Natural Language Processing
CSE 325/425

Sihong Xie

Lecture 12:

- Neural networks revisit

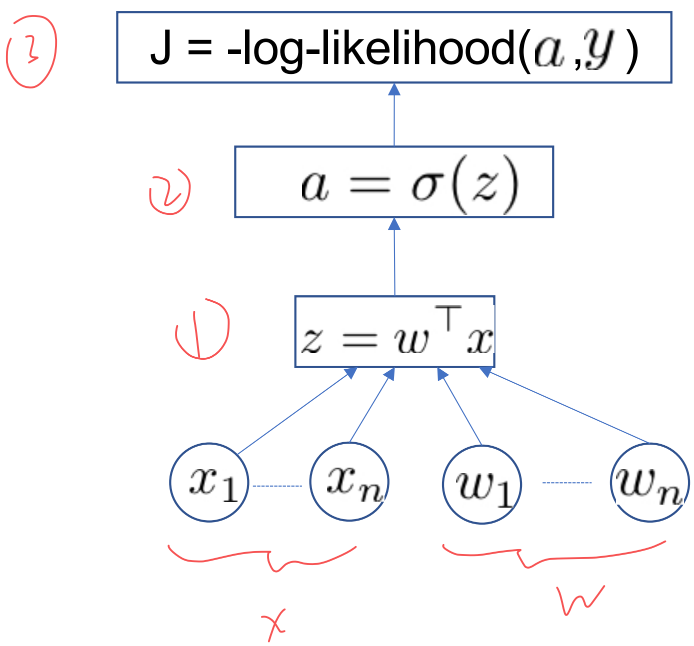
Revisiting neural networks

POS tagging goes neural!

- use neural networks to predict sequences:
  - word sequences: language models
  - POS tag sequences: POS-taggers
- upgrade HMM/MEMM to RNN
  - local normalization to distributions over all words/tags.
- upgrade CRF to CRF-LSTM
  - normalize to distribution over all sequences.

Logistic regression is a neural network

A computation graph is a differentiable system for ***evaluation*** and ***differentiation***.



Forward pass:

- Compute the value of the hidden, output units, and the loss.

① $z = w^T x = \sum_{j=1}^{n} w_i x_i \in \mathbb{R}$

② $a = \sigma(z) = \frac{1}{1+\exp(-z)} \in \mathbb{R}$

③ $J = -y \log a - (1-y) \log(1-a) \in \mathbb{R}$

Back-propagation:

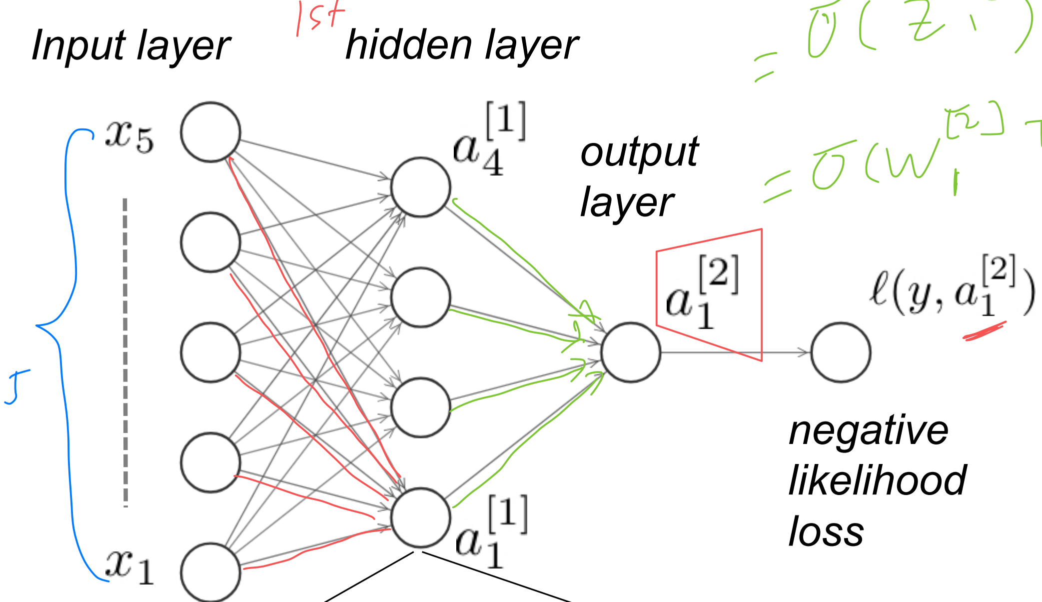
- compute the gradients using the chain rules.
- more later after we review matrix calculus.

Neural Network

Multiple Layer Perceptron (MLP)

A neural network is a computation graph that stacks many Logistic regression models.

