

Introduction to Computer Vision

12. Neural Networks

UCLA – CS 188 – Fall 2019

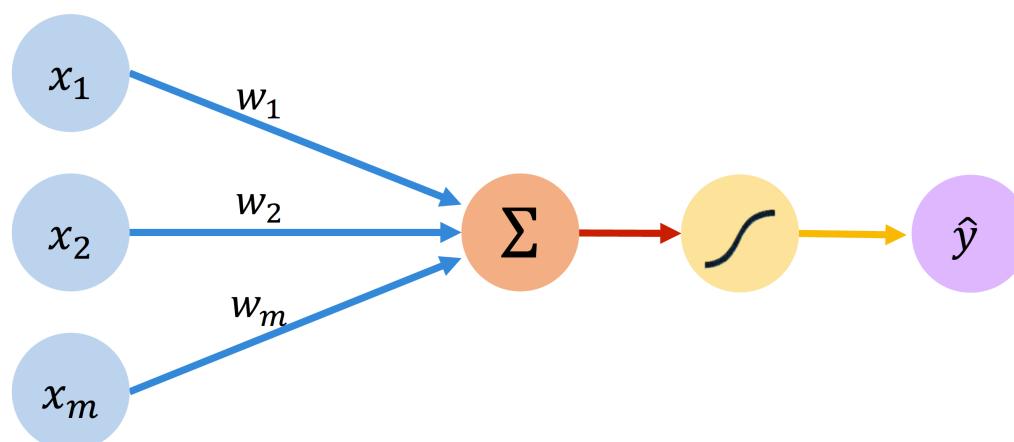
Fabien Scalzo, Ph.D.

Week 1			26-Sep	Introduction
Week 2	1-Oct	Basic Image Processing	3-Oct	Feature Extraction and Classification
Week 3	8-Oct	Feature Tracking/Optical Flow	10-Oct	SVD, 2D camera model, projective plane
Week 4	15-Oct	2D Image transformations, RANSAC	17-Oct	Euclidean geometry, rigid body motion
Week 5	22-Oct	Epipolar Geometry	24-Oct	3D Cameras and processing
Week 6	29-Oct	Midterm	31-Oct	3D Cameras and processing
Week 7	5-Nov	Learning from data	7-Nov	Neural Networks
Week 8	12-Nov	Deep Learning	14-Nov	Deep Learning
Week 9	19-Nov	Object Detection	21-Nov	Generative Models
Week 10	26-Nov	Guest Lecture (Nikhil Naik)	28-Nov	
Week 11	3-Dec	Applications	5-Dec	Recap
Week 12	10-Dec		12-Dec	Final

-
- The Perceptron
 - Neural Networks
 - Single Layer
 - Multi-output
 - Deep Neural Network
 - Gradient descent
 - Back-propagation
 - Chain rule
 - Computational Graph
 - Algorithm
 - Examples
-

The Perceptron (1958)

The neural network unit (or artificial neuron)



Linear combination
of inputs

Output

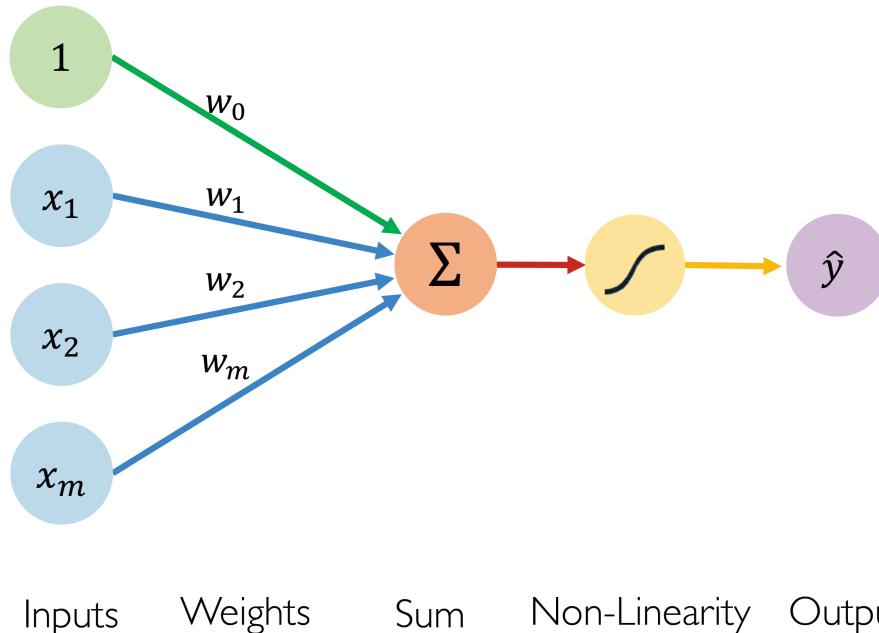
$$\hat{y} = g \left(\sum_{i=1}^m x_i w_i \right)$$

Non-linear activation function

Inputs Weights Sum Non-Linearity Output

The Perceptron (1958)

The neural network unit (or artificial neuron)



Linear combination of inputs

Output

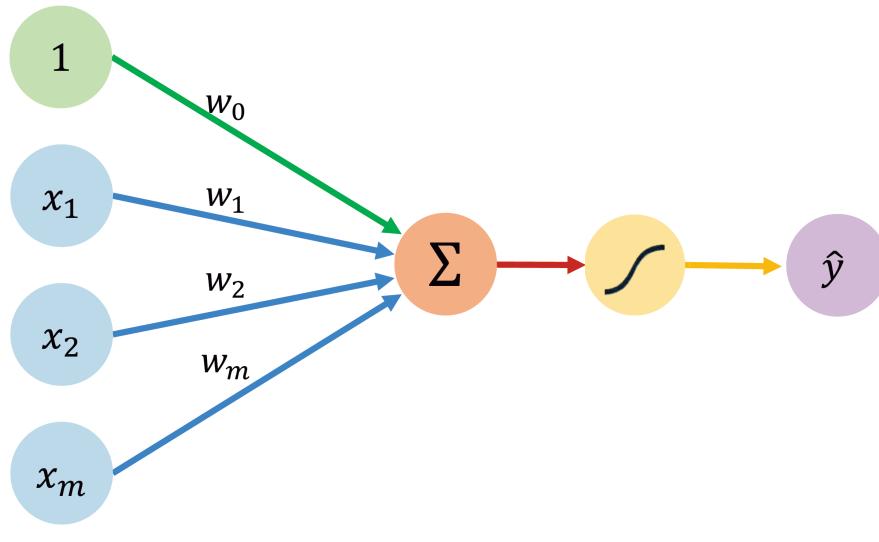
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

Non-linear activation function

Bias

The Perceptron (1958)

The neural network unit (or artificial neuron)



