EECS 595

Natural Language Processing

Lecture 4: Logistic Regression and Neural Network

Instructor: Joyce Chai

Misc.

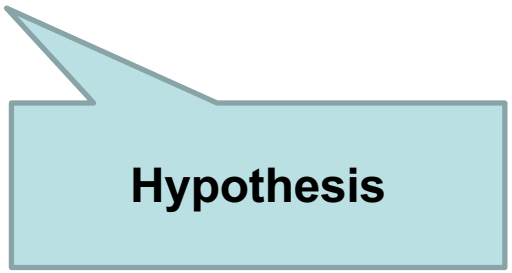
- Homework 1 is due 9/17 11:59pm
- Submit through Canvas
- Both written portion and programming portion
- Written portion: use latex, submit PDF
- Programming portion:
  - Use the provided skeletons for your code.

Learning from Examples

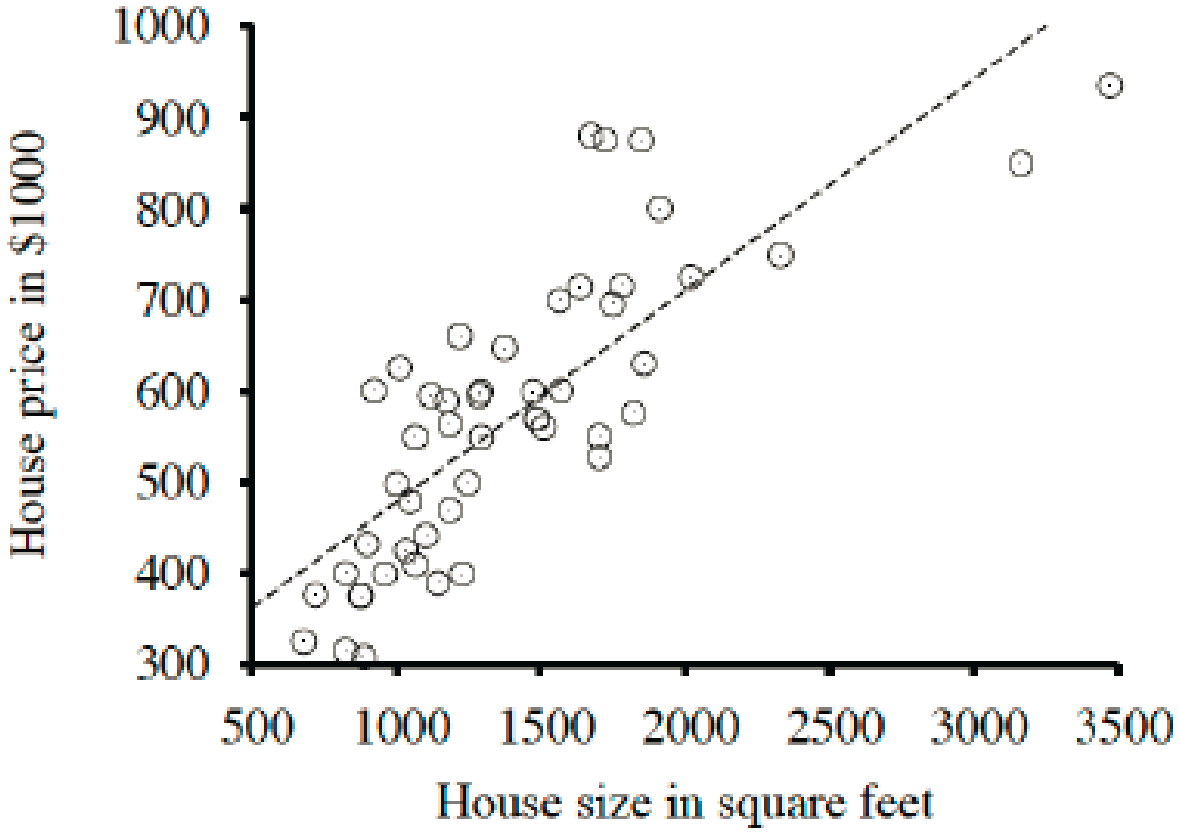
Training Examples: (x1, y1), (x2, y2), … (xn, yn)

Where each yi was generated by an unknown function y=***f***(x)

Discover a function ***h*** that best approximates the true function ***f***



Univariate Linear Regression



Univariate Linear Regression

$\mathbf{w} = [w_0, w_1]$ weight vector

$h_{\mathbf{w}}(x) = w_1 x + w_0$

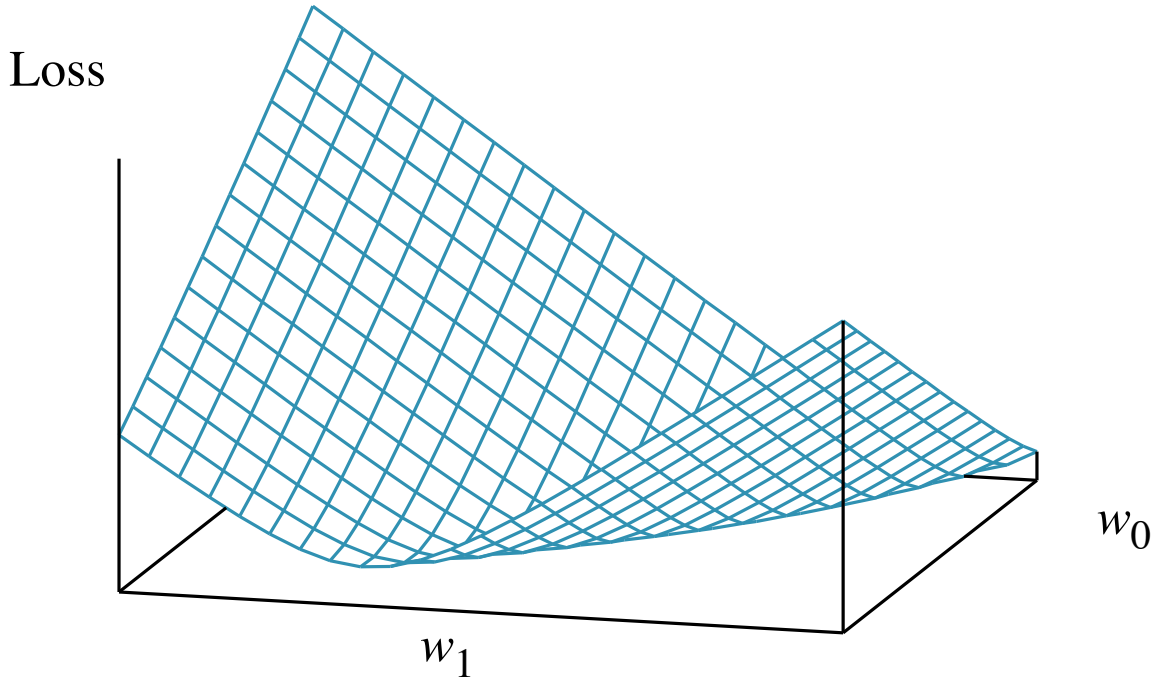
Find weight vector that minimizes empirical loss, e.g., L_2:

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

i.e., find $\mathbf{w}^*$ such that

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} Loss(h_{\mathbf{w}})$$

Weight Space



Finding w*

Find weights such that:

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0 \text{ and } \frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0$$

These equations have a close form solution as follows:

$$w_1=\frac{N(\sum x_jy_j)-(\sum x_j)(\sum y_j)}{N(\sum {x_j}^2)-(\sum x_j)^2}$$

$$w_0=(\sum y_i-w_1(\sum x_j))/N$$

Gradient Descent

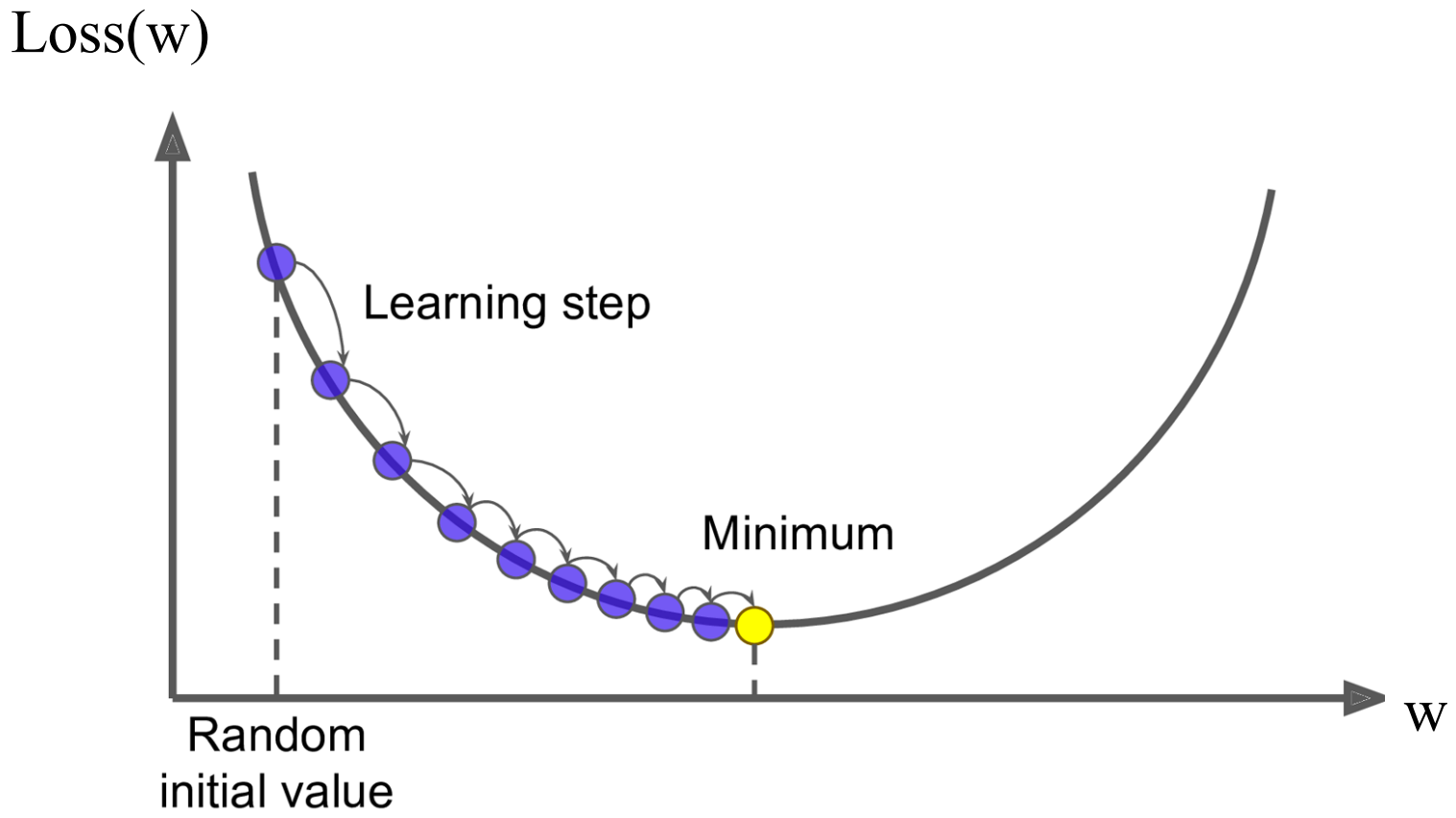
To go beyond linear models, minimizing the loss function often do not have a closed form solution.