$\vec{x} \in \mathbb{R}^5$



$= \sigma(z_1^{[2]})$

$= \sigma(W_1^{[2]T} \vec{a}^{[1]})$

$W_1^{[1]} \in \mathbb{R}^5$

One Logistic regression model

$z_1^{[1]} = W_1^{[1]^\top} x \qquad a_1^{[1]} = \sigma(z_1^{[1]})$ ← activation (σ is an activation function) tanh, ReLU

$W_1^{[1]}$ the vector of weights on edges connecting x to $z_1^{[1]}$

Neural network drawing tool: http://alexlenail.me/NN-SVG/index.html

$z_2^{[1]} = W_2^{[1]T} x$

$a_2^{[1]} = \sigma(z_2^{[1]})$

$$\underset{4\times5}{z^{[1]}} = W^{[1]} \underset{5}{x} \in \mathbb{R}^{4\times 1}$$

$$a^{[1]} = \sigma(z^{[1]}) = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \vdots \\ \sigma(z_4^{[1]}) \end{bmatrix} \in \mathbb{R}^4$$

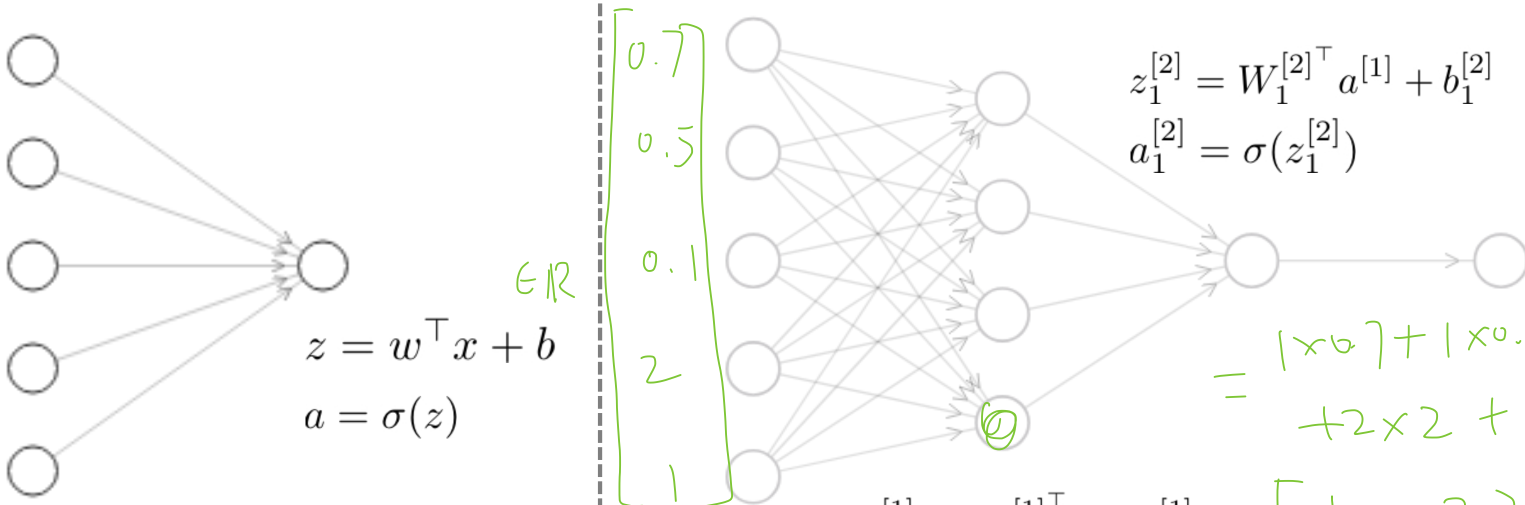
$$W^{[1]} = \begin{bmatrix} — W_1^{[1]T} — \\ — W_2^{[1]T} — \\ — W_3^{[1]T} — \\ — W_4^{[1]T} — \end{bmatrix} \in \mathbb{R}^{4\times 5}$$

{vectorization of NN}

Forward propagation

In general, for the j-th neural on the first layer:

$$z_j^{[1]} = W_j^{[1]^\top} x + b_j^{[1]}$$
$$a_j^{[1]} = \sigma(z_j^{[1]})$$



$$z = w^\top x + b$$
$$a = \sigma(z)$$

$$\vec{x} = \begin{bmatrix} 0.7 \\ 0.5 \\ 0.1 \\ 2 \\ 1 \end{bmatrix} \in \mathbb{R}$$

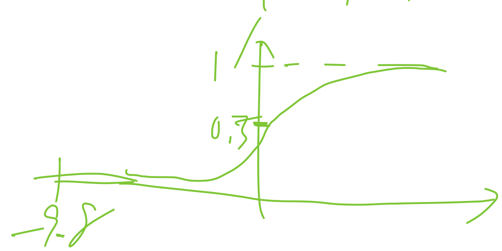
$$z_1^{[2]} = W_1^{[2]^\top} a^{[1]} + b_1^{[2]}$$
$$a_1^{[2]} = \sigma(z_1^{[2]})$$

