Intro to Connectionist Machine Learning

CS4248 Natural Language Processing

Week 05

Shreyas Kuthanoor PRAKESH and Min-Yen KAN

5

Slides from NUS CS3244 and Dan Jurafsky (Stanford)

$\frac{d}{dx}\ln x = \frac{1}{x}$, $\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$, $f(x) = u[v(x)]$, $\frac{df(x)}{dx} = \frac{du}{dv}\cdot\frac{dv}{dx}$

Week 04 Agenda

$$\frac{\partial L_{CE}}{\partial w_j} = \frac{\partial}{\partial w_j}\left[-\left[y\log\sigma(w\cdot x+b) + (1-y)\left(\log(1-\sigma(wx+b))\right)\right]\right]$$

$$= -\left[\frac{\partial}{\partial w_j} y\log\sigma(wx+b) + \frac{\partial}{\partial w_j}(1-y)\left(\log(1-\sigma(wx+b))\right)\right]$$

$$\frac{\partial L_{CE}}{\partial w_j} = \frac{-y}{\sigma(wx+b)}\frac{\partial}{\partial w_j}\sigma(wx+b) - \frac{1-y}{1-\sigma(wx+b)}\frac{\partial}{\partial w_j}1-\sigma(wx+b)$$

$$\frac{\partial L_{CE}}{\partial w_j} = -\left[\frac{y}{\sigma(wx+b)} - \frac{1-y}{1-\sigma(wx+b)}\right]\frac{\partial}{\partial w_j}\sigma(wx+b)$$

Text Classification

Case Study: Sentiment Analysis

TF-IDF

Vector Space Model

Naïve Bayes
and a Runthrough (time permitting)

Evaluating Text Classification

$$\frac{\partial L_{CE}}{\partial w_j} = -\left[\frac{y-\sigma(wx+b)}{\sigma(wx+b)[1-\sigma(wx+b)]}\right]\sigma(wx+b)[1-\sigma(wx+b)]\frac{\partial(wx+b)}{\partial w_j}$$

$$= -\left[\frac{y-\sigma(wx+b)}{\sigma(wx+b)[1-\sigma(wx+b)]}\right]\sigma(wx+b)[1-\sigma(wx+b)]\,x_j$$

Week 05 Agenda

$$= -[y-\sigma(wx+b)]\,x_j$$

$$= [\sigma(wx+b)-y]\,x_j$$

Generative vs. Discriminative Classifiers

Classification with Logistic Regression and a Runthrough

Cross Entropy

Stochastic Gradient Descent

LR as a Probabilistic ML Classifier

Regularization

XOR

Neural Networks

connectionless classification

Generative vs. Discriminative Classifiers

Slide Credits: Dan Jurafsky (Stanford)

Logistic Regression

Important analytic tool in natural and social sciences.

Baseline supervised machine learning tool for classification.

It’s also the foundation of neural networks.

Slide Credits: Dan Jurafsky (Stanford)

Generative vs. Discriminative Classifiers

Naïve Bayes is a generative classifier

But in contrast:

Logistic Regression is a discriminative classifier

Slide Credits: Dan Jurafsky (Stanford)

What are Generative and Discriminative Classifiers?

Suppose we're distinguishing cat from dog images:

Photos sourced from ImageNet. Slide Credits: Dan Jurafsky (Stanford)

Generative Classifier

Build a model of what's in a cat image

How cat looks like

- Knows about whiskers, ears, eyes
- Assigns a probability to any image:
  - How cat-y is this image?
  - Also build a model for dog images

argmax $P(x|y)$

Given a new image at test time:

Run both models and see which one fits better

Photos sourced from ImageNet. Slide Credits: Dan Jurafsky (Stanford)

Discriminative Classifier

find the right class.

Just tries to distinguish dogs from cats. $p(y|x)$

Oh look, dogs have collars! Don't need anything else.

Photos sourced from ImageNet. Slide Credits: Dan Jurafsky (Stanford)

Classifying y given document x in Generative vs Discriminative Classifiers

Naïve Bayes – strong independence assumptions, multiply f_1 and f_2.

$$\hat{y} = argmax\ P(x|y)P(y)$$

Logistic Regression – weight distributed to two correlated features f_1 and f_2. Works

$$\hat{y} = argmax\ P(y|x)$$

Slide Credits: Dan Jurafsky (Stanford)

Components of a probabilistic machine learning classifier

Given m input/output pairs $(\boldsymbol{x}^{(j)}, y^{(j)})$:

need all 4.

1. A feature representation of the input. $\boldsymbol{x}^{(j)} = [x_1, x_2, \ldots, x_i, \ldots, x_n]$. The ith feature is denoted x_i, or more completely $x_i^{(j)}$, sometimes $f_i(\boldsymbol{x})$.

2. A classification function that computes $\hat{y}$, the estimated class, via $p(y|\boldsymbol{x})$, like the sigmoid or softmax functions.

3. An objective function for learning, like cross-entropy loss.

4. An algorithm for optimizing the objective function: stochastic gradient descent.

Slide Credits: Dan Jurafsky (Stanford)

Classification with Logistic Regression

want to find optimal weights.

The Two Phases of Logistic Regression

separate weight

Training: we learn weights θ and bias b using **stochastic gradient descent** and **cross-entropy loss.**

⚠ **Notation Varies:** Weights are also called *parameters*, sometime denoted as w (as used in the SLP3 textbook).

Why is θ separate from b? Mull on that.

Test: Given a test example x, we compute $p(y|x)$ using learned weights θ and bias b, and return whichever label ($y = 1$ or $y = 0$) has higher probability.

Slide Credits: Dan Jurafsky (Stanford)

Logistic Regression: Weighted Features

For feature x_i, weight θ_i tells is how important is x_i:

- x_1 = "review contains *awesome*": $\theta_1 = +10$
- x_2 = "review contains *abysmal*": $\theta_2 = -10$
- x_3 = "review contains *mediocre*": $\theta_3 = -2$

bias

weight

We'll sum up all the weighted features and the bias:

$$z = \left(\sum_{i=1}^{n} \theta_i x_i\right) + b \qquad \boxed{z = \theta \cdot x + b}$$

If this sum is "high", we say $\hat{y} = 1$; if low, then $\hat{y} = 0$.

What about the bias b? What does that correspond to?

Can also view b as a form of w_0 for an x_0 that is always observed.

Then what happens to these formulas?

linear regression. need to make into probability.

Slide Credits: Dan Jurafsky (Stanford)

But we want a probabilistic classifier…

We need to formalize "sum is high".

We'd want a principled classifier that gives us a probability, like Naïve Bayes

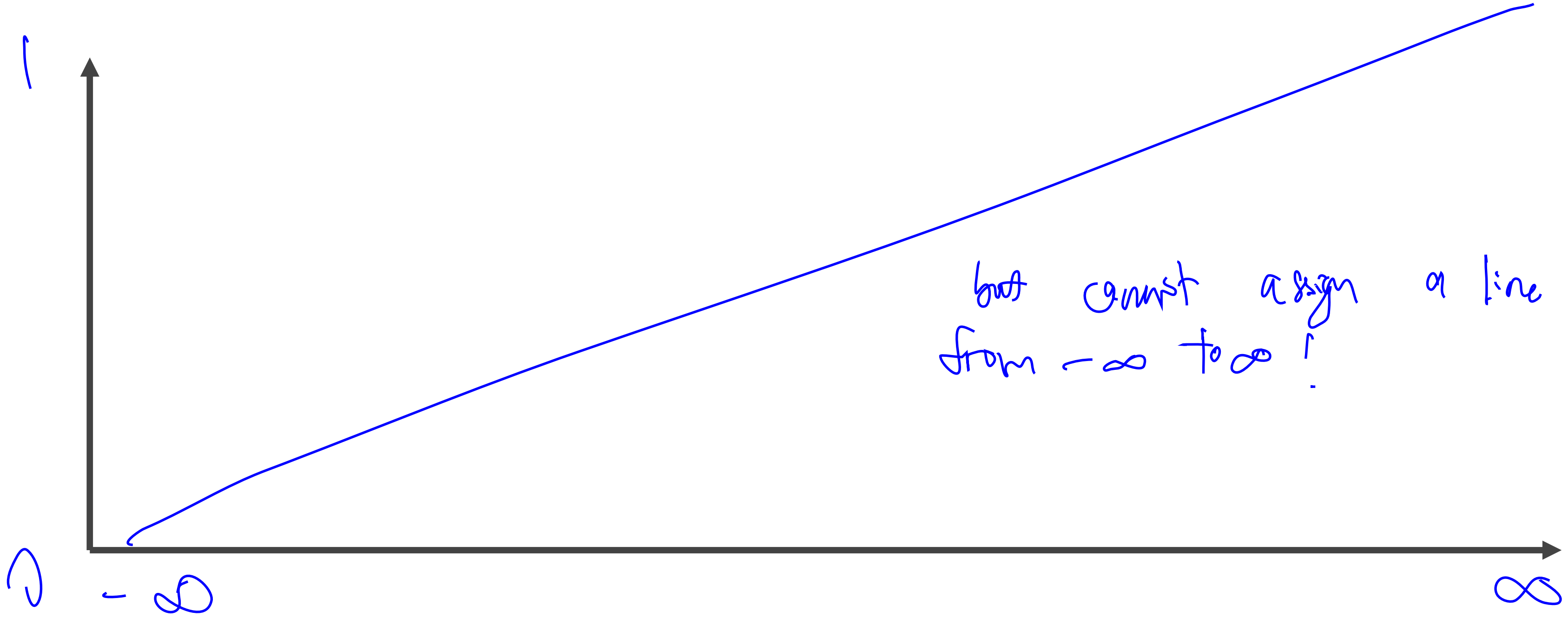
Want a model that tells us $P(y = 1|x; \theta)$ and $P(y = 0|x; \theta)$

But: z isn't a probability, but a number! How do we solve this?

Slide Credits: Dan Jurafsky (Stanford)

Map to ℝ interval [0,1]

Need a function that maps real numbers to the unit interval.

