Introduction to Machine Learning

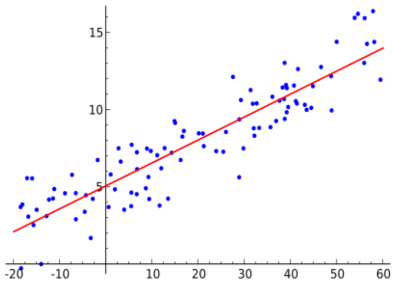
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley

How do we formulate learning problems?

Different types of learning problems

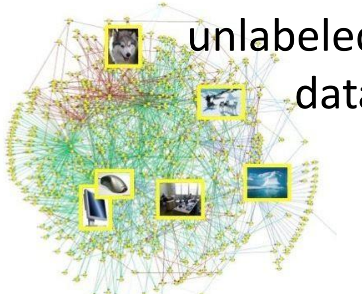


predict y from x

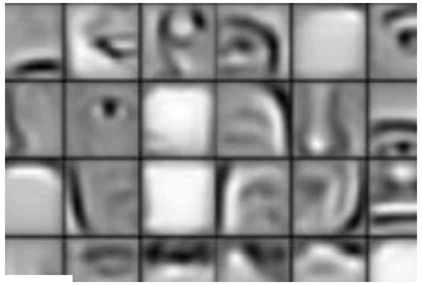
$f_\theta(x) = y$

[object label]

supervised learning

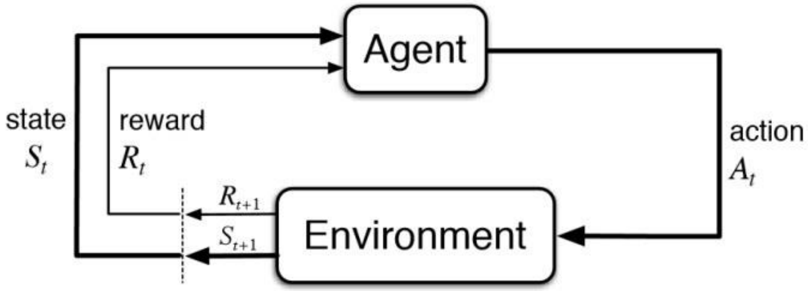


unlabeled data



representation

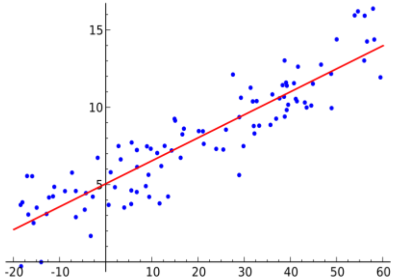
unsupervised learning



reinforcement learning

Supervised learning

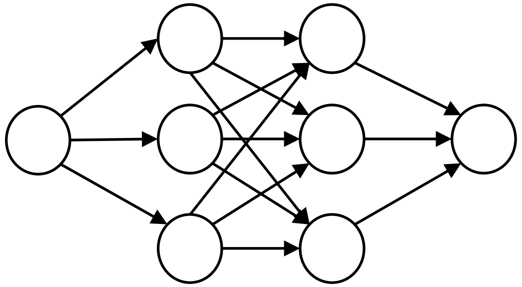
Given: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ learn $f_\theta(x) \approx y$



predict y from x

$f_\theta(x) = y$

[object label]



Questions to answer:

how do we represent $f_\theta(x)$?

$$f_\theta(x) = \theta_1 x_1 + \theta_2 x_2 + \theta_3$$

$$f_\theta(x) = \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

how do we measure difference between $f_\theta(x_i)$ and y_i?

$$||f_\theta(x_i) - y_i||^2$$

probability?

$$\delta(f_\theta(x_i) \neq y_i)$$

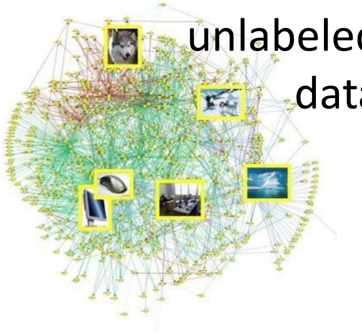
how do we find the best setting of θ?

gradient descent

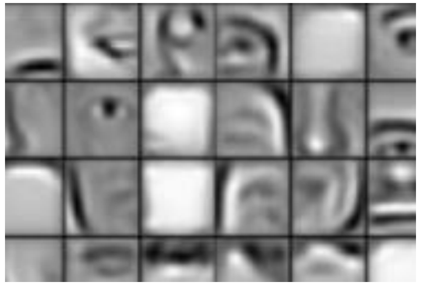
random search

least squares

Unsupervised learning



unlabeled data



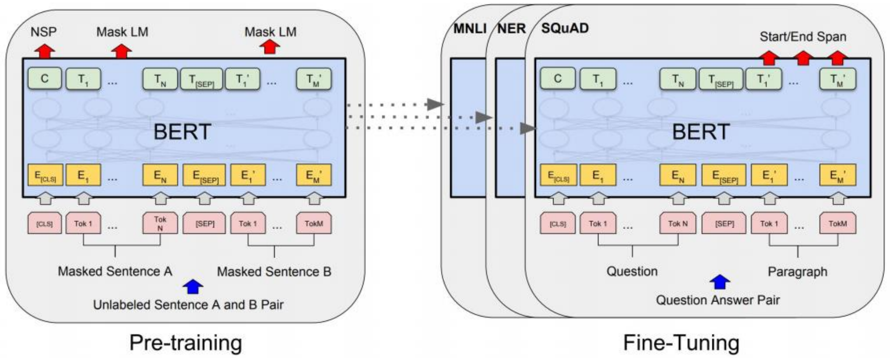
representation

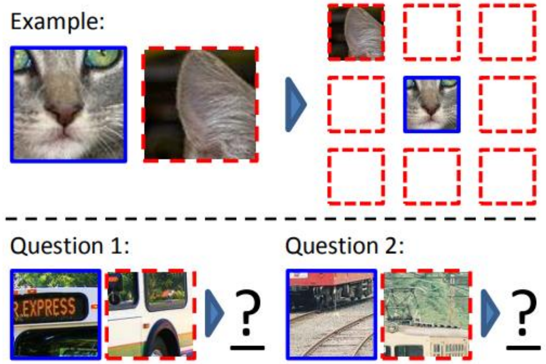
what does that mean?