$= 1\times0.7 + 1\times0.5 + 2\times(0.1) + 2\times2 + 4+1 + 0.4 = 9.8$

$$z_1^{[1]} = W_1^{[1]^\top} x + b_1^{[1]} = \begin{bmatrix} 1 & 1 & 2 & 2 & 4 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.5 \\ 0.1 \\ 2 \\ 1 \end{bmatrix} + 0.4$$
$$a_1^{[1]} = \sigma(z_1^{[1]}) = \frac{1}{1+\exp(-9.8)} \simeq 0$$

$$W_1^{[1]} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 4 \end{bmatrix}$$

$$b_1^{[1]} = 0.4$$



Matrix Calculus

- To train a neural network with parameters w (may include the biases), usually we minimize a scalar loss with respect to the parameters

$$\min_{w \in \mathbb{R}^d} L(w)$$

- Most optimization algorithms need the gradient of the loss with respect to the parameters w :

$$\frac{\partial L}{\partial w}$$

- Logistic regression optimization is a simple case:
  - can you recall what loss function we used?
  - and what's the gradient?

Matrix Calculus

set of all parameters (Not the data space)

- Basic definitions. Given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$
  - differential $f(x+h) = f(x) + d_x f(h) + o(h) \qquad d_x f : \mathbb{R}^n \to \mathbb{R}$
  - gradient: explicit form of $d_x f$

$\lim_{h \to 0} \frac{o(h)}{h} = 0 \qquad d_x f(h) = (\nabla_x f)^T h$

  - partial derivative

$$\nabla_x f : \mathbb{R}^n \to \mathbb{R}^n \qquad \nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

  - Examples:

$$f([x_1, x_2]) = 3x_1 + x_2^2$$

$f : \mathbb{R}^2 \to \mathbb{R}$

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 3 \\ 2x_2 \end{bmatrix}$$

$$\nabla_x f \Big|_{x = \begin{bmatrix} 0 \\ 1 \end{bmatrix}} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$\nabla_x f : \mathbb{R}^2 \to \mathbb{R}^2$

Matrix Calculus

- Generalization of gradient to higher dimensional output space $f : \mathbb{R}^n \to \mathbb{R}^m$

  - Jacobian

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} [\partial f_1 / \partial \vec{x}]^T \\ \vdots \\ [\partial f_m / \partial \vec{x}]^T \end{bmatrix} : \mathbb{R}^n \to \mathbb{R}^{m \times n}$$

  - Examples:

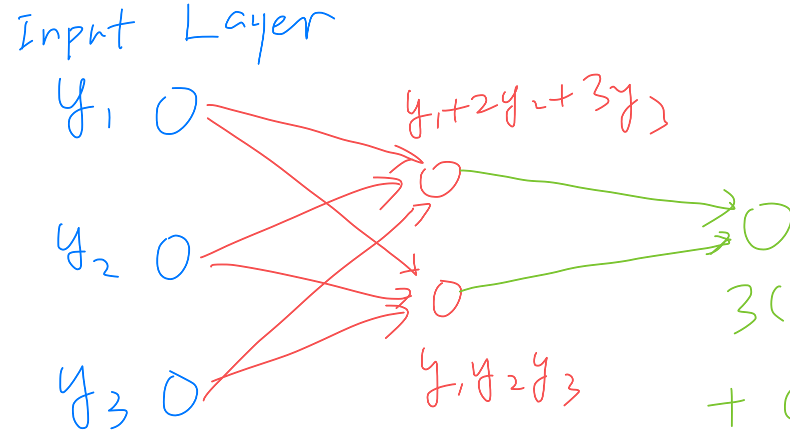
$n=3$

$$g([y_1, y_2, y_3]) = \begin{bmatrix} y_1 + 2y_2 + 3y_3 \\ y_1 y_2 y_3 \end{bmatrix} \quad m=2$$

$$g([y_1 \cdots y_{100}]) = \begin{bmatrix} g_1(y_1, \cdots, y_{100}) \\ g_2(y_1, \cdots, y_{100}) \\ g_3(y_1, \cdots, y_{100}) \end{bmatrix}$$

$$\frac{\partial g}{\partial \vec{y}} = \begin{bmatrix} 1 & 2 & 3 \\ y_2 y_3 & y_1 y_3 & y_1 y_2 \end{bmatrix} \in \mathbb{R}^{2 \times 3}$$

$$\frac{\partial f}{\partial y} \in \mathbb{R}^{3 \times 100}$$



$3(y_1+2y_2+3y_3)$
$+(y_1y_2y_3)^2$
$= f\circ g\,(y_1,y_2,y_3)$

Matrix Calculus

- Chain rule for derivatives:
  - Given two differentiable functions $g:\mathbb{R}^p \to \mathbb{R}^n$ $\quad f:\mathbb{R}^n \to \mathbb{R}^m$
  - Composition of two functions $f\circ g:\mathbb{R}^p \to \mathbb{R}^m$
  - Jacobian of $f\circ g$ is the product of the Jacobians