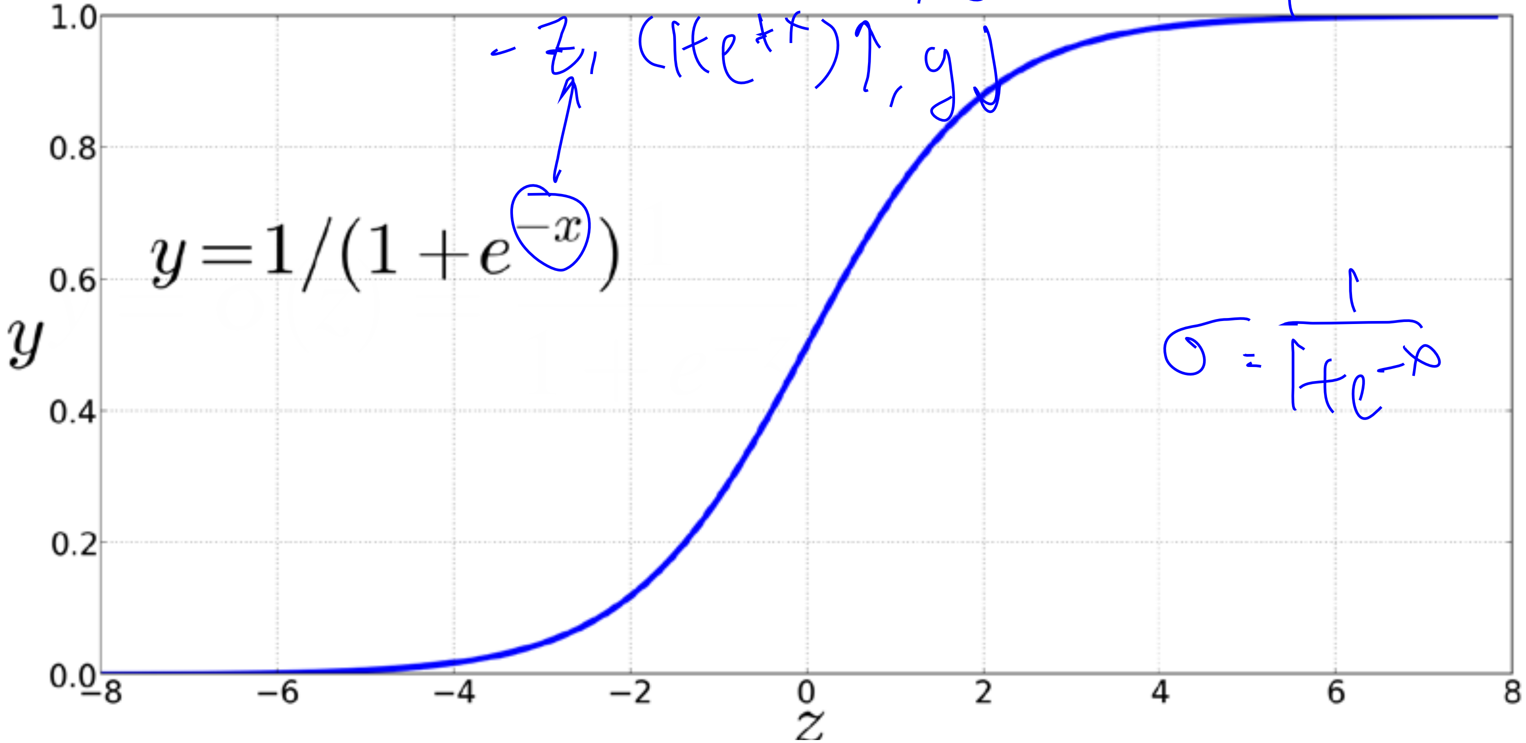


The sigmoid or logistic function

cannot be uniformly high

$+z, (1+e^{-x}) \downarrow, y \uparrow$

$-z, (1+e^{+x}) \uparrow, y \downarrow$



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

Idea of logistic regression

We'll compute our signal $z = \theta \cdot x$

conv. infinite range $\longrightarrow$ [0,1] range

We'll pass it through the sigmoid function $\sigma(\theta \cdot x)$

And we'll just treat it as a probability.

Slide Credits: Dan Jurafsky (Stanford)

Making probabilities with sigmoids

$P(y = 1) = \boxed{\sigma(\theta \cdot x)} \longrightarrow$ This returns a probability $\in [0, 1]$.

$$= \frac{1}{1 + \exp(-(\theta \cdot x))}$$

$P(y = 0) = 1 - \sigma(\theta \cdot x) \longrightarrow 1 - P(y=1)$

$$= 1 - \frac{1}{1+e^{-\theta x}}$$

$$= \frac{1+e^{-\theta x} - 1}{1+e^{-\theta x}} = \frac{e^{-\theta x}}{1+e^{-\theta x}}$$

Slide Credits: Dan Jurafsky (Stanford)

Making probabilities with sigmoids

$$P(y=1) = \sigma(\theta \cdot x)$$

$$= \frac{1}{1 + \exp(-(\theta \cdot x))}$$

$$P(y=0) = 1 - \sigma(\theta \cdot x)$$

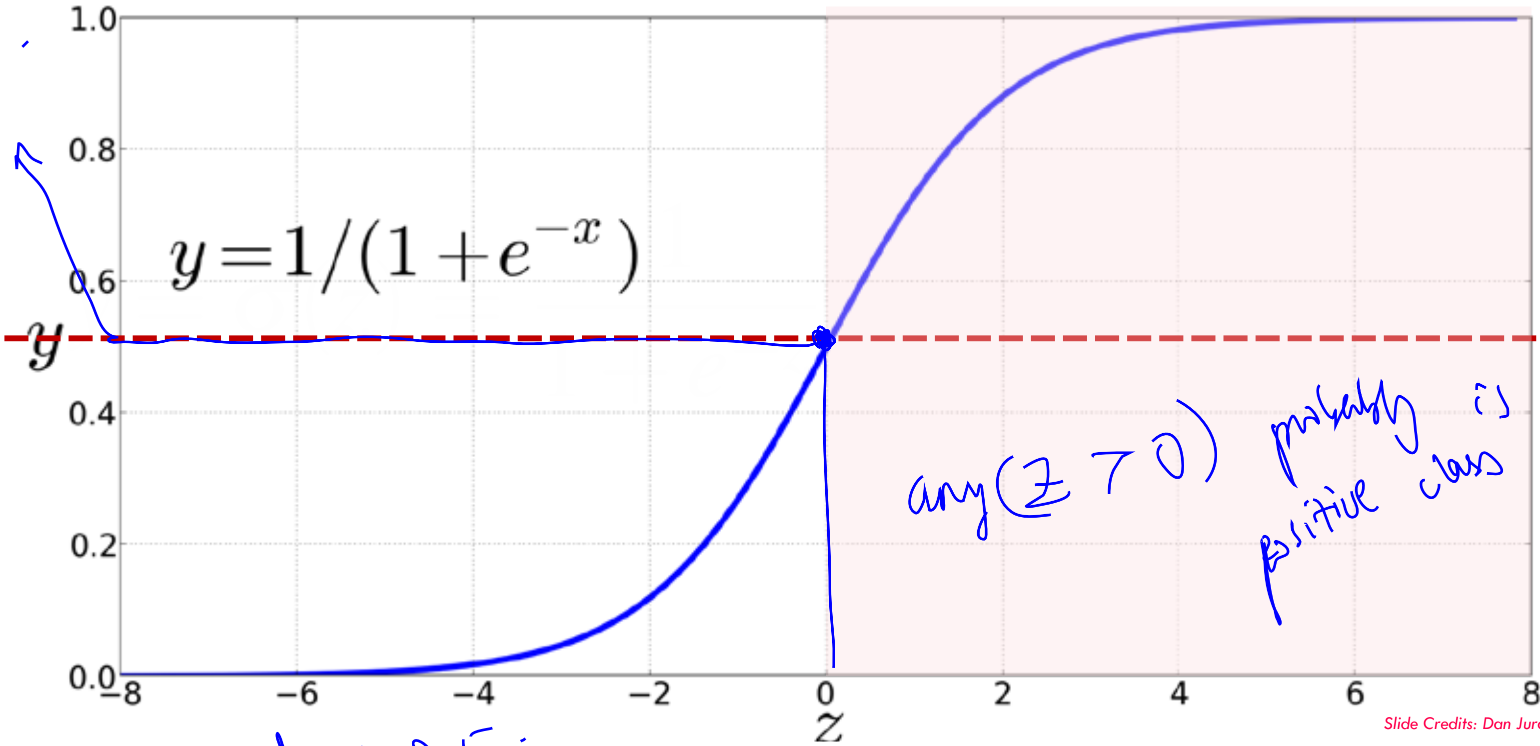
$$= 1 - \frac{1}{1 + \exp\big(-(\theta \cdot x)\big)}$$

$$= \frac{\exp\big(-(\theta \cdot x)\big)}{1 + \exp\big(-(\theta \cdot x)\big)}$$

Slide Credits: Dan Jurafsky (Stanford)

The sigmoid or logistic function

0.5 .

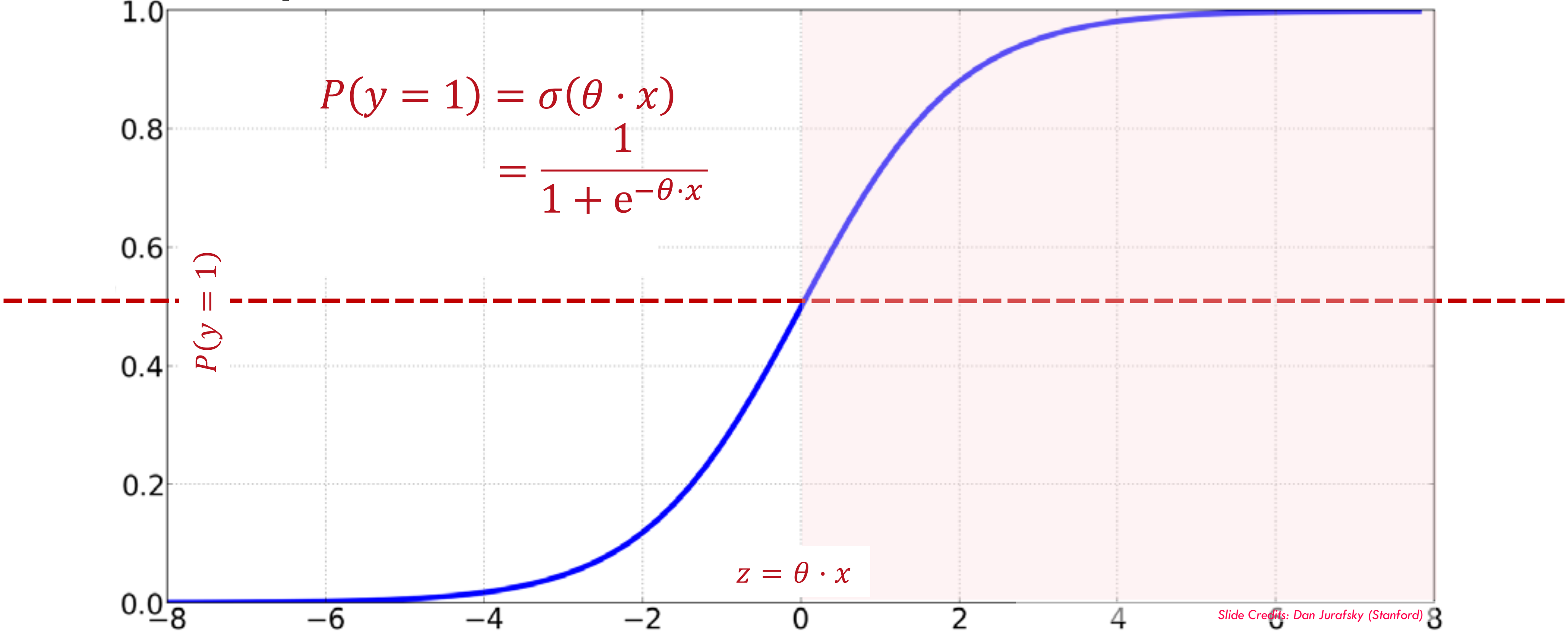


any (z > 0) probability is positive class

Slide Credits: Dan Jurafsky (Stanford)

∴ threshold = 0.5 .

