

# CS559 Machine Learning Neural Networks

Tian Han

Department of Computer Science  
Stevens Institute of Technology

Week 12

# Outline

- Basics of neural network
- Implementation of Simple NN

# Basics of Neural Networks

## Some Reviews

- Last few lectures, we focused on **unsupervised learning** problems (e.g., density estimation, clustering etc), and we discussed the general algorithm and model for that (e.g., **EM algorithm** for **latent variable model**). We also show that the model can be represented and do inference using *Bayesian Network* which encodes conditional relations between different random variables.

## Some Reviews

- Last few lectures, we focused on **unsupervised learning** problems (e.g., density estimation, clustering etc), and we discussed the general algorithm and model for that (e.g., **EM algorithm** for **latent variable model**). We also show that the model can be represented and do inference using *Bayesian Network* which encodes conditional relations between different random variables.
- Back to the **supervised learning** scenarios, but consider *non-linear* models on that.

## Some Reviews

- Last few lectures, we focused on **unsupervised learning** problems (e.g., density estimation, clustering etc), and we discussed the general algorithm and model for that (e.g., **EM algorithm** for **latent variable model**). We also show that the model can be represented and do inference using *Bayesian Network* which encodes conditional relations between different random variables.
- Back to the **supervised learning** scenarios, but consider *non-linear* models on that.
- One way is to use *Neural Network* which presents *non-linear* transformation/mapping of the input.

## Summary of supervised learning

$$\underbrace{\mathbf{w}^T \mathbf{x} + b}_{\text{score}}$$

	Classification	Regression
Predictor	sign	score
Relate to y	margin (score, y)	residual (score, y)
Loss	zero-one, logistic	squared, absolute deviation
Algorithm	GD	GD

## Review: Optimization problem

## Minimize Training Loss

$$\min_{\mathbf{w} \in \mathbb{R}^d} \text{TrainLoss}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N L(\mathbf{x}_n, y_n, \mathbf{w})$$

## Review: Optimization problem

### Gradient Descent (GD)

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

$\nabla_{\mathbf{w}} \text{TrainLoss}$  denotes the gradient of the (average) total training loss with respect to  $\mathbf{w}$

## Review: Optimization problem

### Gradient Descent (GD)

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

$\nabla_{\mathbf{w}} \text{TrainLoss}$  denotes the gradient of the (average) total training loss with respect to  $\mathbf{w}$

### Stochastic Gradient Descent (SGD)

For each  $(x, y) \in D_{train}$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(x, y, \mathbf{w})$$

$\nabla_{\mathbf{w}} L$  denotes the gradient of one example loss with respect to  $w$

## Notation clarification

- $\mathbf{w}, \mathbf{x}$ : bold letters are usually vectors in dimension  $d$
- $w_i$ : the  $i$ -th element in vector  $\mathbf{w}$
- $\mathbf{x}_n$ : the  $n$ -th example in the training data set, its corresponding target value:  $y_n$
- $x_i$ : feature  $i$  in current example  $\mathbf{x}$
- $x_{ni}$ : feature  $i$  in the  $n$ -th example
- $V, W$ : capital letters usually denote matrices.

# What is Neural Network?

- Often associated with biological devices (brains), electronic devices, or network diagrams;
- But the best conceptualization for this presentation is none of these: think of a neural network as a mathematical function.