We can use hill-climbing to search for a solution in a continuous weight space.

w ⟵ any point in the parameter space
loop until convergence do
    for each w_i in **w** do

$$w_i \leftarrow w_i - \alpha \frac{\partial Loss(\mathbf{w})}{\partial w_i}$$

Gradient Descent



Partial Derivative with One Training Example

$$\frac{\partial Loss(\mathbf{w})}{\partial w_i} = \frac{\partial}{\partial w_i}(\mathrm{y} - h_{\mathbf{w}}(\mathrm{x}))^2 = 2(\mathrm{y} - h_{\mathbf{w}}(\mathrm{x})) \times \frac{\partial}{\partial w_i}(\mathrm{y} - h_{\mathbf{w}}(\mathrm{x}))$$

$$= 2(\mathrm{y} - h_{\mathbf{w}}(\mathrm{x})) \times \frac{\partial}{\partial w_i}(\mathrm{y} - (w_1\mathrm{x} + w_0))$$

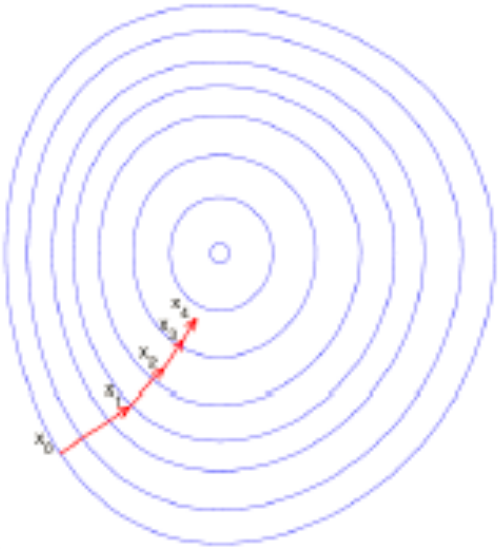
$$\frac{\partial Loss(\mathbf{w})}{\partial w_0} = -2(\mathrm{y} - h_{\mathbf{w}}(\mathrm{x})) \qquad \frac{\partial Loss(\mathbf{w})}{\partial w_1} = -2(\mathrm{y} - h_{\mathbf{w}}(\mathrm{x})) \times x$$

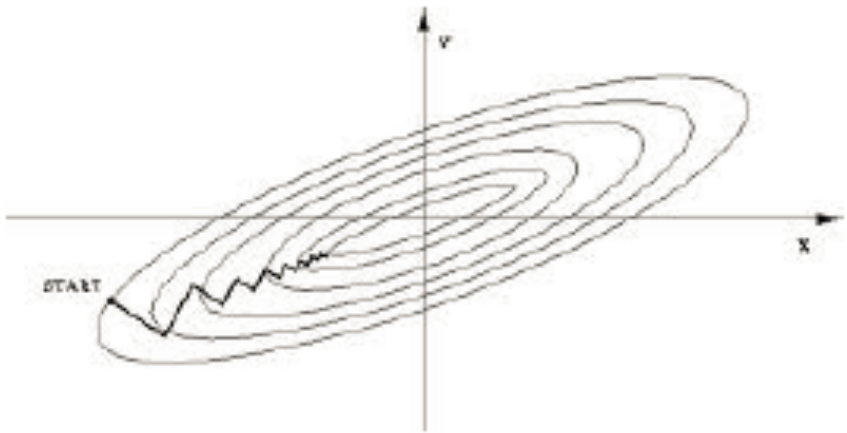
$$w_0 \leftarrow w_0 + \alpha(y - h_{\mathbf{w}}(\mathrm{x})) \quad w_1 \leftarrow w_1 + \alpha(y - h_{\mathbf{w}}(\mathrm{x})) \times x$$

Gradient Descent Cont.

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

step size or learning rate





Gradient Descent Cont.

For one training example (x, y):

$$w_0 \leftarrow w_0 + \alpha(y - h_{\mathbf{w}}(x)) \quad \text{and} \quad w_1 \leftarrow w_1 + \alpha(y - h_{\mathbf{w}}(x))x$$

For *N* training examples: minimize the sum of individual losses

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \quad \text{and} \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))x_j$$

batch gradient descent

stochastic gradient descent: take a step for one training example at a time

Multivariate case

$$h_{sw}(\mathbf{x}_j) = w_0 + w_1 x_{j1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

Augmented vectors: add a feature to each $\mathbf{x}$ by tacking on a 1: $x_{j,0} = 1$

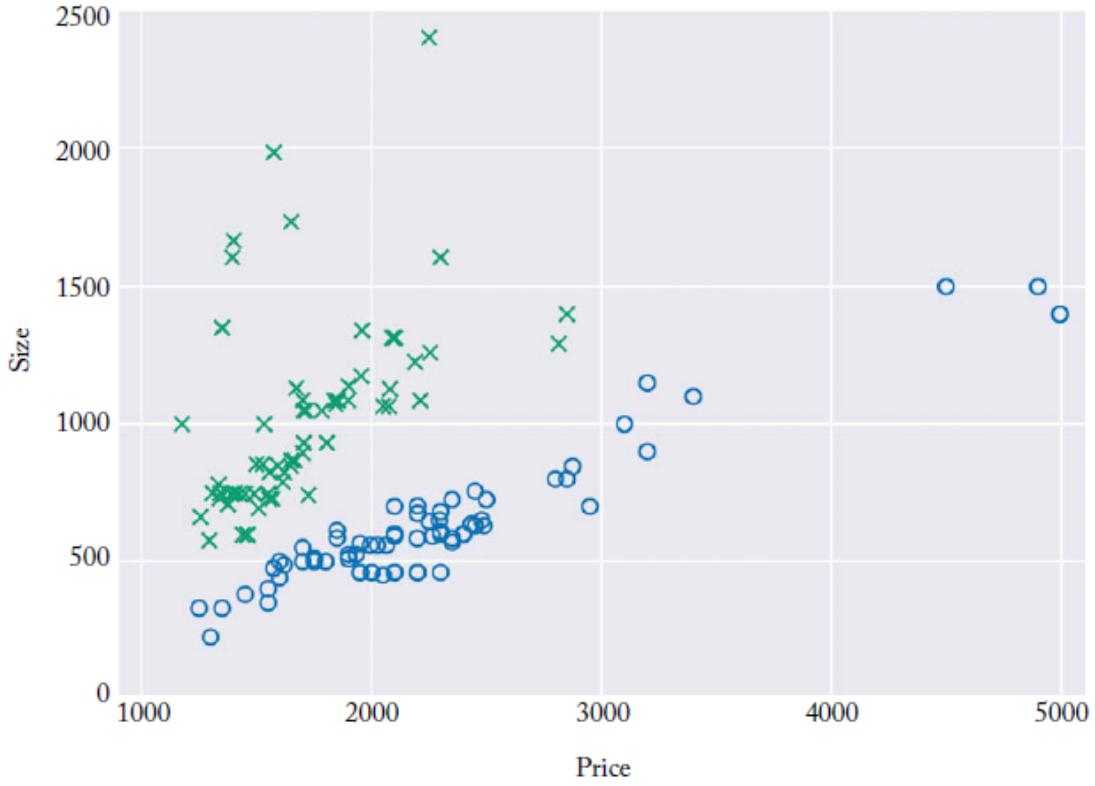
Then:

$$h_{sw}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^T \mathbf{x}_j = \sum_i w_i x_{j,i}$$

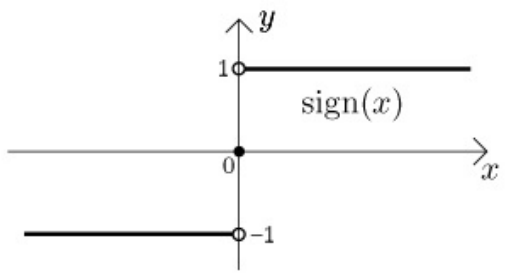
And batch gradient descent update becomes:

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) x_{j,i}$$

Linear Classifiers

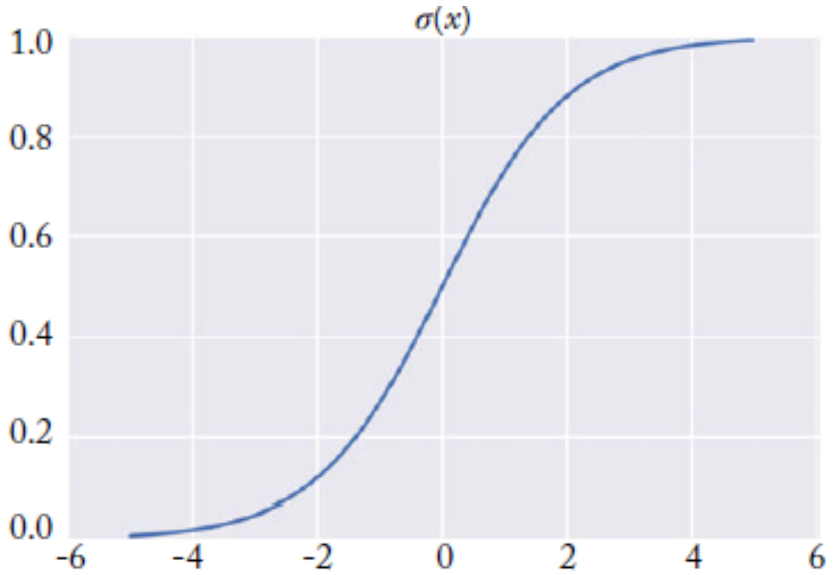


$$\text{sign}(x) = \begin{cases} 1, & x > 0; \\ 0, & x = 0; \\ -1, & x < 0. \end{cases}$$



$$\hat{y} = \text{sign}(f(x)) = \text{sign}(x \cdot w + b)$$
$$= \text{sign}(\text{size} \times w_1 + \text{price} \times w_2 + b),$$

Log-Linear Binary Classifiers



Sigmoid function, logistic function

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

$$\hat{y} = \sigma\big(f(x)\big) = \frac{1}{1 + e^{-(Wx+b)}}$$

Elements of a Classifier

Training data: N pairs of data instances $(x^{(j)},y^{(j)})$ where $x^{(j)}$ is an input, $y^{(j)}$ is a class label.

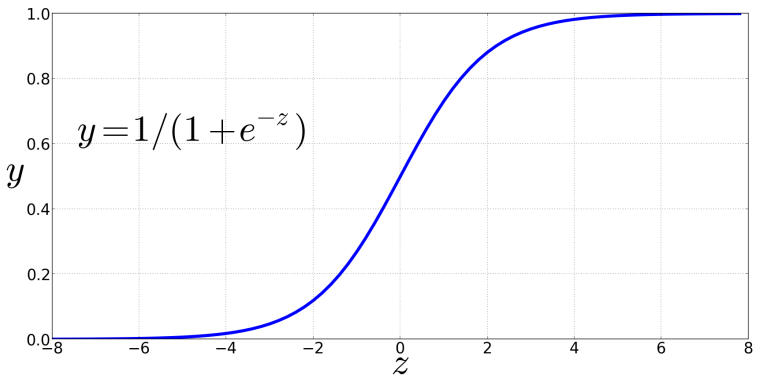
Each input $x^{(j)}$ can be represented by a feature vector $[x_1^{(j)},x_2^{(j)},...,x_n^{(j)}]$, feature i for input $x^{(j)}$ is $x_i^{(j)}$ or $f_i(x^{(j)})$

Classification function $p(y|x)$ returns an estimated class label $\hat{y}$.

An objective function for learning, usually involves minimizing error on the training data

An algorithm for optimizing the object function.

Logistic Regression



- Return a probability value
- Differentiable

$$P(y=1) = \sigma(w \cdot x + b) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

$$P(y=0) = 1-\sigma(w \cdot x + b) = 1-\frac{1}{1+e^{-(w \cdot x+b)}} = \frac{e^{-(w \cdot x+b)}}{1+e^{-(w \cdot x+b)}}$$

Decision making:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

An Example on Sentiment Analysis

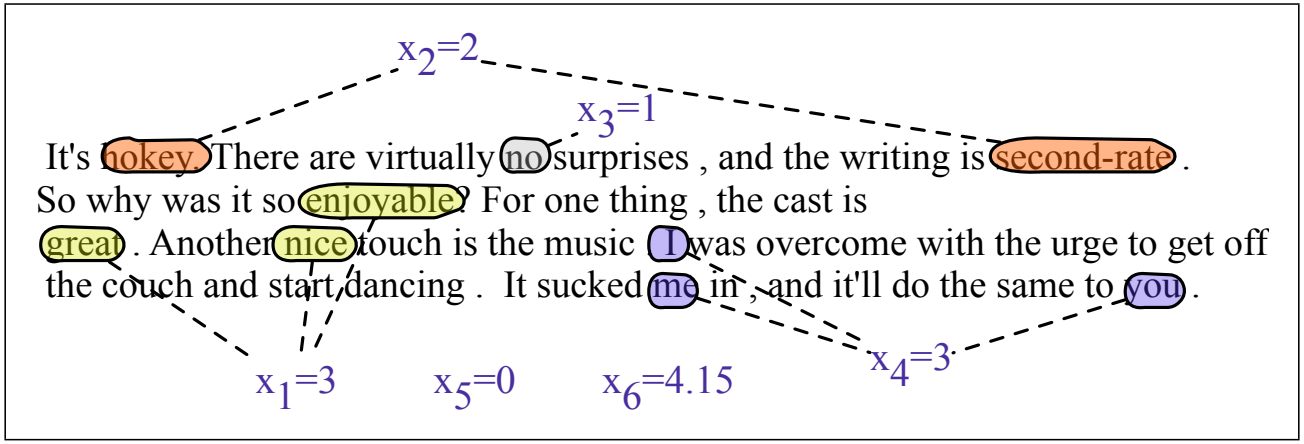


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Feature engineering

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

An Example on Sentiment Analysis

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Suppose w=[2.5, -5.0, -1.2, 0.5, 2.0, 0.7], b = 0.1

$$\begin{aligned} p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\ &= \sigma(.805) \\ &= 0.69 \\ p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\ &= 0.31 \end{aligned}$$

Weights: (1) importance of corresponding feature in prediction;
(2) meanings of positive and negative weights

Feature Engineering

- Features are often designed to capture linguistic intuition and/or domain properties or knowledge
- A careful analysis using training/development datasets.
- Combination of features to capture feature interaction.
- Feature template: design a template so it can automatically extract features from training set. (e.g., capture previous word or previous two words).
- Recent work has focused on representational learning by avoiding feature engineering.

Learning log linear model: loss function

We need a loss function that expresses, for an observation x, how close the classifier output ($\hat{y} = \sigma(w \cdot x + b)$) is to the correct output (y, which is 0 or 1). We'll call this:

$$L(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y$$

- Maximize the log probability of the true y labels in the training data -> conditional MLE.

  y = 0 or 1 $\qquad p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$

- Minimize the loss -> cross entropy loss

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$$

$$L_{CE}(w, b) = -[y \log \sigma(w \cdot x + b) + (1-y) \log(1 - \sigma(w \cdot x + b))]$$

Learning log linear model: loss function

Extend from one data point to the entire training set

$$\begin{aligned}
\log p(\text{training labels}) &= \log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}) \\
&= \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) \\
&= -\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})
\end{aligned}$$

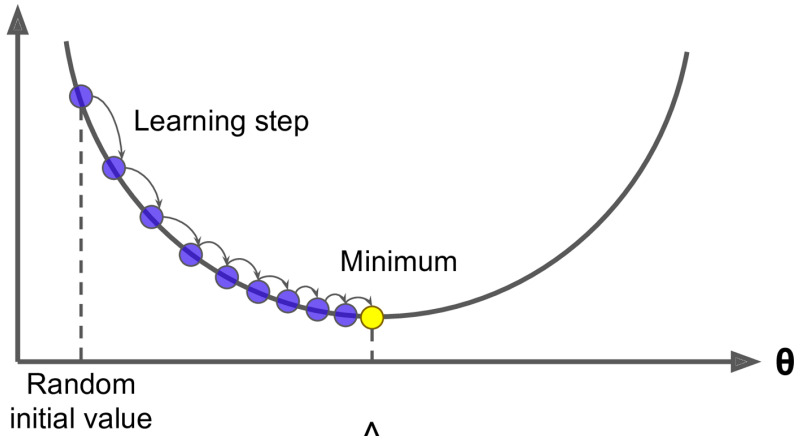
$$\begin{aligned}
Cost(w,b) &= \frac{1}{m} \sum_{i=1} L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\
&= -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\end{aligned}$$

Gradient Descent

Objective is to find parameters to minimize

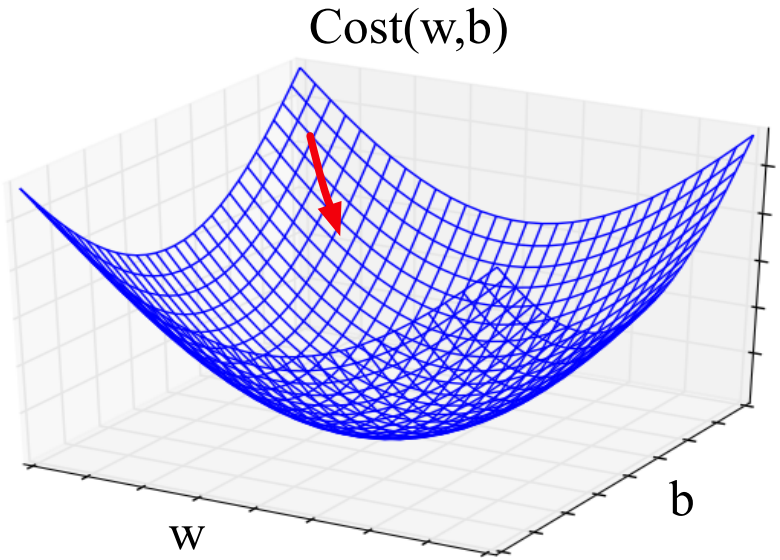
$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

A convex function, has a global minimum. Gradient descent from any point guarantees to find the minimum.