The probabilistic classifier



LR Runthrough

Sentiment Analysis Case Study

Slide Credits: NUS CS3244

Movie Review: does $\hat{y} = 1$ or 0?

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Feature	Description	Value
x_1	Count of words in +ve lexicon	
x_2	Count of words in −ve lexicon	
x_3	1 if "no" in doc; 0 otherwise	
x_4	Count of 1^{st} & 2^{nd} person pronouns	
x_5	1 if "!" in doc; 0 otherwise	
x_6	Log of the word count	

It's **hokey** . There are virtually no surprises , and the writing is **second-rate** . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . *I* was overcome with the urge to get off the couch and start dancing . It sucked *me* in , and it'll do the same to *you* .

Feature	Description	Value
x_1	Count of words in +ve lexicon	3
x_2	Count of words in −ve lexicon	**2**
x_3	1 if "no" in doc; 0 otherwise	1
x_4	Count of 1st &2nd person pronouns	3
x_5	1 if "!" in doc; 0 otherwise	0
x_6	ln of the word count	ln(66) = 4.19

⚠ **Important:** LR and NB both require **feature engineering** as they do not combine primitive features together to make composite ones.

Adapted from Dan Jurafsky (Stanford)

Now factor in the weights

Feature	Description	Value (x)	Weight (θ; Assumed)	Product (θx)
x_0	Bias b	1	0.1	
x_1	Count of words in +ve lexicon	3	2.5	
x_2	Count of words in −ve lexicon	**2**	−5.0	
x_3	1 if "no" in doc; 0 otherwise	1	−1.2	
x_4	Count of 1st & 2nd person pronouns	3	0.5	
x_5	1 if "!" in doc; 0 otherwise	0	2.0	
x_6	ln of the word count	4.19	0.7	

Adapted from Dan Jurafsky (Stanford)

$$z = \sum_{i=1}^{n} \theta_i x_i = 1 \times 0.1 + 3 \times 2.5 + 2 \times (-5) + 1 \times (-1.2) + 3 \times 0.5 + 4.19 \times 0.7 =$$

LR Calculation

Feature	Description	Value (x)	Weight (θ)	Product (θx)
x_0	Bias b	1	0.1	0.1
x_1	Count of words in +ve lexicon	3	2.5	7.5
x_2	Count of words in –ve lexicon	**2**	–5.0	–10.0
x_3	1 if "no" in doc; 0 otherwise	<u>1</u>	–1.2	–1.2
x_4	Count of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in doc; 0 otherwise	0	2.0	0
x_6	ln of the word count	4.19	0.7	2.933

2.833

$P(+|x) = P(y = 1|x) = \sigma(\theta \cdot x) =$ $\frac{1}{1+e^{-2.833}}$

$P(+|x) = P(y = 0|x) = 1 - \sigma(\theta \cdot x) =$

Adapted from Dan Jurafsky (Stanford)

LR Calculation

Feature	Description	Value (x)	Weight (θ; Assumed)	Product (θx)
x_0	Bias b	1	0.1	0.1
x_1	Count of words in +ve lexicon	3	2.5	7.5
x_2	Count of words in –ve lexicon	**2**	–5.0	–10.0
x_3	1 if "no" in doc; 0 otherwise	1	–1.2	–1.2
x_4	Count of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in doc; 0 otherwise	0	2.0	0
x_6	ln of the word count	4.19	0.7	+ 2.933

$$P(+|x) = P(y=1|x) = \sigma(\theta \cdot x) =$$
$$P(-|x) = P(y=0|x) = 1 - \sigma(\theta \cdot x) =$$

Adapted from Dan Jurafsky (Stanford)

LR Calculation

Feature	Description	Value (x)	Weight (θ; Assumed)	Product (θx)
x_0	Bias b	1	0.1	0.1
x_1	Count of words in +ve lexicon	3	2.5	7.5
x_2	Count of words in –ve lexicon	**2**	–5.0	–10.0
x_3	1 if "no" in doc; 0 otherwise	1	–1.2	–1.2
x_4	Count of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in doc; 0 otherwise	0	2.0	0
x_6	ln of the word count	4.19	0.7	+ 2.933
				0.833

$P(+|x) = P(y=1|x) = \sigma(\theta \cdot x) = \quad \sigma(0.833) = 0.7$

$P(-|x) = P(y=0|x) = 1 - \sigma(\theta \cdot x) = \quad 1 - \sigma(0.833) = 1 - 0.7 = 0.3$

Adapted from Dan Jurafsky (Stanford)

LR Calculation: ***It's positive!***

Feature	Description	Value (x)	Weight (θ; Assumed)	Product (θx)
x_0	Bias b	1	0.1	0.1
x_1	Count of words in +ve lexicon	3	2.5	7.5
x_2	Count of words in −ve lexicon	2	−5.0	−10.0
x_3	1 if "no" in doc; 0 otherwise	1	−1.2	−1.2
x_4	Count of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in doc; 0 otherwise	0	2.0	0
x_6	ln of the word count	4.19	0.7	+ 2.933
				0.833

$P(+|x) = P(y=1|x) = \sigma(\theta \cdot x) = \quad \sigma(0.833) = 0.7$

$P(-|x) = P(y=0|x) = 1 - \sigma(\theta \cdot x) = \quad 1 - \sigma(0.833) = 1 - 0.7 = 0.3$

$\boldsymbol{P(+|x) > 0.5, \hat{y} = +}$ classified as positive

Adapted from Dan Jurafsky (Stanford)

Cross Entropy

What's the goal of learning?
To get the best weights.

Slide Credits: NUS CS3244

Wait, where did the θ's come from?

Supervised classification: we know the correct label y (either 0 or 1) for each x.

What the system produces is an estimate, $\hat{y}$.

We want to set θ to minimize the difference between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.

1. We need a metric: a **loss function** (also termed **cost function**); and
2. We need an **optimization algorithm** to update θ to minimize the loss.

Let's deal with #1 first.

Adapted from Dan Jurafsky (Stanford)

Our Objective Function

We want to know how far is the classifier output:

$$\hat{y} = \sigma(\theta x)$$

from the true output:

$$y \text{ [= either 0 or 1]}$$

We'll call this difference:

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

Machine Learning likes to minimize loss, hence L.

maximize utility

Slide Credit: Dan Jurafsky (Stanford)

Negative log likelihood loss

(Also termed cross-entropy loss)

A case of conditional maximum likelihood estimation:

We choose the parameters θ

That maximize the log probability of the true y labels in the training data

Given the observations x.

Slide Credit: Dan Jurafsky (Stanford)

Cross-entropy Loss

Goal: maximize probability of the correct label $p(y|x)$

Since there are 2 discrete outcomes (0 or 1), we can express the probability $p(y|x)$ from our classifier (what we want to maximize) as:

$$\begin{cases} 1-\hat{y}, & \text{if } y=0; \\ \hat{y}, & \text{if } y=1. \end{cases}$$

– negative labels

– positive labels

We can combine both cases into one formula this way:

$y=0$, $P(y|x) = 1-\hat{y}$

$y=1$, $P(y|x) = \hat{y}$

$$P(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

Slide Credit: Dan Jurafsky (Stanford)

Cross-entropy Loss

Goal: maximize $P(y|x) = \hat{y}^y(1-\hat{y})^{1-y}$.

Take logs of both sides (monotonically equivalent):

Slide Credit: Dan Jurafsky (Stanford)

Cross-entropy Loss

Goal: maximize $P(y|x) = \hat{y}^y(1-\hat{y})^{1-y}$.

Take logs of both sides (monotonically equivalent):

maximize $\log P(y|x) = \log[\hat{y}^y(1-\hat{y})^{1-y}]$

maximize $\log P(y|x) = y \log \hat{y} + (1-y)\log(1-\hat{y})$

Slide Credit: Dan Jurafsky (Stanford)

Cross-entropy Loss

maximize $\log P(y|x) = y \log \hat{y} + (1-y)\log(1-\hat{y})$

We have a maximization, but ML terminology prefers losses to minimize (as in the title). Let's flip the sign.

The result is cross-entropy loss (cross entropy between $\hat{y}$ and y):

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

Slide Credit: Dan Jurafsky (Stanford)

Does it work?

Loss should be:

- Smaller if the model estimate is close to correct
- Larger when the model is confused

For our sentiment example, let's examine it, pretending if it was positive or negative.

" It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you . "

Slide Credit: Dan Jurafsky (Stanford)

We calculated:

$$P(+|x) = \sigma(0.833) = 0.7$$
$$P(-|x) = 1 - \sigma(0.833) = 0.3$$

Feature	Description	Value (x)	Weight (θ)	Product (θx)
x_0	Bias b	1	0.1	0.1
x_1	Count of words in +ve lexicon	3	2.5	7.5
x_2	Count of words in −ve lexicon	2	−5.0	−10.0
x_3	1 if "no" in doc; 0 otherwise	1	−1.2	−1.2
x_4	Count of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in doc; 0 otherwise	0	2.0	0
x_6	ln of the word count	4.19	0.7	2.933

$$L_{ce}(\hat{y}, y) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

If the model was right ($y = 1$)

$-1(\log 0.7)$

$=$

If the model was wrong ($y = 0$)

$-\log(0.3)$

$=$

Slide Credit: Dan Jurafsky (Stanford)

Loss is bigger when the model is wrong

We calculated:

$P(+|x) = \sigma(0.833) = 0.7$

$P(-|x) = 1 - \sigma(0.833) = 0.3$

Feature	Description	Value (x)	Weight (θ)	Product (θx)
x_0	Bias b	1	0.1	0.1
x_1	Count of words in +ve lexicon	3	2.5	7.5
x_2	Count of words in −ve lexicon	2	−5.0	−10.0
x_3	1 if "no" in doc; 0 otherwise	1	−1.2	−1.2
x_4	Count of 1st & 2nd person pronouns	3	0.5	1.5
x_5	1 if "!" in doc; 0 otherwise	0	2.0	0
x_6	ln of the word count	4.19	0.7	2.933

$$L_{ce}(\hat{y}, y) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

<u>If the model was right ($y = 1$)</u>

$$= -[\log(\hat{y})]$$
$$= -[\log(0.7)]$$
$$= 0.36$$

<u>If the model was wrong ($y = 0$)</u>

$$= -[\log(1-\hat{y})]$$
$$= -[\log(0.3)]$$
$$= 1.2$$

Slide Credit: Dan Jurafsky (Stanford)

loss bigger when model wrong

Multiclass with Logistic Regression

We can generalize logistic regression for 2 or more classes:

- Generalize to Multinomial Logistic Regression

Also termed *Maximum Entropy Modeling* (MaxEnt)

- Features have separate weights for each of the k classes

- Upgrade Sigmoid to the Softmax, keeping the $\mathbb{R} \rightarrow [0,1]$ idea:

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}, 1 \leq i \leq k$$

Sums to unity

(e.g., $z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1] \rightarrow [0.055, 0.090, 0.0067, 0.1, 0.74, 0.01]$)

Slide Credit: Dan Jurafsky (Stanford)

$$\frac{1.82212}{33.2349}$$

Stochastic Gradient Descent

Now that we know how we're doing,
how can we do better?