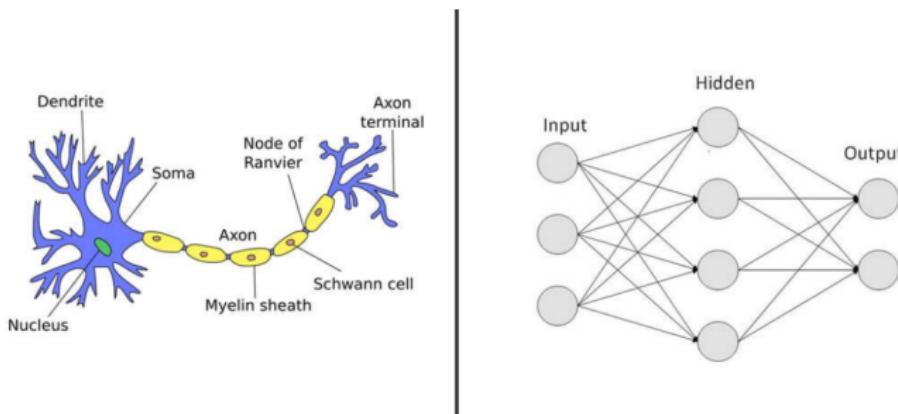


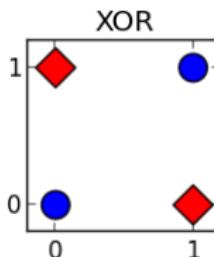
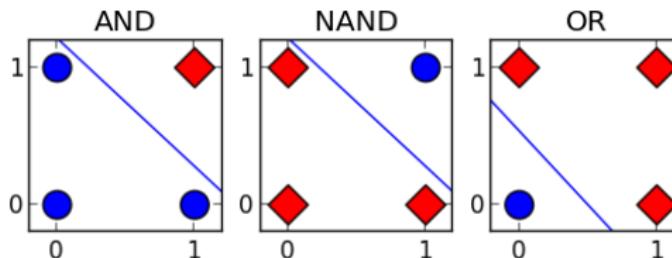
Figure: [Jacob Joseph]

## The pros of Neural Network

- Successfully used on **a variety of domains**: computer vision, speech recognition, gaming etc.
- Can provide solutions to very **complex and nonlinear** problems;
- If provided with **sufficient amount of data**, can solve classification and detection problems accurately and easily
- Once trained, **prediction is fast**;

# Motivation

- Perceptron: limitations;
- Feedforward networks and Backpropagation;

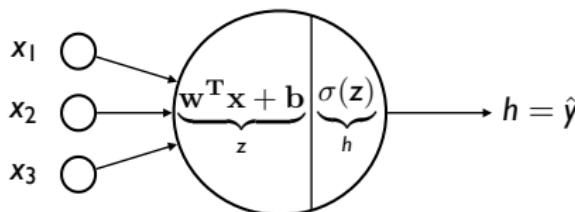


# Motivation

- Allow to learn **non linearly** separable transformations from input to output;
- A single **hidden layer** allows to compute any input/output transformation;

# No Hidden Units: Logistic Regression

- Sigmoid activation function:



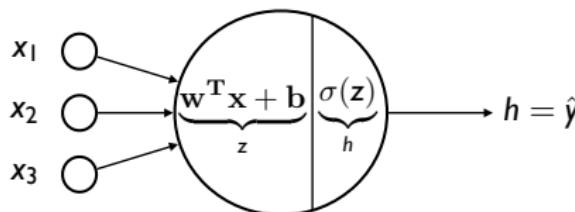
- Output score: sigmoid function  $\hat{y} = \sigma(w^T x + b)$
- Recall sigmoid function:

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

Derivative:

# No Hidden Units: Logistic Regression

- Sigmoid activation function:



- Output score: sigmoid function  $\hat{y} = \sigma(\mathbf{w}^T \mathbf{x} + b)$
- Recall sigmoid function:

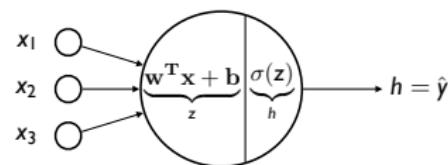
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Derivative:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

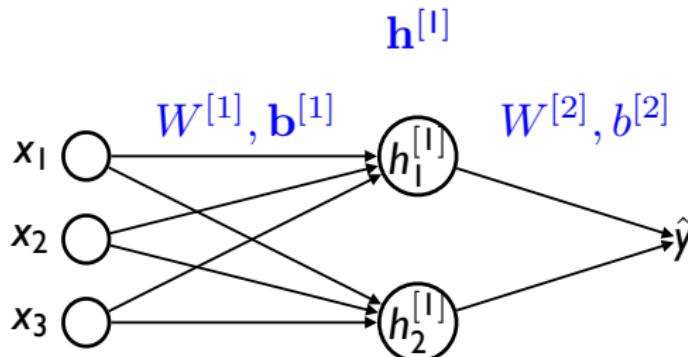
# Learning Algorithm: No hidden units

- Logistic loss:  $J(\mathbf{x}, y, \mathbf{w}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$
- $\hat{y} = \frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}}} = \sigma(\mathbf{w}^T \mathbf{x})$
- $\frac{\partial \hat{y}}{\partial w_i} = \hat{y}(1 - \hat{y})x_i$
- $\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i} = (\hat{y} - y)x_i$
- Applying gradient descent ( $\eta$  is the learning rate):
  - Element (feature) operation:  
 $w_i^{t+1} = w_i^t - \eta \frac{\partial J}{\partial w_i}$
  - Vector operation:  
 $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial J}{\partial \mathbf{w}}$