generative modeling:

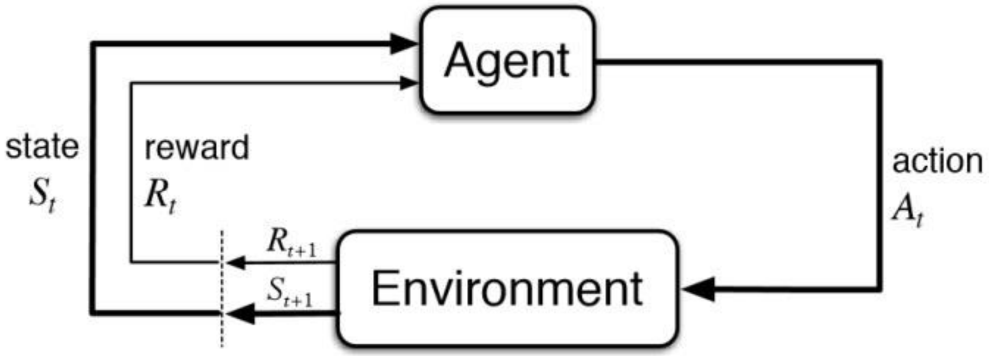
GANs
VAEs
pixel RNN, etc.

self-supervised representation learning:





Reinforcement learning



choose $f_\theta(s_t) = a_t$

to maximize $\sum_{t=1}^{H} r(s_t, a_t)$

actually subsumes (generalizes) supervised learning!

supervised learning: get $f_\theta(x_i)$ to match y_i

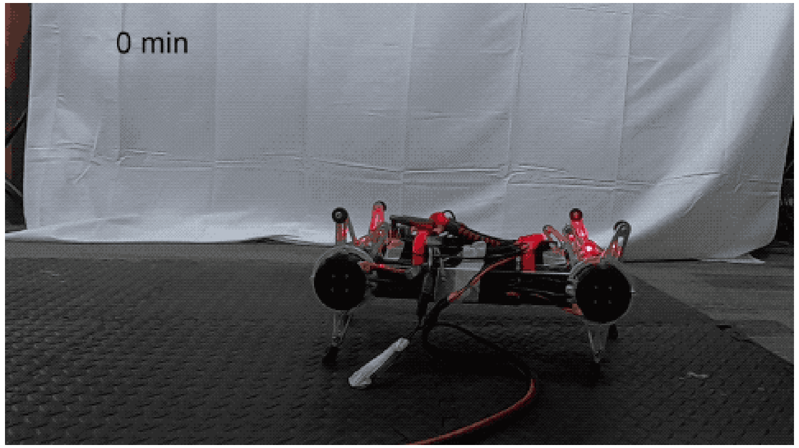
reinforcement learning: get $f_\theta(s_t)$ to maximize reward (could be anything)

Actions: muscle contractions
Observations: sight, smell
Rewards: food

Actions: motor current or torque
Observations: camera images
Rewards: task success measure (e.g., running speed)

Actions: what to purchase
Observations: inventory levels
Rewards: profit

Reinforcement learning



Haarnoja et al., 2019

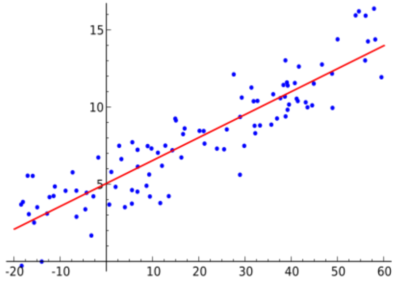
But many other application areas too!

- Education (recommend which topic to study next)
- YouTube recommendations!
- Ad placement
- Healthcare (recommending treatments)

Let’s start with supervised learning…

Supervised learning

Given: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ learn $f_\theta(x) \approx y$



predict y from x

$f_\theta(x) = y$

[object label]

The overwhelming majority of machine learning that is used in industry is supervised learning

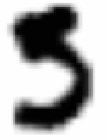
- Encompasses all prediction/recognition models trained from ground truth data
- Multi-billion $/year industry!
- Simple basic principles

Example supervised learning problems

Given: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ learn $f_\theta(x) \approx y$

Predict…	**Based on…**
category of object	image
sentence in French	sentence in English
presence of disease	X-ray image
text of a phrase	audio utterance
y	x

Prediction is difficult

		0	1	2	3	4	5	6	7	8	9
	5?	0%	0%	0%	0%	0%	90%	8%	0%	2%	0%
	9?	4%	0%	0%	0%	11%	0%	4%	0%	6%	75%
	3?	5%	0%	0%	40%	0%	30%	20%	0%	5%	0%
	4?	5%	0%	0%	0%	50%	0%	3%	0%	2%	40%
	0?	70%	0%	20%	0%	0%	0%	0%	0%	10%	0%

Predicting probabilities

Often makes more sense than predicting discrete labels

We'll see later why it is also **easier** to learn, due to smoothness

- Intuitively, we can't change a discrete label "a tiny bit," it's all or nothing
- But we **can** change a probability "a tiny bit"

Given: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$

learn $\cancel{f_\theta(x) \approx y}$ $p_\theta(y|x)$

$p_\theta(y|x)$

Conditional probabilities

x random variable representing the **input**

why is it a **random** variable?

y random variable representing the **output**

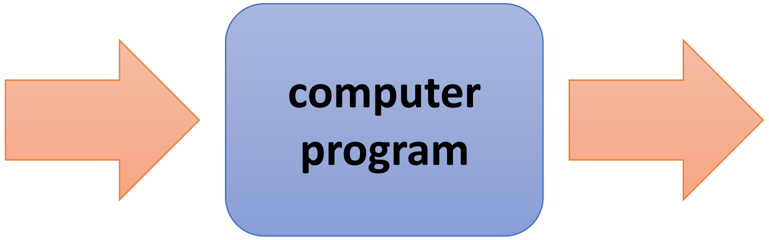
$$p(x, y) = p(x)p(y|x)$$

chain rule

$$p(y|x) = \frac{p(x, y)}{p(x)}$$

definition of conditional probability

How do we represent it?