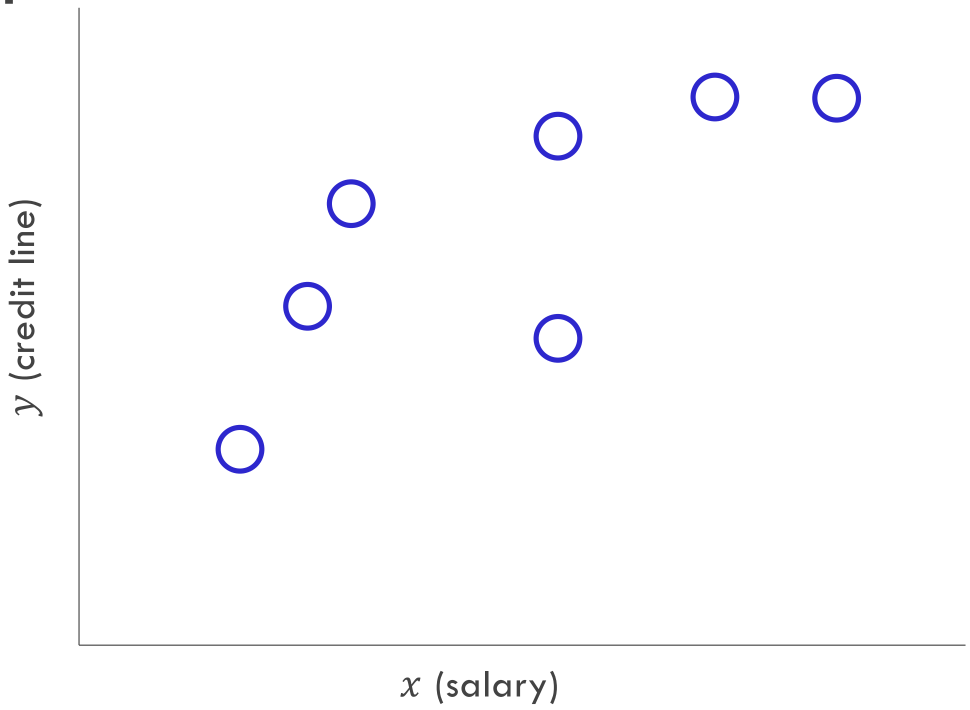
Slide Credits: NUS CS3244 and Dan Jurafsky (Stanford)

Gradient Descent

Climbing up (down)
one step at a time

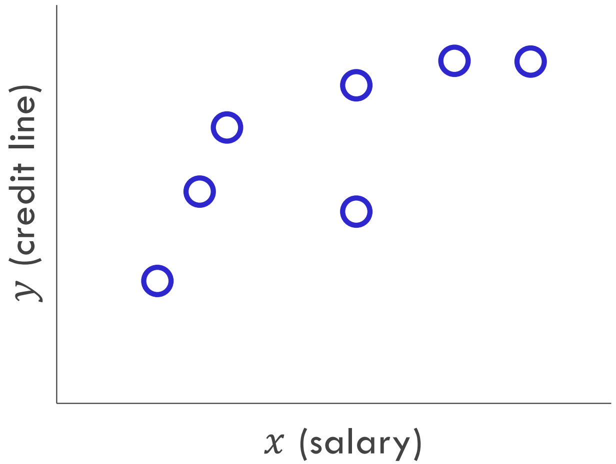
Slide Credits: NUS CS3244

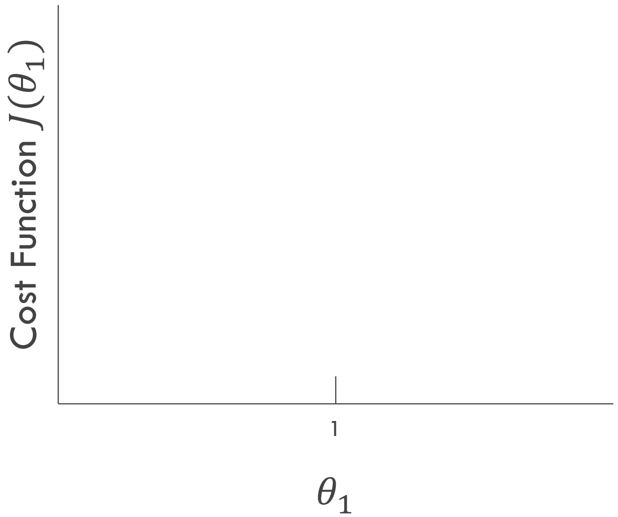
Univariate Linear Regression: Salary to predict Credit Line



Slide Credits: NUS CS3244

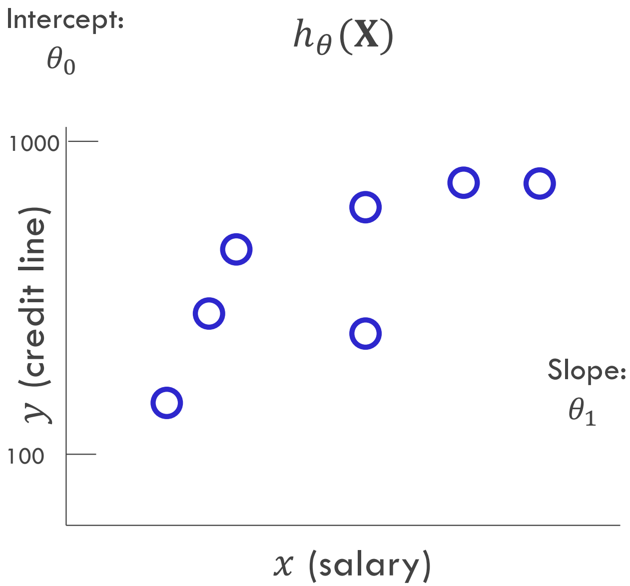
Ignoring the bias term

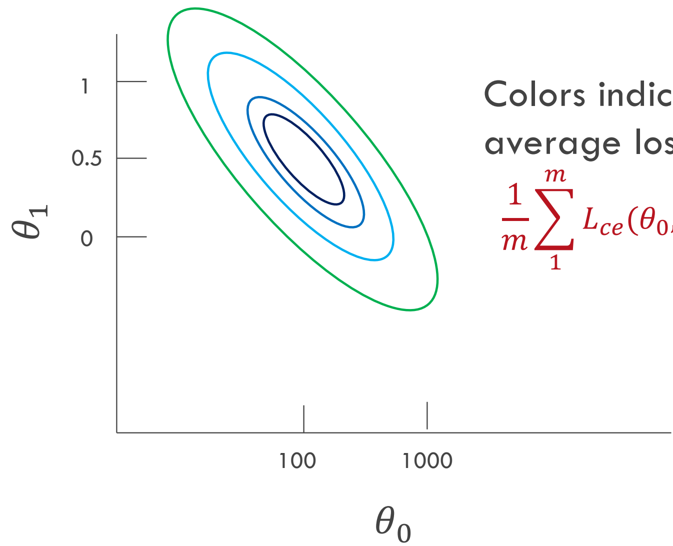




Slide Credits: NUS CS3244

Loss Function $L_{ce}(\theta_0, \theta_1)$



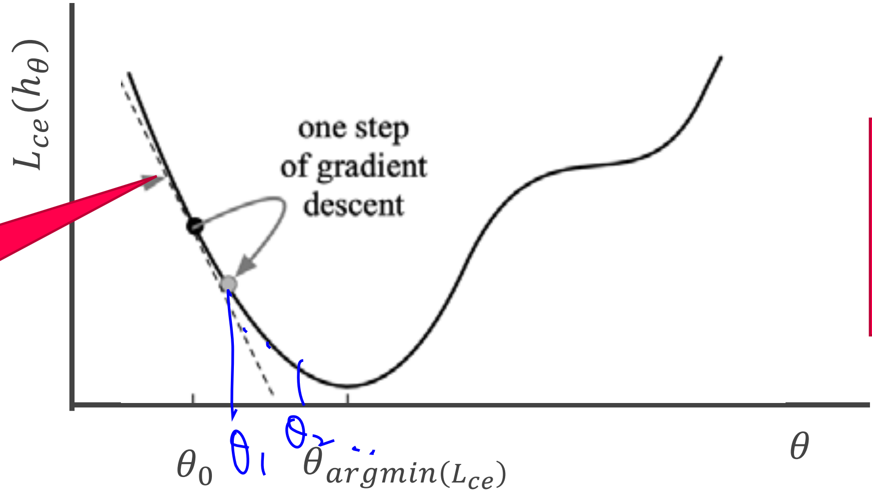


Colors indicate average loss

$$\frac{1}{m}\sum_{1}^{m} L_{ce}(\theta_0, \theta_1)$$

Slide Credits: NUS CS3244

Cross Entropy Loss Function Curve



Slope of loss at θ_0 is −ve; so we'll move a bit +ve.

N.B. for neural networks our loss function is more complex and generally not convex.

… Because L_{ce} for linear regression is a convex function of θ.

Slide Credits: NUS CS3244 and Dan Jurafsky (Stanford)

Gradients

The gradient of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

Gradient Descent: Find the gradient of the loss function at the current point and move in the opposite direction.

Slide Credit: Dan Jurafsky (Stanford)

Iterative method: gradient descent

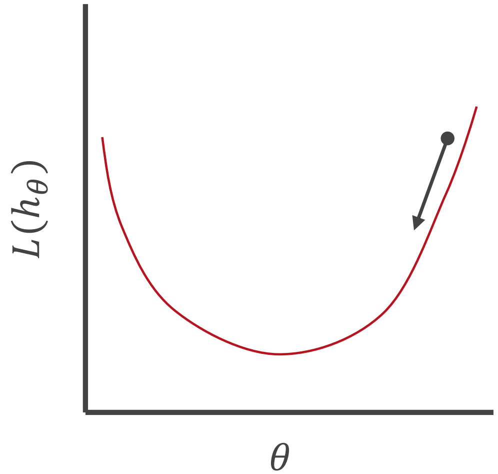
General method for nonlinear optimization

Start at $\theta(t)$; take a step along the direction with the steepest gradient.

How big a step / How fast to learn?

