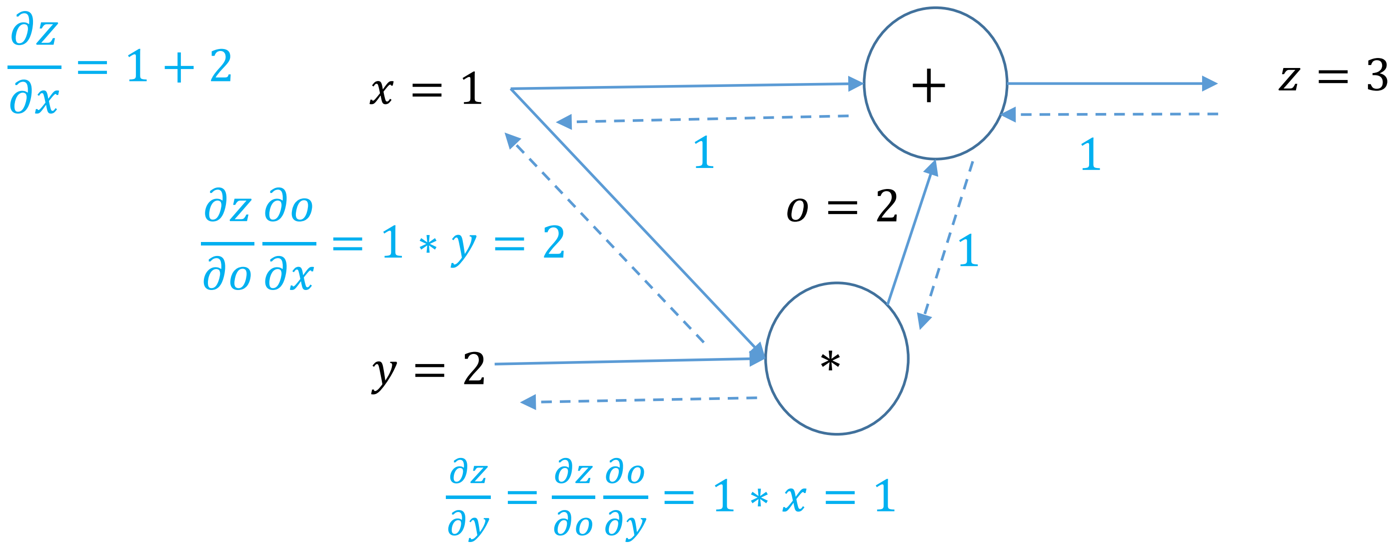
- $v_i = f_1(v_l)$
- $v_j = f_2(v_l)$
- v_k is the final output

$$\frac{\partial v_k}{\partial v_l} = mul\left(\frac{\partial v_k}{\partial v_j}, \frac{\partial v_j}{\partial v_l}\right) + mul\left(\frac{\partial v_k}{\partial v_i}, \frac{\partial v_i}{\partial v_l}\right)$$



