

# Machine Learning

## Lecture 11: Neural Networks 2

Hao Tang and Hiroshi Shimodaira

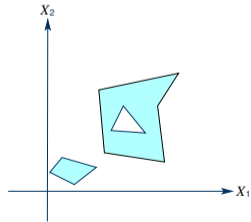
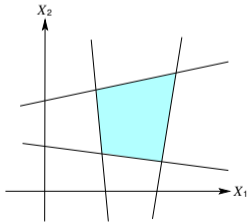
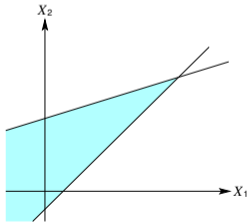
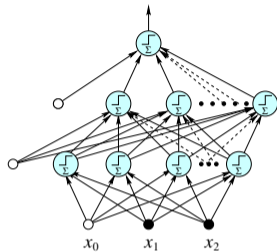
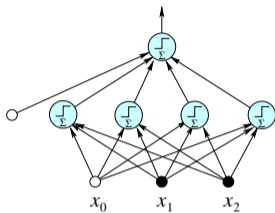
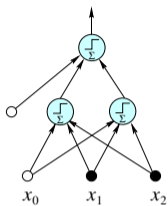
14 October 2022

*Ver. 1.0*

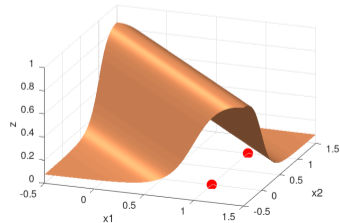
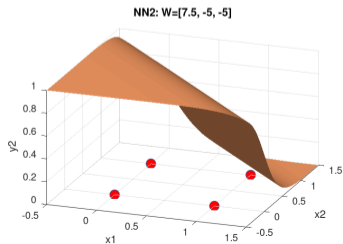
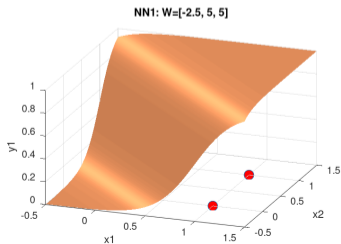
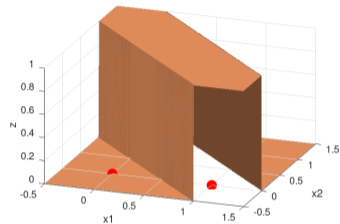
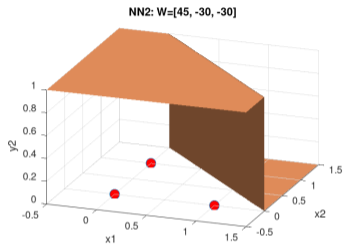
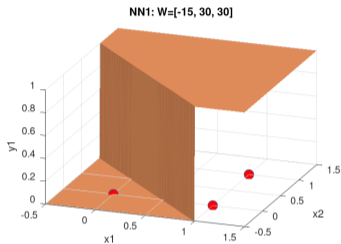
## Topics - you should be able to explain after this week

- Neural network with logistic sigmoid activation functions
- Interpretation of the output
- Training of single-layer neural network with MSE / cross entropy
- Gradient descent
- Activation functions
- Training of multi-layer neural network – error back propagation
- Computation graphs

# Recap: Perceptron structures and decision boundaries



# Recap: Output of NN – threshold func. vs sigmoid func.



## Recap: Ability of neural networks

- Universal approximation theorem

- “Univariate function and a set of affine functionals can uniformly approximate any continuous function of  $n$  real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. “ (G. Cybenko (1989)

→

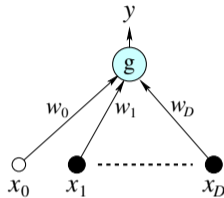
A single-output node neural network with a single hidden layer with a finite neurons can approximate continuous functions.