Inputs Weights Sum Non-Linearity Output

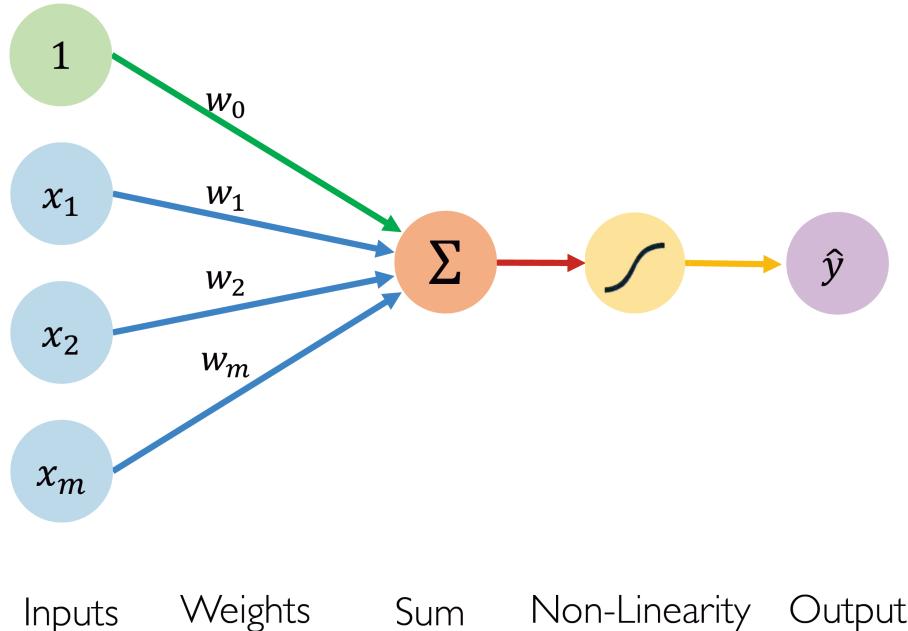
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

$$\hat{y} = g (w_0 + \mathbf{X}^T \mathbf{W})$$

where: $\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

The Perceptron (1958)

The neural network unit (or artificial neuron)

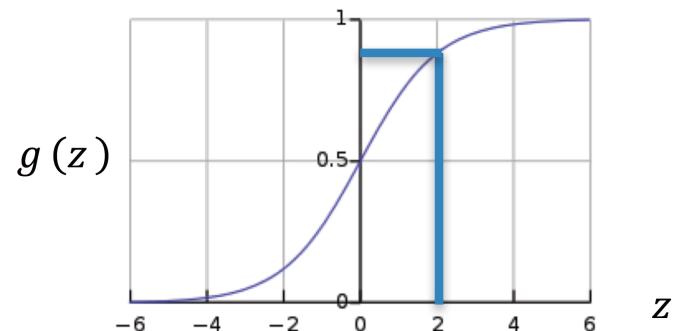


Activation Functions

$$\hat{y} = g(w_0 + \mathbf{X}^T \mathbf{W})$$

- Example: sigmoid function

$$g(z) = \frac{1}{1 + e^{-z}}$$



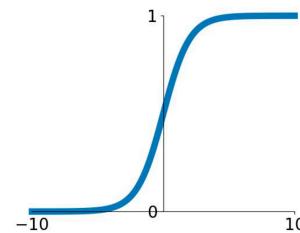
The Perceptron (1958)

Activation Functions



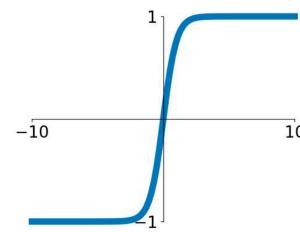
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



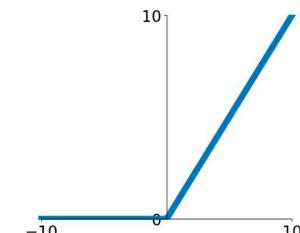
tanh

$$\tanh(x)$$



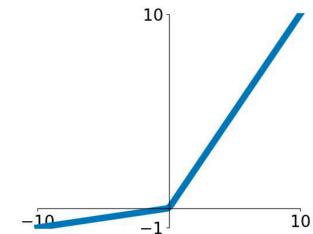
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

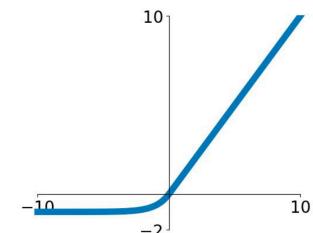


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

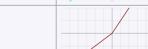


ReLU: Rectified Linear Unit

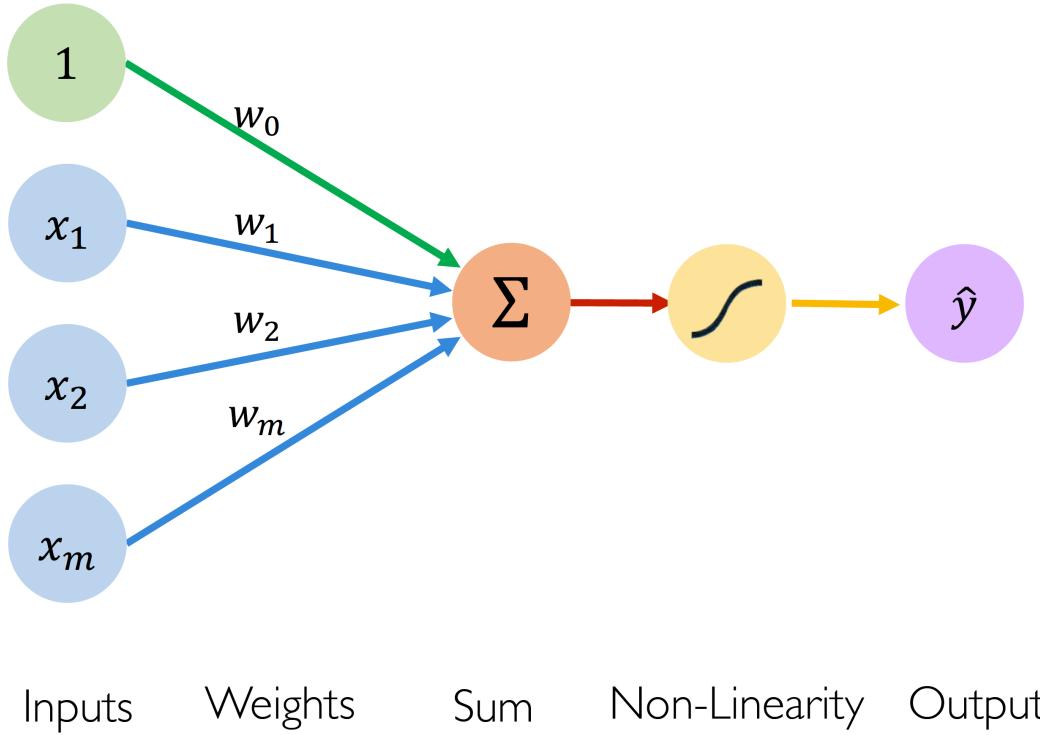
The Perceptron (1958)