## Neural Networks: one hidden layer

- One hidden layer:



- Hidden layer representation

$$z_1^{[1]} = \mathbf{w}_1^{[1]T} \mathbf{x} + b_1^{[1]}, h_1^{[1]} = \sigma(z_1^{[1]})$$

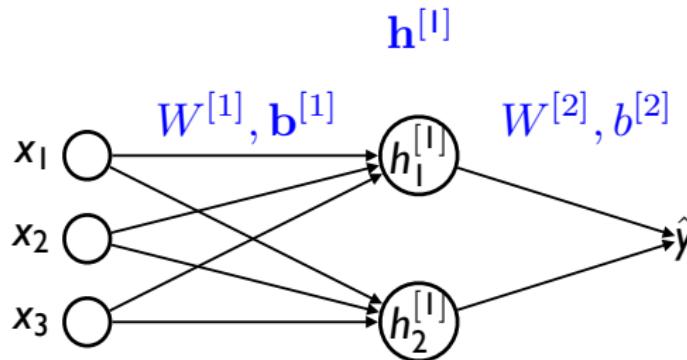
$$z_2^{[1]} = \mathbf{w}_2^{[1]T} \mathbf{x} + b_2^{[1]}, h_2^{[1]} = \sigma(z_2^{[1]})$$

\$\Rightarrow\$

$$\mathbf{z}^{[1]} = \underbrace{W^{[1]}_{2 \times 3}}_{2 \times 3} \underbrace{\mathbf{x}_{3 \times 1}}_{3 \times 1} + \underbrace{\mathbf{b}^{[1]}_{2 \times 1}}_{2 \times 1}$$

## Neural Networks: one hidden layer

- One hidden layer:



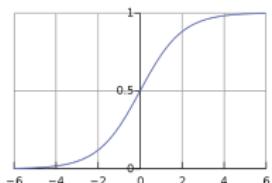
- Output layer

$$z^{[2]} = \underbrace{W^{[2]}}_{1 \times 2} \underbrace{\mathbf{h}^{[1]}}_{2 \times 1} + b^{[2]}$$

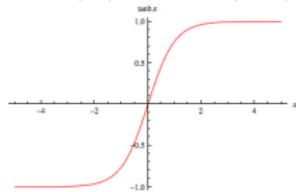
$$\hat{y} = \sigma(z^{[2]})$$

# Activation Functions

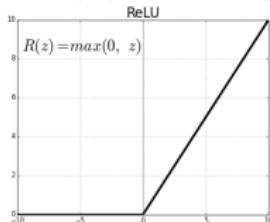
**Sigmoid:**  $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$