~~[object label]~~

[object probability]

0	1	2	3	4	5	6	7	8	9
0%	0%	0%	0%	0%	90%	8%	0%	2%	0%

10 possible labels, output 10 numbers

(that are positive and sum to 1.0)

How do we represent it?

computer program

~~[object label]~~

[object probability]

~~if below_line(x) then:~~
~~else:~~

how about:

$$p(y = \text{dog}|x) = x^T \theta_{\text{dog}}$$

$$p(y = \text{cat}|x) = x^T \theta_{\text{cat}}$$

$$\vec{\theta} = \{\theta_{\text{dog}}, \theta_{\text{cat}}\}$$

(that are positive and sum to 1.0)

How do we represent it?

→ **computer program** →

~~[object label]~~

[object probability]

$$\text{if below\_line(x) then:}$$
$$\text{else:}$$

why **any** function?

how about:

$$f_{\text{dog}}(x) = x^T\theta_{\text{dog}}$$

$$f_{\text{cat}}(x) = x^T\theta_{\text{cat}}$$

$$\vec{\theta} = \{\theta_{\text{dog}}, \theta_{\text{cat}}\}$$

$$p(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x))$$

could be any (ideally one to one & onto) function that takes these inputs and outputs probabilities that are **positive** and **sum to 1**

How do we represent it?

how about:

$$f_{\text{dog}}(x) = x^T \theta_{\text{dog}}$$

$$f_{\text{cat}}(x) = x^T \theta_{\text{cat}}$$

$$\vec{\theta} = \{\theta_{\text{dog}}, \theta_{\text{cat}}\}$$

$$p(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x))$$

could be any (ideally one to one & onto) function that takes these inputs and outputs probabilities that are **positive** and **sum to 1**

how to make a number z positive?

$$z^2 \quad |z| \quad \max(0, z) \quad \exp(z)$$

especially convenient because it's one to one & onto maps entire real number line to entire set of positive reals (but don't overthink it, any one of these would work)

how to make a bunch of numbers sum to 1?

$$\frac{z_1}{z_1 + z_2} \qquad \frac{z_1}{\sum_{i=1}^n z_i}$$

How do we represent it?

how about:

$$f_{\text{dog}}(x) = x^T \theta_{\text{dog}} \qquad p(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x))$$

$$f_{\text{cat}}(x) = x^T \theta_{\text{cat}}$$

$$\vec{\theta} = \{\theta_{\text{dog}}, \theta_{\text{cat}}\}$$

makes it positive

makes it sum to 1

$$\text{softmax}_{\text{dog}}(f_{\text{dog}}(x), f_{\text{cat}}(x)) = \frac{\exp(f_{\text{dog}}(x))}{\exp(f_{\text{dog}}(x)) + \exp(f_{\text{cat}}(x))}$$

There is nothing magical about this

It's not the only way to do it

Just need to get the numbers to be positive and sum to 1!