Activation Functions

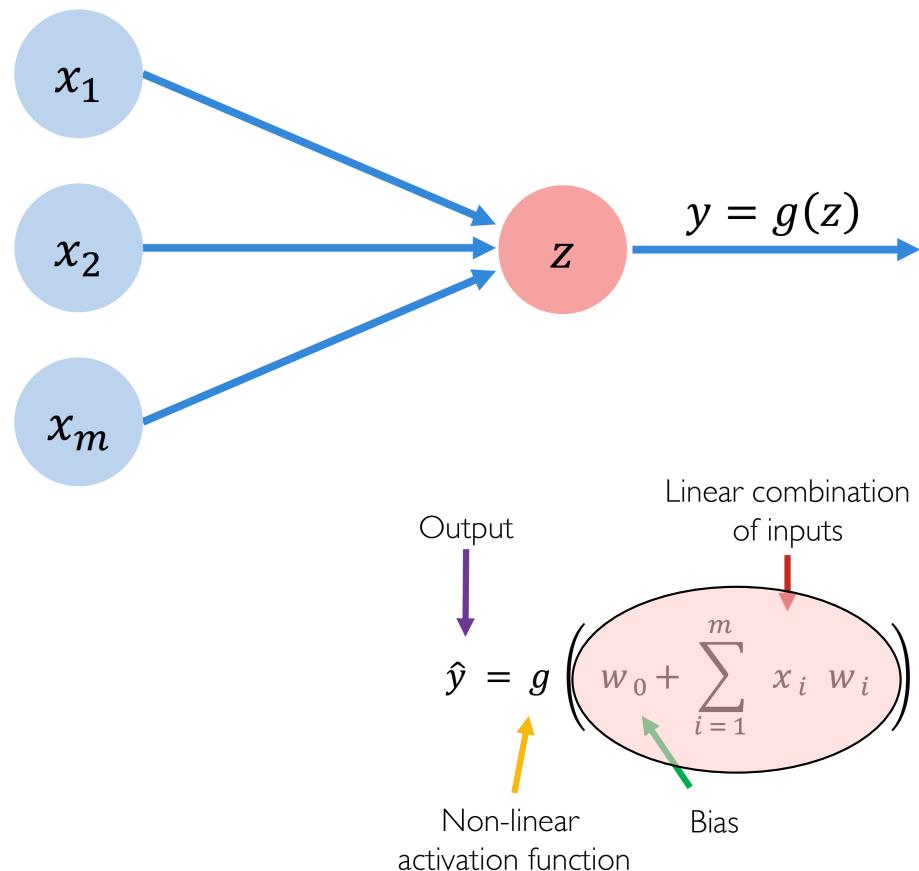
Scaled exponential linear unit (SELU) ^[21]	$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$	$f'(\alpha, x) = \lambda \begin{cases} \alpha(e^x) & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
S-shaped rectified linear activation unit (SReLU) ^[22]	$f_{t_l, a_l, t_r, a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ t_l, a_l, t_r, a_r are parameters.	$f'_{t_l, a_l, t_r, a_r}(x) = \begin{cases} a_l & \text{for } x \leq t_l \\ 1 & \text{for } t_l < x < t_r \\ a_r & \text{for } x \geq t_r \end{cases}$
Adaptive piecewise linear (APL) ^[23]	$f(x) = \max(0, x) + \sum_{i=1}^S a_i^* \max(0, -x + b_i^*)$	$f'(x) = H(x) - \sum_{i=1}^S a_i^* H(-x + b_i^*)$ ^[4]
SoftPlus ^[24]	$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$
Bent identity	$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2 + 1}} + 1$
GELU ^[25]	$f(x) = x\Phi(x) = x(1 + \text{erf}(x/\sqrt{2}))/2$	$f'(x) = \Phi(x) + x\phi(x)$
Sigmoid Linear Unit (SiLU) ^[26] (AKA SiL ^[26] and Swish- ^[17])	No	
SoftExponential ^[28]	$f(\alpha, x) = \begin{cases} \frac{\ln(1 - \alpha(x+\alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{\alpha^2 - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1 - \alpha(x+\alpha)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$
Soft Clipping ^[29]	$f(\alpha, x) = \frac{1}{\alpha} \log \frac{1 + e^{\alpha x}}{1 + e^{\alpha(x-1)}}$	$f'(\alpha, x) = \frac{1}{2} \sinh\left(\frac{p}{2}\right) \operatorname{sech}\left(\frac{px}{2}\right) \operatorname{sech}\left(\frac{p}{2}(1-x)\right)$
Sinoid ^[30]	$f(x) = \sin(x)$	$f'(x) = \cos(x)$
Sinc	$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$
Gaussian	$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$
SQ-RBF	$f(x) = \begin{cases} 1 - \frac{x^2}{2} & : x \leq 1.0 \\ 2 - \frac{2-x^2}{2} & : 1.0 \leq x < 2.0 \\ 0 & : x > 2.0 \end{cases}$	$f'(x) = \begin{cases} -x & : x \leq 1.0 \\ 2 - x & : 1.0 \leq x < 2.0 \\ 0 & : x > 2.0 \end{cases}$

Name	Plot	Equation	Derivative (with respect to x)
Identity			
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ ^[1]	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
SQNL ^[9]		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$	$f'(x) = 1 \mp \frac{x}{2}$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
ArSinH		$f(x) = \sinh^{-1}(x) = \ln(x + \sqrt{x^2 + 1})$	$f'(x) = \frac{1}{\sqrt{x^2 + 1}}$
ElliottSig ^{[10][11]} Softsign ^{[12][13]}		$f(x) = \frac{x}{1 + x }$	$f'(x) = \frac{1}{(1 + x)^2}$
Inverse square root unit (ISRU) ^[14]		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3$
Inverse square root linear unit (ISRLU) ^[14]		$f(x) = \begin{cases} \frac{x}{\sqrt{1 + x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \left(\frac{1}{\sqrt{1+x^2}}\right)^3 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Rectified linear unit (ReLU) ^[15]		$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$
Bipolar rectified linear unit (BReLU) ^[16]		$f(x_i) = \begin{cases} \text{ReLU}(x_i) & \text{if } i \bmod 2 = 0 \\ -\text{ReLU}(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$f'(x_i) = \begin{cases} \text{ReLU}'(x_i) & \text{if } i \bmod 2 = 0 \\ -\text{ReLU}'(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU) ^[17]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric rectified linear unit (PReLU) ^[18]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Randomized leaky rectified linear unit (RReLU) ^[19]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential linear unit (ELU) ^[20]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$