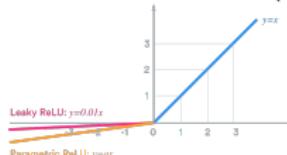
**tanh:**  $f(x) = 2\sigma(2x) - 1$



**ReLU:**  $f(x) = \max(0, x)$



**Leaky ReLU:**  $f(x) = \max(\alpha x, x)$

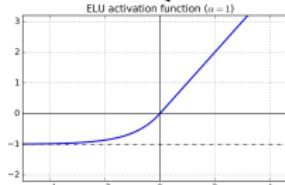


**Maxout**

$$\max(\mathbf{w}_1^T \mathbf{x} + b_1, \mathbf{w}_2^T \mathbf{x} + b_2)$$

**ELU:**

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



## Neural Networks: one hidden layer

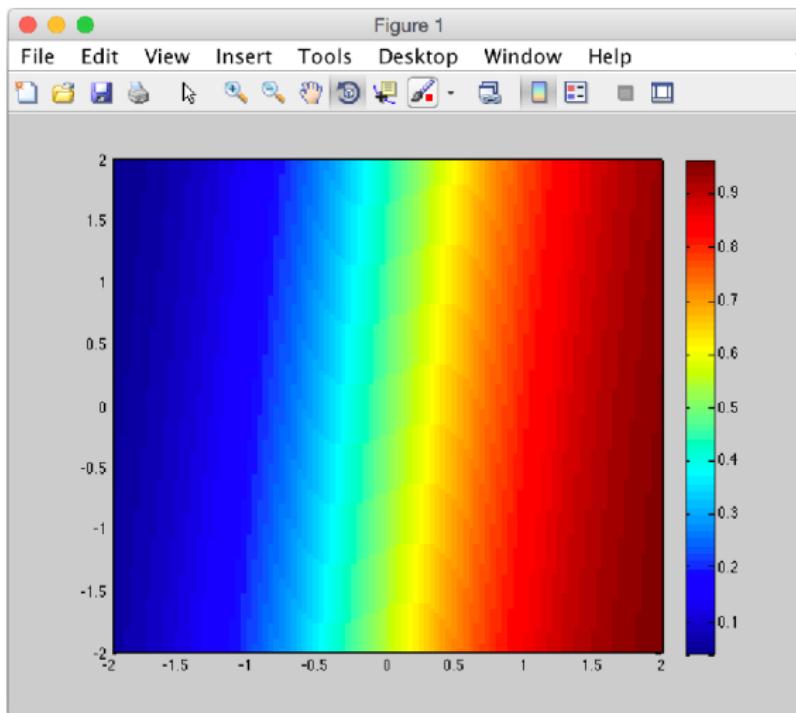
- Think of intermediate hidden units as learned features of a linear predictor.
- Feature learning: manually specified features:

$\mathbf{x}$

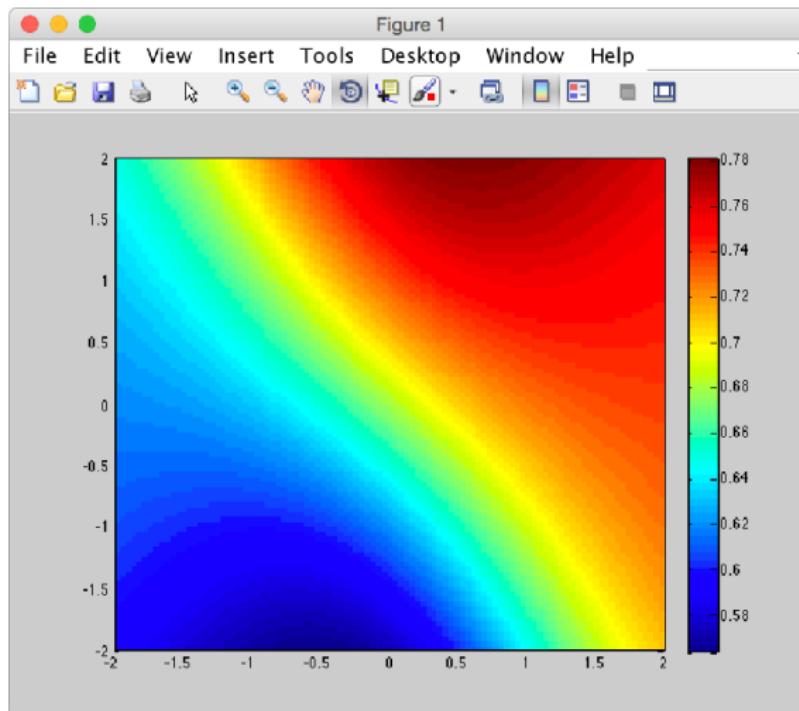
automatically learned features:

$$h(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_k(\mathbf{x})]$$

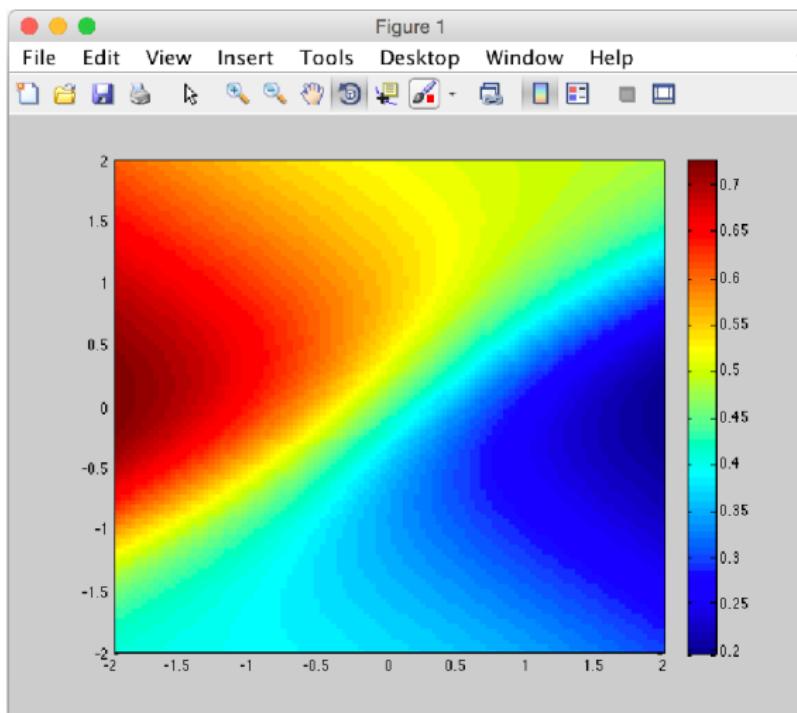
# Decision Boundary: Logistic Regression



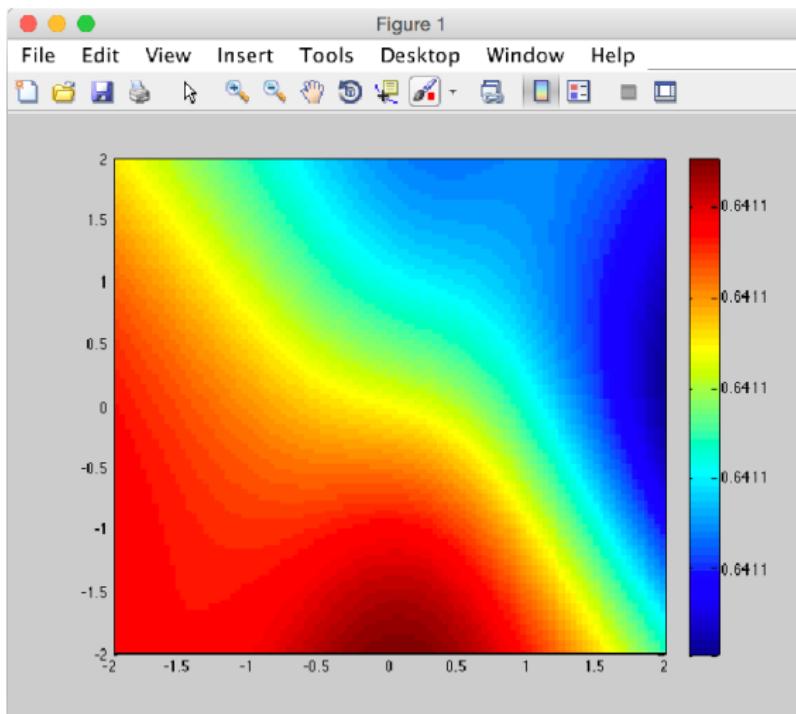
# Decision Boundary: 2 hidden layers, 2 hidden units



# Decision Boundary: 2 hidden layers, 3 hidden units



# Decision Boundary: 10 hidden layers, 5 hidden units



## Loss Minimization

- Optimization problem

$$\text{TrainLoss}(\mathbf{W}, \mathbf{b}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{train}} \text{Loss}(\mathbf{x}, y, \mathbf{W}, \mathbf{b})$$

$$J = \text{Loss}(\mathbf{x}, y, \mathbf{W}, \mathbf{b}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

$$\hat{y} = h^{[2]}$$

- Goal: compute gradient

$$\nabla_{\mathbf{W}, \mathbf{b}} J(\mathbf{W}, \mathbf{b})$$

## Stochastic Gradient Descent

- Goal: compute gradient

$$\nabla_{\mathbf{W}, \mathbf{b}} J(\mathbf{W}, \mathbf{b})$$

- Repeat, given a sample  $(\mathbf{x}^{(i)}, y^{(i)})$ :

$$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}, \quad W^{[1]} = W^{[1]} - \eta dW^{[1]}$$

$$d\mathbf{b}^{[1]} = \frac{\partial J}{\partial \mathbf{b}^{[1]}}, \quad \mathbf{b}^{[1]} = \mathbf{b}^{[1]} - \eta d\mathbf{b}^{[1]}$$

$$dW^{[2]} = \frac{\partial J}{\partial W^{[2]}}, \quad W^{[2]} = W^{[2]} - \eta dW^{[2]}$$