- K. Hornik (1990) doi:10.1016/0893-6080(91)90009-T
- N. Guliyev, V. Ismailov (2018) doi:10.31219/osf.io/xgnw8

## Output of logistic sigmoid activation function

- Consider a single-layer network with a single output node logistic sigmoid activation function: (cf. logistic regression)

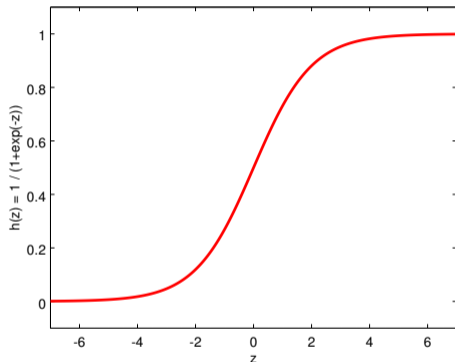
$$y = g(a) = \frac{1}{1 + \exp(-a)}, \quad \text{where } a = \sum_{i=0}^D w_i x_i$$
$$= \frac{1}{1 + \exp\left(-\sum_{i=0}^D w_i x_i\right)}$$



- Consider a two class problem, with classes  $C_1$  and  $C_2$ . The posterior probability of  $C_1$ :

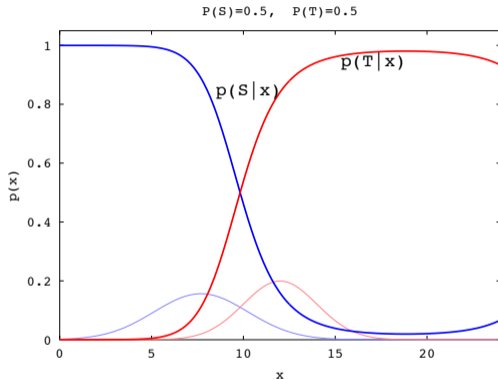
$$P(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1) P(C_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_1) P(C_1)}{p(\mathbf{x}|C_1) P(C_1) + p(\mathbf{x}|C_2) P(C_2)}$$
$$= \frac{1}{1 + \frac{p(\mathbf{x}|C_2) P(C_2)}{p(\mathbf{x}|C_1) P(C_1)}} = \frac{1}{1 + \exp\left(-\log \frac{p(\mathbf{x}|C_1) P(C_1)}{p(\mathbf{x}|C_2) P(C_2)}\right)}$$

# Approximation of posterior probabilities



Logistic sigmoid function:

$$g(a) = \frac{1}{1 + \exp(-a)}$$



Posterior probabilities of two classes with Gaussian distributions:

# Training single layer neural network with MSE

- Training set :  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ , where  $y_i \in \{0, 1\}$

- Error function:

$$\begin{aligned} E_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2} \sum_{n=1}^N (\hat{y}_n - y_n)^2 \\ &= \frac{1}{2} \sum_{n=1}^N \left( g(\mathbf{w}^T \mathbf{x}_n) - y_n \right)^2 \\ &= \frac{1}{2} \sum_{n=1}^N \left( g \left( \sum_{i=0}^D w_i x_{ni} \right) - y_n \right)^2 \end{aligned}$$

- Definition of the training problem as an optimisation problem

$$\min_{\mathbf{w}} E_{\text{MSE}}(\mathbf{w})$$



## Training single layer neural network with MSE (*cont.*)

- Optimisation problem:  $\min_{\mathbf{w}} E_{\text{MSE}}(\mathbf{w})$
- No analytical solution
- Employ an iterative method (requires initial values)  
e.g. **Gradient descent** (steepest descent), Newton's method, Conjugate gradient methods

- Gradient descent  
(scalar form)

$$w_i^{(\text{new})} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(\mathbf{w}), \quad (\eta > 0)$$

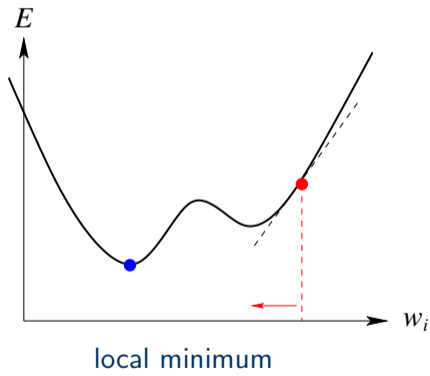
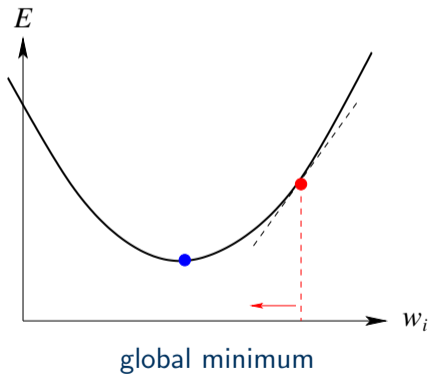
(vector form)

$$\mathbf{w}^{(\text{new})} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E(\mathbf{w}), \quad (\eta > 0)$$

- **Online/stochastic gradient descent** (cf. Batch training)  
Sample  $(x_n, y_n)$  from the data set and update the weights at a time.