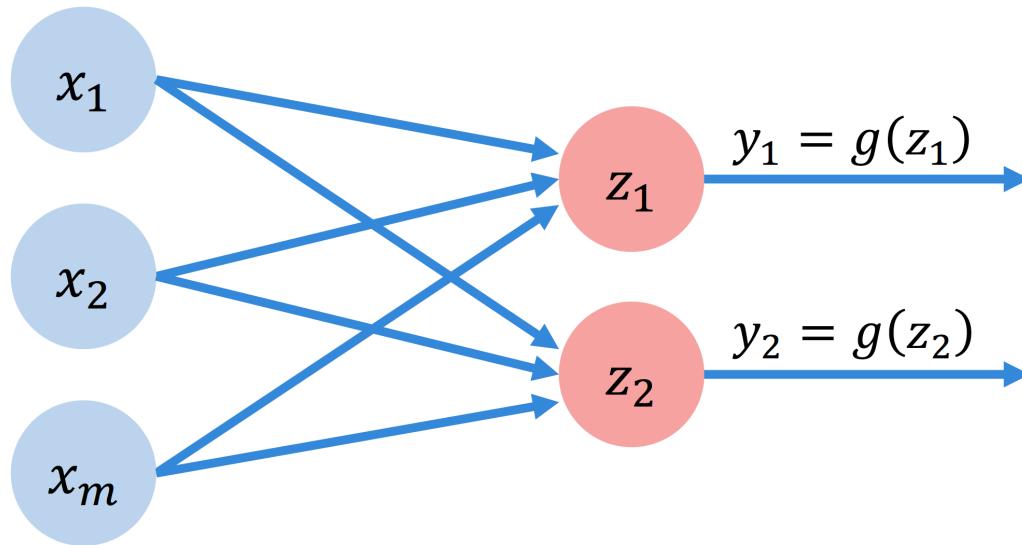
From Perceptron to Neural Networks



From Perceptron to Neural Networks

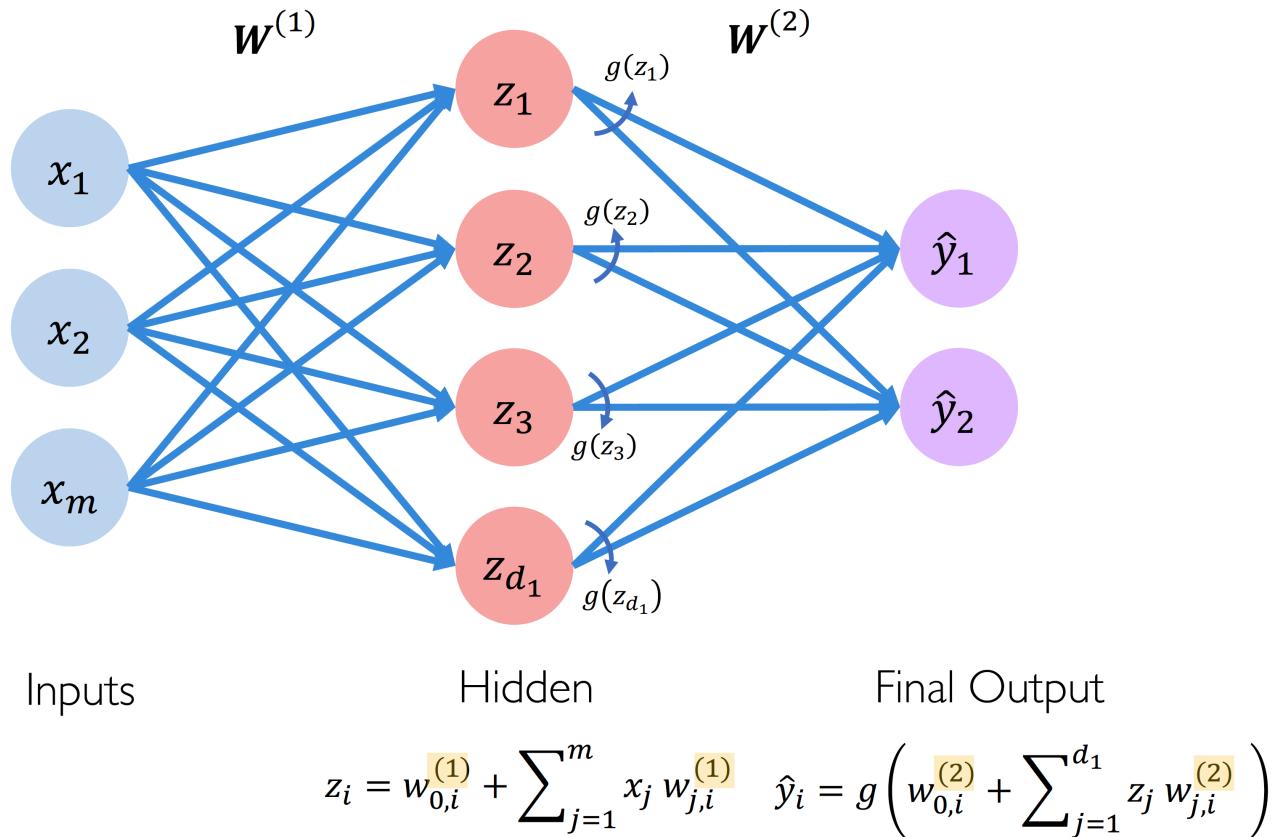


From Perceptron to Neural Networks

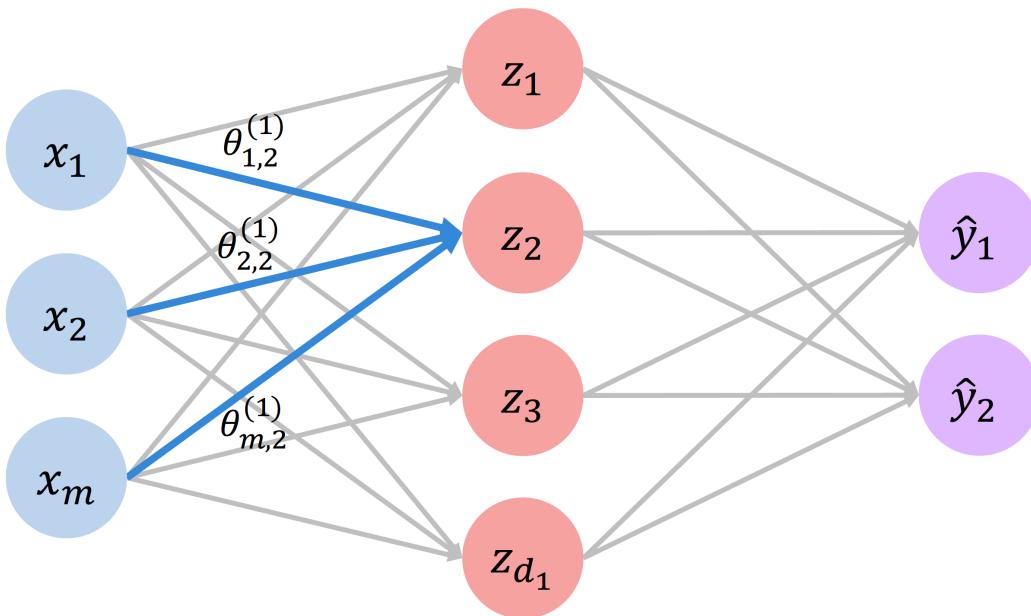


$$z_i = w_{0,i} + \sum_{j=1}^m x_j w_{j,i}$$