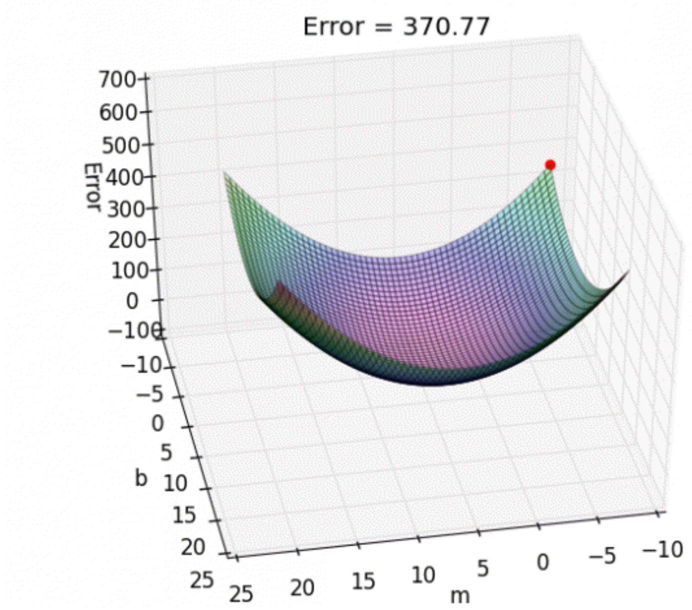
Dependent on a fixed step size η :

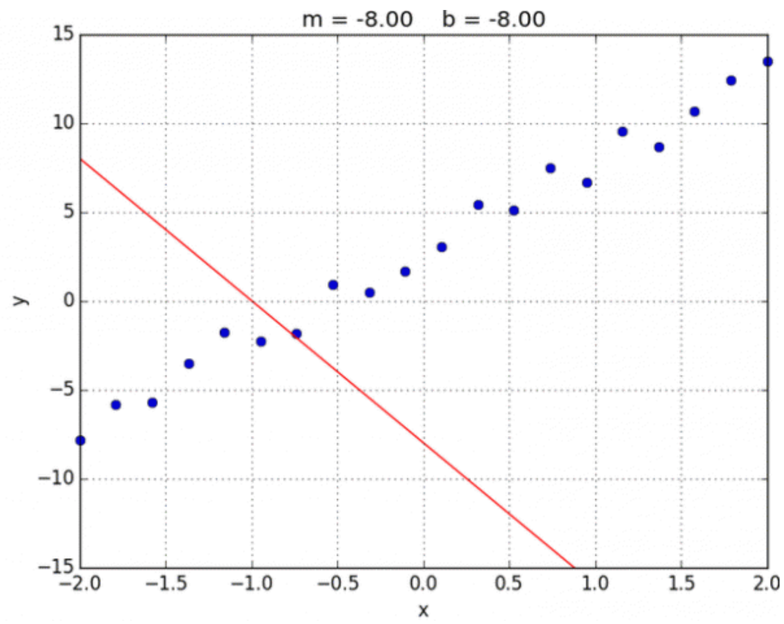
$$\theta(t+1) = \theta(t) - \eta \cdot \frac{d}{d\theta} h_\theta(\boldsymbol{x})$$



Gradient descent can minimize any *smooth* function (= needs a derivative).

Slide Credits: NUS CS3244





Credits: Alykhan Tejani's Medium Post

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

The Gradient

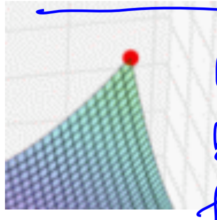
We'll represent y as $h(x;\theta)$ to make the dependence on θ more obvious:

gradient of loss fn.

$$\nabla_\theta L(h(x;\theta), y) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} L(h(x;\theta), y) \\ \vdots \\ \frac{\partial}{\partial \theta_n} L(h(x;\theta), y) \end{bmatrix}$$

$\hat{y}$

$[\sigma(wx+b) - y]x_0$



difference between true y and $\hat{y} = \sigma(wx+b)$, multiplied by x_j.

The final equation for updating θ based on the gradient is thus:

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

step size

Slide Credit: Dan Jurafsky (Stanford)

LR as a Probabilistic ML classifier

Putting it together

Click to edit Master Attribution style.

Components of a probabilistic machine learning classifier

Given m input/output pairs $(\boldsymbol{x}^{(j)}, y^{(j)})$:

1. A feature representation of the input. $\boldsymbol{x}^{(j)} = [x_1, x_2, \ldots, x_i, \ldots, x_n]$. The ith feature is denoted x_i, or more completely $x_i^{(j)}$, sometimes $f_i(\boldsymbol{x})$.

2. A classification function that computes $\hat{y}$, the estimated class, via $p(y|\boldsymbol{x})$, like the sigmoid or softmax functions.

3. An objective function for learning, like cross-entropy loss.

4. An algorithm for optimizing the objective function: stochastic gradient descent.

Slide Credits: Dan Jurafsky (Stanford)

Logistic regression algorithm

1. Initialize the weights at $t = 0$ to $\theta(0)$ → doesn't matter can start anywhere in the hyperbole, gradient descent later.
2. Do
3. Compute the gradient

gradient loss

$$\nabla(t) = \boxed{\nabla L_{ce}(\theta(t))} = -\frac{1}{m}\sum_{j=1}^{m}\frac{y^{(j)}x^{(j)}}{1 + e^{y^{(j)}\theta(t)\cdot x^{(j)}}}$$

sigmoid

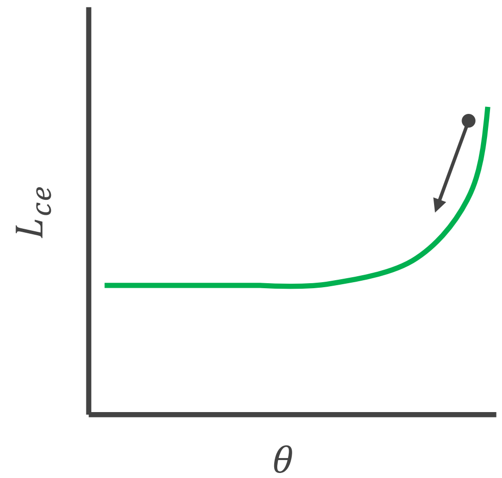
4. // Move in the direction $v(t) = -\nabla(t)$
   Update the weights $\theta(t+1) = \theta(t) - \alpha\nabla L_{ce}$ ← move in direction opposite gradient
5. Continue to next iteration, until it is time to stop.
6. Return the final weights θ^*

Slide Credits: NUS CS3244

Termination Condition

When to stop?

Natural choice: *gradient* < *threshold*



But lots of flat regions in most spaces:

Instead, use criteria:

1. error change is small and/or;
2. error is small;
3. maximum number of iterations is reached.

Slide Credits: NUS CS3244

Mini-batch training

Stochastic gradient descent chooses a single random example at a time. That can result in choppy movements

More common to compute gradient over batches of training instances.

mini batches = easily vectorized → computational efficiency.

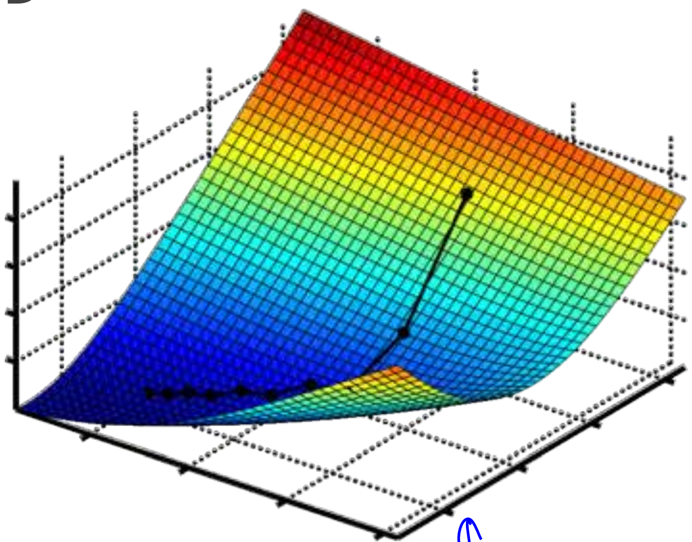
- **Batch training:** entire dataset
- **Mini-batch training:** m examples (512, or 1024)

Mini batch gradient: $\frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}\left[\sigma(w\cdot x^{(i)}+b) - y^{(i)}\right]x_j^{(i)}$

Slide Credit: Dan Jurafsky (Stanford)

GD vs. SGD on $m = 10$

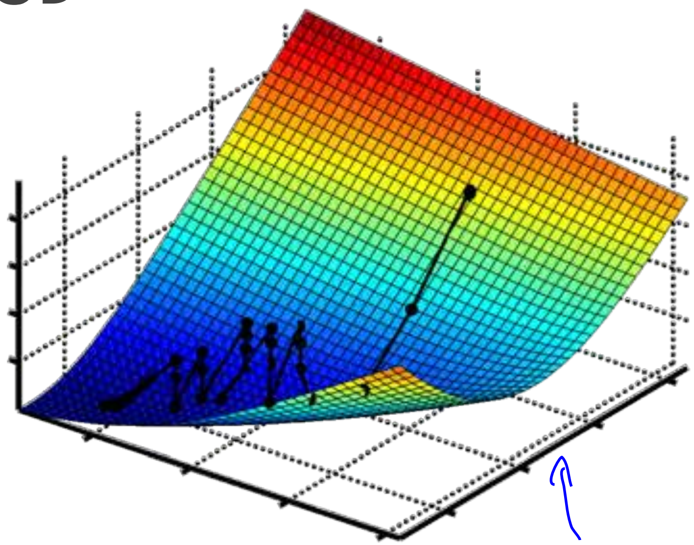
GD



10 steps

accurate but expensive

SGD



cheap

30 steps

Slide Credits: NUS CS3244

Regularization

Sample Lecture Title

Click to edit Master Attribution style.

Overfitting

A model that perfectly matches the training data often has a problem.

It may **overfit** to the data, modeling noise

- A random word that perfectly predicts *y* (it happens to only occur in one class) will get a very high weight.
- Failing to generalize to a test set without this word.

A good model should be able to **generalize**

Slide Credit: Dan Jurafsky (Stanford)

Overfitting

What are some good n-gram features?

Your answers:

\+ This movie drew me in, and it'll do the same to you.

– I can't tell you how much I hated this movie. It sucked.

Slide Credit: Dan Jurafsky (Stanford)

Overfitting

What are some good n-gram features?

+ This movie drew me in, and it'll do the same to you.

– I can't tell you how much I hated this movie. It sucked.

Your answers:

How do you feel about these?

x_1 = “the same to you”

x_2 = “tell you how much”

4-gram features that are very specific that just “memorize” training set might cause problems.

Slide Credit: Dan Jurafsky (Stanford)

Overfitting

4-gram model on tiny data will just memorize the data

- 100% accuracy on the training set

But it will be surprised by the novel 4-grams in the test data

- Low accuracy on test set

Models that are too powerful can overfit the data

- Fitting the details of the training data so exactly that the model doesn't generalize well to the test set

Slide Credit: Dan Jurafsky (Stanford)

Regularization

A solution for overfitting

Add an overfit penalty $\Omega(\theta)$ to the loss function:

$$\hat{\theta} = argmax \sum_{j=1}^{m} \log P\left(y^{(j)} \middle| x^{(j)}\right) - \Omega(\theta)$$

Idea: choose a $\Omega(\theta)$ that penalizes large weights.

Fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

Slide Credit: Dan Jurafsky (Stanford)

Regularization Variants

L2 Regularization (= Ridge Regression)

$$\hat{\theta} = argmax \sum_{j=1}^{m} \log P\left(y^{(j)} \middle| x^{(j)}\right) - \sum_{i=1}^{n} \theta_i^2$$

L1 Regularization (= Lasso Regression)

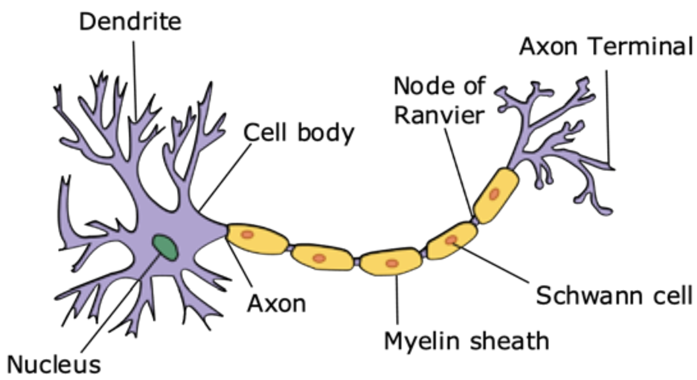
$$\hat{\theta} = argmax \sum_{j=1}^{m} \log P\left(y^{(j)} \middle| x^{(j)}\right) - \sum_{i=1}^{n} |\theta_i|$$

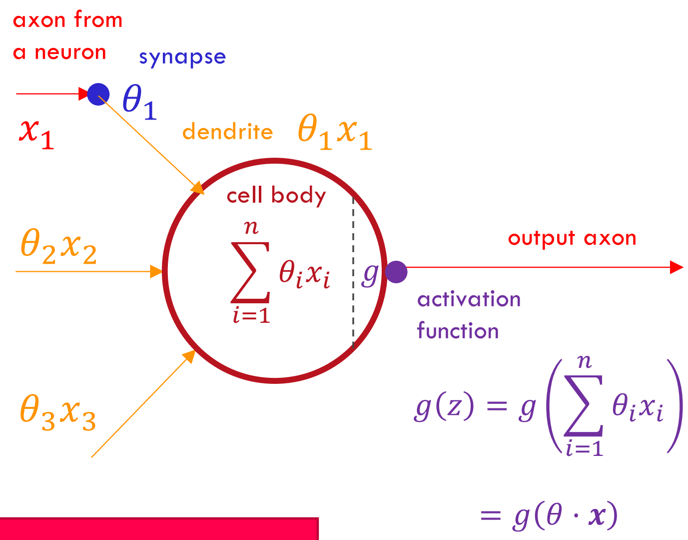
XOR

Motivating Neural Networks

Slide Credits: CS3244

Biological Inspiration





Looks familiar? A neuron is a LR unit!

Diagram credits: Dhp1080 - Own work, CC BY-SA 3.0 via Wikimedia Commons.

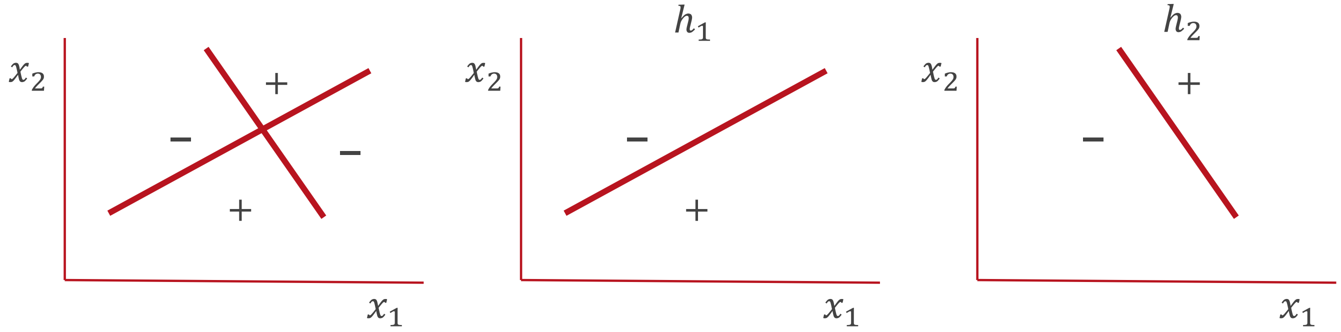
Stacked Logistic Regression

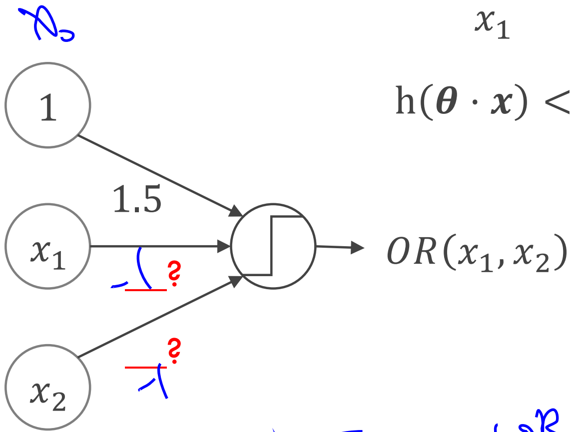
Logistic Regression can represent any linear combination of the features, mapping them non-linearly to a probability.

But what if we want to represent some non-linear relationship between features? Sorry, can't do it.

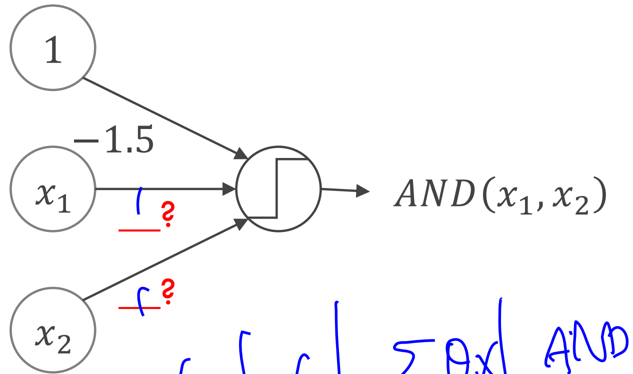
Idea: Use LR to create such non-linear features. Feed LR outputs from other LRs.

Combining Linear Classifiers





$\mathrm{h}(\boldsymbol{\theta} \cdot \boldsymbol{x}) <> 0.5 \quad y = \pm 1$

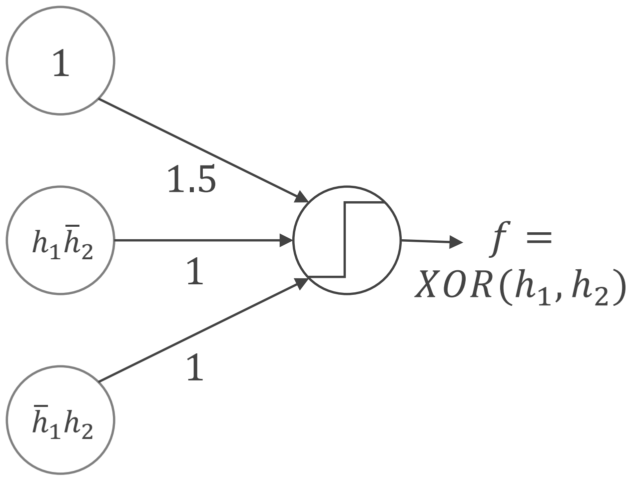


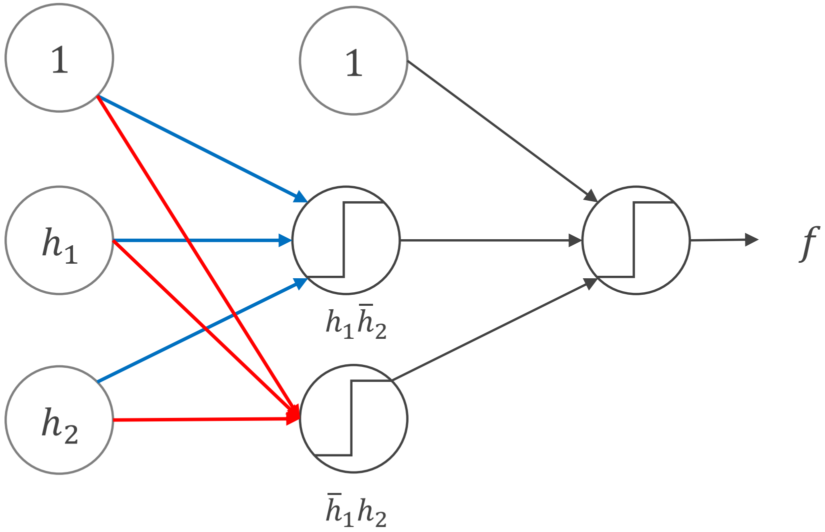
x_1	x_2	$\Sigma\theta x$	$OR(x_1, x_2)$
0	0	1.5	1
0	1	0.5	1
1	0	0.5	1
1	1	−0.5	−1

x_1	x_2	$\Sigma\theta x$	$AND(x_1, x_2)$
0	0	−1.5	−1
0	1	−0.5	−1
1	0	−0.5	−1
1	1	0.5	1