The softmax in general

0	1	2	3	4	5	6	7	8	9
0%	0%	0%	0%	0%	90%	8%	0%	2%	0%

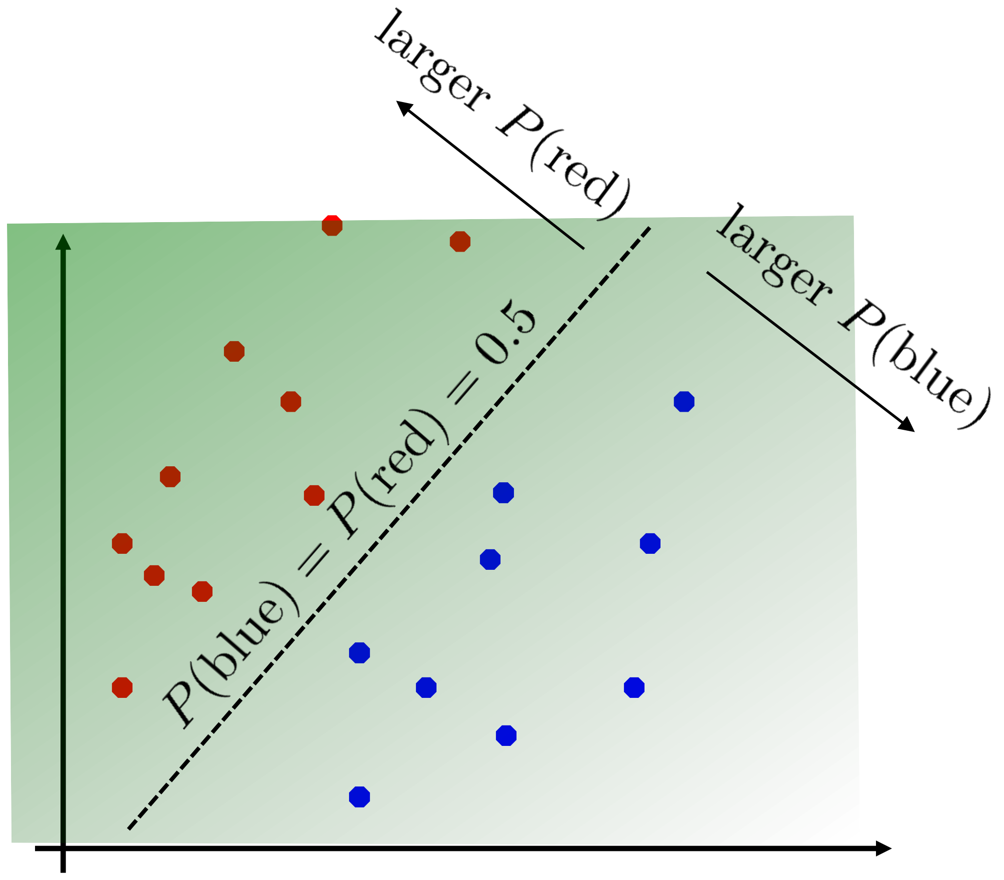
N possible labels

$p(y|x)$ – vector with N elements

$f_\theta(x)$ – vector-valued function with N outputs

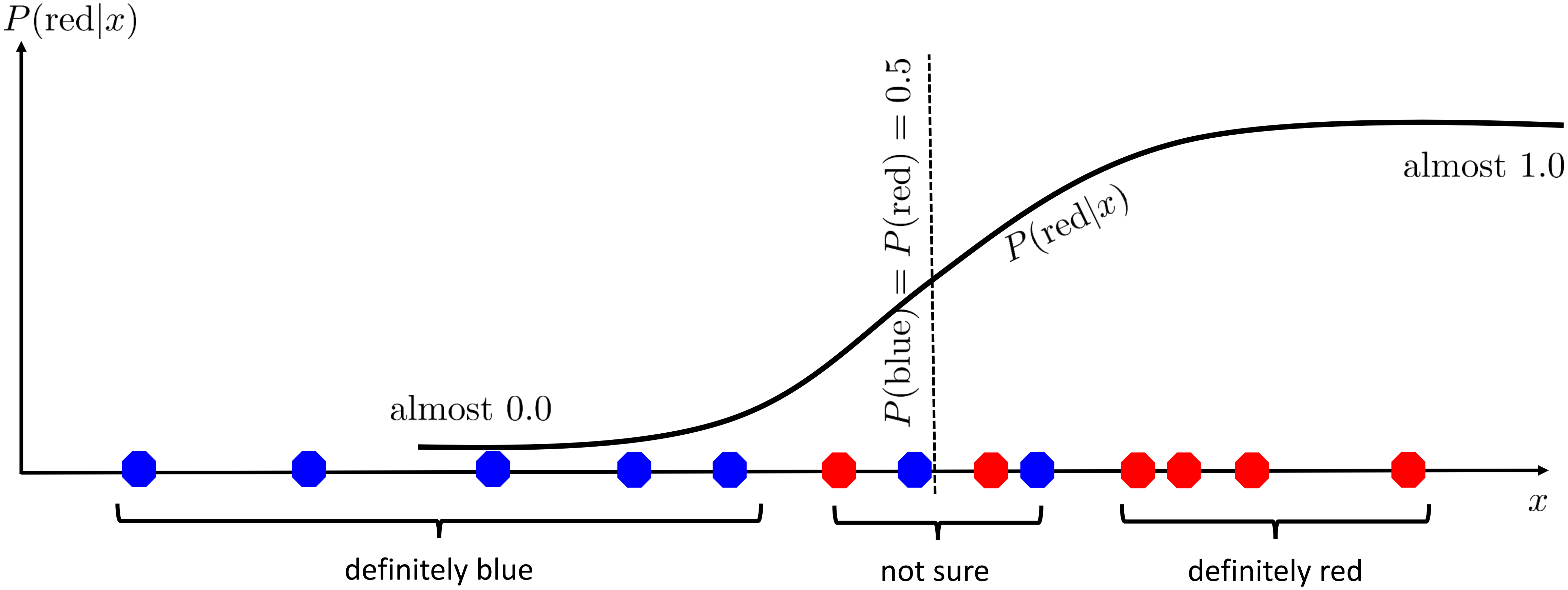
$$p(y = i|x) = \text{softmax}(f_\theta(x))[i] = \frac{\exp(f_{\theta,i}(x))}{\sum_{j=1}^{N} \exp(f_{\theta,j}(x))}$$

An illustration: 2D case



As $\theta_y^T x$ gets bigger, $p(y|x)$ gets bigger

An illustration: 1D case

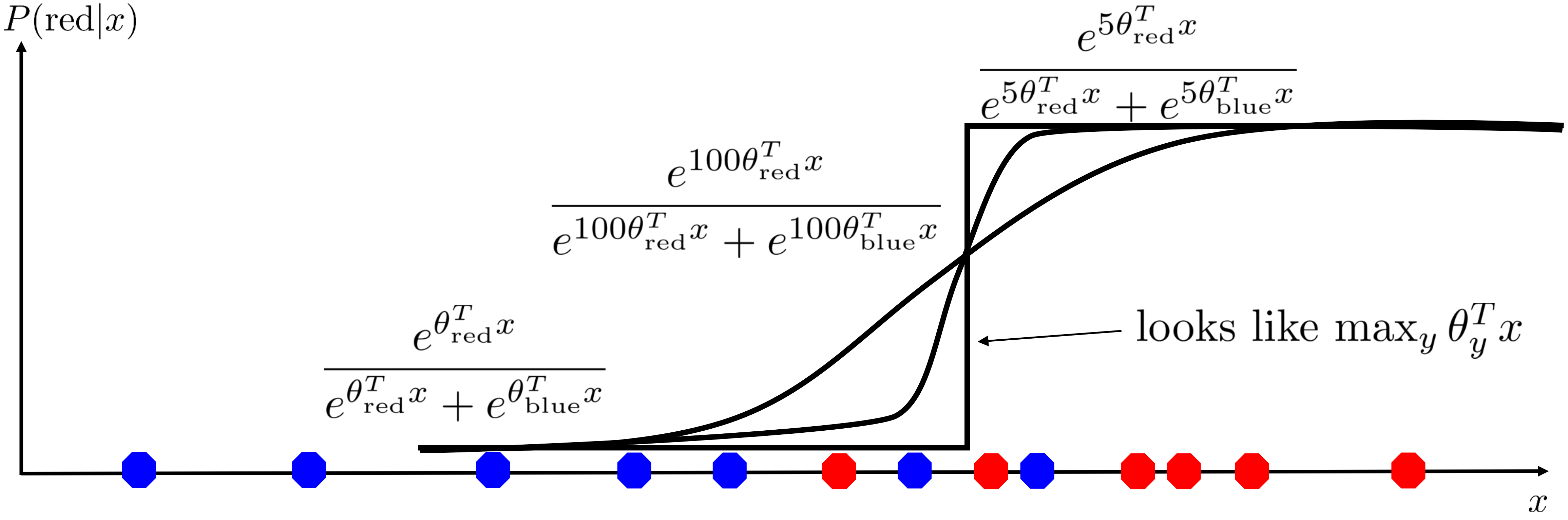


$$P(\text{red}|x) = \frac{e^{\theta_{\text{red}}^T x}}{e^{\theta_{\text{red}}^T x} + e^{\theta_{\text{blue}}^T x}}$$

probability increases exponentially as we move away from boundary

normalizer

Why is it called a *soft*max?