$$d_x(f\circ g) = d_{\underline{g(x)}}(f)\circ d_x(g) \qquad\qquad J_{f\circ g} = J_f \circ J_g$$

output from g at x

  - Examples:

$$g([y_1,y_2,y_3]) = \begin{bmatrix} y_1+2y_2+3y_3 \\ y_1y_2y_3 \end{bmatrix} \qquad \frac{\partial g}{\partial y} = \begin{bmatrix} 1 & 2 & 3 \\ y_2y_3 & y_1y_3 & y_1y_2 \end{bmatrix}$$

$$f([x_1,x_2]) = 3x_1 + x_2^2 \qquad \frac{\partial f}{\partial \vec{x}} = \begin{bmatrix} 3 \\ 2x_2 \end{bmatrix}$$

$$d_y(f\circ g)\big|_{y=[0,1,2]}$$

$$= d_{g(y)}(f)\big|_{g(y)=g(0,1,2)} \circ d_y(g)\big|_{y=[0,1,2]} = [3\ ,0]\times\begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 0 \end{bmatrix}$$

$x \in \mathbb{R}^d$

Matrix Calculus

$$\frac{\partial}{\partial w}(w_1x_1 + w_2x_2 + \cdots w_nx_n) = \begin{bmatrix} \frac{\partial}{\partial w_1}(w_1x_1 + \cdots + w_nx_n) \\ \vdots \\ \frac{\partial}{\partial w_n}(w_1x_1 + \cdots + w_nx_n) \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- Common examples
  - Vector inner product — Logistic regression $$\frac{\partial}{\partial w}(w^\top x) = x$$
  - Matrix-vector (w.r.t. vector) — MLP $$\frac{\partial}{\partial x}Wx = W$$ (use def of Jacobian)
  - Vector-matrix (w.r.t. vector) $$\frac{\partial}{\partial x}x^\top W = W^\top$$ $\leftarrow [\ ]$
  - Matrix-vector (w.r.t. matrix) $$\boxed{\frac{\partial}{\partial W}Wx}$$ of the shape of (#rows of W × shape of W)
    - Usually don't directly find this Jacobian: why?
    - Rather, embed this in chain-rule for some scalar function
    - Example in the next two slides.

NN param

$$J = f(z) \qquad z = Wx$$

$$\underbrace{\frac{\partial J}{\partial W}}_{\text{of the same shape of } W} = \boxed{\frac{\partial z}{\partial W}}\frac{\partial J}{\partial z}$$

① $z = w^T x + b$

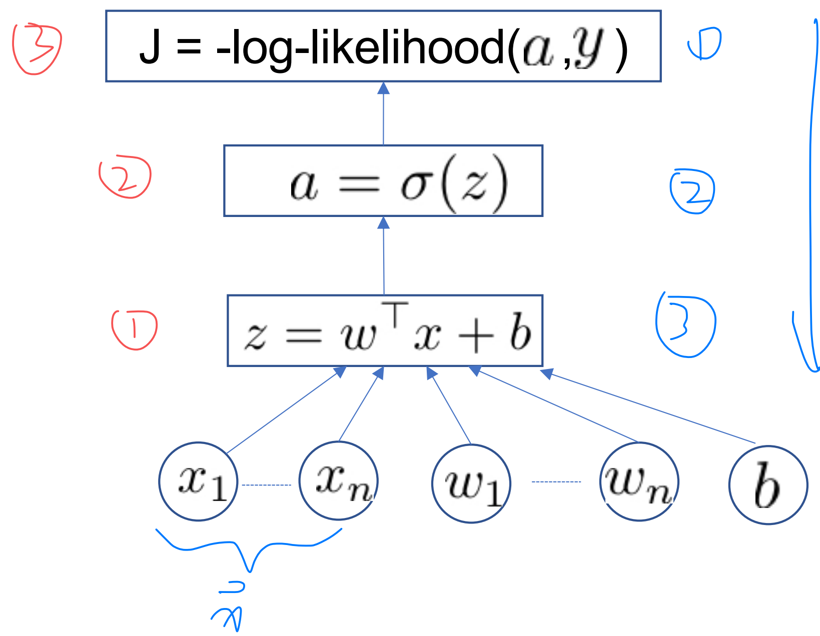
② $a = \sigma(z) = \frac{1}{1+\exp(-z)} = \Pr(y=1|x)$

③ $J = -y\log a - (1-y)\log(1-a)$

Prob. Interpretation:

forward prop.