# Gradient descent

$$w_i^{(\text{new})} \leftarrow w_i - \eta \frac{\partial}{\partial w_i} E(\mathbf{w}), \quad (\eta > 0)$$



## Training single layer neural network with MSE (cont.)

$$E_{\text{MSE}}(\mathbf{w}) = \sum_{n=1}^N E_n, \quad \text{where } E_n = \frac{1}{2} (\hat{y}_n - y_n)^2 = \frac{1}{2} \left( g \left( \sum_{i=0}^D w_i x_{ni} \right) - y_n \right)^2$$

where  $\hat{y}_n = g(a_n)$ ,  $a_n = \sum_{i=0}^D w_i x_{ni}$ ,  $\frac{\partial a_n}{\partial w_i} = x_{ni}$

$$\begin{aligned} \frac{\partial E_n(\mathbf{w})}{\partial w_i} &= \frac{\partial E_n(\mathbf{w})}{\partial \hat{y}_n} \frac{\partial \hat{y}_n}{\partial a_n} \frac{\partial a_n}{\partial w_i} \\ &= (\hat{y}_n - y_n) \frac{\partial g(a_n)}{\partial a_n} \frac{\partial a_n}{\partial w_i} \\ &= (\hat{y}_n - y_n) g'(a_n) x_{ni} \\ &= (\hat{y}_n - y_n) g(a_n) (1 - g(a_n)) x_{ni} \quad \text{if } g(\cdot) \text{ is a sigmoid function} \end{aligned}$$

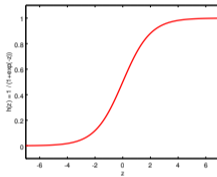
## Another training criterion – cross-entropy error

- Training problem with the mean squared error (MSE) criterion with the sigmoid function

$$E_{\text{MSE}}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (\hat{y}_n - y_n)^2, \quad \hat{y}_n = g(a_n)$$

$$\frac{\partial E_{\text{MSE}}(\mathbf{w})}{\partial w_i} = \sum_{n=1}^N (\hat{y}_n - y_n) g'(a_n) x_{ni}, \quad g'(a) = g(a)(1 - g(a))$$

$g'(a) \approx 0$  for such  $a$  that  $g(a) \approx 0$  or  $1$ .



- Cross-entropy error

$$E_{\text{H}}(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N \{ y_n \log \hat{y}_n + (1 - y_n) \log (1 - \hat{y}_n) \}$$

For multi classes,  $E_{\text{H}}(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N \sum_i y_i \log \hat{y}_i$

It can be shown that:

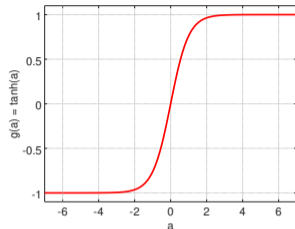
$$\frac{\partial E_{\text{H}}(\mathbf{w})}{\partial w_i} = \frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n) x_{ni}$$

## Other activation functions

- Tanh

$$g(a) = \tanh(a) = \frac{1 - e^{-2a}}{1 + e^{-2a}}$$

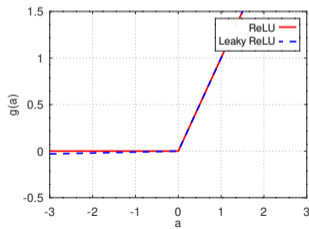
- Mapping  $(-\infty, +\infty) \rightarrow (-1, 1)$
- 0 (zero) centred  $\rightarrow$  faster convergence than sigmoid



- ReLU (Rectified Linear Unit)