Slide Credits: CS3244

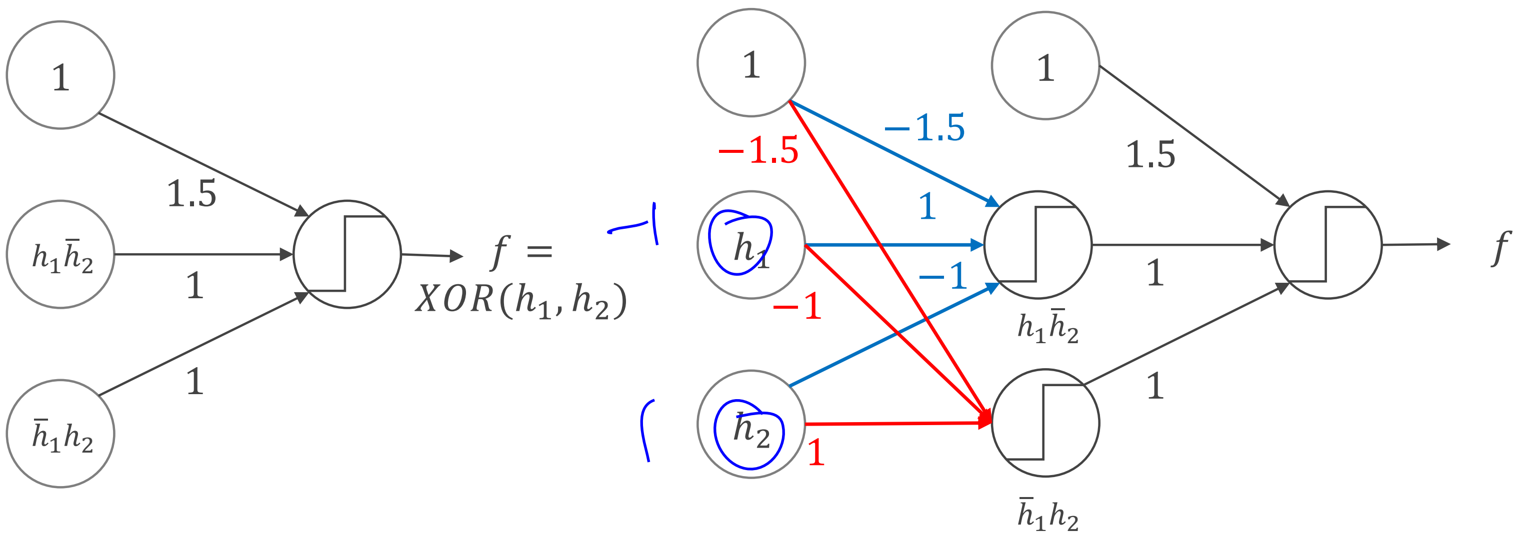
Creating Layers





Slide Credits: CS3244

Creating Layers

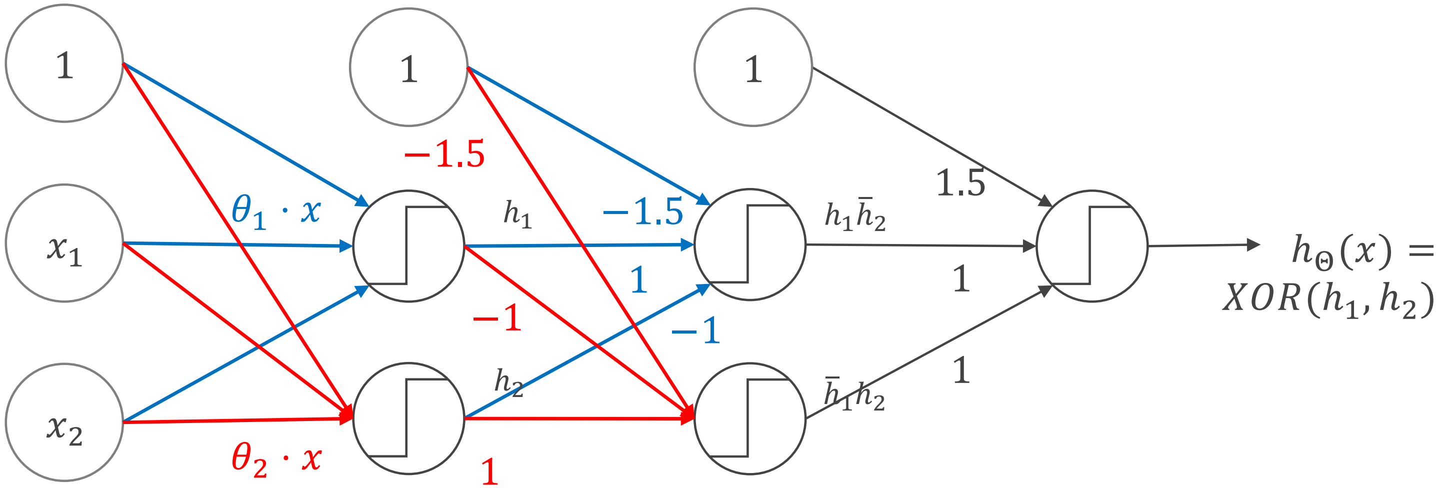


Slide Credits: CS3244

NN = Stacked Linear Regression

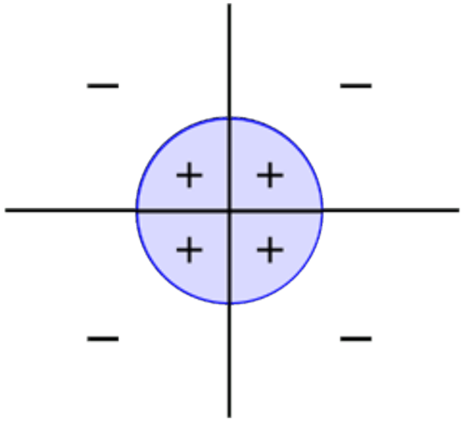
3 layers

"feedforward" ≡ no loops

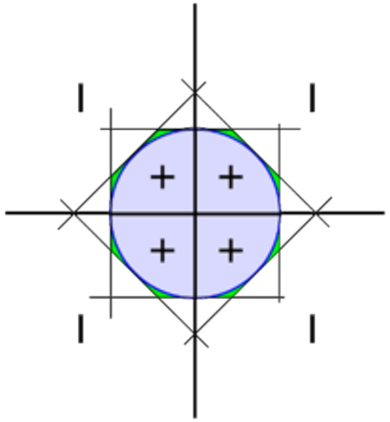


Slide Credits: CS3244

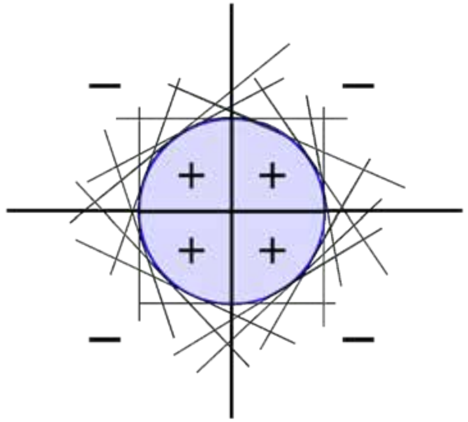
A Powerful Model



Target



8 perceptrons

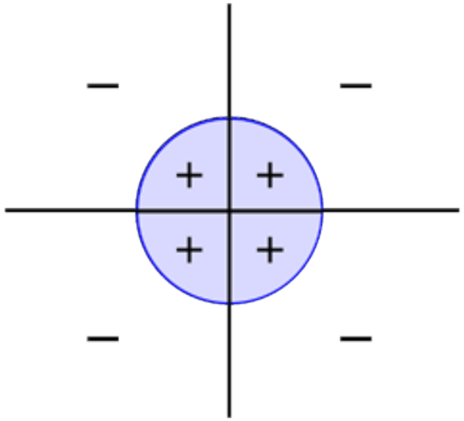


16 perceptrons

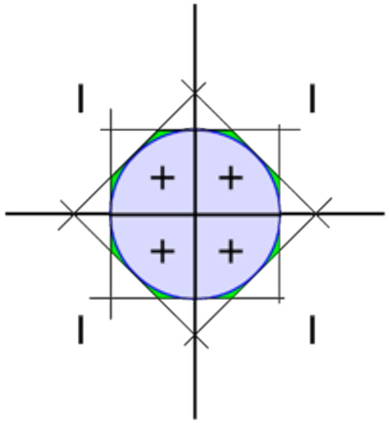
2 red flags:

Slide Credits: CS3244

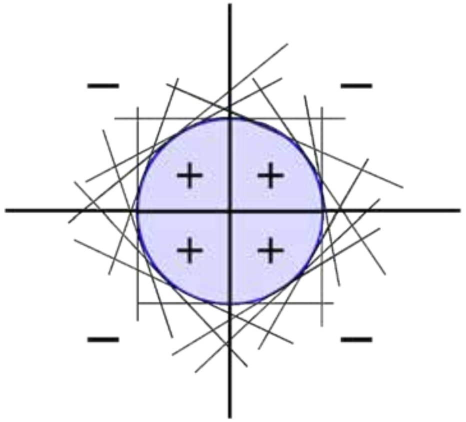
A Powerful Model



Target



8 perceptrons



16 perceptrons

2 red flags:

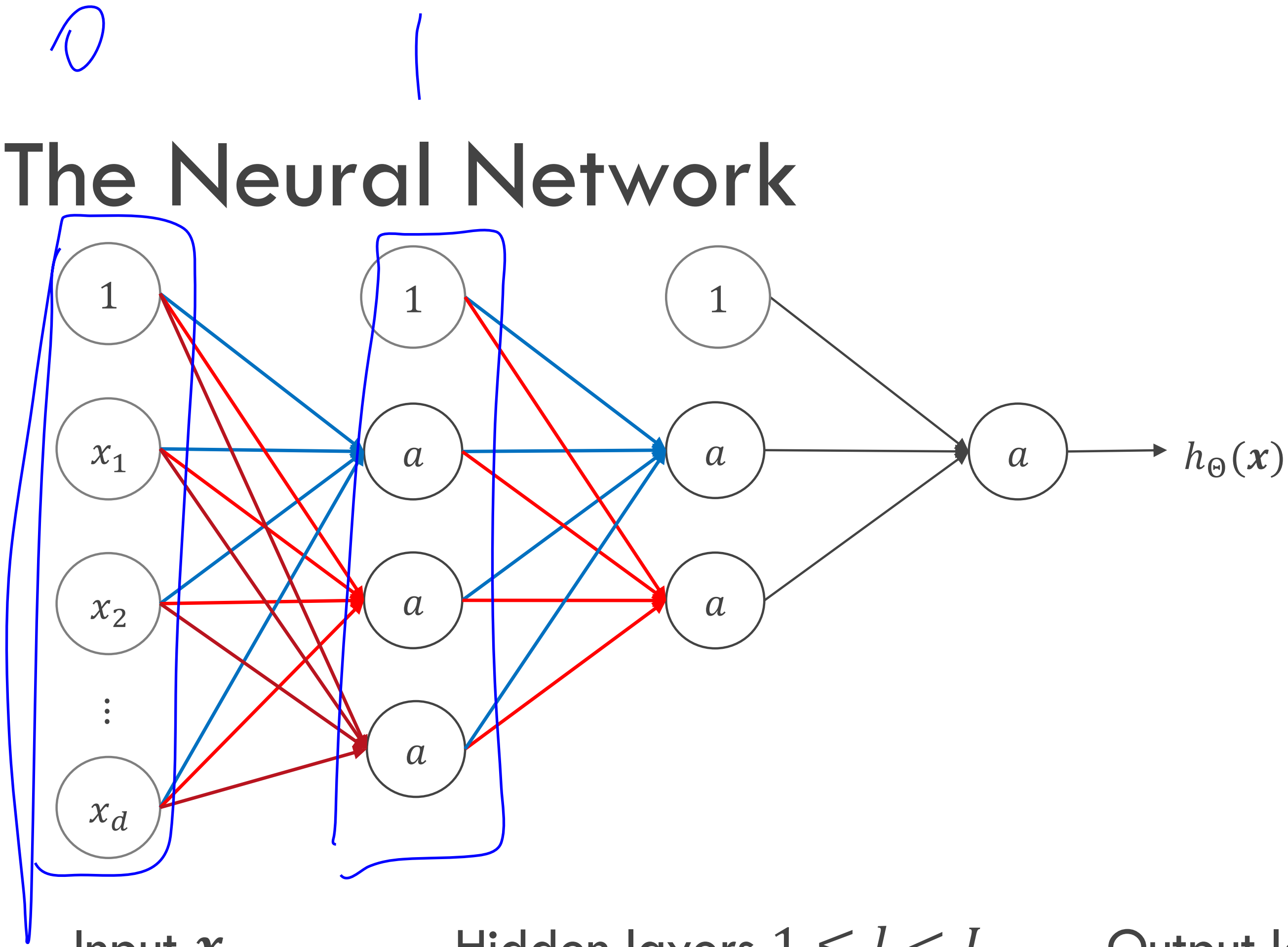
for generalization and
for optimization

Slide Credits: CS3244

Neural Networks

Slide Credits: NUS CS3244

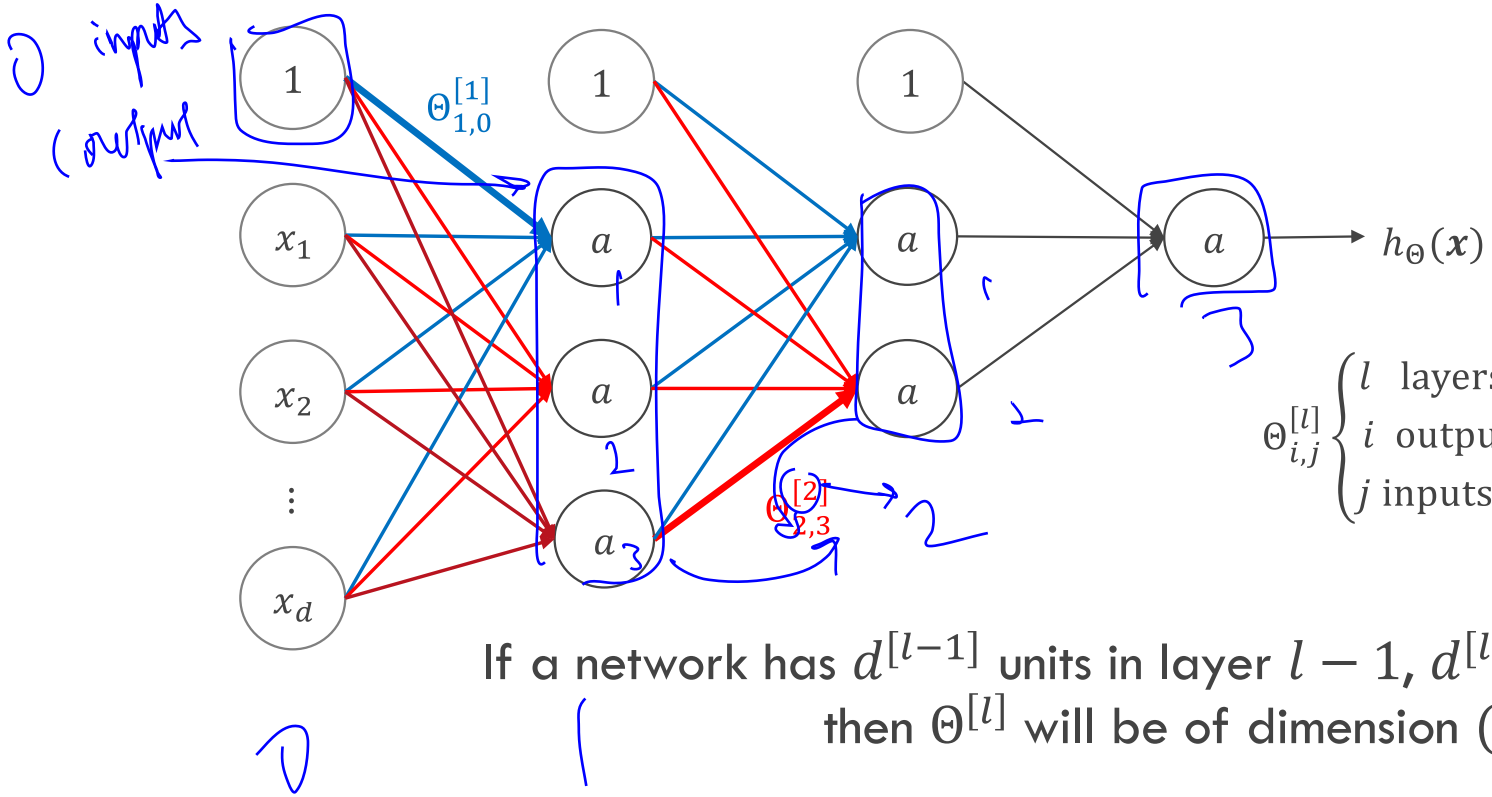
The Neural Network



Input $\boldsymbol{x}$ Hidden layers $1 \leq l < L$ Output layer $l = L$

Slide Credits: CS3244

NN Indices

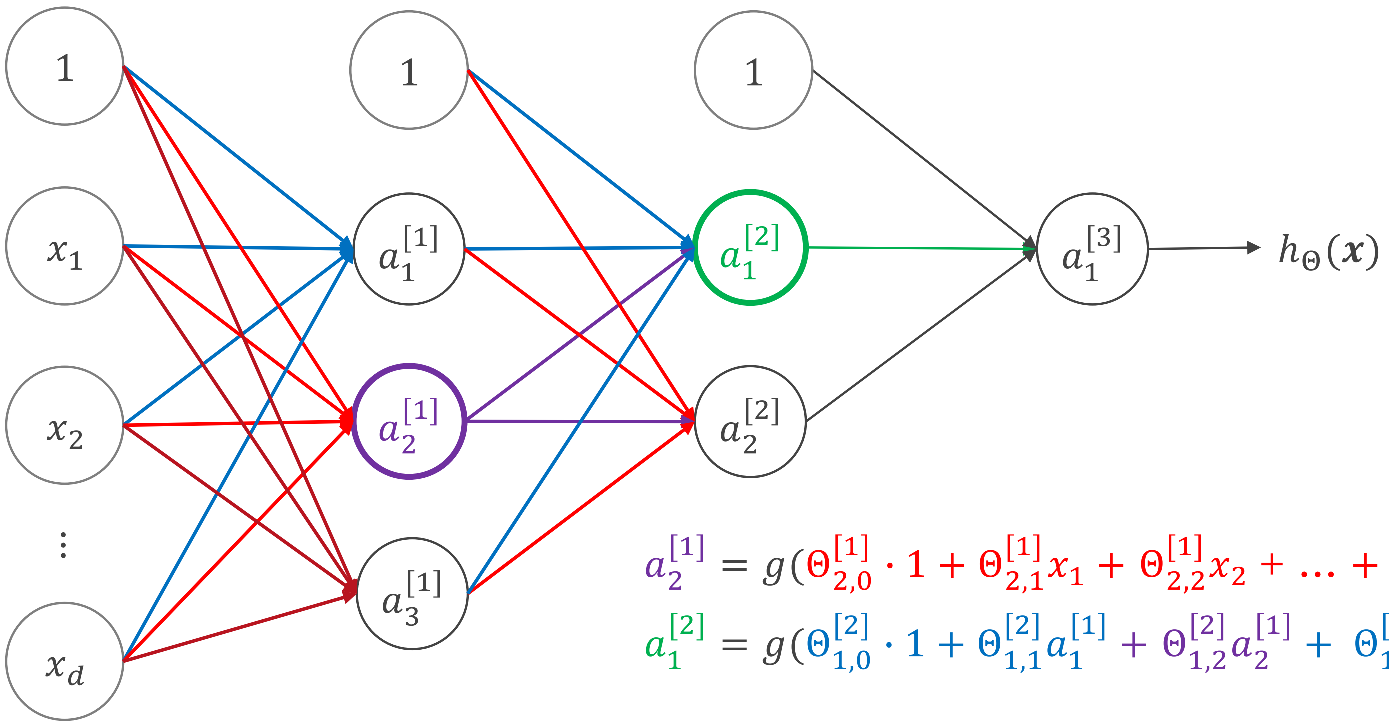


$$\Theta_{i,j}^{[l]} \begin{cases} l \ \text{ layers}\,,\, 1 \ \leq l \leq L. \\ i \ \text{ outputs}\,,\, 1 \leq i \leq d^{[l]} \\ j \ \text{inputs}\,,\, 0 \leq j \leq d^{[l-1]} \end{cases}$$

If a network has $d^{[l-1]}$ units in layer $l-1$, $d^{[l]}$ units at layer l, then $\Theta^{[l]}$ will be of dimension $(d^{[l-1]}+1)\times d^{[l]}$.

Slide Credits: CS3244

NN Activations



$$a_2^{[1]} = g(\Theta_{2,0}^{[1]} \cdot 1 + \Theta_{2,1}^{[1]} x_1 + \Theta_{2,2}^{[1]} x_2 + \ldots + \Theta_{2,d}^{[1]} x_d)$$

$$a_1^{[2]} = g(\Theta_{1,0}^{[2]} \cdot 1 + \Theta_{1,1}^{[2]} a_1^{[1]} + \Theta_{1,2}^{[2]} a_2^{[1]} + \Theta_{1,3}^{[2]} a_3^{[1]})$$

Slide Credits: CS3244

How the Network Operates

$$\Theta_{i,j}^{[l]} \begin{cases} 1 \leq l \leq L & \text{layers} \\ 1 \leq i \leq d^{[l]} & \text{outputs} \\ 0 \leq j \leq d^{[l-1]} & \text{inputs} \end{cases}$$

$$x_i^{[l]} = a_i^{[l]} = g\left(z_i^{[l]}\right) = g\left(\sum_{j=0}^{d^{[l-1]}} \Theta_{i,j}^{[l]} x_j^{[l-1]}\right)$$

$$= g((\Theta_i^{[l]}) \cdot \ \boldsymbol{x}^{[l-1]})$$

Feedforward: Apply $\boldsymbol{x}$ to $x_1^{[0]} \ldots x_{d^{[0]}}^{[0]} \rightarrow x_1^{[L]} = h(\boldsymbol{x})$

Slide Credits: CS3244

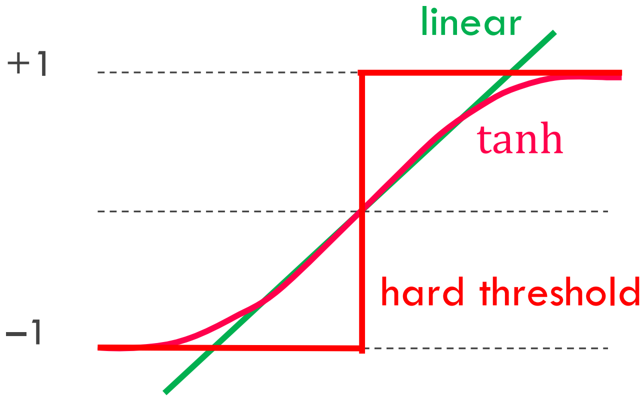
Mapped sigmoid from [0,1] to [-1,1].

Activation Functions

Final Layer

Replace sigmoid [0,1] with hyperbolic tangent [−1,1] if we want −ve and +ve classes.

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

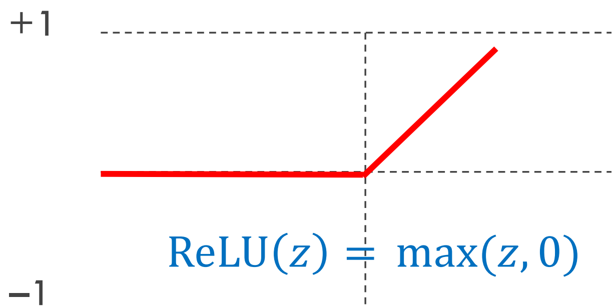


Intermediate Layer(s):

Outputs are features for an upstream unit.

Don't need a probabilistic interpretation, since features can have any value; just need to be non-linear. does not activate all the neurons at the same time

Use the Rectified Linear Unit (ReLU):



$$\mathrm{ReLU}(z) = \max(z, 0)$$

if result zero, neuron does not get activated.

NN Summary

Intermediate NN layers create real valued features.

- Use a simpler activation function, the ReLU.

With NN being complex, our loss function is no longer convex with one minimum.

- Then SGD finds a local minimum, rather than a global.
- Good initialization is more important.

The overfitting issue is more extreme in NNs due to their complexity.

- Regularization more important
- One method you'll hear about: **dropout**.