Backpropagation for binary Logistic Regression



Going Backwards

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

② $\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z}$ (chain rule)

$= \left(-\frac{y}{a} + \frac{1-y}{1-a}\right)\times(a)\times(1-a)$

$= -y(1-a) + a(1-y)$

$= a - y$

③ $\boxed{\frac{\partial J}{\partial w}} = \frac{\partial J}{\partial z}\times\frac{\partial z}{\partial w}$

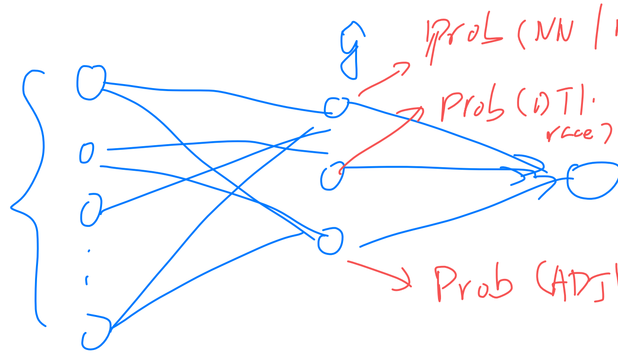
$= (a-y)\times\vec{x}$

gradient descent:

$w^{(t+1)} \leftarrow w^{(t)} - \eta\times\frac{\partial J}{\partial w} = w^{(t)} + \eta(y\vec{x} - a\vec{x})$

MEMM

$\mathbb{R}^d \ni \vec{x}$



$CE(y, \hat{y})$

y : one-hot vector

Backpropagation for multi-class Logistic Regression

- Example: predicting more than two classes.
  - Cross-entropy loss (negative log-likelihood loss)
  - Used in logistic regression and neural networks for classification.

$y \in \{NN, DT, ADJ\}$

$$z = Wx = \begin{bmatrix} \text{---} \; w_1^{[1]T} \; \text{---} \\ \text{---} \; w_2^{[1]T} \; \text{---} \\ \text{---} \; w_3^{[1]T} \; \text{---} \end{bmatrix} = \begin{bmatrix} - W_{NN} - \\ - W_{DT} - \\ - W_{ADJ} - \end{bmatrix}$$

"race" → $\vec{x}$ → $y = [1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix} = \hat{y} = \text{softmax}(z)$$

$$\hat{y}_i = \exp(z_i) / \sum_{j=1}^{3} \exp(z_j)$$

$$CE(y, \hat{y}) = -\sum_{i=1}^{3} y_i \log \hat{y}_i$$

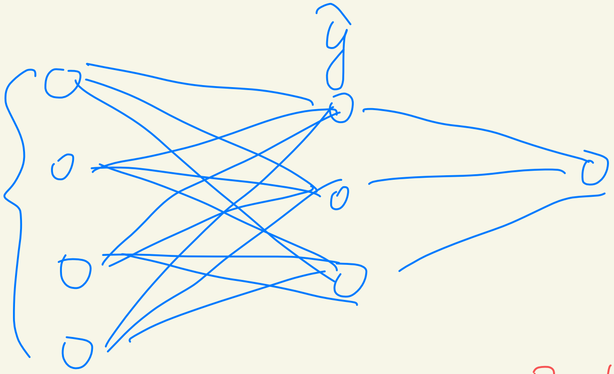
$\hat{y}_i \triangleq$ Prob of class i predicted by the model

$$= -1 \times \log \hat{y}_1 - 0 \times \log \hat{y}_2 - 0 \times \log \hat{y}_3 = -\log \hat{y}_1$$

$\triangleq$ negative log-likelihood of NN give "race"

$\vec{x}$

for "race"



$CE(\hat{y}, y) = J$

$\in \mathbb{R}$

① $z = Wx$

② $\hat{y} = \text{softmax}(z)$

③ $J = CE(\hat{y}, y)$

Backprop.

① $\dfrac{\partial J}{\partial \hat{y}}$

② $\dfrac{\partial J}{\partial z} = \dfrac{\partial J}{\partial \hat{y}} \dfrac{\partial \hat{y}}{\partial z}$

$= \hat{y} - y$

predicted prob vector

one hot vector

③ $\dfrac{\partial J}{\partial W} = \left(\dfrac{\partial J}{\partial z}\right)\left(\dfrac{\partial z}{\partial W}\right)^T$