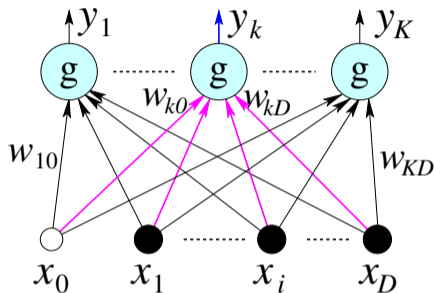
$$g(a) = \max(0, a)$$

- Several times faster than tanh.
- 'Dying ReLU' problem – a unit of outputting 0 always  $\rightarrow$  use Leaky ReLU instead



See M1 Sec. 13.4.3 for details

## Single-layer network with multiple output nodes



$$y_1(\mathbf{x}) = g(\mathbf{w}_1^T \mathbf{x} + w_{10})$$

$$\vdots$$

$$y_K(\mathbf{x}) = g(\mathbf{w}_K^T \mathbf{x} + w_{K0})$$

$$\begin{pmatrix} y_1 \\ \vdots \\ y_K \end{pmatrix} = g \left( \begin{pmatrix} w_{10} & w_{11} & \dots & w_{1D} \\ \vdots & & \ddots & \vdots \\ w_{K0} & w_{K1} & \dots & w_{KD} \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix} \right)$$

$$\mathbf{y} = g(\mathbf{W}\hat{\mathbf{x}})$$

- $K$  output nodes:  $y_1, \dots, y_K$ .
- For  $\mathbf{x}_n = (x_{n0}, \dots, x_{nD})^T$ ,

$$\hat{y}_{nk} = g\left(\sum_{d=0}^D w_{kd} x_{nd}\right) = g(a_{nk}), \quad a_{nk} = \sum_{d=0}^D w_{kd} x_{nd}$$

## Training of single-layer network with multiple output nodes

- Training set :  $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$   
where  $\mathbf{y}_n = (y_{n1}, \dots, y_{nK})$  and  $y_{nk} \in \{0, 1\}$
- Error function:

$$\begin{aligned} E_{\text{MSE}}(\mathbf{w}) &= \frac{1}{2} \sum_{n=1}^N \|\hat{\mathbf{y}}_n - \mathbf{y}_n\|^2 \\ &= \sum_{n=1}^N E_n, \quad \text{where } E_n = \frac{1}{2} \|\hat{\mathbf{y}}_n - \mathbf{y}_n\|^2 = \frac{1}{2} \sum_{k=1}^K (\hat{y}_{nk} - y_{nk})^2 \end{aligned}$$

- Training with the gradient descent:

$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}, \quad (\eta > 0)$$

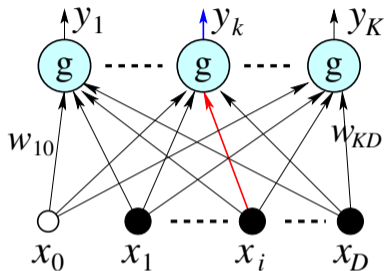
## The derivatives of the error function (single-layer)

$$E_n = \frac{1}{2} \sum_{k=1}^K (\hat{y}_{nk} - y_{nk})^2$$

$$\hat{y}_{nk} = g(a_{nk})$$

$$a_{nk} = \sum_{i=0}^D w_{ki} x_{ni}$$

$$\begin{aligned} \frac{\partial E_n}{\partial w_{ki}} &= \frac{\partial E_n}{\partial \hat{y}_{nk}} \frac{\partial \hat{y}_{nk}}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{ki}} \\ &= (\hat{y}_{nk} - y_{nk}) g'(a_{nk}) x_{ni} \end{aligned}$$





## Normalisation of output nodes - softmax

- Outputs with sigmoid activation function:

$$\sum_{k=1}^K y_k \neq 1$$

$$y_k = g(a_k) = \frac{1}{1 + \exp(-a_k)}, \quad a_k = \sum_{i=0}^D w_{ki} x_i$$

- **Softmax** activation function:

$$y_k = \frac{\exp(a_k)}{\sum_{\ell=1}^K \exp(a_{\ell})}$$

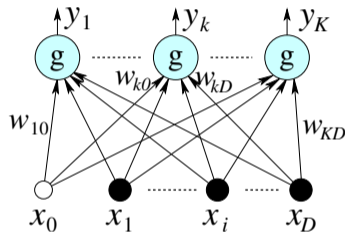
- Properties of the softmax function

(i)  $0 \leq y_k \leq 1$

(iii) differentiable

(ii)  $\sum_{k=1}^K y_k = 1$

(iv)  $y_k \approx P(C_k | \mathbf{x}) = \frac{p(\mathbf{x} | C_k) P(C_k)}{\sum_{\ell=1}^K p(\mathbf{x} | C_{\ell}) P(C_{\ell})}$