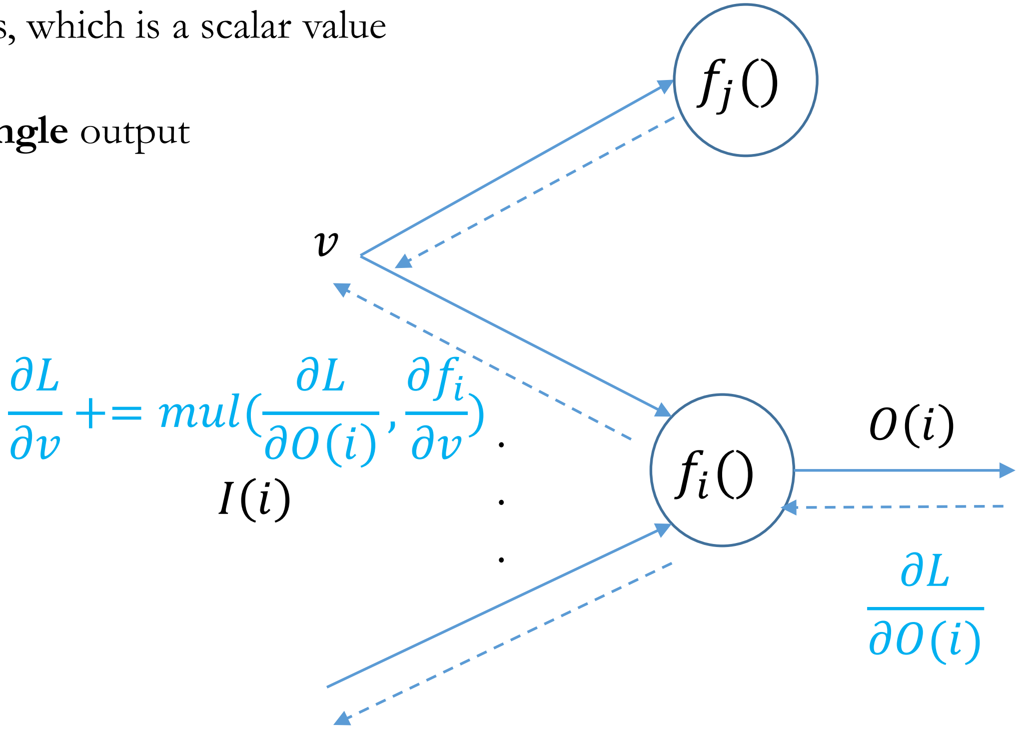
Practice

- $z = f(x, y) = x * y + x$
- Given $\mathrm{x} = 1, \mathrm{y} = 2$, compute $\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}$?

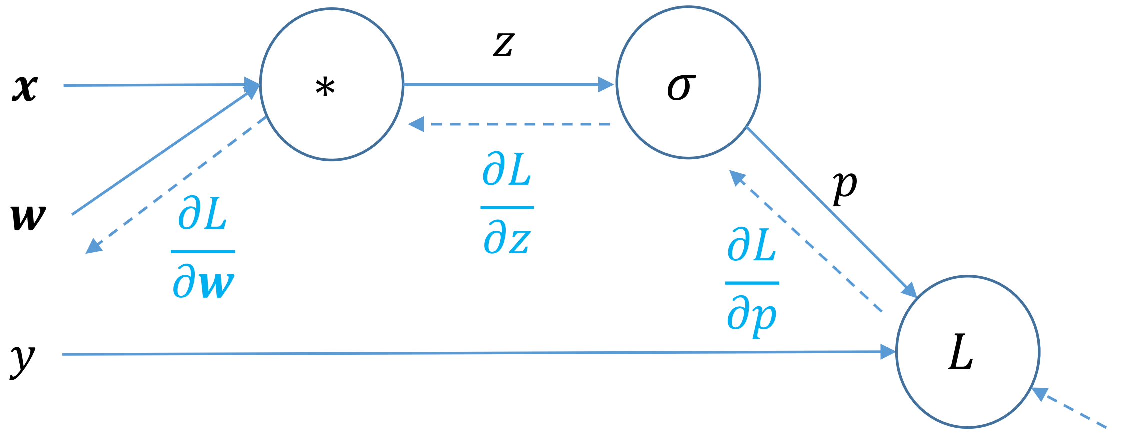


Backpropagation (BP)

- Computation graph
  - Denote all variables as $v_1, v_2, \ldots$ sorted in topological order
    - The last variable is the loss (L) for neural network models, which is a scalar value
  - Denote all operations as $f_1, f_2, \ldots f_k$
    - Each operation accepts multiple inputs and generate **a single** output
      - which may be used by multiple other operations
    - $I(i)$ denotes all the input variables to f_i
    - $O(i)$ denotes the single output variable of f_i
- Forward pass
  - For i = 1, 2, … k
    - Run operation f_i: $I(i) \rightarrow O(i)$
- Backward pass
  - Initialize $\frac{\partial L}{\partial v} = 0$ for all v
  - For i = k, … 1 Note the reverse order!
    - For each $v \in I(i)$ Compute $\frac{\partial L}{\partial v} += mul(\frac{\partial L}{\partial O(i)}, \frac{\partial f_i(I(i))}{\partial v})$



