How do we formulate learning problems?
Different types of learning problems

Supervised learning:
- Predict $y$ from $x$
- $f_\theta(x) = y$
- [object label]

Unsupervised learning:
- Unlabeled data
- Representation

Reinforcement learning:
- Agent (state $S_t$, reward $R_t$, action $A_t$)
- Environment (state $S_{t+1}$, reward $R_{t+1}$)
Supervised learning

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

learn $f_\theta(x) \approx y$

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 $y_i$?

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

how do we find the best setting of $\theta$?

- 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)
Reinforcement learning

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

learn $f_\theta(x) \approx y$

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: \( D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)

learn \( f_\theta(x) \approx y \)

Predict...
- category of object
- sentence in French
- presence of disease
- text of a phrase

Based on...
- image
- sentence in English
- X-ray image
- audio utterance

\( Y \)

\( X \)
Prediction is difficult

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>5?</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>90%</td>
<td>8%</td>
<td>0%</td>
<td>2%</td>
<td>0%</td>
</tr>
<tr>
<td>9?</td>
<td>4%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>11%</td>
<td>0%</td>
<td>4%</td>
<td>0%</td>
<td>6%</td>
<td>75%</td>
</tr>
<tr>
<td>3?</td>
<td>5%</td>
<td>0%</td>
<td>0%</td>
<td>40%</td>
<td>0%</td>
<td>30%</td>
<td>20%</td>
<td>0%</td>
<td>5%</td>
<td>0%</td>
</tr>
<tr>
<td>4?</td>
<td>5%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>50%</td>
<td>0%</td>
<td>3%</td>
<td>0%</td>
<td>2%</td>
<td>40%</td>
</tr>
<tr>
<td>0?</td>
<td>70%</td>
<td>0%</td>
<td>20%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>10%</td>
<td>0%</td>
</tr>
</tbody>
</table>
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: \( \mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)

learn \( f_\theta(x) \approx y \quad p_\theta(y|x) \)
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?

10 possible labels, output 10 numbers (that are positive and sum to 1.0)
How do we represent it?

How about:

\[ p(y = \text{dog}|x) = x^T \theta_{\text{dog}} \]
\[ p(y = \text{cat}|x) = x^T \theta_{\text{cat}} \]
\[ \tilde{\theta} = \{ \theta_{\text{dog}}, \theta_{\text{cat}} \} \]

(that are positive and sum to 1.0)
How do we represent it?

If below line($x$) then
else:

how about:

\[ f_{\text{dog}}(x) = x^T \theta_{\text{dog}} \]
\[ f_{\text{cat}}(x) = x^T \theta_{\text{cat}} \]
\[ \tilde{\theta} = \{ \theta_{\text{dog}}, \theta_{\text{cat}} \} \]

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

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:

\[ f_{\text{dog}}(x) = x^T \theta_{\text{dog}} \quad p(y|x) = \text{softmax}(f_{\text{dog}}(x), f_{\text{cat}}(x)) \]

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

\[ \theta = \{ \theta_{\text{dog}}, \theta_{\text{cat}} \} \]

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?

\[ \frac{z_1}{z_1 + z_2} \quad \frac{z_1}{\sum_{i=1}^{n} z_i} \]
How do we represent it?

how about:

\[ f_{\text{dog}}(x) = x^T \theta_{\text{dog}} \]
\[ f_{\text{cat}}(x) = x^T \theta_{\text{cat}} \]
\[ \vec{\theta} = \{ \theta_{\text{dog}}, \theta_{\text{cat}} \} \]

![Image of mathematical expressions]

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

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

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

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

\(N\) possible labels

\(p(y|x)\) – vector with \(N\) elements

\(f_\theta(x)\) – vector-valued function with \(N\) outputs
An illustration: 2D case

As $\theta^T_y x$ gets bigger, $p(y|x)$ gets bigger
An illustration: 1D case

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

- definitely blue
- not sure
- definitely red

Probability increases exponentially as we move away from boundary.
Why is it called a softmax?

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

This function looks like \( \max_y \theta_y^T x \).
Loss functions
So far...

\[ f_{\text{dog}}(x) = x^T \theta_{\text{dog}} \]
\[ f_{\text{cat}}(x) = x^T \theta_{\text{cat}} \]
\[ \tilde{\theta} = \{ \theta_{\text{dog}}, \theta_{\text{cat}} \} \]

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

How do we select \( \tilde{\theta} \)?
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

2. Define your loss function

3. Pick your optimizer

4. Run it on a big GPU
How is the dataset “generated”?

\[ \sim p(x) \quad \text{probability distribution over photos} \]

\[ \text{“dog”} \sim p(y|x) \quad \text{conditional probability distribution over labels} \]

result: \( (x, y) \sim p(x, y) \)
How is the dataset “generated”?