Neural Network

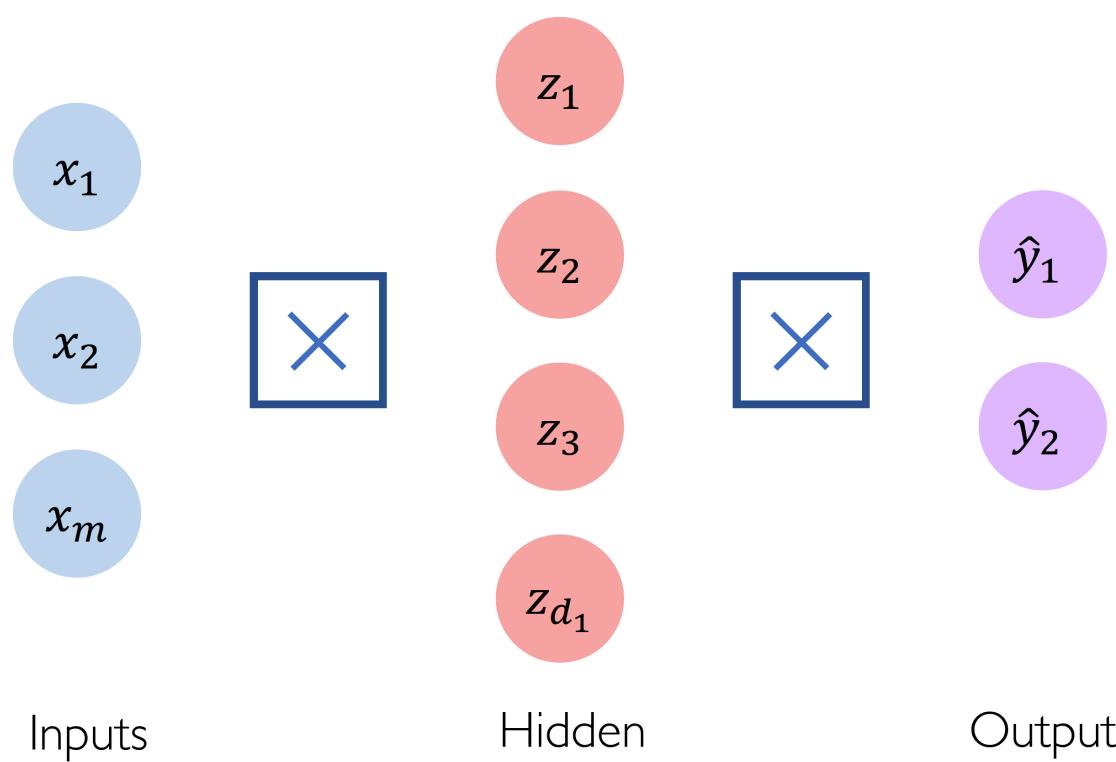


Neural Network

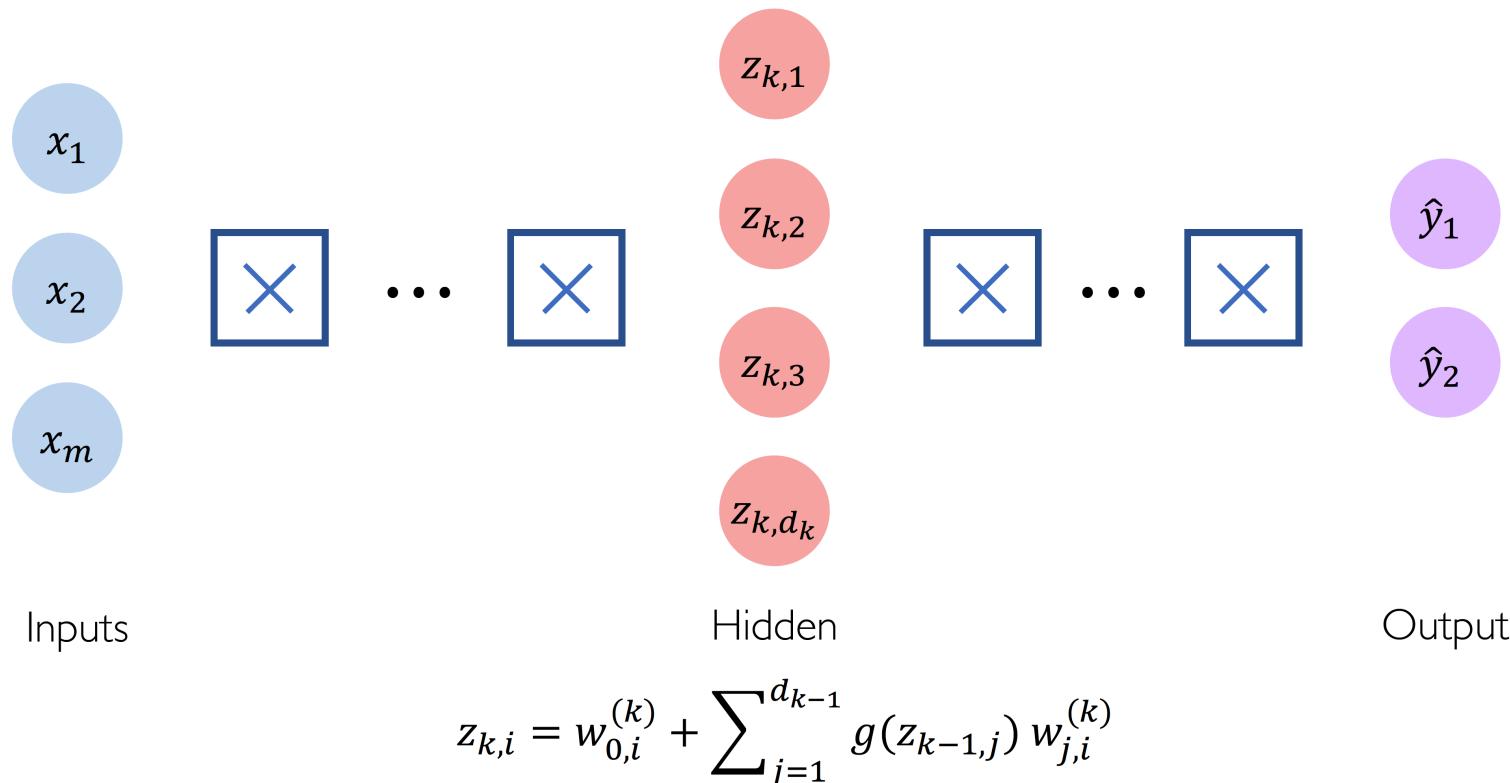


$$\begin{aligned} z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^m x_j w_{j,2}^{(1)} \\ &= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)} \end{aligned}$$