$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

Gradient Descent: N dimensions



$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

Stochastic Gradient Descent

Computing gradient after each training example

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(n)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(n)}$

$\theta \leftarrow 0$
repeat T times
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
  Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
  Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
  $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?
  $\theta \leftarrow \theta - \eta\, g$ # go the other way instead
return θ

Can result in very choppy movement.

Minibatch: over batches of training example rather than an single example.

Some Basic Derivative

$$\frac{d}{dx}\ln(x)=\frac{1}{x} \qquad \frac{d}{dx}\left(\mathbf{e}^{x}\right)=\mathbf{e}^{x}$$

Basic Properties/Formulas/Rules

$\frac{d}{dx}\left(cf(x)\right)=cf'(x)$, c is any constant. $\left(f(x)\pm g(x)\right)'=f'(x)\pm g'(x)$

$\frac{d}{dx}\left(x^{n}\right)=nx^{n-1}$, n is any number. $\frac{d}{dx}(c)=0$, c is any constant.

$(fg)'=f'g+fg'$ – **(Product Rule)** $\left(\frac{f}{g}\right)'=\frac{f'g-fg'}{g^{2}}$ – **(Quotient Rule)**

$\frac{d}{dx}\left(f\left(g(x)\right)\right)=f'\left(g(x)\right)g'(x)$ **(Chain Rule)**

$\frac{d}{dx}\left(\mathbf{e}^{g(x)}\right)=g'(x)\mathbf{e}^{g(x)}$ $\frac{d}{dx}\left(\ln g(x)\right)=\frac{g'(x)}{g(x)}$

Common Derivatives

Polynomials

$\frac{d}{dx}(c)=0$ $\frac{d}{dx}(x)=1$ $\frac{d}{dx}(cx)=c$ $\frac{d}{dx}\left(x^{n}\right)=nx^{n-1}$ $\frac{d}{dx}\left(cx^{n}\right)=ncx^{n-1}$

Derivation for weight learning

Chain rule: $f(x) = u(v(x)) \quad \text{-->} \quad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$

Derivative of the sigmoid function $\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$

$$\begin{aligned}
\frac{\partial LL(w,b)}{\partial w_j} &= \frac{\partial}{\partial w_j} - [y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\
&= -\left[\frac{\partial}{\partial w_j} y \log \sigma(w \cdot x + b) + \frac{\partial}{\partial w_j} (1-y) \log [1 - \sigma(w \cdot x + b)] \right]
\end{aligned}$$

$$\frac{\partial LL(w,b)}{\partial w_j} = -\frac{y}{\sigma(w \cdot x + b)} \frac{\partial}{\partial w_j} \sigma(w \cdot x + b) - \frac{1-y}{1 - \sigma(w \cdot x + b)} \frac{\partial}{\partial w_j} [1 - \sigma(w \cdot x + b)]$$

Rearranging terms:

$$\frac{\partial LL(w,b)}{\partial w_j} = -\left[\frac{y}{\sigma(w\cdot x+b)} - \frac{1-y}{1-\sigma(w\cdot x+b)}\right]\frac{\partial}{\partial w_j}\sigma(w\cdot x+b)$$

And now plugging in the derivative of the sigmoid, and using the chain rule

$$\begin{aligned}
\frac{\partial LL(w,b)}{\partial w_j} &= -\left[\frac{y-\sigma(w\cdot x+b)}{\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]}\right]\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]\frac{\partial(w\cdot x+b)}{\partial w_j} \\
&= -\left[\frac{y-\sigma(w\cdot x+b)}{\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]}\right]\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]x_j \\
&= -[y-\sigma(w\cdot x+b)]x_j \\
&= [\sigma(w\cdot x+b)-y]x_j
\end{aligned}$$

And the gradient for multiple data points is the sum of the individual gradients::

$$\frac{\partial Cost(w,b)}{\partial w_j} = \sum_{i=1}^{m}\left[\sigma(w\cdot x^{(i)}+b)-y^{(i)}\right]x_j^{(i)}$$

An Example

Suppose one data point with two features, y = 1 (positive review)

$$x_1 = 3 \quad \text{(count of positive lexicon words)}$$
$$x_2 = 2 \quad \text{(count of negative lexicon words)}$$

Let's assume the initial weights and bias in θ^0 are all set to 0, and the initial learning rate η is 0.1:

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$

The single update step requires that we compute the gradient, multiplied by the learning rate

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$$

Computer the derivatives:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(w,b)}{\partial w_1} \\ \frac{\partial L_{CE}(w,b)}{\partial w_2} \\ \frac{\partial L_{CE}(w,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Parameter update:

$$\theta^2 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Regularization

Avoid overfitting training data -> avoid very high weights for features that also fit the noise of the data. A regularization term is added to the objective function.

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_{1=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(w)$$

L1 regularization (Losso Regression): $R(W) = ||W||_1 = \sum_{i=1}^{N} |w_i|$

$$\hat{w} = \underset{w}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |w_j|$$

L2 regularization (Ridge Regression): $R(W) = ||W||_2^2 = \sum_{j=1}^{N} w_j^2$

$$\hat{w} = \underset{w}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} w_j^2$$

Multinomial Logistic Regression: Softmax

More than two classes. $c \in C,\ p(y = c|x)$

For a vector z of dimensionality k, the softmax is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq k$$

The softmax of an input vector $z = [z_1, z_2, ..., z_k]$ is thus a vector itself:

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, ..., \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}} \right]$$

Multinomial Logistic Regression: Softmax

More than two classes. $c \in C,\ p(y = c|x)$

Thus for example given a vector:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

the result softmax(z) is

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Multinomial Logistic Regression: Softmax

More than two classes. $c \in C,\ p(y = c|x)$

$$p(y = c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

Loss function for a single instance: (K classes)

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\sum_{k=1}^{K} 1\{y = k\} \log p(y = k|x) \\ &= -\sum_{k=1}^{K} 1\{y = k\} \log \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^{K} e^{w_j \cdot x + b_j}} \end{aligned}$$

function 1{} returns to 1 if the condition in the brackets is true and to 0 otherwise.

Multinomial Logistic Regression: Softmax

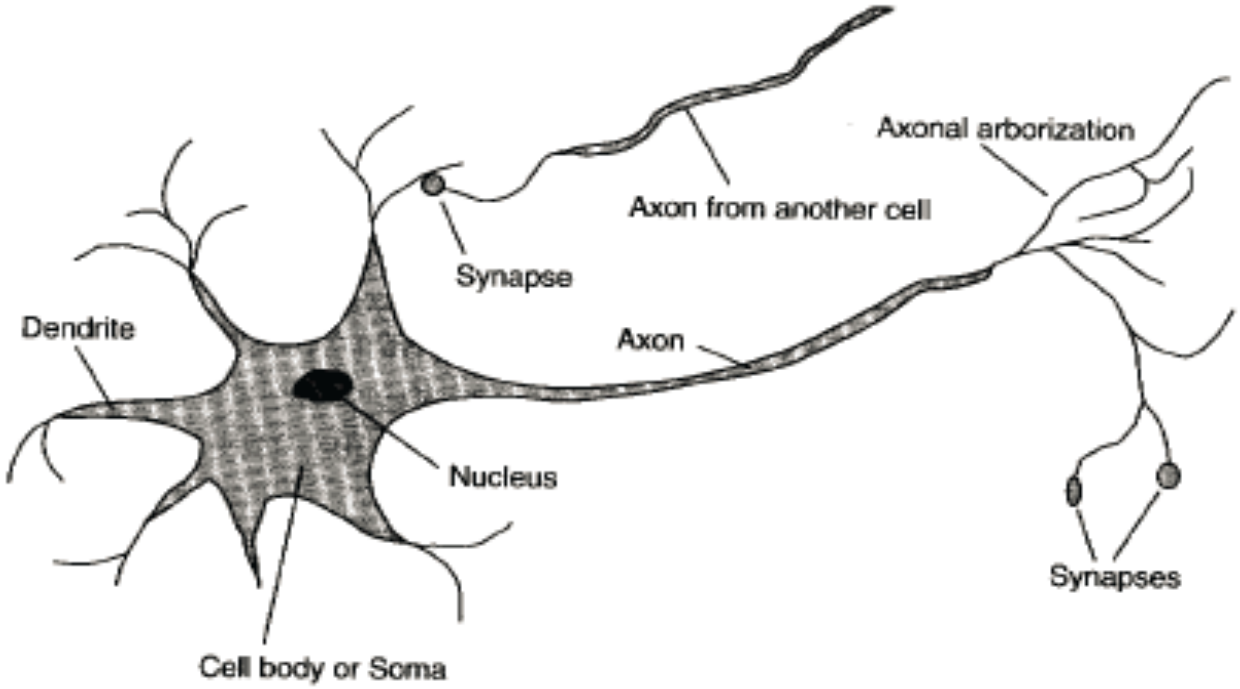
Derivative is similar with the logistic regression

$$\begin{aligned}\frac{\partial L_{CE}}{\partial w_k} &= -(1\{y=k\} - p(y=k|x))x_k \\ &= -\left(1\{y=k\} - \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^{K} e^{w_j \cdot x + b_j}}\right) x_k\end{aligned}$$

