Lecture 2: Machine Learning Basics

CS 182 Spring 2021 – Taught by Sergey Levine

Notes by Michael Zhu

Types of Machine Learning Problems

1. **Supervised Learning**

- have dataset of labeled data
- want to predict y from x

2. **Unsupervised learning**

- have unlabeled data
- want to learn a useful representation of the data

3. **Reinforcement learning**

- have an agent and environment
- agent outputs action and the environment outputs the next state and reward
- want to learn an optimal action strategy

Supervised Learning

Given a labeled dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$

Learn $f_\theta(x) = y$

Considerations:

1. **How do we represent $f_\theta(x)$?**

   This course goes over **neural network** representations of $f_\theta(x)$

2. **How do we measure the performance of $f_\theta(x)$?**

   We compute the **loss** which is usually a function of the difference between $f_\theta(x_i)$ and y_i

3. **How do we find the best θ?**

   We use an **optimzation** algorithm. Examples include random search, local search, gradient descent, etc.

Examples of supervised learning problems

Predict	**Based on...**
category of object	image
sentence in French	sentence in English
presense of disease	X-ray image
text phrase	audio utterance

Predicting probabilities

Often more useful to predict a **probability distribution** of the set of possible outputs $\{y_i\}$ $\forall i$

This is easier to learn due to **smoothness**.

Probabilities: It's possible to update probability distribution by small amounts

Discrete Labels: By definition, it's impossible to update a discrete label by small amounts; you either change labels or you don't

Previous goal: ~~learn $f_\theta(x) = y$~~

New goal: **learn** $f_\theta(x) = p_\theta(y|x)$

Conditions:

- probabilties must be positive $\rightarrow \forall i \; p_\theta(y_i|x) > 0$
- probabilites must sum to one $\rightarrow \sum_i p_\theta(y_i|x) = 1$

Softmax function

Any function that takes some input vector and outputs a probability vector that is **positive** and **sums to one**.

Ideally this is a one to one & onto function to prevent information loss

Ways to make a number z **positive**: $z^2 \quad |z| \quad max(0, z) \quad exp(z)$

Note: $exp(z)$ is one to one & onto; it maps the set of real numbers to the set of positive reals $R \rightarrow R^+$

Normalization: forces numbers to sum to one by dividing each number by the total sum $z_i / \sum_i z_i$

Use $exp(x)$ and normalization to define our softmax

$$p(y_i|x) = softmax_i(f_\theta(x)) = \frac{exp(f_{\theta,i}(x))}{\sum_j exp(f_{\theta,j}(x))}$$

There are many ways to get positive numbers that some to one, but the softmax is very commmonly used

Dataset generation

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

We want our dataset to be as representative of the real world as possible

The set of inputs $\{x_i\}$ are sampled from a probability distribution $p(x)$ based on the **real world**

The set of labels $\{y_i\}$ are generated from a conditional probability distribution $p(y|x)$ based on the **labelling process** (i.e. human observation, measuring tools, experimentation, etc)

By chain rule, the probability of each sample is:

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

If we assume that each sample is **independent** and **identically distributed** (i.i.d.), then:

$$p(D) = \prod_i p(x_i, y_i)$$

$$p(D) = \prod_i p(x_i) p(y_i|x_i)$$

In machine learning, we are learning $p_\theta(y|x)$, which is a **model** of the true $p(y|x)$

A good model should make the dataset look probable, so we want to maximize $p(D)$ as defined with $p_\theta(y|x)$

$$\arg\max_\theta p(D) = \prod_i p(x_i)p_\theta(y_i|x_i)$$

Loss Function

Issue: the above definition of $p(D)$ is numerically challenging since it will almost always be very close to zero (multiplying many numbers <= 1)

Instead we can take the logarithm $\log p(D)$ to use a **summation** rather than product.

$$\log p(D) = \sum_i \log p(x_i) + \log p_\theta(y_i|x_i)$$

We can **ignore** $\log p(x_i)$ because it doesn't depend on θ

Optimization **objective**:

$$\theta^* \leftarrow \arg\max_\theta \sum_i \log p_\theta(y_i|x_i) \quad \text{maxiumum likelihood estimation (MLE)}$$

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

Examples of loss functions

Intuitively, loss functions tell us how **bad** θ is

Zero-one loss: $\sum_i \delta(f_\theta(x_i)! = y_i)$

Mean squared error: $\sum_i^n \frac{1}{n}||f_\theta(x_i) - y_i||^2$

Negative log-likelihood: $-\sum *i \log p * \theta(y * i|x * i)$ (aka **cross-entropy**)

And many more...

Aside: **cross-entropy** measures the similarity between two distributions p and p_θ

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

Optimization: Gradient Descent

We have our loss function $L(\theta)$ as derived from above

General minimization algorithm:

1. Find a direction v where $L(\theta)$ decreases
2. Update $\theta \leftarrow \theta + \alpha v$

Gradient descent algorithm:

Gradient: $\nabla_\theta L(\theta) = \begin{pmatrix} dL(\theta)/d\theta_1 \\ dL(\theta)/d\theta_2 \\ \dots \\ dL(\theta)/d\theta_n \end{pmatrix}$

1. Compute $\nabla_\theta L(\theta)$
2. Update $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$

Unsupervised Learning

Given unlabeled data $D = \{x_1, x_2, ..., x_n\}$

Learn a **representation** of the data

Another way of thinking about this is that the data comes from some **continuous probability distribution**, and the goal of unsupervised learning is to help us figure out and *model* that distribution

Two formulations of unsupervised learning

Generative Modeling

Neural network that generates outputs that are similar to the data that it was trained on (i.e. GANs, VAEs, pixel RNN, etc).

Example: Generating faces; the model must come up with an internal representation of what makes up a face

Self-supervised Representation Learning

Task: remove parts of the data and try to predict the missing parts

The task itself is not that important, but it forces the model to learn useful representations that can then be used for other tasks

Example: BERT language model

- dataset of sentences
- train model by removing words and then predicting the missing words
- model uses word embeddings to make these predictions, and as it trains, it updates the embeddings to represent the corpus better
- these embeddings can then be used for other language tasks like machine translation and sentiment analysis

Reinforcement Learning

Given:

1. Agent:

- takes in **state** s_t as input
- outputs an **action** a_t

2. Environment:

- takes in **action** a_t as input
- outputs the next **state** s_{t+1} and a corresponding **reward** r_{t+1}

Want to *learn* $f_\theta(s_t) = a_t$ **which *maximizes* total reward** $\sum_{t=1}^{H} r(s_t, a_t)$

Note: Supervised learning can be thought of as a specific type of reinforcement learning where the reward is the negative loss

Example: training a dog

- action: muscle movement
- state/observations: sight, smell, sound, etc
- reward: food

Example: training a robot to walk

- action: motor current or torque
- state/observations: camera images or video feed

- reward: distance travelled

Example: supply chain management

- action: what to purchase, method of distribution, etc
- state/observations: inventory levels
- reward: profit

Example: reccomendations (ad placement, youtube videos, healthcare treatment)

- action: giving a recommendation
- state/observations: user history
- reward: if user accepts the recommendation $$