Multi-output Neural Network

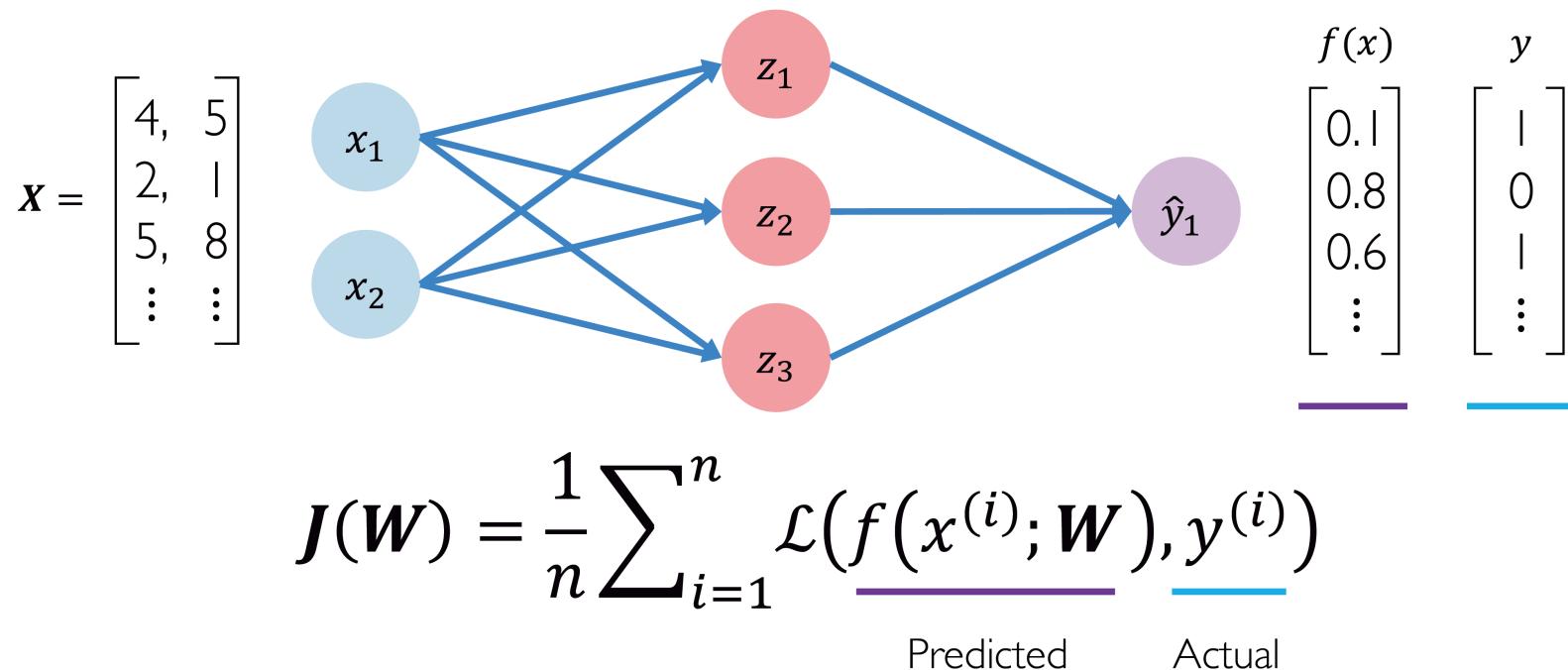


Multi-output Neural Network



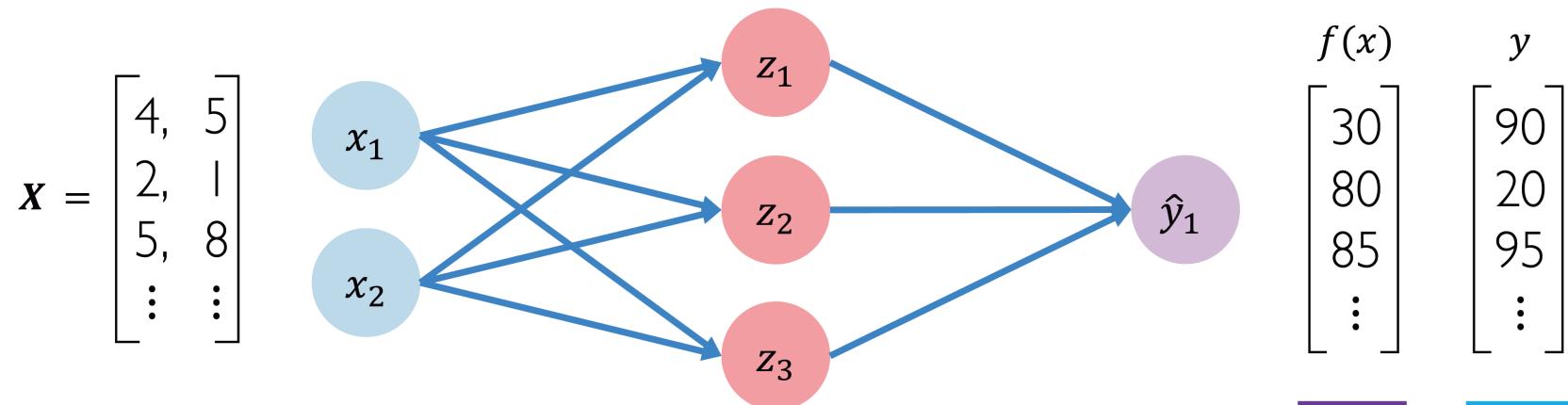
Learning the Parameters of the Network

The **empirical loss** measures the total loss over our entire dataset



Learning the Parameters of the Network

Mean squared error loss can be used with regression models that output continuous real numbers



$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \left(\underline{y^{(i)}} - \underline{f(x^{(i)}; \mathbf{W})} \right)^2$$

Actual Predicted

Learning the Parameters of the Network

We want to *find the network weights that achieve the lowest loss*

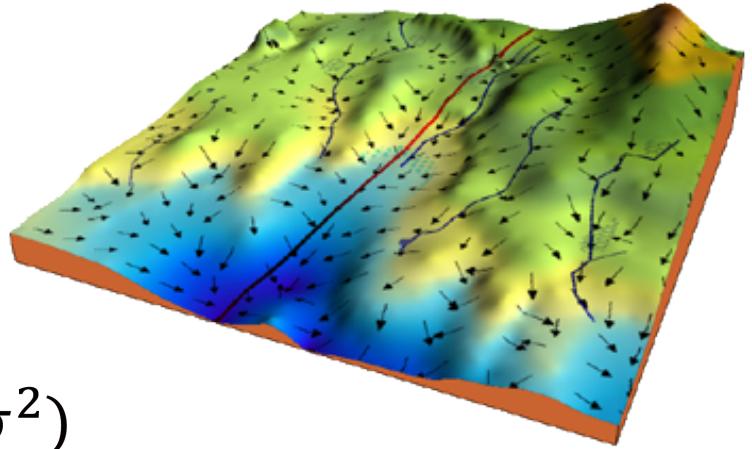
$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights



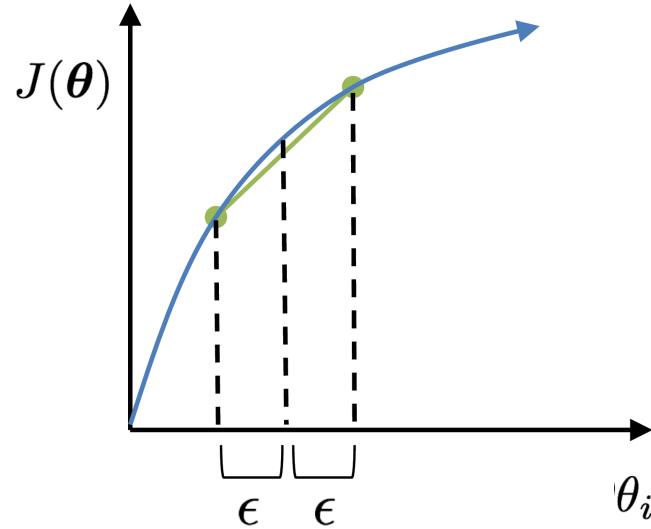
Computing the Gradient

- Numerical Approximation

The centered finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{(J(\boldsymbol{\theta} + \epsilon \cdot \mathbf{d}_i) - J(\boldsymbol{\theta} - \epsilon \cdot \mathbf{d}_i))}{2\epsilon}$$

where \mathbf{d}_i is a 1-hot vector consisting of all zeros except for the i th entry of \mathbf{d}_i , which has value 1.



Computing the Gradient

- Analytical

Given the cost function:

$$f(m, b) = \frac{1}{N} \sum_{i=1}^n (y_i - (mx_i + b))^2$$

The gradient can be calculated as:

$$f'(m, b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N} \sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with SGD:

(take small steps
opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Chain rule

If $h(x)$ is a composite function defined by $h(x) = g(f(x))$, and f and g are differentiable, then $h(x)$ is differentiable and h' is given by the product:

$$(h(x))' = (g(f(x)))' = \textcolor{red}{g'(f(x))} \cdot \textcolor{red}{f'(x)}$$

derive the outside,
keep the inside

derive the
inside

$$\frac{d}{dx}[f(g(x))] = f'(g(x))g'(x)$$

Chain rule

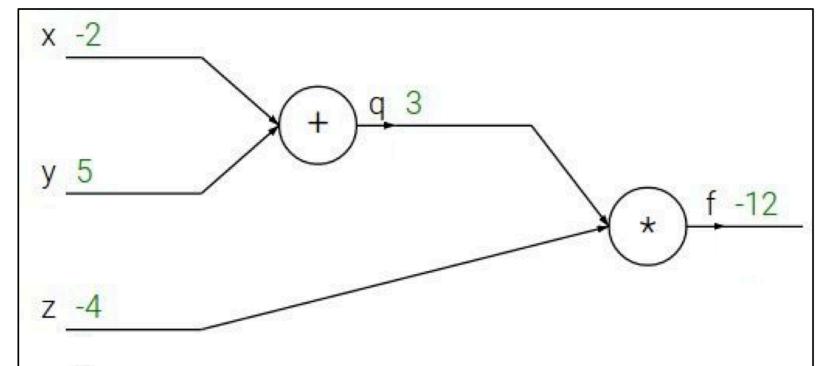
$$f(x, y, z) = (x + y)z$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