# Training of multi-layer neural networks

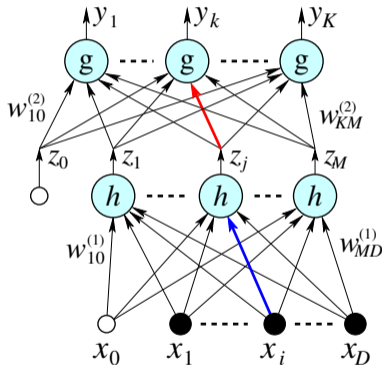
## Multi-layer perceptron (MLP)

- Hidden-to-output weights:

$$w_{kj}^{(2)} \leftarrow w_{kj}^{(2)} - \eta \frac{\partial E}{\partial w_{kj}^{(2)}}$$

- Input-to-hidden weights:

$$w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} - \eta \frac{\partial E}{\partial w_{ji}^{(1)}}$$



# Training of MLP

- 1940s Warren McCulloch and Walter Pitts : 'threshold logic'  
Donald Hebb : 'Hebbian learning'
- 1957 Frank Rosenblatt : 'Perceptron'
- 1969 Marvin Minsky and Seymour Papert : limitations of neural networks
- 1980 Kunihiro Fukushima: 'Neocognitoron'
- 1986 D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors" (1974, Paul Werbos)

## The derivatives of the error function (two-layers)

$$E_n = \frac{1}{2} \sum_{k=1}^K (\hat{y}_{nk} - y_{nk})^2$$

$$\hat{y}_{nk} = g(a_{nk}), \quad a_{nk} = \sum_{j=1}^M w_{kj}^{(2)} z_{nj}$$

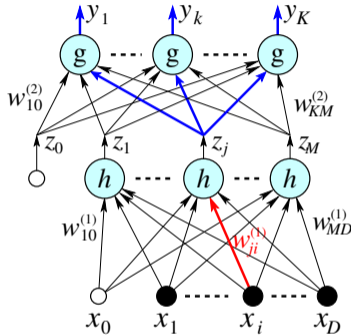
$$z_{nj} = h(b_{nj}), \quad b_{nj} = \sum_{i=0}^D w_{ji}^{(1)} x_{ni}$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial \hat{y}_{nk}} \frac{\partial \hat{y}_{nk}}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{kj}^{(2)}}$$

$$= (\hat{y}_{nk} - y_{nk}) g'(a_{nk}) z_{nj}$$

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial b_{nj}} \frac{\partial b_{nj}}{\partial w_{ji}^{(1)}} = \left( \sum_{k=1}^K \frac{\partial E_n}{\partial \hat{y}_{nk}} \frac{\partial \hat{y}_{nk}}{\partial z_{nj}} \right) h'(b_{nj}) x_{ni}$$

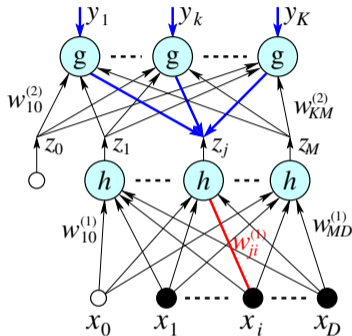
$$= \left( \sum_{k=1}^K (\hat{y}_{nk} - y_{nk}) g'(a_{nk}) w_{kj}^{(2)} \right) h'(b_{nj}) x_{ni}$$



