### Lecture 2: ML review (1)

CS 182/282A ("Deep Learning")

2022/01/24

### Today's lecture

- In this lecture and the next lecture, we will go over concepts at the core of machine learning as a whole
  - We will focus on concepts that are the most relevant to deep learning
- Much of this will be review if you have already taken a machine learning course
- Today, we will focus on the supervised learning problem setup, go over the general machine learning method, and define **probabilistic models**, **likelihood based loss functions**, and **gradient based optimization**

#### Different classes of learning problems (non exhaustive)

Supervised learning





Unsupervised learning



Reinforcement learning







### Let's start with supervised learning

### Supervised learning



- In supervised learning, we are given a dataset  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$
- Our goal is to learn a model that predicts outputs given inputs:  $f_{\theta}(\mathbf{x}) = y$
- This setup encompasses the overwhelming majority of machine learning that is used in industry (a multi-billion \$/year industry)
- The basic principles are simple, and we will cover them in this lecture

## Examples of supervised learning problems (that deep learning has done really well on)

X	У				
image of object	category of object				
sentence in English	sentence in French				
audio utterance	text of what was said				
amino acid sequence	3D protein structure				

## Should the model just output *y*? What could go wrong?

Image	0?	1?	2?	3?	4?	5?	6?	7?	8?	9?
5	0%	0%	0%	60%	0%	35%	0%	0%	0%	5%
9	0%	0%	0%	0%	50%	0%	0%	0%	0%	50%
0	30%	0%	70%	0%	0%	0%	0%	0%	0%	0%

### Predicting probabilities

- Often, it makes more sense to have the model predict output *probabilities*, rather than the outputs themselves
  - This can better capture when the model is *uncertain* about difficult inputs
  - We'll also see later why this makes the learning process easier
- So instead of the model output  $f_{\theta}(\mathbf{x})$  being a single y, it will instead be an entire distribution over all possible y!
  - E.g., for digit recognition, the output will be 10 numbers between 0 and 1 that sum to 1

### How do we output probabilities?

- How do we make our model output numbers between 0 and 1 that sum to 1?
- Idea: first let our model output whatever numbers it wants
  - Then, make all the numbers positive and *normalize* (divide by the sum)
- There are many ways to make a number *z* positive
  - In this context, the most commonly used choice is exp(z), which is bijective
  - In this case, the (raw) model outputs are called **logits**

## A probabilistic model for discrete labels

if there are K possible labels, then  $f_{\theta}(\mathbf{x})$  is a vector of length K

we represent the final probabilities using the **softmax** function:

softmax 
$$(f_{\theta}(x))_{c} = \frac{exp\{f_{\theta}(x)_{c}\}}{\sum_{i=1}^{K} exp\{f_{\theta}(x)_{i}\}} = p_{\theta}(y=c|x)$$

(aside: sometimes,  $f_{\theta}(\mathbf{x})$  is used to denote  $\arg \max_{y} p_{\theta}(y \mid \mathbf{x})$ , but I won't do that)

# Some examples of the softmax function

supposing K = 4, let's work through some examples softmax([0, 0, 0, 0]) = [0.15, 0.15, 0.25, 0.25]softmax([-100, -100, -100]) = [0.25, 0.25, 0.25, 0.25] softmax([0, 0, 100, 0]) ≈ [0,0,1,0] softmax([-100, -100, 0, -100]) ≈ [0,0,1,0] softmax([2, 1, 0, 0]) = [0.6103, 0.2245, 0.0826, 0.0826]



- So far, we have defined what our probabilistic model is going to look like
  - In the case of discrete labels, it will output K numbers that will be exponentiated and normalized to form an output distribution
- What else do we need?
  - How do we know whether or not the model parameters are good?
  - How do we find good parameters?

## The machine learning method (or, at least, the deep learning method)

- 1. Define your **model** which neural network, what does it output, ...
- 2. Define your **loss function** which parameters are good vs. bad?
- 3. Define your **optimizer** how do we find good parameters?
- 4. Run it on a big GPU

The machine learning method (or, at least, the deep learning method)

1. Define your **model** — which neural network, what does it output, ...

2. Define your **loss function** - which parameters are good vs. bad?

3. Define your **optimizer** — how do we find good parameters?

4. Run it on a big GPU

### What loss function should we use?

- In deciding on a loss function, we have a few desiderata:
  - If our parameters perfectly explain the data, we should incur minimal loss
  - The loss should be "easy" to optimize
  - We don't want to have to engineer new loss functions for every problem
- We will satisfy these desiderata by leveraging the most widely used tool in statistical inference maximum likelihood estimation (MLE)