$$P(\text{red}|x) = \frac{e^{\theta_{\text{red}}^T x}}{e^{\theta_{\text{red}}^T x} + e^{\theta_{\text{blue}}^T x}}$$

Loss functions

So far...

computer program

[object probability]

$$f_{\text{dog}}(x) = x^T \theta_{\text{dog}}$$

$$f_{\text{cat}}(x) = x^T \theta_{\text{cat}}$$

$$\vec{\theta} = \{\theta_{\text{dog}}, \theta_{\text{cat}}\}$$

this has learned parameters

$$p(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x))$$

$$p(y = i|x) = \text{softmax}(f_\theta(x))[i] = \frac{\exp(f_{\theta,i}(x))}{\sum_{j=1}^N \exp(f_{\theta,j}(x))}$$

How do we select $\vec{\theta}$?

The machine learning method

for solving any problem ever

1. Define your **model class**

How do **represent** the "program"

We (mostly) did this in the last section

(though we'll spend a lot more time on this later)

2. Define your **loss function**

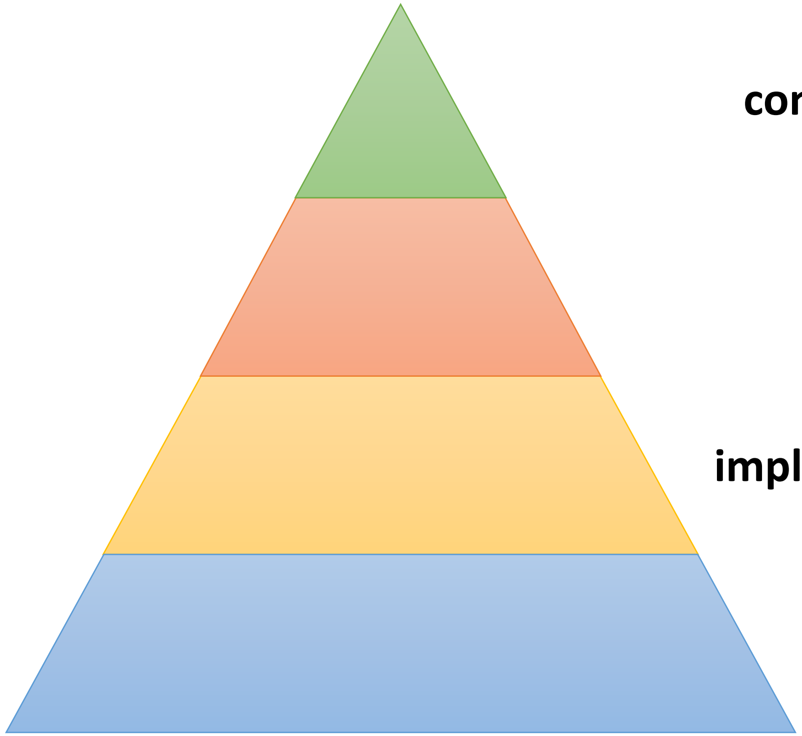
How to measure if one **model** in your **model class** is better than another?

3. Pick your **optimizer**

How to **search** the **model class** to find the model that minimizes the **loss function?**

4. Run it on a big GPU

Aside: Marr's levels of analysis



computational	**"why?"**	e.g., loss function
algorithmic	**"what?"**	e.g., the model
implementation	**"how?"**	e.g., the optimization algorithm
	"on which GPU?"	

There are many variants on this basic idea…

The machine learning method

for solving any problem ever

1. Define your **model class**

How do **represent** the “program”

We (mostly) did this in the last section

(though we’ll spend a lot more time on this later)

2. Define your **loss function**

How to measure if one **model** in your **model class** is better than another?

3. Pick your **optimizer**

How to **search** the **model class** to find the model that minimizes the **loss function?**

4. Run it on a big GPU

How is the dataset "generated"?

$\sim \; p(x)$

probability distribution over photos

$$\text{“dog”} \sim p(y|x)$$

conditional probability distribution over labels

$$\text{result: } (x, y) \sim p(x, y)$$

How is the dataset “generated”?

$(x, y) \sim p(x, y)$

Training set: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$

what is $p(\mathcal{D})$?

every (x_i, y_i) independent of each (x_j, y_j)
when is this true? when is this false?

key assumption: *independent* and *identically distributed* (i.i.d.)

exactly the same for all i

$(x_i, y_i) \sim \underline{p(x, y)}$

when i.i.d.: $p(\mathcal{D}) = \prod_i p(x_i, y_i)$

How is the dataset "generated"?