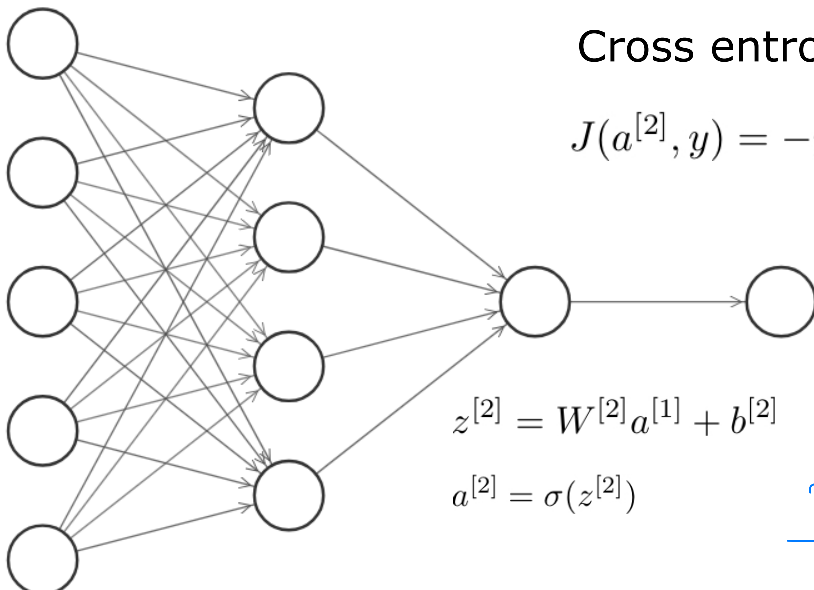
(swapped due to matrix shape compatibility.)

$= (\hat{y} - y) \quad x^T$

$$\begin{bmatrix} - w_1 - \\ - w_2 - \\ - w_3 - \end{bmatrix}$$

(outer product)

Backpropagation for Neural Network



Cross entropy loss for binary classication:

$$J(a^{[2]}, y) = -y \log a^{[2]} - (1-y)\log(1-a^{[2]})$$

Based on the gradient of Logistic Regression, $\frac{\partial J}{\partial z^{[2]}} = a^{[2]} - y$ (error term)

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

To train $W^{[2]}$ & $b^{[2]}$, we need

$$\frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial z^{[2]}} \boxed{\frac{\partial z^{[2]}}{\partial W^{[2]}}}$$ ← too large to evaluate

$$= (a^{[2]} - y)\,(a^{[1]})^T$$ ← this is correct if verified each element of the outer product.

Input vector x

$$z^{[1]} = W^{[1]} x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

Get the dimensions right!

$$\frac{\partial J}{\partial b^{[2]}} = \frac{\partial J}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}}$$

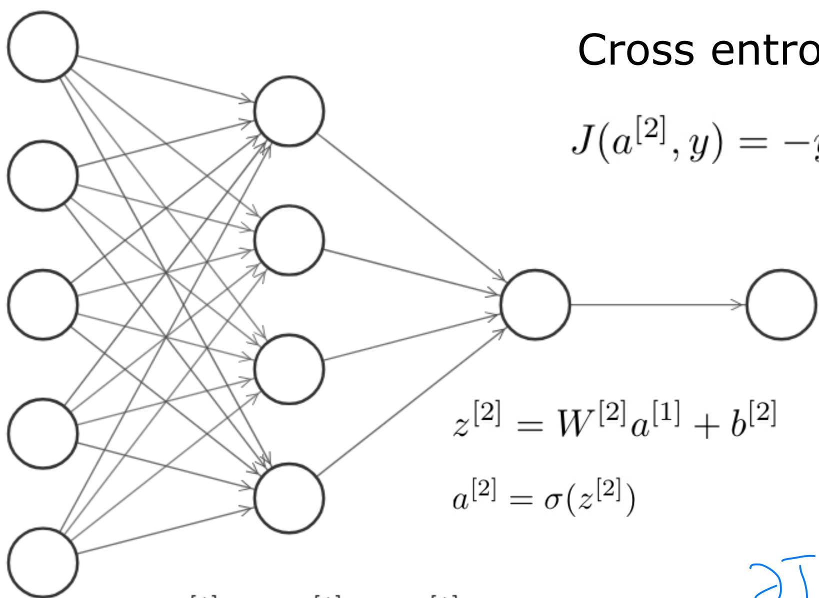
$$= (a^{[2]} - y)$$ ← $z^{[2]} = W^{[2]} a^{[1]} + (I) b^{[2]}$ (Identity matrix)

$$\frac{\partial J}{\partial w^{[1]}} = \frac{\partial J}{\partial z^{[1]}} x^T$$

($z^{[1]}$ has the same number of rows as $w^{[1]}$, & x^T has the same number of columns of $w^{[1]}$)

$$\frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial z^{[1]}}$$

Backpropagation for Neural Network



$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Input vector x

$$z^{[1]} = W^{[1]} x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

Cross entropy loss for binary classication:

$$J(a^{[2]}, y) = -y \log a^{[2]} - (1-y) \log(1 - a^{[2]})$$

Need $\frac{\partial J}{\partial z^{[1]}}$ since that will help finding $\frac{\partial J}{\partial W^{[1]}}$ & $\frac{\partial J}{\partial b^{[1]}}$ for training $W^{[1]}$ & $b^{[1]}$.