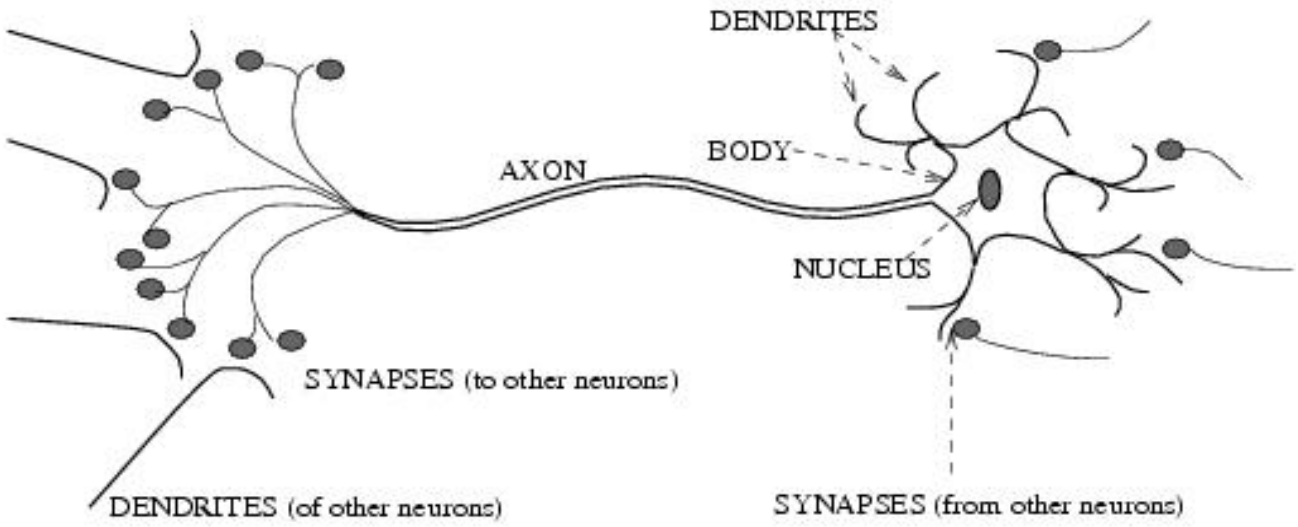
A Brief Intro to Artificial Neural Network

How does the brain work?

- Remains a great mystery of science!
- Basic component: the neuron.



Biological Neural Activity



- Each neuron has a *body*, an *axon*, and many *dendrites*
  - Can be in one of the two states: *firing* and *rest.*
  - Neuron fires if the total incoming stimulus exceeds the threshold
- *Synapse*: thin gap between axon of one neuron and dendrite of another.
  - Signal exchange
  - Synaptic strength

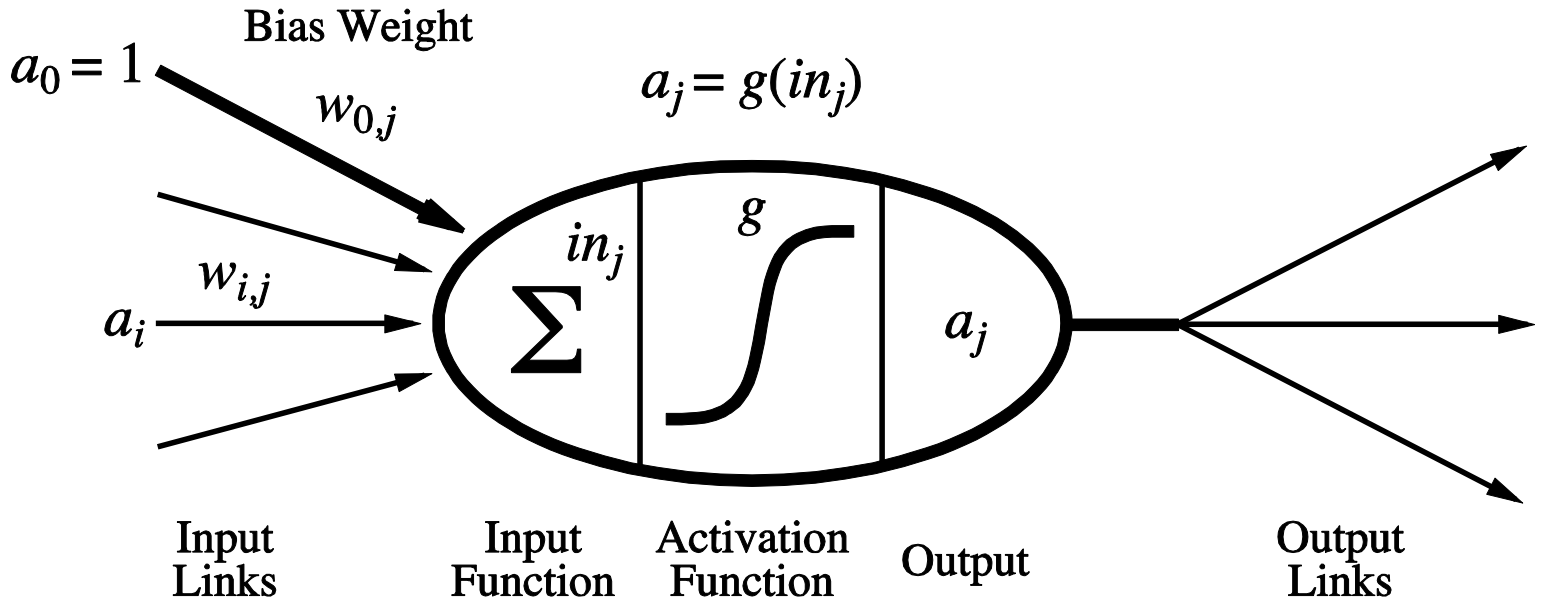
Introduction of ANN

- **What is an (artificial) neural network**
  - A set of **nodes** (units, neurons, processing elements)
    - Each node has input and output
    - Each node performs a simple computation by its **node function**
  - **Weighted connections** between nodes
    - Connectivity gives the structure/architecture of the net
    - What can be computed by a NN is primarily determined by the connections and their weights
  - A very much simplified version of networks of neurons in animal nerve systems

Introduction of ANN

ANN	Bio NN
• **Nodes** – input – output – node function • **Connections** – connection strength	• **Cell body** – signal from other neurons – firing frequency – firing mechanism • **Synapses** – synaptic strength

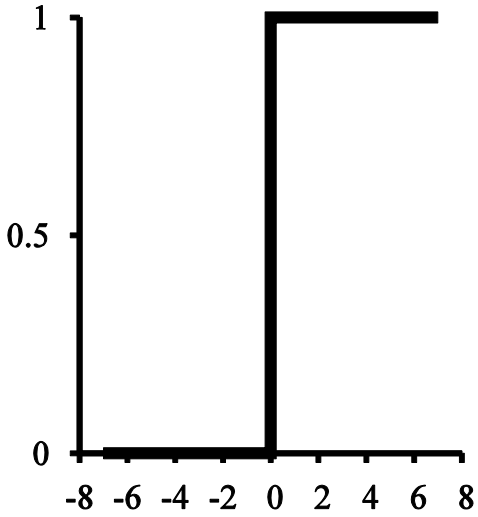
Introduction of ANN



- **Input function:** $in_j = \sum_{i=0}^{n} w_{i,j} a_i$
- **Activation function:** $a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i)$

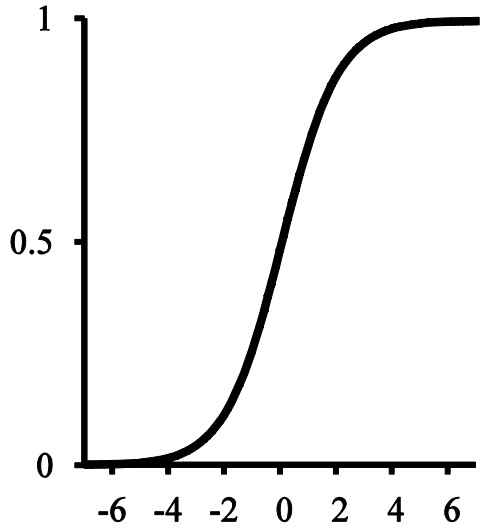
Activation Function

- threshold function-> the unit is called perceptron



- Logistic function -> sigmoid perceptron

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



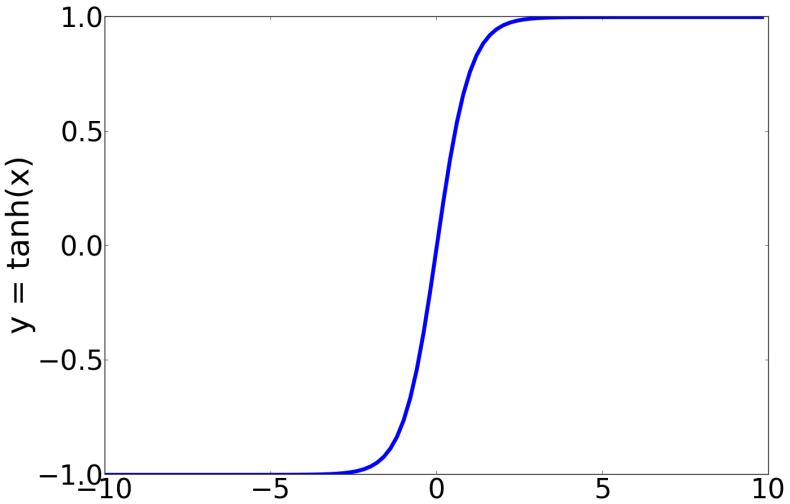
Both are nonlinear functions. Logistic function is differentiable

Activation Function

Tanh Activation (hyperbolic tangent)