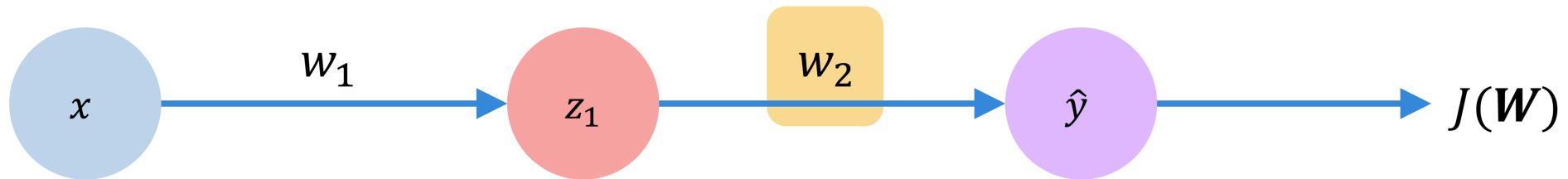


Computational graph

directed graph where the nodes correspond to **operations** or **variables**.
Variables can feed their value into operations, and operations can feed their output into other operations.

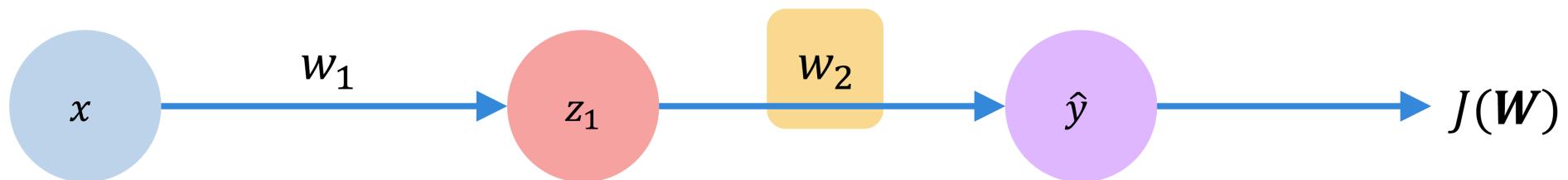
Backpropagation

How do we compute the gradient



How does a small change in one weight (ex. w_2) affect the final loss $J(\mathbf{W})$?

Backpropagation

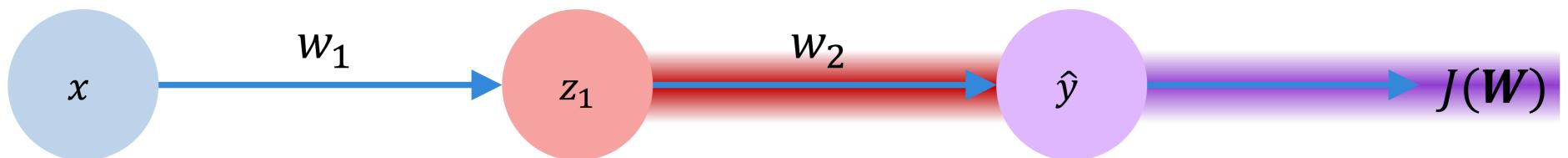


$$\frac{\partial J(\mathbf{W})}{\partial w_2} =$$

Let's use the chain rule!

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Backpropagation

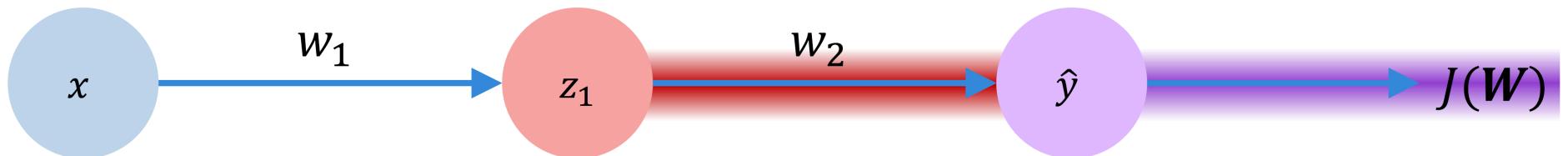


$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \underline{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}} * \underline{\frac{\partial \hat{y}}{\partial w_2}}$$

Yellow horizontal bar

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

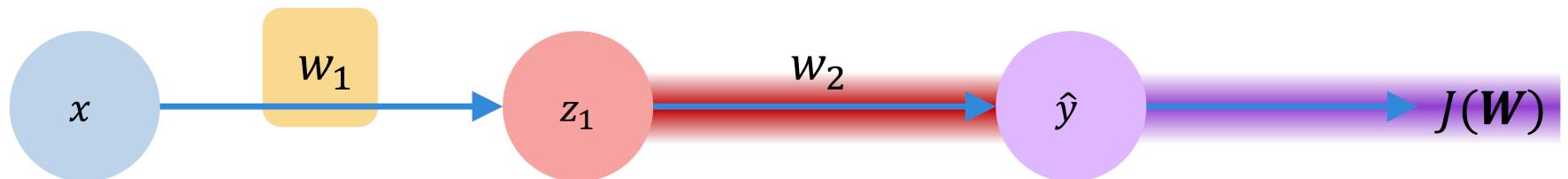
Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \underline{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}} * \underline{\frac{\partial \hat{y}}{\partial w_2}}$$


$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Backpropagation



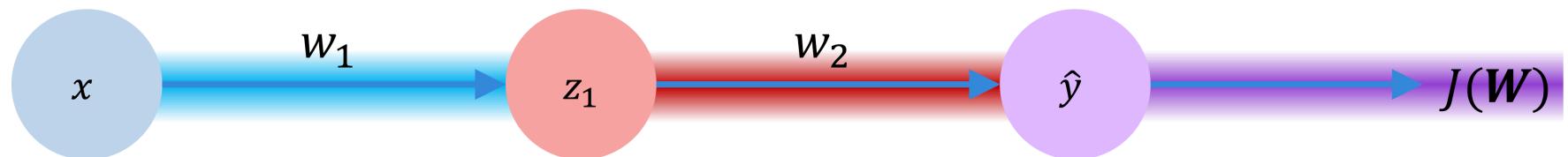
$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