when i.i.d.: $p(\mathcal{D}) = \prod_i p(x_i, y_i) = \prod_i p(x_i)p(y_i|x_i)$

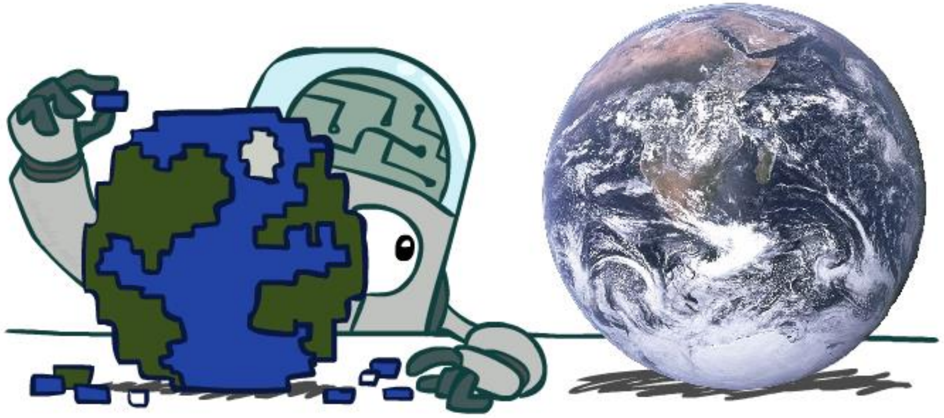
we are learning $p_\theta(y|x)$ it's a "model" of the true $p(y|x)$

a good model should make the data look probable

idea: choose θ such that

$$p(\mathcal{D}) = \prod_i p(x_i)p_\theta(y_i|x_i)$$

is maximized



what's the problem?

How is the dataset “generated”?

$$p(\mathcal{D}) = \prod_i p(x_i) p_\theta(y_i|x_i)$$

multiplying together many numbers ≤ 1

$$\log p(\mathcal{D}) = \sum_i \log p(x_i) + \log p_\theta(y_i|x_i) = \sum_i \log p_\theta(y_i|x_i) + \cancel{\text{const}}$$

$$\theta^\star \leftarrow \arg\max_\theta \sum_i \log p_\theta(y_i|x_i)$$

maximum likelihood estimation (MLE)

$$\theta^\star \leftarrow \arg\min_\theta -\sum_i \log p_\theta(y_i|x_i)$$

negative log-likelihood (NLL)

this is our **loss function**!

Loss functions

aside: cross-entropy

how similar are two distributions, p_θ and p?

$$H(p, p_\theta) = -\sum_y p(y|x_i) \log p_\theta(y|x_i)$$

assume $y_i \sim p(y|x_i)$

$$H(p, p_\theta) \approx -\log p_\theta(y_i|x_i)$$

In general:

the *loss function* quantifies how *bad* θ is

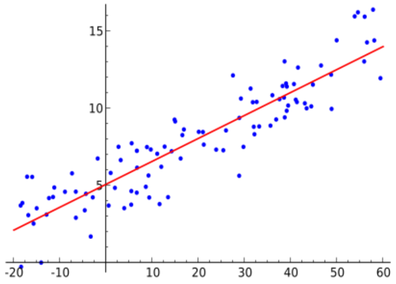
we want the *least bad* (best) θ

Examples:

negative log-likelihood: $-\sum_i \log p_\theta(y_i|x_i)$ also called *cross-entropy* why?

zero-one loss: $\sum_i \delta(f_\theta(x_i) \neq y_i)$

mean squared error: $\sum_i \frac{1}{2}||f_\theta(x_i) - y_i||^2$ actually just negative log-likelihood! why?



Optimization

The machine learning method

for solving any problem ever

1. Define your **model class**

$$f_{\text{dog}}(x) = x^T \theta_{\text{dog}}$$
$$f_{\text{cat}}(x) = x^T \theta_{\text{cat}}$$
$$p_\theta(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x))$$

2. Define your **loss function**

negative log-likelihood: $-\sum_i \log p_\theta(y_i|x_i)$

3. Pick your **optimizer**

4. Run it on a big GPU

The loss "landscape"

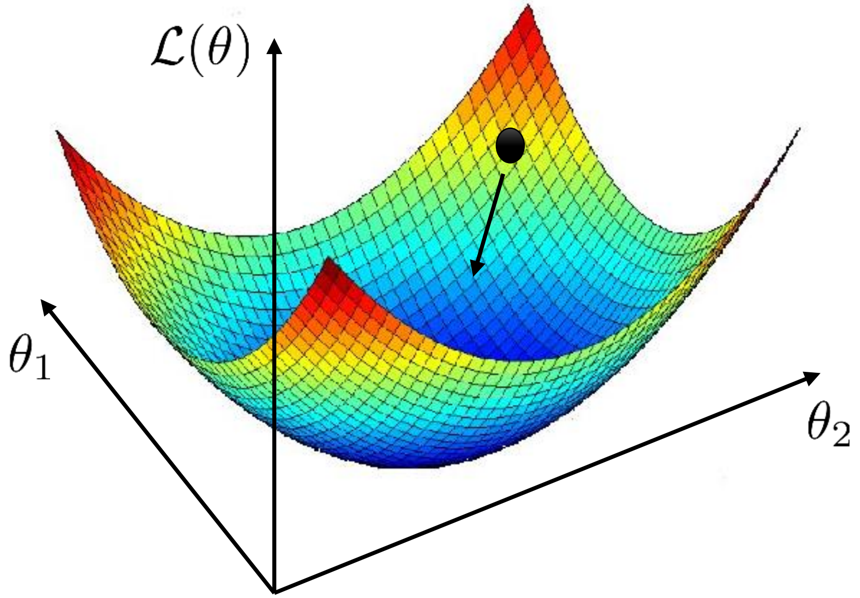
$$\theta^\star \leftarrow \arg\min_\theta \underbrace{-\sum_i \log p_\theta(y_i|x_i)}_{\mathcal{L}(\theta)}$$

let's say θ is 2D

An algorithm:

1. Find a *direction* v where $\mathcal{L}(\theta)$ decreases
2. $\theta \leftarrow \theta + \alpha v$

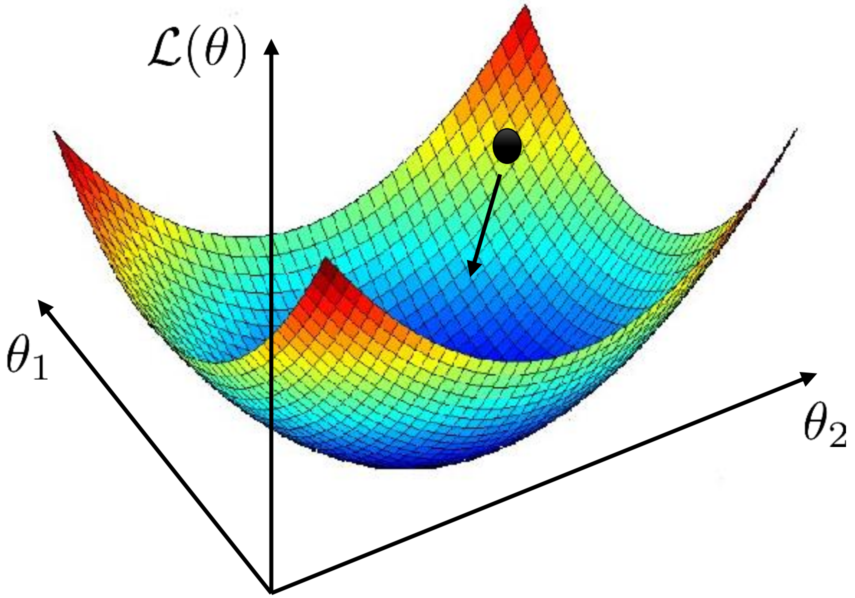
some small constant called "learning rate" or "step size"



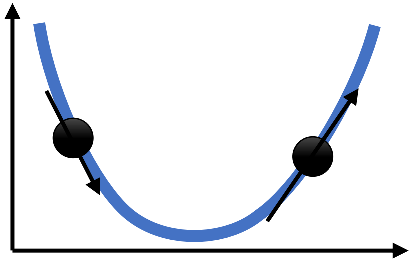
Gradient descent

An algorithm:

1. Find a *direction* v where $\mathcal{L}(\theta)$ decreases
2. $\theta \leftarrow \theta + \alpha v$



Which way does $\mathcal{L}(\theta)$ decrease?



negative slope = go to the right

positive slope = go to the left

in general:

for each dimension, go in the direction opposite the slope **along that dimension**

$$v_1 = -\frac{d\mathcal{L}(\theta)}{d\theta_1} \qquad v_2 = -\frac{d\mathcal{L}(\theta)}{d\theta_2} \qquad \text{etc.}$$

gradient:

$$\nabla_\theta \mathcal{L}(\theta) = \begin{pmatrix} \frac{d\mathcal{L}(\theta)}{d\theta_1} \\ \frac{d\mathcal{L}(\theta)}{d\theta_2} \\ \vdots \\ \frac{d\mathcal{L}(\theta)}{d\theta_n} \end{pmatrix}$$

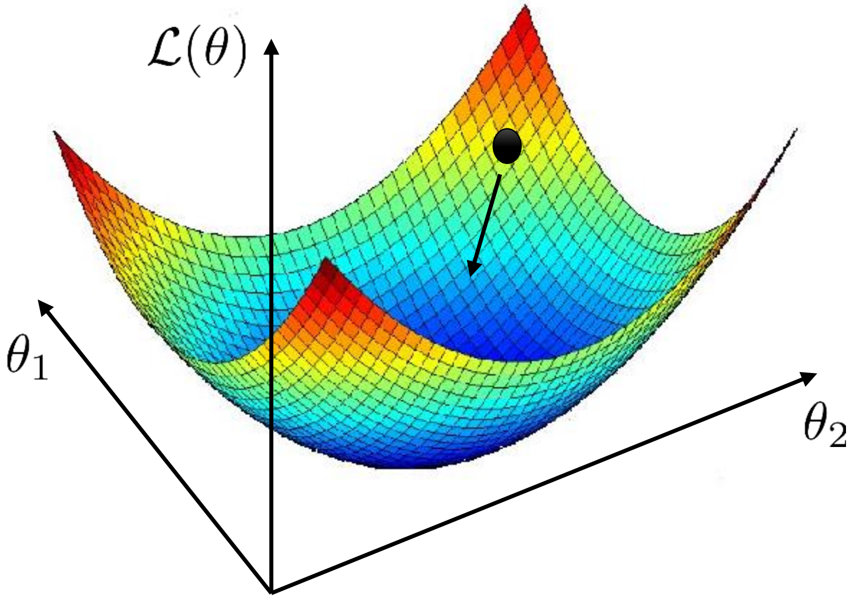
Gradient descent

An algorithm:

1. Find a *direction* v where $\mathcal{L}(\theta)$ decreases
2. $\theta \leftarrow \theta + \alpha v$

Gradient descent:

1. Compute $\nabla_\theta \mathcal{L}(\theta)$
2. $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$



$$\nabla_\theta \mathcal{L}(\theta) = \begin{pmatrix} \frac{d\mathcal{L}(\theta)}{d\theta_1} \\ \frac{d\mathcal{L}(\theta)}{d\theta_2} \\ \vdots \\ \frac{d\mathcal{L}(\theta)}{d\theta_n} \end{pmatrix}$$

We'll go into a lot more detail about gradient descent and related methods in a later lecture!

The machine learning method

for solving any problem ever

1. Define your **model class**

$$f_{\text{dog}}(x) = x^T\theta_{\text{dog}}$$
$$f_{\text{cat}}(x) = x^T\theta_{\text{cat}}$$
$$p_\theta(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x))$$

2. Define your **loss function**

negative log-likelihood: $-\sum_i \log p_\theta(y_i|x_i)$

3. Pick your **optimizer**

Gradient descent:

1. Compute $\nabla_\theta \mathcal{L}(\theta)$
2. $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$

4. Run it on a big GPU

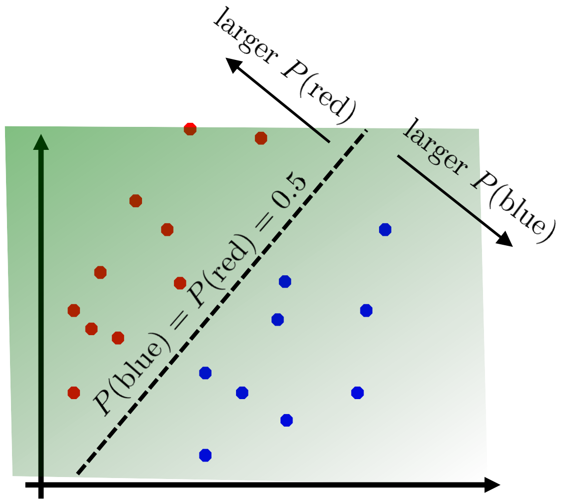
Logistic regression

$$f_\theta(x) = \begin{bmatrix} x^T\theta_{y_1} \\ x^T\theta_{y_2} \\ \dots \\ x^T\theta_{y_m} \end{bmatrix}$$

$$f_\theta(x) = x^T\theta$$

matrix

$$x^T \times \begin{bmatrix} \theta_{y_1} & \theta_{y_2} & \theta_{y_3} \end{bmatrix} = \begin{bmatrix} x^T\theta_{y_1} \\ x^T\theta_{y_2} \\ \dots \\ x^T\theta_{y_m} \end{bmatrix}$$



$$p_\theta(y = i|x) = \text{softmax}(f_\theta(x))[i] = \frac{\exp(f_{\theta,i}(x))}{\sum_{j=1}^{m} \exp(f_{\theta,j}(x))}$$

Gradient descent:

1. Compute $\nabla_\theta \mathcal{L}(\theta)$
2. $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$

$$\mathcal{L}(\theta) = -\sum_{i=1}^{n} \log p_\theta(y_i|x_i)$$

Special case: binary classification

What if we have only two classes?

This is a bit *redundant*

$$P(y_1|x) = \frac{e^{\theta_{y_1}^T x}}{e^{\theta_{y_1}^T x} + e^{\theta_{y_2}^T x}}$$

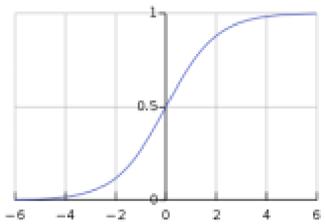
Why? $P(y_1|x) + P(y_2|x) = 1$

if we know $P(y_1|x)$, we know $P(y_2|x)$

multiply top and bottom by $e^{-\theta_{y_1}^T x}$

$$P(y_1|x) = \frac{e^{\theta_{y_1}^T x}}{e^{\theta_{y_1}^T x} + e^{\theta_{y_2}^T x}} = \frac{e^{\theta_{y_1}^T x} e^{-\theta_{y_1}^T x}}{(e^{\theta_{y_1}^T x} + e^{\theta_{y_2}^T x}) e^{-\theta_{y_1}^T x}} = \frac{\overbrace{e^{\theta_{y_1}^T x - \theta_{y_1}^T x}}^{=1}}{\underbrace{e^{\theta_{y_1}^T x - \theta_{y_1}^T x}}_{=1} + e^{\theta_{y_2}^T x - \theta_{y_1}^T x}}$$

Let $\theta_+ = \theta_{y_1} - \theta_{y_2}$



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

this is called the logistic equation
also referred to as a sigmoid

Empirical risk and true risk

zero-one loss: $\sum_i \delta(f_\theta(x_i) \neq y_i)$ 1 if wrong, 0 if right

Risk: probability you will get it wrong

expected value of our loss quantifies this

can be generalized to other losses

(e.g., NLL)

$\sim \; p(x)$

$\text{Risk} = E_{x \sim p(x), y \sim p(y|x)}[\mathcal{L}(x, y, \theta)]$

$y \sim p(y|x)$ how likely is it that $f_\theta(x)$ is wrong?

During training, we can't sample $x \sim p(x)$, we just have $\mathcal{D}$

$\text{Empirical risk} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(x_i, y_i, \theta) \approx E_{x \sim p(x), y \sim p(y|x)}[\mathcal{L}(x, y, \theta)]$

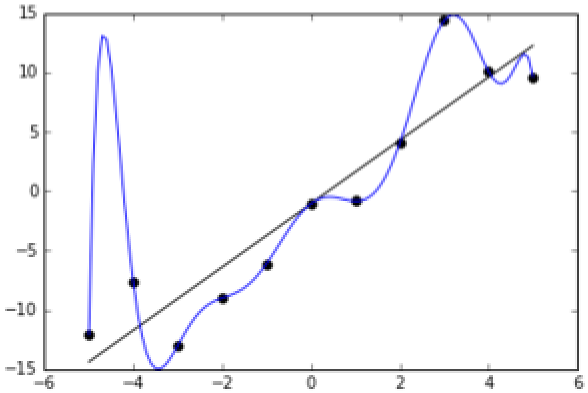
is this a **good** approximation?

Empirical risk minimization

$$\text{Empirical risk} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(x_i, y_i, \theta) \approx E_{x\sim p(x), y\sim p(y|x)}[\mathcal{L}(x, y, \theta)]$$

Supervised learning is (usually) *empirical* risk minimization

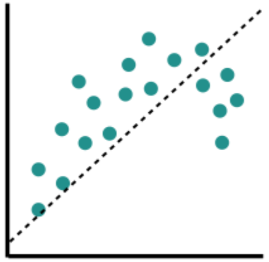
Is this the same as *true* risk minimization?



Overfitting: when the empirical risk is low, but the true risk is high

can happen if the dataset is too small

can happen if the model is too powerful (has too many parameters/capacity)



Underfitting: when the empirical risk is high, and the true risk is high

can happen if the model is too weak (has too few parameters/capacity)

can happen if your optimizer is not configured well (e.g., wrong learning rate)

This is very important, and we will discuss this in much more detail later!

Summary

1. Define your **model class**

$$f_{\text{dog}}(x) = x^T \theta_{\text{dog}}$$
$$f_{\text{cat}}(x) = x^T \theta_{\text{cat}}$$

$$p_\theta(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x))$$

2. Define your **loss function**

negative log-likelihood: $-\sum_i \log p_\theta(y_i|x_i)$

3. Pick your **optimizer**

Gradient descent:

1. Compute $\nabla_\theta \mathcal{L}(\theta)$
2. $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$

4. Run it on a big GPU