Logistic regression



$$dz = \frac{\partial L}{\partial z}, \qquad dp = \frac{\partial L}{\partial p} \qquad \frac{\partial L}{\partial L} = 1$$

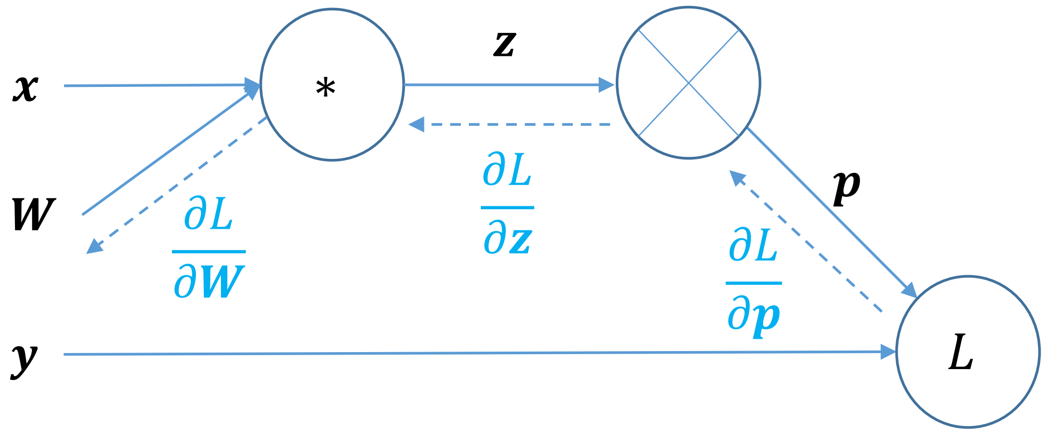
- $z = \boldsymbol{w}^T \boldsymbol{x}, \boldsymbol{x} \in R^{n \times 1}, \boldsymbol{w} \in R^{n \times 1}$
- $p = \sigma(z)$
- $L(p, y) = -ylogp - (1 - y) \log(1 - p)$
- Dot
  - Forward$(\boldsymbol{x}, \boldsymbol{w})$: $\boldsymbol{w}^T \boldsymbol{x}$
  - Backward$(dz, \boldsymbol{x}, \boldsymbol{w})$: $dz\ \boldsymbol{x}$
- Logistic
  - Forward(z): $\sigma(z)$
  - Backward(dp, z) : $\mathrm{p} = \sigma(z);\ dp * p * (1 - p) = (-\frac{y}{p} + \frac{1-y}{1-p}) * p * (1 - p) = p - y$
- Binary-Cross-Entropy
  - Forward(p, y): $-ylogp - (1 - y) \log(1 - p)$
  - Backward$([\mathrm{dL}], p, y)$: $-\frac{y}{p} + \frac{1-y}{1-p}$

  dL is redundant since it is 1

This operation can also be further decomposed; but since binary cross-entropy is used so commonly, we treat it as a whole stand-alone unit.