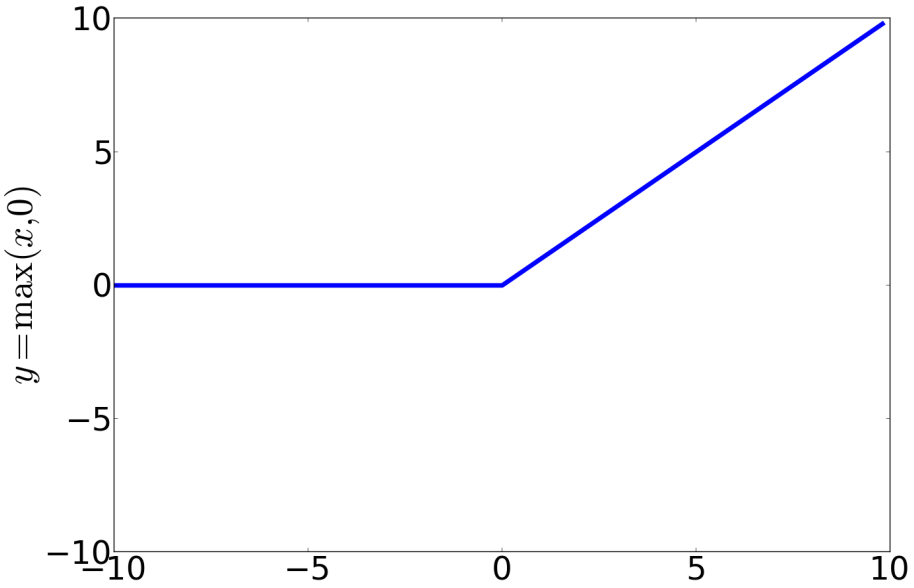
- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

$$y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



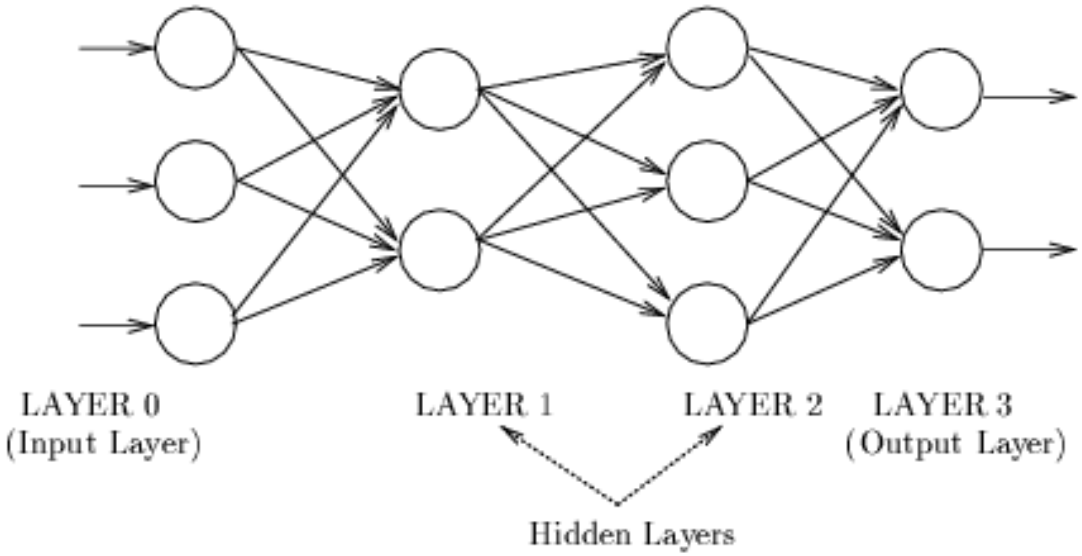
Activation Function

Rectified linear unit. ReLU: g(x) = max(0, x)



Network Connections

- Feed-forward Network
  - Has connections only in one direction -> directed acyclic



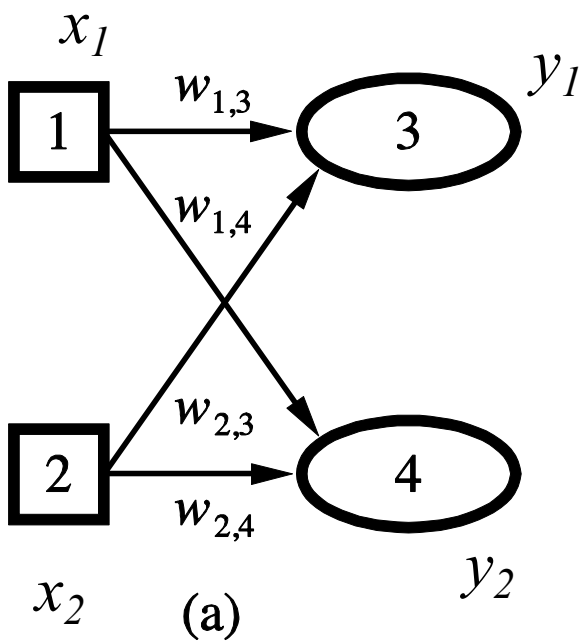
Conceptually, nodes at higher levels successively abstract features from preceding layers

- Recurrent Network
  - Feeds its outputs back into its own inputs

Single-layer feed-forward Perceptron

- Single-layer feed-forward perceptrons
  - A network with all the inputs connected directly to the outputs

An example: two-bit add function



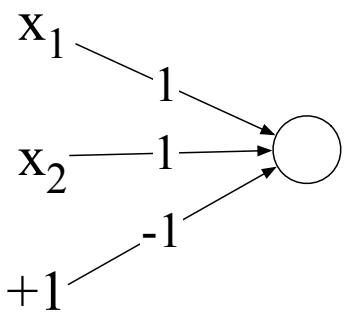
x_1	x_2	y_1 (carry)	y_2 (sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

"carry function" (unit 3) can be learned easily, but not the "sum function" (unit 4)

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

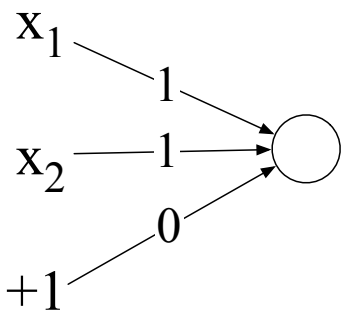
A simple perceptron

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



(a)

AND



(b)

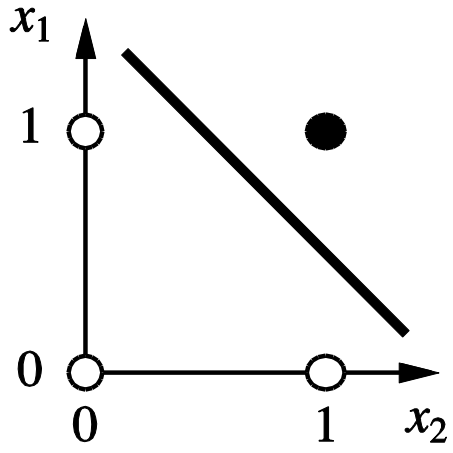
OR

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

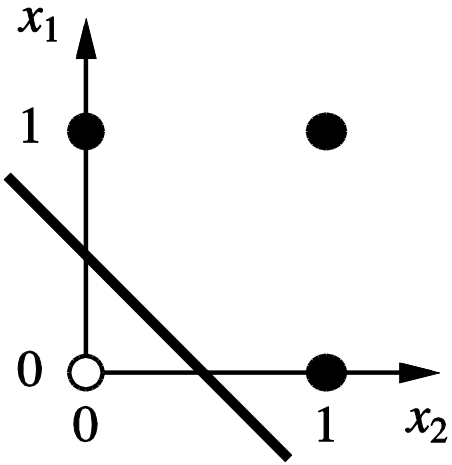
A simple perceptron

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

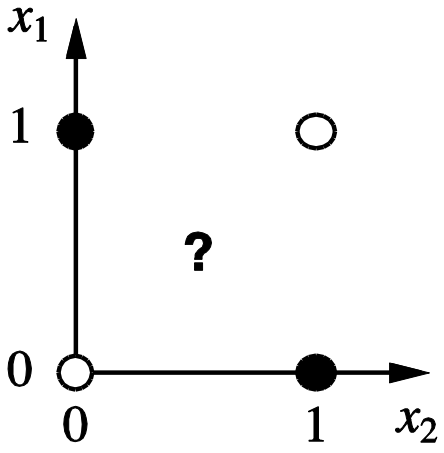
- Perceptrons cannot learn the function that is not linearly separable



(a) x_1 **and** x_2



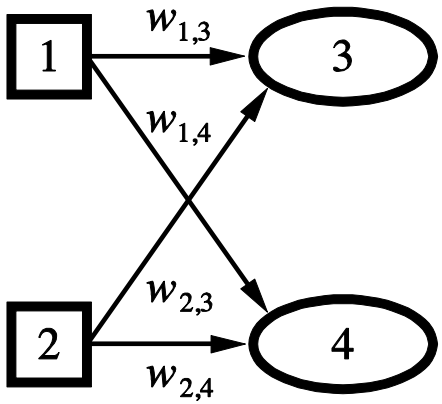
(b) x_1 **or** x_2



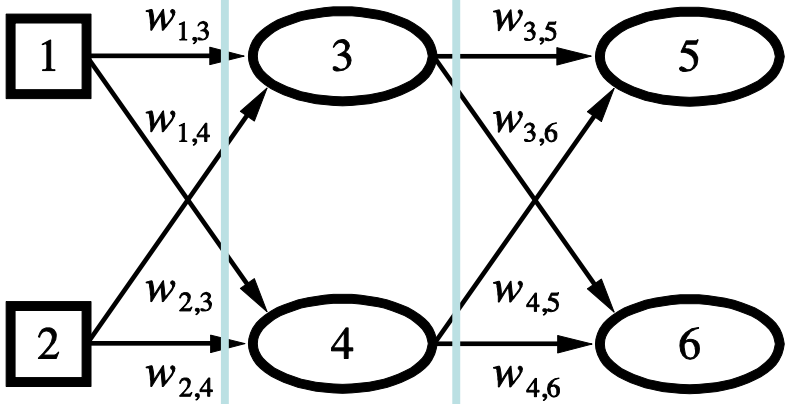
(c) x_1 **xor** x_2

Solution: Multilayer feed-forward NN

- Add hidden layers to incorporate non-linearity into the network.