$$\frac{\partial J}{\partial a^{[1]}} = \left(\frac{\partial z^{[2]}}{\partial a^{[1]}}\right)^T \frac{\partial J}{\partial z^{[2]}} = \left(W^{[2]}\right)^T \frac{\partial J}{\partial z^{[2]}} = \left(W^{[2]}\right)^T \left(a^{[2]} - y\right)$$

$$\frac{\partial J}{\partial z^{[1]}} = \frac{\partial J}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}}$$

Jacobian of the vector function $a^{[1]}$ with input $z^{[1]}$

Continue at the top

Get the dimensions right!

$$a^{[1]} = \sigma(z^{[1]}) = [\sigma(z_1^{[1]}), \ldots, \sigma(z_4^{[1]})]$$

$$\frac{\partial a^{[1]}}{\partial z^{[1]}} = \mathrm{diag}\left(\left[\frac{\partial a_1^{[1]}}{\partial z_1^{[1]}}, \ldots, \frac{\partial a_4^{[1]}}{\partial z_4^{[1]}}\right]\right)$$

$$W^{[2]} = \begin{bmatrix} w^{[2]}_{11} & w^{[2]}_{12} & w^{[2]}_{13} & w^{[2]}_{14} \\ w^{[2]}_{21} & \cdots & w^{[2]}_{23} & \vdots \\ w^{[2]}_{31} & \cdots & w^{[2]}_{33} & \vdots \end{bmatrix}$$

Since $a^{[1]}_3 \in \mathbb{R}$ contributes to J through $z^{[2]}_1$, $z^{[2]}_2$, & $z^{[2]}_3$, with weights $w^{[2]}_{13}$, $w^{[2]}_{23}$ & $w^{[2]}_{33}$, so

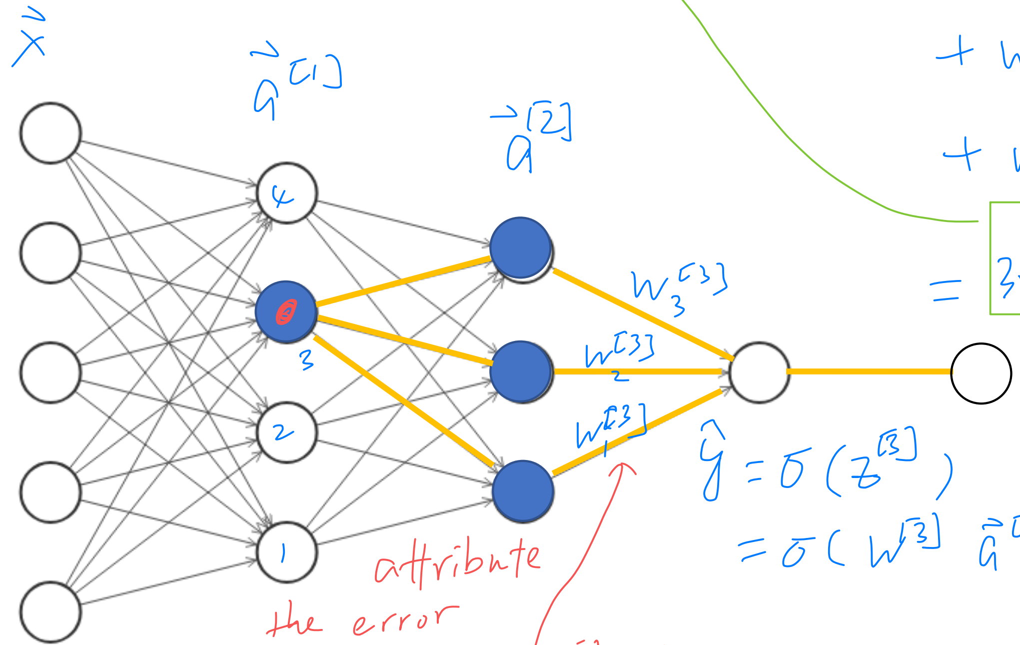
$$\frac{\partial J}{\partial a^{[1]}_3} = w^{[2]}_{13}\frac{\partial J}{\partial z^{[2]}_1} + w^{[2]}_{23}\frac{\partial J}{\partial z^{[2]}_2} + w^{[2]}_{33}\frac{\partial J}{\partial z^{[2]}_3}$$

$$= \boxed{\text{3rd column of } W^{[2]}} \times \frac{\partial J}{\partial z^{[2]}}$$

Backpropagation intuition

How much J will change if $a^{[1]}_3$ is slightly perturbed to $a^{[1]}_3 + h$, with $h \to 0$

$$\frac{\partial J}{\partial a^{[1]}_3} = \lim_{h \to 0} \frac{J - J(a^{[1]}_3 + h)}{h}$$



$J = CE(\hat{y}, y)$

$\hat{y} = \sigma(z^{[3]})$

$= \sigma(W^{[3]}\vec{a}^{[2]} + b^{[3]})$

attribute the error $\hat{y} - y$ to each $a^{[2]}$ element

Backprop: $\frac{\partial J}{\partial z^{[3]}} = \hat{y} - y$, $\frac{\partial J}{\partial a^{[2]}} = (W^{[3]})^T \frac{\partial J}{\partial z^{[3]}}$