Softmax regression

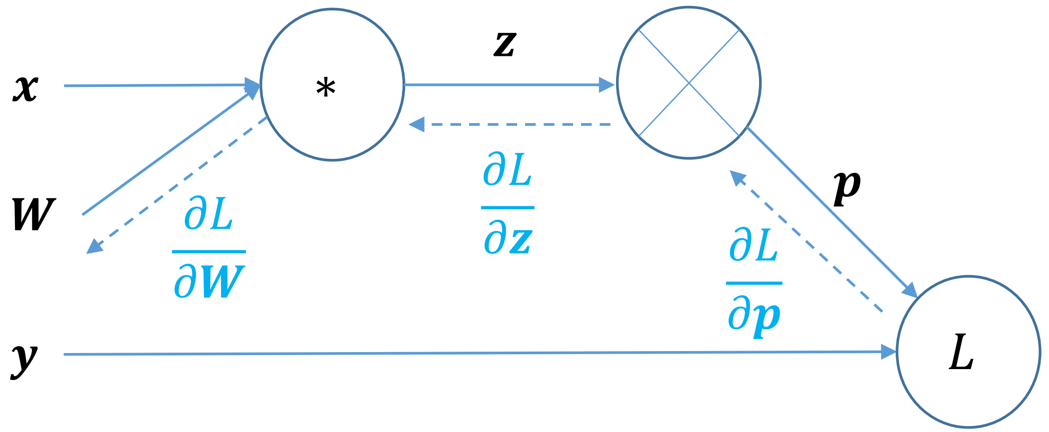
- $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x},$ $\quad \boldsymbol{x} \in R^n, W \in R^{k*n}$
- $\boldsymbol{p} = softmax(\boldsymbol{z}) \quad \boldsymbol{p} \in R^k$
- $L(\boldsymbol{p}, \boldsymbol{y}) = \sum_{i=1} -y_i \log p_i$



- Matmul matrix multiplication
  - Forward$(\boldsymbol{x}, \boldsymbol{W})$: $\boldsymbol{W}\boldsymbol{x}$
  - Backward$(\boldsymbol{dz}, \boldsymbol{x}, \boldsymbol{W})$
- Softmax
  - Forward$(\boldsymbol{z})$: $\boldsymbol{p}$
  - Backward$(\boldsymbol{dp}, \boldsymbol{z})$
- Cross-entropy
  - Forward$(\boldsymbol{p}, \boldsymbol{y})$: $\sum_{i=1} -y_i \log p_i$
  - Backward$(\boldsymbol{p}, \boldsymbol{y})$

Softmax regression

- $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x},$ $\quad \boldsymbol{x} \in R^n, W \in R^{k*n}$
- $\boldsymbol{p} = softmax(\boldsymbol{z})$ $\quad \boldsymbol{p} \in R^k$
- $L(\boldsymbol{p}, \boldsymbol{y}) = \sum_{i=1} -y_i \log p_i$