(a)



(b) Hidden layer

XOR

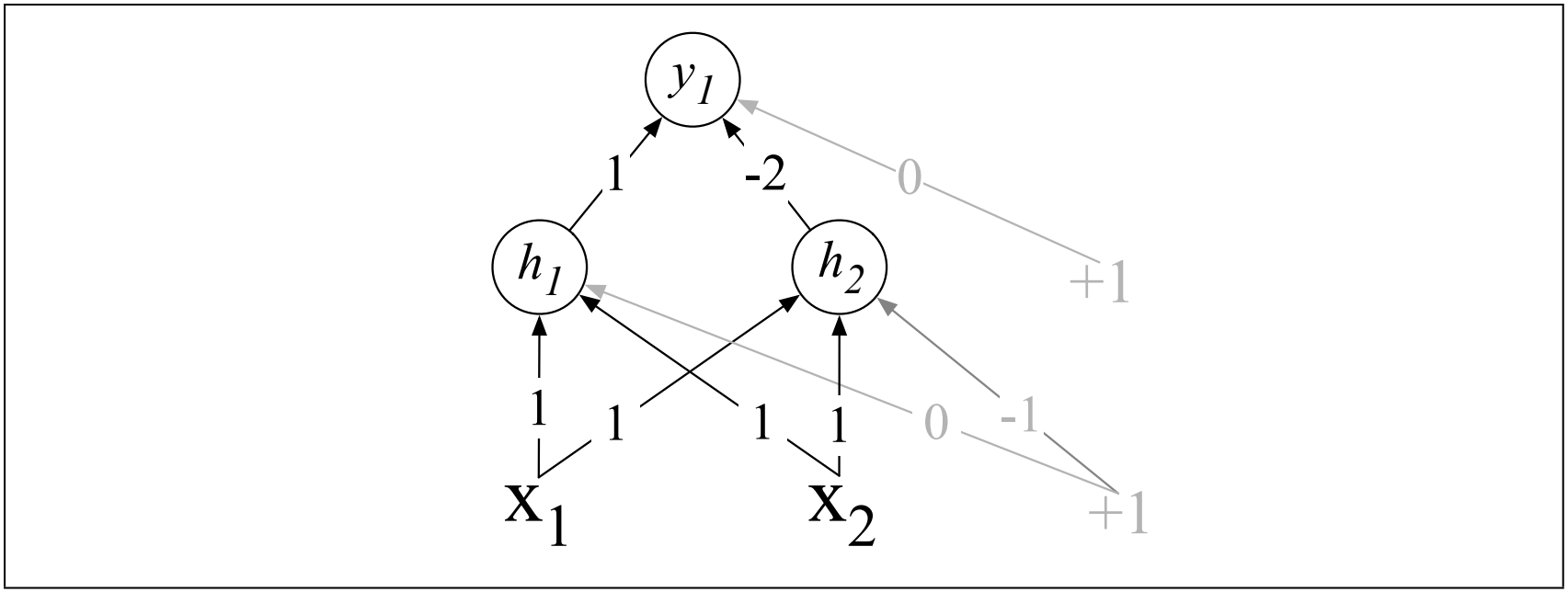
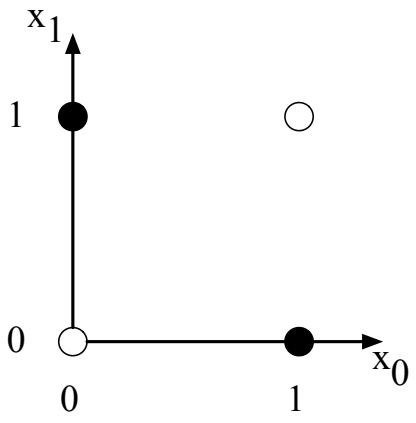
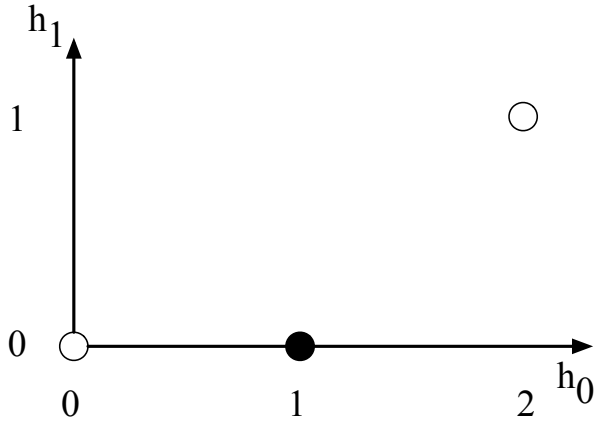


Figure 7.6 XOR solution after Goodfellow et al. (2016). There are three ReLU units, in two layers; we've called them h_1, h_2 (h for "hidden layer") and y_1. As before, the numbers on the arrows represent the weights w for each unit, and we represent the bias b as a weight on a unit clamped to +1, with the bias weights/units in gray.

XOR



a) The original *x* space



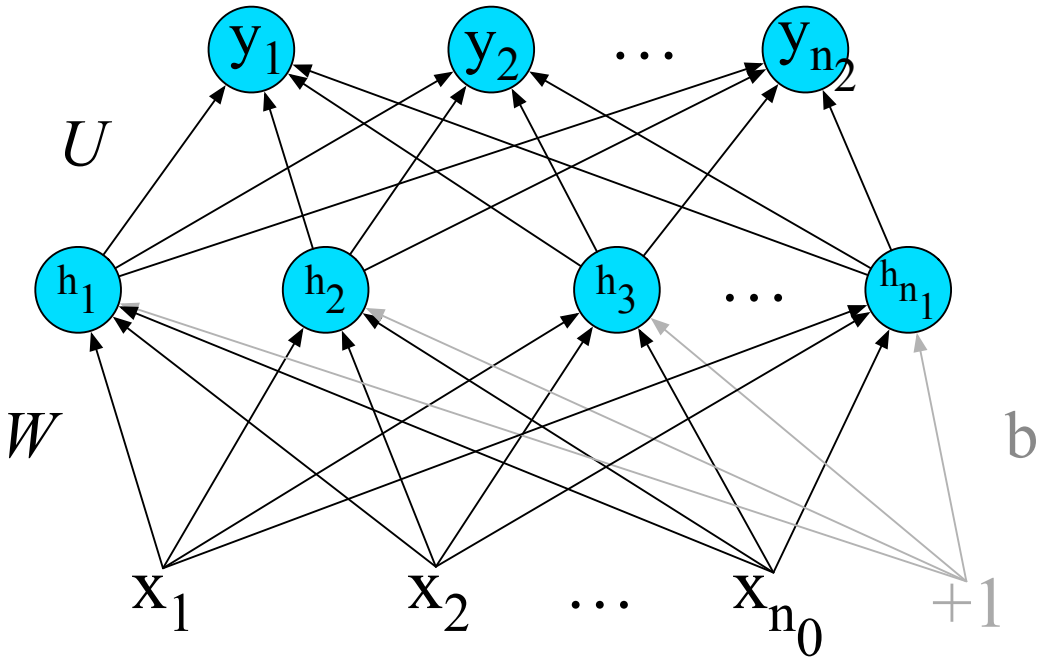
b) The new *h* space

x0	x1	h0	h1	y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0

The hidden layer forms a new representation of the input. Notice that the input point [0 1] has been collapsed with the input point [1 0], making it possible to linearly separate the positive and negative cases of XOR.

A Simple 2-layer Feed-forward Network

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} \quad 1 \le i \le d$$



$$y = \text{softmax}(z)$$

$$z = Uh$$
$$U \in \mathbb{R}^{n_2 \times n_1}$$

$$h = \sigma(Wx + b)$$
$$x \in \mathbb{R}^{n_0} \quad h \in \mathbb{R}^{n_1}$$
$$b \in \mathbb{R}^{n_1} \quad W \in \mathbb{R}^{n_1 \times n_0}$$

Softmax Activation Function

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c|\mathbf{x})$
- Softmax activation function at the output:

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp a_1}{\sum_c \exp a_c}, \ldots, \frac{\exp a_C}{\sum_c \exp a_c}\right]^T$$

  - strictly positive
  - sums to one
- Predicted class: one with highest estimated probability

Training NN

- Given labeled examples (with classification labels), training is to learn the weights in the network.
- We will need some loss function to measure the distance between the predicted output and the ground-truth output.
  - Cross entropy loss, etc.
- Find parameters (weights, biases) that minimize the loss
  - Gradient descent
- Need to estimate the weights all the way to layer 1 even the loss is happening after many layers at the output.
  - Back-propogation:

Cross Entropy Loss

- Let y be the a vector over the C classes representing the true probability distribution, $\hat{y}$ be the vector of distribution of system prediction.