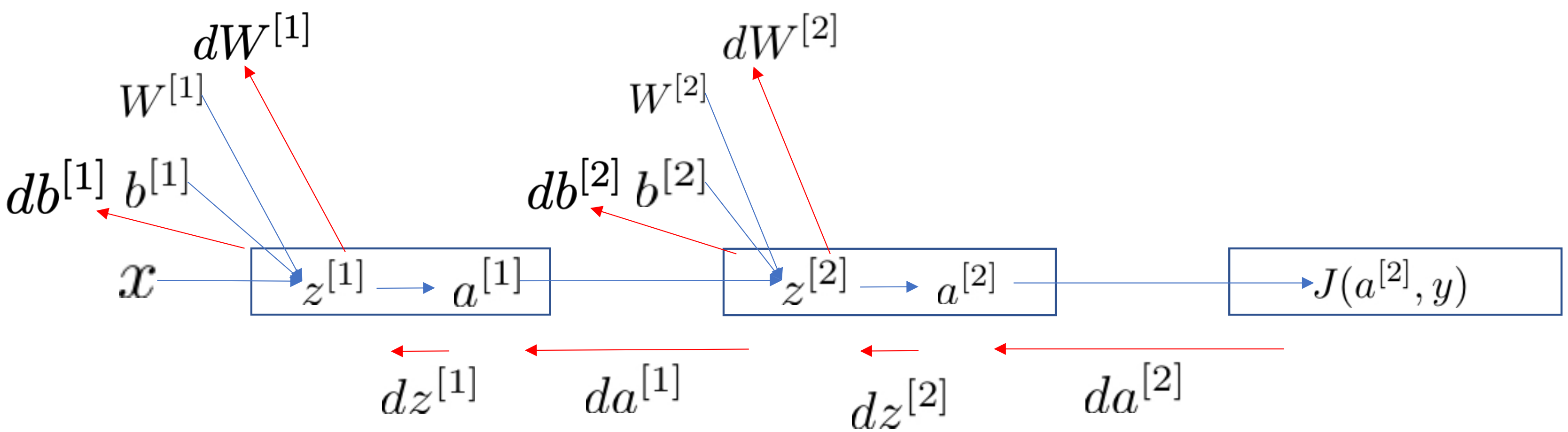
$$= (W^{[3]})^T(\hat{y} - y)$$

$$= \begin{bmatrix} w^{[3]}_1 \times (\hat{y} - y) \\ \vdots \\ w^{[3]}_3 \times (\hat{y} - y) \end{bmatrix}$$

$$\frac{\partial J}{\partial z^{[2]}} = (W^{[3]})^T(\hat{y} - y) \times \mathrm{diag}\left(\frac{\partial a^{[2]}}{\partial z^{[2]}}\right)$$

Backpropagation for Neural Network

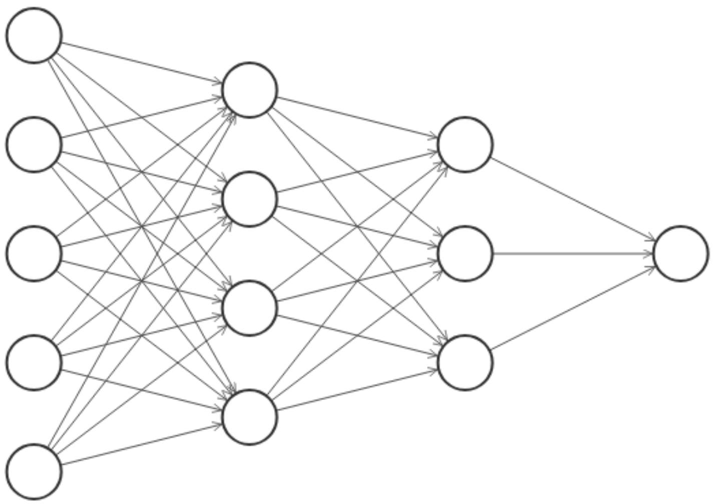
The big picture



Get the dimensions right!

Gradient descent for network training

1. Initialize network weights randomly.
2. Compute gradients through backpropagation.
3. Update weights (learning rate?).
4. Repeat 2 – 3 until convergence.



$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\partial J}{\partial w}\Big|_{w=w^{(t)}}$$

$$w^{(t+2)} \leftarrow w^{(t+1)} - \eta \frac{\partial J}{\partial w}\Big|_{w=w^{(t+1)}}$$

$$\vdots$$

$$w^{(*)} \leftarrow w^{(*)} - \eta \boxed{\frac{\partial J}{\partial w}\Big|_{w=w^{*}}}$$

$$= w^{(*)}$$

$$\approx 0$$