The maximum likelihood principle po(x,y)=p(x)po(y|x) given data  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$  $\{p_{\Theta}: \Theta \in \Theta\}$ assume a set (family) of distributions on  $(\mathbf{x}, \mathbf{y})$ assume some plate generated D

the parameters  $\theta$  dictate the conditional distribution of y given **x** 

he objective/definition: "recover 
$$\hat{\Theta}$$
" (sort of)  
 $\Theta_{mle} = \underset{\Theta \in \Theta}{\operatorname{arg.max}} p(\Theta | \Theta) = \underset{\Theta \in \Theta}{\operatorname{arg.max}} \underset{i=1}{\overset{N}{\prod}} p(x_i) p_{\Theta}(y_i | x_i)$ 

# From MLE to a loss function

we are given  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ our goal is to find  $\underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{arg max}} \prod_{i=1}^{N} p(\mathbf{x}_i) p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i)$ 

working with a product of terms is tricky and messy...

idea: take the log instead! this leads to the **negative log likelihood** loss function:

$$\begin{array}{l} arg \max_{i=1}^{n} \sum_{i=1}^{n} \log p(x_i) + \log p_{\theta}(y_i | x_i) = arg \max_{i=1}^{n} \sum_{i=1}^{n} \log p_{\theta}(y_i | x_i) \\ constant w.r.t. \\ \theta = arg \min_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \min_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} \sum_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta = arg \max_{i=1}^{n} - \log p_{\theta}(y_i | x_i) \\ \theta$$

## The negative log likelihood loss function

this loss is oftentimes called the **cross-entropy** loss — what is cross-entropy?

$$H(p,q) = -\sum_{x} p(x) \log q(x) = \mathbb{E}_{p}[-\log q(x)]$$

let's plug in  $p_{data}$  (the true data distribution) for p and some  $p_{\theta}$  for q: H( $p_{dota}$ ,  $p_{\theta}$ ) =  $\mathbb{E}_{p_{data}} \left[ -\log p_{\theta}(X, Y) \right]$   $= \mathbb{E}_{p_{data}} \left[ -\log p(X) - \log p_{\theta}(Y|X) \right] \approx \sum_{i=1}^{N} -\log p(x_i) - \log p_{\theta}(y_i|X_i)$ 

maximizing log likelihood is approximately equivalent to minimizing cross-entropy!

#### Should we use the negative log likelihood loss? Revisiting our desiderata

- If our parameters perfectly explain the data, we should incur minimal loss
  - Given sufficient data, the log likelihood is maximized by the "true" parameters, if our model is able to represent the underlying data distribution
  - This is related to an attractive property of MLE called *consistency*
- The loss should be "easy" to optimize more on this next
- We don't want to have to engineer new loss functions for every problem
  - Many commonly used loss functions, such as **squared error** for regression, can be derived/motivated from log likelihood for different modeling assumptions

## The machine learning method (or, at least, the deep learning method)

- 1. Define your **model** which neural network, what does it output, ...
- 2. Define your **loss function** which parameters are good vs. bad?

3. Define your **optimizer**  $\rightarrow$  how do we find good parameters?

4. Run it on a big GPU

#### What optimizer should we use?

- Deep learning relies on **iterative optimization** to find good parameters
  - Starting from an initial "guess", continually refine that guess until we are satisfied with our final answer
- By far the most commonly used set of iterative optimization techniques in deep learning is (first order) gradient based optimization and variants thereof
  - Basically, move the parameters in the direction of the *negative gradient* of the average loss:  $\theta \leftarrow \theta \alpha \nabla_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; \mathbf{x}_i, y_i)$

### Synergy between loss function and optimizer

- The gradient tells us how the loss value changes for small parameter changes
  - We decrease the loss if we move (with a small enough  $\alpha$ ) along the direction of the negative gradient (basically, go "opposite the slope" in each dimension)
- This motivates choosing the loss function and model carefully, such that the loss function is *differentiable* with respect to the model parameters
  - The negative log likelihood fulfills this for many reasonable problem setups
  - What loss function would not be differentiable?
  - For example, the **0-1 loss function**: 0 if the model is correct, 1 otherwise

A small example: logistic regression The "linear neural network", if we're being weird

- Given  $\mathbf{x} \in \mathbb{R}^d$ , define  $f_{\theta}(\mathbf{x}) = \theta^{\top} \mathbf{x}$ , where  $\theta$  is a  $d \times K$  matrix
- Then, for class  $c \in \{0, ..., K-1\}$ , we have  $p_{\theta}(y = c \mid \mathbf{x}) = \operatorname{softmax}(f_{\theta}(\mathbf{x}))_c$
- Loss function:  $\ell(\theta; \mathbf{x}, y) = -\log p_{\theta}(y \mid \mathbf{x})$

• Optimization: 
$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; \mathbf{x}_i, y_i)$$

## The machine learning method (or, at least, the deep learning method)

- 1. Define your **model** which neural network, what does it output, ...
- 2. Define your **loss function** which parameters are good vs. bad?
- 3. Define your **optimizer** how do we find good parameters?
- 4. Run it on a big GPU