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

- In hard classification (one class is true), y is a one-hot vector. CE loss is simplified to the log probability of the correct class: **Negative log likelihood loss**

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i \qquad L_{CE}(\hat{y}, y) = -\log \frac{e^{z_i}}{\sum_{j-1}^{K} e^{z_j}}$$

- Take the derivative of loss function $\frac{\partial L_{CE}}{\partial w_k}$, which allows you to update weights in the last layer

Back Propagation

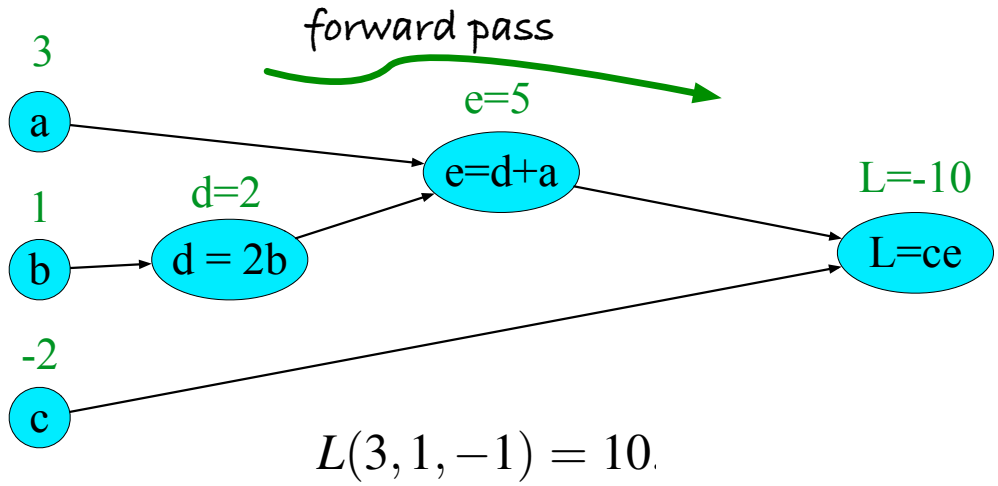
- We need to update weights all the way to layer 1.
- Calculate gradients for earlier layers through error back-propagation
- Same as a more general procedure called backward differentiation, which depends on computation graphs

Computation Graphs

- Model the process of computation which is broken down to separate operations. Each operation is a node in the graph.
- Forward pass is used to compute the value of a function given an input.

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



Backward Differentiation

- Backpass to compute derivatives $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$
- Use the chain rule: a composite function $f(x) = u(v(w(x)))$,

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

$$L(a,b,c) = c(a+2b)$$

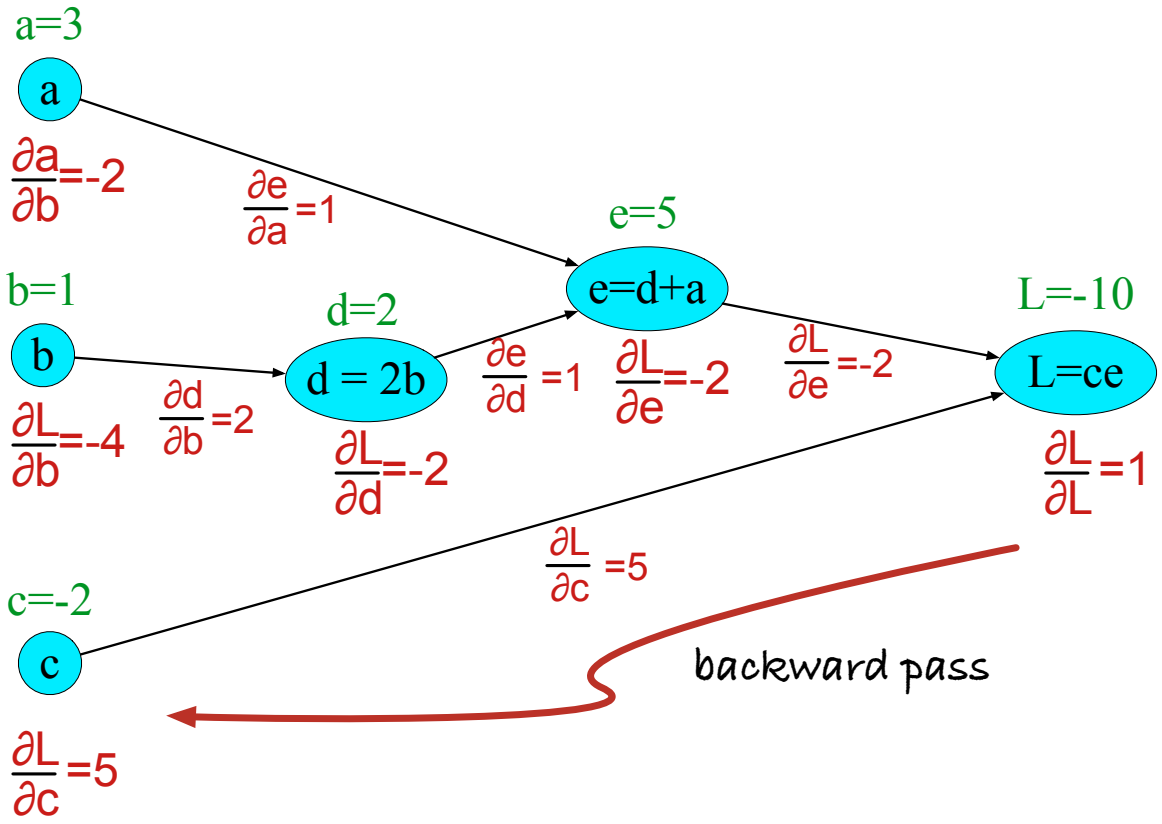
$$\begin{aligned} d &= 2 * b \\ e &= a + d \\ L &= c * e \end{aligned}$$

$$\frac{\partial L}{\partial c} = e \qquad \begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

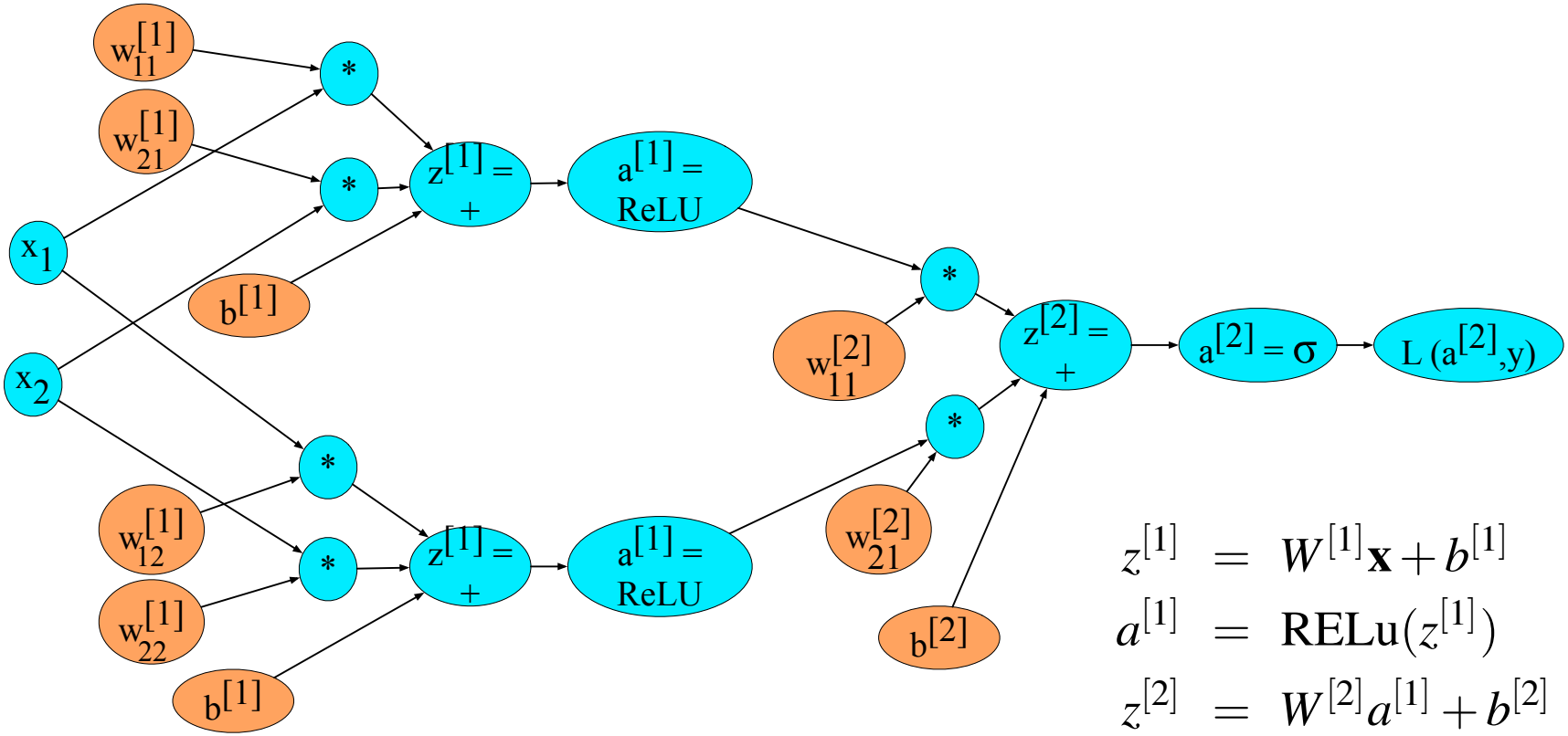
$$\begin{aligned} L = ce &: \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e \\ e = a + d &: \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1 \\ d = 2b &: \quad \frac{\partial d}{\partial b} = 2 \end{aligned}$$

Backpass Computation

- At each node, (1) compute the partial derivative with respect to the parent, (2) multiply it by the partial derivative passed down by the parent; (3) pass it to the child.



Computation Graphs for NN



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \text{RELu}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$