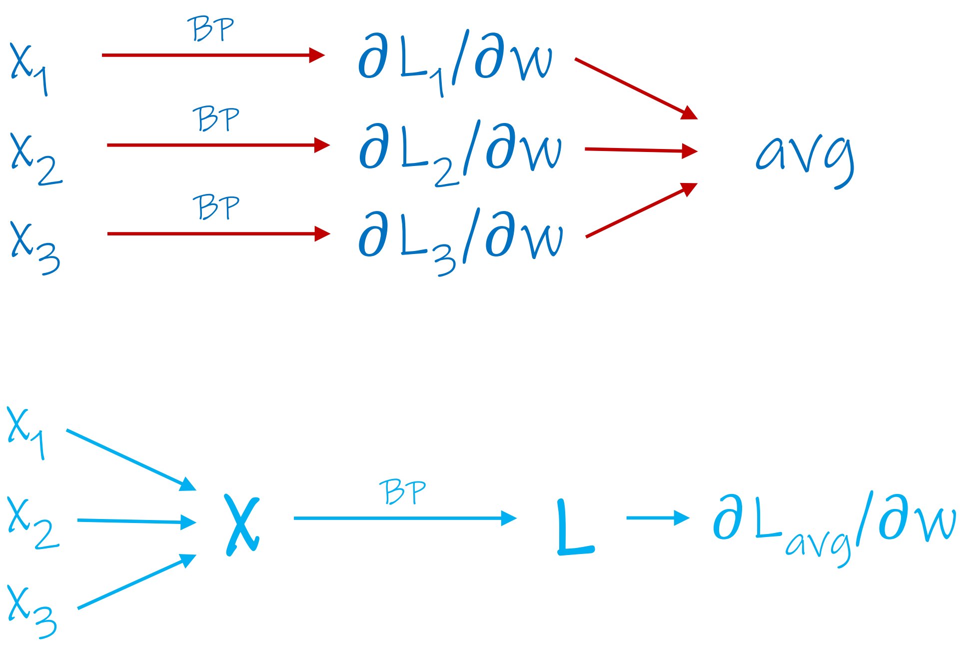
L is a complex node

- Matmul
  - Forward$(\boldsymbol{x}, \boldsymbol{W})$: $\boldsymbol{W}\boldsymbol{x}$
  - Backward$(d\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{W})$: $d\boldsymbol{z}\ \boldsymbol{x}^{\boldsymbol{T}}$
- Softmax-Cross-Entropy
  - Forward$(\boldsymbol{z})$: $\boldsymbol{p}$; $\sum_{i=1} -y_i \log p_i$
  - Backward$(\boldsymbol{z}, \boldsymbol{y})$: $\boldsymbol{p} - \boldsymbol{y}$

BP for multiple examples

So far, we've considered backpropagation for a single example.
How should we handle all of our training samples?

- Approach I (individual)
  - Backpropagation for each example separately
  - Average the gradients across all examples
- Approach II (vectorised)
  - Use matrix (one row per example) in the BP
  - Average the loss
  - Compute gradient based on averaged loss



Which approach is better and why?

Slide Credit: Angela Yao

BP operations

These are all derived via chain rule

- Add_bias e.g. $z = XW + b$
  - $\boldsymbol{A}$ is a matrix; b is a row vector
  - Forward$(\boldsymbol{A}, b)$: $\boldsymbol{C} = A + b$
  - Backward$(d\boldsymbol{C}, \boldsymbol{A}, b)$: $d\boldsymbol{A} = d\boldsymbol{C}$, $db = \mathbf{1}^T\, d\boldsymbol{C}$
- Array and scalar multiplication
  - $\boldsymbol{v}$ is an array, $\boldsymbol{k}$ is a scalar
  - Forward$(\boldsymbol{v}, k)$: $c = k\ v$
  - Backward$(d\boldsymbol{c}, \boldsymbol{v}, k)$:$d\boldsymbol{v} = k\ d\boldsymbol{c}$

BP operations

- Matmul matrix multiplication operation
  - $\boldsymbol{A} \in R^{m*k}, \boldsymbol{B} \in R^{k*n}$ (including matrix with a single column or row)
  - Forward($\boldsymbol{A}, \boldsymbol{B}$): $\boldsymbol{C} = AB \in R^{m*n}$
  - Backward($\boldsymbol{dC}, \boldsymbol{A}, \boldsymbol{B}$): $d\boldsymbol{A} =?$, $d\boldsymbol{B} =?$
- Logistic operation
  - a is an array of any shape
  - Forward(a): $b = \sigma(a)$
  - Backward(db, a): $da =?$
- Softmax-Cross-entropy operation
  - $Z \text{ and } Y \in R^{m*k}$ are matrix of the same shape; each row of a (or b) sums to 1
  - Forward ($\boldsymbol{Z}, \boldsymbol{Y}$): $\boldsymbol{P} = softmax(\boldsymbol{Z}); L = sum\ (-\boldsymbol{Y} log \boldsymbol{P})/m$
  - Backward($\boldsymbol{Z}, \boldsymbol{Y}$): $d\boldsymbol{Z} =?$

Summary

- Extending simple Perceptron to Multilayer Perceptron model
- Backpropagation algorithm
  - A modular way to compute the gradient of the loss w.r.t parameters
  - Based on chain rules and matrix calculus

Homework & Practice

- Homework
  - slide 6 & 38,
- Practice
  - Colab notebook (implement the BP operations in Python)