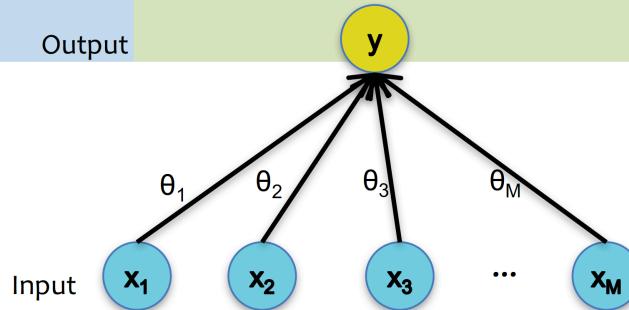
— — —

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

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

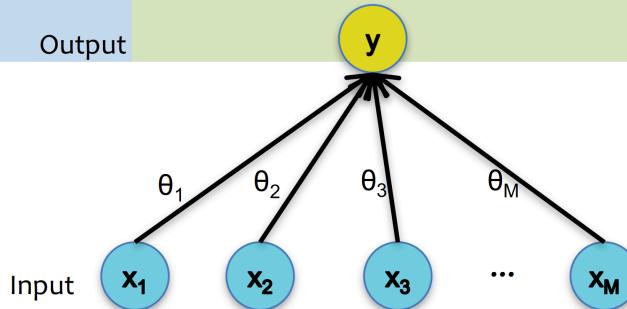
$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

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

$$a = \sum_{j=0}^D \theta_j x_j$$

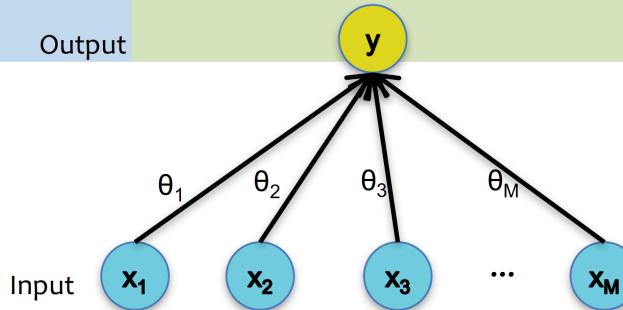
Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

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

$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

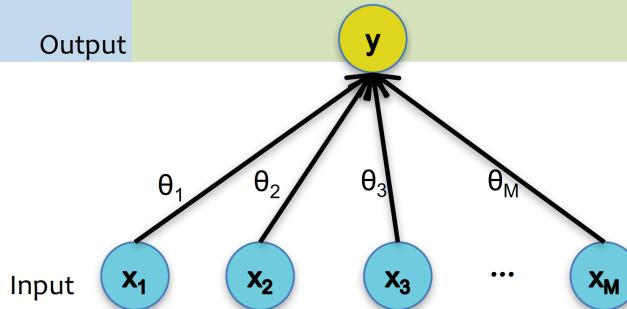
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

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

$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

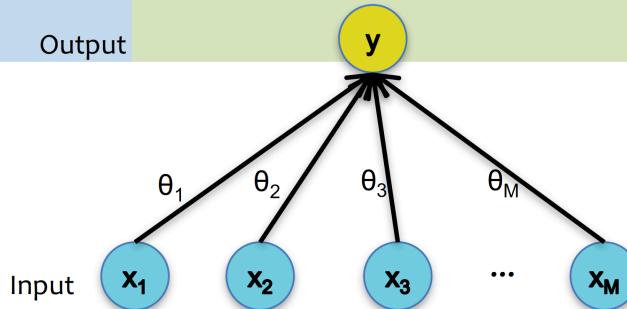
$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da} \frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

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

$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

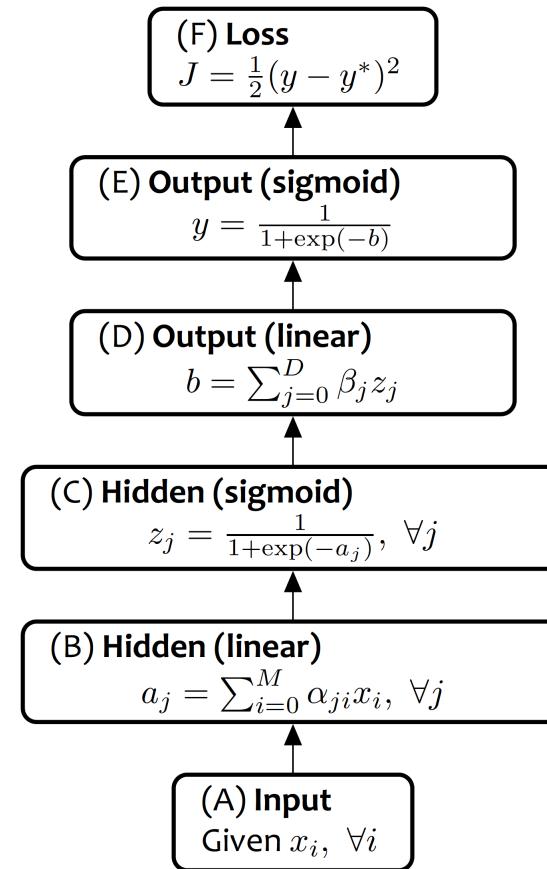
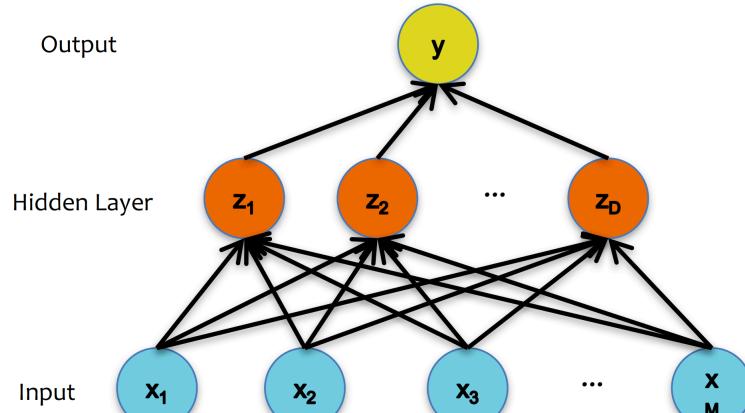
$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da} \frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da} \frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Training

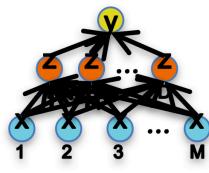
Backpropagation



Training

Backpropagation

Case 2:
**Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

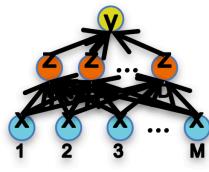
$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

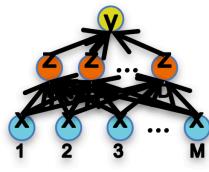
Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

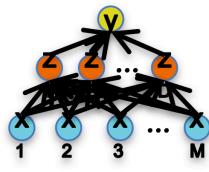
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

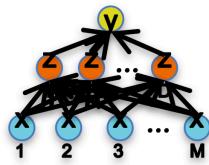
$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

Training

Backpropagation

Case 2: Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

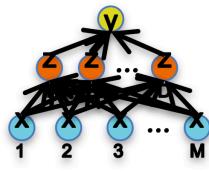
$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^D \alpha_{ji}$$

Forward Computation

1. Write an **algorithm** for evaluating the function $y = f(x)$. The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the “**computation graph**”)
2. Visit each node in **topological order**.
For variable u_i with inputs v_1, \dots, v_N
 - a. Compute $u_i = g_i(v_1, \dots, v_N)$
 - b. Store the result at the node

Backward Computation

Visit each node in **reverse topological order**.

For variable $u_i = g_i(v_1, \dots, v_N)$

- a. We already know dy/du_i
- b. Increment dy/dv_j by $(dy/du_i)(du_i/dv_j)$
(Choice of algorithm ensures computing (du_i/dv_j) is easy)