## Error back propagation

$$\begin{aligned} \frac{\partial E_n}{\partial w_{kj}^{(2)}} &= \frac{\partial E_n}{\partial \hat{y}_{nk}} \frac{\partial \hat{y}_{nk}}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{kj}^{(2)}} \\ &= (\hat{y}_{nk} - y_{nk}) g'(a_{nk}) z_{nj} \\ &= \delta_{nk}^{(2)} z_{nj}, \quad \delta_{nk}^{(2)} = \frac{\partial E_n}{\partial a_{nk}} \end{aligned}$$

$$\begin{aligned} \frac{\partial E_n}{\partial w_{ji}^{(1)}} &= \frac{\partial E_n}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial b_{nj}} \frac{\partial b_{nj}}{\partial w_{ji}^{(1)}} \\ &= \left( \sum_{k=1}^K (\hat{y}_{nk} - y_{nk}) g'(a_{nk}) w_{kj}^{(2)} \right) h'(b_{nj}) x_{ni} \\ &= \left( \sum_{k=1}^K \delta_{nk}^{(2)} w_{kj}^{(2)} \right) h'(b_{nj}) x_{ni} \end{aligned}$$



# Practical representations - computation graph

- Consider a two-layer neural network with softmax output

- 1st layer with sigmoid activation functions:

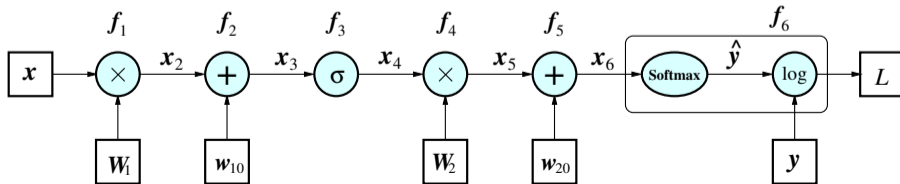
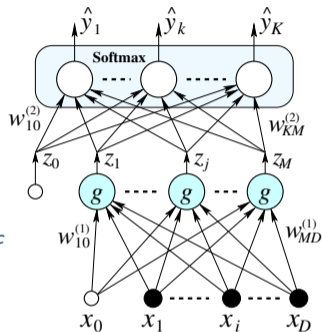
$$\mathbf{z} = g(\mathbf{x}) = \sigma(W_1 \mathbf{x} + w_{10})$$

- 2nd layer with a softmax activation function:

$$\hat{\mathbf{y}} = \text{softmax}(W_2 \mathbf{z} + w_{20})$$

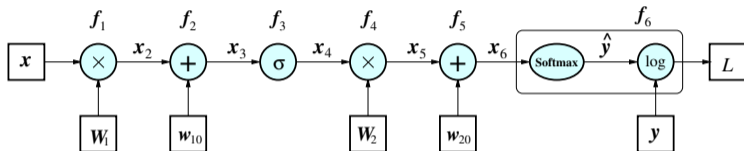
- Cross-entropy loss function

$$L = -\sum y_i \log \hat{y}_i = -\log \hat{y}_c = -\log \text{softmax}(W_2 \mathbf{z} + w_{20})_c$$



# Computation graph

Represents computation as a directed graph comprising of simple operations on vectors and matrices  $\Rightarrow$  Automatic differentiation (NE)



$$L = f(\mathbf{x}) = f_6(f_5(f_4(f_3(f_2(f_1(\mathbf{x}))))))$$

$$f = f_6 \circ f_5 \circ f_4 \circ f_3 \circ f_2 \circ f_1$$

$$f_1 : \mathbf{x}_2 = W_1 \mathbf{x}$$

$$f_2 : \mathbf{x}_3 = \mathbf{x}_2 + w_{10}$$

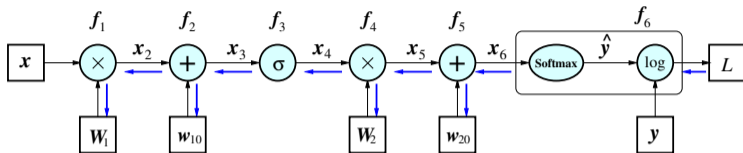
$$f_3 : \mathbf{x}_4 = \sigma(\mathbf{x}_3)$$

$$f_4 : \mathbf{x}_5 = W_2 \mathbf{x}_4$$

$$f_5 : \mathbf{x}_6 = \mathbf{x}_5 + w_{20}$$

$$f_6 : L = \log \text{softmax}(\mathbf{x}_6)_{i=y}$$

## Computation graph (cont.)



$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \mathbf{x}_6} \frac{\partial \mathbf{x}_6}{\partial \mathbf{x}_5} \frac{\partial \mathbf{x}_5}{\partial W_2}$$