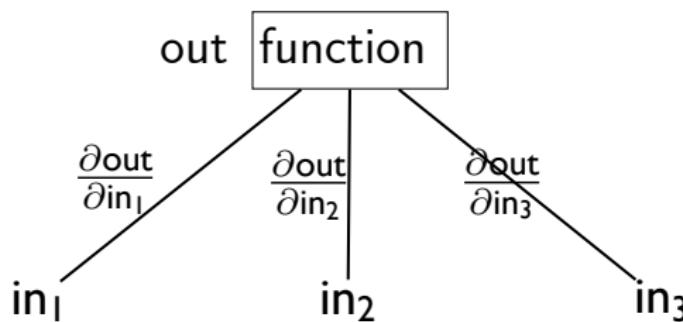
$$db^{[2]} = \frac{\partial J}{\partial b^{[2]}}, \quad b^{[2]} = b^{[2]} - \eta db^{[2]}$$

# Approach

- Mathematical: just grind through the **chain rule**
- Next: visualize the computation using a computation graph
- Advantages:
  - Avoid long equations
  - Reveal structure of computations (modularity, efficiency, dependencies)

## Functions as boxes

- Partial derivatives (gradients): how much does the output change if an input changes?



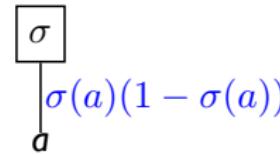
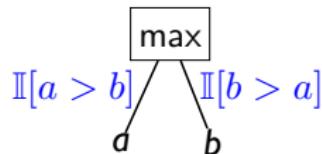
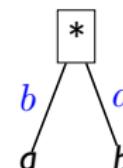
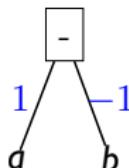
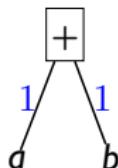
- Example:

$$\text{out} = 2\text{in}_1 + \text{in}_2\text{in}_3$$

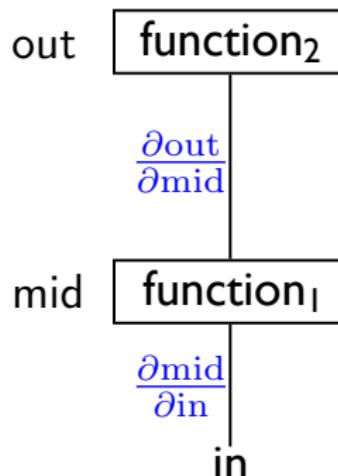
$$2\text{in}_1 + (\text{in}_2 + \epsilon)\text{in}_3 = \text{out} + \epsilon\text{in}_3$$

- Partial derivatives (gradients): a measure of sensitivity

# Basic building blocks



# Composing Functions



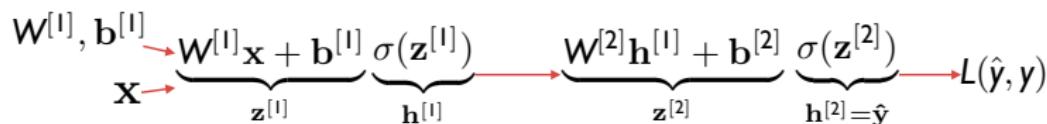
## Chain rule

$$\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$$

## Back propagation

- Goal: learn the weights so that the loss (error) is minimized.
- It provides an efficient procedure to compute derivatives.
- For a fixed sample  $(\mathbf{x}, y)$ , we want to learn  $\hat{y} = f(\mathbf{x})$
- Negative Log-likelihood over input  $\mathbf{x}_n$  is (assume  $y_n \in \{0, 1\}$ ):  
$$L_n = -y_n \log(\hat{y}_n) - (1 - y_n) \log(1 - \hat{y}_n)$$
- Total error of training examples is  $E = \sum_n L_n$