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

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

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

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

\((x_i, y_i) \sim p(x, y)\)

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 \( \theta \) 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”?

\[ p(\mathcal{D}) = \prod_{i} p(x_i)p_\theta(y_i|x_i) \]

\[ \log p(\mathcal{D}) = \sum_{i} \log p(x_i) + \log p_\theta(y_i|x_i) = \sum_{i} \log p_\theta(y_i|x_i) + \text{const} \]

\( \theta^* \leftarrow \arg \max_{\theta} \sum_{i} \log p_\theta(y_i|x_i) \)  \quad \text{maximum likelihood estimation (MLE)}

\( \theta^* \leftarrow \arg \min_{\theta} -\sum_{i} \log p_\theta(y_i|x_i) \)  \quad \text{negative log-likelihood (NLL)}

this is our \textit{loss function}!
Loss functions

In general:
the loss function quantifies how bad $\theta$ is
we want the least bad (best) $\theta$

Examples:
negative log-likelihood: $- \sum_i \log p_\theta(y_i|x_i)$
zero-one loss: $\sum_i \delta(f_\theta(x_i) \neq y_i)$
mean squared error: $\sum_i \frac{1}{2} ||f_\theta(x_i) - y_i||^2$

aside: cross-entropy
how similar are two distributions, $p_\theta$ and $p$?

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

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

also called cross-entropy why?

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”

\[ \theta^* \leftarrow \arg \min_{\theta} - \sum_i \log p_{\theta}(y_i|x_i) \]

let’s say \( \theta \) 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?

In general:
- for each dimension, go in the direction opposite the slope along that dimension

\[ v_1 = -\frac{d\mathcal{L}(\theta)}{d\theta_1} \quad v_2 = -\frac{d\mathcal{L}(\theta)}{d\theta_2} \quad \text{etc.} \]
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. $\theta \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta)$

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

$$\nabla_{\theta} \mathcal{L}(\theta) = \begin{bmatrix} \frac{d\mathcal{L}(\theta)}{d\theta_1} \\ \frac{d\mathcal{L}(\theta)}{d\theta_2} \\ \vdots \\ \frac{d\mathcal{L}(\theta)}{d\theta_n} \end{bmatrix}$$
The machine learning method for solving any problem ever

1. Define your **model class**
   
   \[ f_{\text{dog}}(x) = x^T \theta_{\text{dog}} \]
   
   \[ f_{\text{cat}}(x) = x^T \theta_{\text{cat}} \]

2. Define your **loss function**
   
   negative log-likelihood:
   
   \[ -\sum_i \log p_{\theta}(y_i|x_i) \]

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

3. Pick your **optimizer**

4. Run it on a big GPU
Logistic regression

\[ f_\theta(x) = \begin{bmatrix} x^T \theta_{y_1} \\ x^T \theta_{y_2} \\ \vdots \\ x^T \theta_{y_m} \end{bmatrix} \]

\[ f_\theta(x) = x^T \theta \]

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

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?

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

This is a bit redundant

Why? \[ P(y_1 | x) + P(y_2 | x) = 1 \]

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

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

\[ = \frac{e^{\theta_{y_1}^T x} e^{-\theta_{y_1}^T x}}{(e^{\theta_{y_1}^T x} + e^{\theta_{y_2}^T x}) e^{-\theta_{y_1}^T x}} \]

\[ = \frac{e^{\theta_{y_1}^T x} - \theta_{y_1}^T x}{e^{\theta_{y_1}^T x} - \theta_{y_1}^T x + e^{\theta_{y_2}^T x} - \theta_{y_1}^T x} = 1 \]

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
(e.g., NLL)

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

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

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

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

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

2. Define your **loss function**
   
   negative log-likelihood: \[- \sum_i \log p_{\theta}(y_i|x_i)\]

3. Pick your **optimizer**
   
   Gradient descent:
   
   \begin{align*}
   &1. \quad \text{Compute } \nabla_{\theta} \mathcal{L}(\theta) \\
   &2. \quad \theta \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta)
   \end{align*}

4. Run it on a big GPU