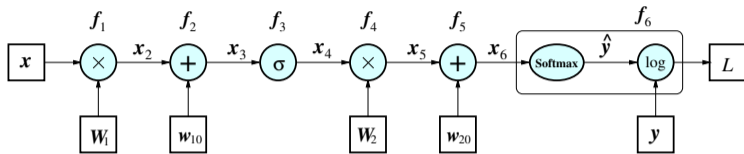
NB: matrix transpose is omitted for simplicity

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \mathbf{x}_6} \frac{\partial \mathbf{x}_6}{\partial \mathbf{x}_5} \frac{\partial \mathbf{x}_5}{\partial \mathbf{x}_4} \frac{\partial \mathbf{x}_4}{\partial \mathbf{x}_3} \frac{\partial \mathbf{x}_3}{\partial W_1}$$

- Forward pass: compute  $\mathbf{x}_2, \dots, \mathbf{x}_6, \hat{\mathbf{y}}, L$ .
- Backward pass: compute  $\frac{\partial L}{\partial w_{20}}, \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial w_{10}}, \frac{\partial L}{\partial W_1}$ .



## Computation graph – cross entropy layer



$$L = -\log \hat{y}_c = -\log \frac{e^{a_c}}{\sum_j e^{a_j}}, \quad \text{where } c \text{ is the true class, } \mathbf{a} = \mathbf{x}_4$$

$$\frac{\partial L}{\partial a_i} = \frac{\partial}{\partial a_i} \left( \log(\sum_j e^{a_j}) - a_c \right) = \frac{e^{a_i}}{\sum_j e^{a_j}} - \mathbb{1}_{i=c} = \hat{y}_i - \mathbb{1}_{i=c}$$

$$\frac{\partial L}{\partial \mathbf{x}_6} = \hat{\mathbf{y}} - \mathbf{y}$$

## Quizzes

- On slide 12, show the following:

$$\frac{\partial E_H(\mathbf{w})}{\partial w_j} = \frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n) x_{ni}$$

- On Slide 24, find the following:

- $\frac{\partial \mathbf{x}_6}{\partial w_{20}}$

- $\frac{\partial \mathbf{x}_5}{\partial W_2}$

- $\frac{\partial \mathbf{x}_4}{\partial \mathbf{x}_3}$

## Appendix – derivatives

- Derivatives of functions of one variable

$$\frac{df}{dx} = f'(x) = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

e.g.,  $f(x) = 4x^3$ ,  $f'(x) = 12x^2$

- Partial derivatives of functions of more than one variable

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon, y) - f(x, y)}{\epsilon}$$

e.g.,  $f(x, y) = y^3x^2$ ,  $\frac{\partial f}{\partial x} = 2y^3x$

# Derivative rules

	Function	Derivative
Constant	$c$	$0$
	$x$	$1$
Power	$x^n$	$nx^{n-1}$
	$\frac{1}{x}$	$-\frac{1}{x^2}$
	$\sqrt{x}$	$\frac{1}{2}x^{-\frac{1}{2}}$
Exponential	$e^x$	$e^x$
Logarithms	$\ln(x)$	$\frac{1}{x}$
Sum rule	$f(x) + g(x)$	$f'(x) + g'(x)$
Product rule	$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
Reciprocal rule	$\frac{1}{f(x)}$	$-\frac{f'(x)}{f^2(x)}$
	$\frac{f(x)}{g(x)}$	$-\frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)}$
Chain rule	$f(g(x))$	$f'(g(x))g'(x)$
	$z = f(y), y = g(x)$	$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$

## Vectors of derivatives

Consider  $f(\mathbf{x})$ , where  $\mathbf{x} = (x_1, \dots, x_D)^T$

Notation: all partial derivatives put in a vector:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_D} \right)^T$$

Example:  $f(\mathbf{x}) = x_1^3 x_2^2$

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{pmatrix} 3x_1^2 x_2^2 \\ 2x_1^3 x_2 \end{pmatrix}$$

Fact:  $f(\mathbf{x})$  changes most quickly in direction  $\nabla_{\mathbf{x}} f(\mathbf{x})$

See M1 Sec. 7.8 for details.