# Back propagation



Loss function :  $L = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$

$$d\mathbf{z}^{[1]} = \frac{\partial L}{\partial \mathbf{h}^{[1]}} \frac{\partial \mathbf{h}^{[1]}}{\partial \mathbf{z}^{[1]}} = W^{[2]T} dz^{[2]} \circ \mathbf{h}^{[1]} \circ (1 - \mathbf{h}^{[1]}) \quad dz^{[2]} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{[2]}} = \hat{y} - y$$

$$dW^{[1]} = d\mathbf{z}^{[1]} \mathbf{x}^T \quad dW^{[2]} = dz^{[2]} \mathbf{h}^{[1]T}$$

$$db^{[1]} = d\mathbf{z}^{[1]}$$

$$db^{[2]} = dz^{[2]}$$

# Summary: computing derivatives

## Backpropagation

Forward propagation:

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

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

$$z^{[2]} = W^{[2]}\mathbf{h}^{[1]} + b^{[2]}$$

$$\hat{y} = h^{[2]} = \sigma(z^{[2]})$$

$$dz^{[2]} = h^{[2]} - y$$

$$dW^{[2]} = dz^{[2]}\mathbf{h}^{[1]}{}^T$$

$$db^{[2]} = dz^{[2]}$$

$$d\mathbf{z}^{[1]} = W^{[2]}{}^T dz^{[2]} \circ \mathbf{h}^{[1]} \circ (1 - \mathbf{h}^{[1]})$$

$$dW^{[1]} = d\mathbf{z}^{[1]}\mathbf{x}^T$$

$$d\mathbf{b}^{[1]} = d\mathbf{z}^{[1]}$$

# Vectorizing across multiple examples - forward

for  $n = 1$  to N:

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

$$\mathbf{h}_n^{[1]} = \sigma(\mathbf{z}_n^{[1]})$$

$$\mathbf{z}_n^{[2]} = W^{[2]}\mathbf{h}_n^{[1]} + b^{[2]}$$

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

Vectorizing

(Row: hidden units,  
column: examples):

$$Z^{[1]} = W^{[1]}X + \mathbf{b}^{[1]}$$

$$H^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}H^{[1]} + b^{[2]}$$

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

# Vectorizing across multiple examples - backpropagation

Vectorizing:

Given one data point:

$$dz^{[2]} = h^{[2]} - y$$

$$dZ^{[2]} = H^{[2]} - \mathbf{y}$$

$$dW^{[2]} = dz^{[2]} \mathbf{h}^{[1]T}$$

$$dW^{[2]} = \frac{1}{N} dZ^{[2]} H^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$db^{[2]} = \frac{1}{N} \sum dZ^{[2]}$$

$$d\mathbf{z}^{[1]} = dz^{[2]} W^{[2]T} \circ \mathbf{h}^{[1]} \circ (1 - \mathbf{h}^{[1]})$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} \circ H^{[1]} \circ (1 - H^{[1]})$$

$$dW^{[1]} = d\mathbf{z}^{[1]} \mathbf{x}^T$$

$$dW^{[1]} = \frac{1}{N} dZ^{[1]} X^T$$

$$d\mathbf{b}^{[1]} = d\mathbf{z}^{[1]}$$

$$d\mathbf{b}^{[1]} = \frac{1}{N} \sum d\mathbf{z}^{[1]}$$

## Adaptive learning rates

- Popular and simple idea: reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g.  $1/t$  decay:  $\eta^t = \frac{\eta}{\sqrt{t+1}}$
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates.

## Adaptive learning rates

- Popular and simple idea: reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g.  $1/t$  decay:  $\eta^t = \frac{\eta}{\sqrt{t+1}}$
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates.
  - Adagrad, RMSprop, Adam etc

# Implementation of simple NN

# Implementation

Python (Numpy) for 1 hidden layer neural network. (Link)

# Implementation

## Implementation

Deep Learning packages (e.g., Tensorflow, Pytorch MXNet) for multiple layer (convolutional) neural network.

Usefully resources: [Tensorflow web](#), [Pytorch web](#)

## Acknowledgement and Further Reading

Slides are adapted from Dr. Y. Ning's Spring 19 offering of CS-559.