CS 182, Lecture Notes

Intro to Deep Learning

Professor: Marvin Zhang
Scribe: Vishal Raman

Contents

§1 Lecture 1: 1/19/22

See the course website for logistical details.

§1.1 Review of Machine Learning

What is machine learning?

- Machine has three core components: **model**, **optimization**, and **data**.
- The model is a function from inputs to outputs, but is **not** programmed by hand - instead, we have parameters to be optimized, and the optimization algorithm finds good parameters that fit the data.
- The process is highly data-driven - this course exists because we need to handle massive datasets and techniques that scale.

The classical view of machine learning is as followss:

- Regression problems: we try to predict outputs from new inputs given a dataset $\mathcal{D} = \{(X_i, y_i)\}$.
- Classification models: In this example, we have a two-dimensional input (x_1, x_2). One way we can do this is through establishing a linear decision boundary. Algebraicly, we can establish this as $\theta_1 x_1 + \theta_2 x_2 + \theta_3 = 0$. The sign of $f(x, \theta) = \theta_1 x_1 + \theta_2 x_2 + \theta_3$ determines the class in the classification model. The function $f_\theta(x) := f(x, \theta)$ is called the **model**.

§1.2 Intro to Deep Learning

Most data is very high dimensional. For example,

- An image consists of millions of pixels, each of which has some number corresponding to pixel intensity. For example, an image could be $224 \times 224 \times 3 = 150528$ dimensional, and these are relatively low-dimensional.
- For language models, vectors are the size of the vocabulary, which corresponds to 10000's of dimensions.
- For audio models, one second can be 16000 time steps of 16-bit integer values.

Our simple linear model from before will not be sufficient.

Deep learning is about learning representations - namely, in order to handle a complex input, we need to learn a good representation that scales. The power of deep learning lies in its ability to learn representations automatically from data.

§1.3 Shallow/Feature-Based Learning

Before deep learning, a common approach was to use a fixed function for extracting features from the input. One example of this is in a model determining orientation of an image, they use an edge detector and a histogram of edges as the feature space, which is low dimensional.

This is a compromise solution because although we don't hand program the model, we hand program the features. Learning on top of these features is very simple; however, coming up with good features is very hard.

In the shallow approach, we use the input and map them through a fixed feature extractor to get to hand-programmed features. Then, we send them through a classifier to obtain labels. In the deep learning approach, we use learned feature extrators to obtain learned features.

§1.4 End-to-end Learning

In deep learning, we process our input through multiple layers of learned transformations. Each layer can be relatively simple, but we can obtain complex representations through composing large numbers of layers. Higher level representations tend to be more abstract and more invariant to information that is not relevant to the label.

The overall model that represents the transformation from input to output is a deep neural network. The parameters at each layer are usually trained with respect to the overall objective/loss. This is referred to end-to-end learning.

In order to work, we need

- Big models with many layers
  - Is more layers better? In training on a dataset called ImageSet, we had LeNet(7 Layers, 1989) to AlexNet(8 Layers, 2012), and the modern ResNet-152 has 152 layers(2015).
- Large datasets with many examples
  - One of the first datasets people considered was MNIST, which predicted digits from handwritten characters. This consisted of 60000 images.
  - Another popular one was CalTech 101, 2003 which had 9000 color images.
  - A similar dataset is CIFAR-10, 2009 with 60000 images.
  - An important one ILSVRC(ImageNet), 2009 has 1.5 million images.
  - The most modern one consists of billions of images, and we can scale as long as we have enough compute.
- Enough compute power to handle all of this: more is better.
  - GPUs(Graphical Processing Units) are very useful for parallel computations.
  - TPUs(Tensor Processing Units) are optimized for matrix operations.

§1.5 Underlying Themes

Overall, we are seeing the perspective that deep learning is very expensive. But deep learning also scales very well, which differs from the old shallow approaches.

The underlying themes:

- Deep learning acquires representations by using high capacity models and lots of data, without requiring engineering features or representations.
- We don't need to know what the good features are, we can have the model figure it out from the data.
- This results in better performance, because when the representations are learned end-to-end, they are better tailored to the current task.
- Scaling is the ability of an algorithm to work better as more data and model capacity grows. Deep learning models are very good at this.

- Inductive bias vs. learning: are we getting better performance from designer insight or learning from the data?
  - Inductive bias is the model we build into the model to make it learn effectively.
  - Although deep learning moves away from heavy inductive bias, we cannot get away from it entirely.
  - Deep Network Models generally overtake the next best model after we figure out the right inductive biases for that application.

Some successes of deep learning:

- Object Recognition: Mask R-CNN(2017)
- Speech Recognition
- Image Generation: BigGAN(2018)
- Text Generation: GPT-2(2019)
- Mastering Go: AlphaGoZero
- Protein Fold Prediction: AlphaFold(2021)

§2 Lecture 2: 1/24/2022

Today, we go over machine learning basics.

§2.1 Supervised Learning

We are given a dataset $\mathcal{D} = \{(X_1, y_1), \ldots, (X_N, y_N)\}$. Our goal is to learn a function that predicts outputs given inputs: $f_\theta(X) = y$.

Some things we are good at doing with supervised learning:

- Learning the category of an object from the image
- Learning the sentence in French from the sentence in English
- Learning the text of what was said using audio utterance
- Learning the 3D protein structure from an amino acid sequence.

Should the model just output y? Consider the example of MNIST, digit classification. Handwritten digits have a lot of ambiguity, so it makes more sense to output a probability distribution, rather than predicting the output.

The way we do this is make the model output whatever it wants, and we make all the numbers positive and normalize. We usually do this with the exp function.

§2.2 Probabilistic Model for Discrete Labels

If there are K possible labels, $f_\theta(x)$ is a vector of length K. We represent the final probability using the softmax function:

$$\text{softmax}\left(f_\theta(x)\right)_c = \frac{\exp\{f_\theta(x)_c\}}{\sum_{i=1}^{K} \exp\{f_\theta(x)_i\}} = p_\theta(y = c|x)$$

How do we know whether model parameters are good and how do we find good parameters?

§2.3 The Machine 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 GPU

§2.3.1 Loss Function

In deciding on a loss function, we have some desiderata:

- If our parameters perfectly explain 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 satisfy these using **maximum likelihood estimation(MLE)** - given data $\mathcal{D} = \{(X_1, y_1), \ldots, (X_N, y_n)\}$, assume a set of distributions on (X, y) given by $\{p_\theta : \theta \in \Theta\}$. We assume some p_{data} generated $\mathcal{D}$, with Then, we seek to maximize:

$$\theta_{MLE} = \underset{\theta \in \Theta}{\operatorname{argmax}}\, p(\mathcal{D}|\theta) = \underset{\theta \in \Theta}{\operatorname{argmax}} \prod_{i=1}^{N} p(x_i)p(y_i|x_i).$$

A good way to find this is by taking a logarithm:

$$\theta_{MLE} = \underset{\theta \in \Theta}{\operatorname{argmin}} - \log p(\mathcal{D}|\theta) = \underset{\theta \in \Theta}{\operatorname{argmin}} - \sum_{i=1}^{N} \log p_\theta(y_i|x_i) =: \sum_{i=1}^{N} \ell(\theta : x_i, y_i).$$

This is referred to as the cross entropy loss:

$$H(p, q) := -\sum_x p(x) \log q(x) = \mathbb{E}_p(-\log q(X)).$$

Note that

$$H(p_{data}, p_\theta) = \mathbb{E}_{p_{data}}[-\log p_\theta(X, y)] \approx \frac{1}{N} \sum_{i=1}^{N} -\log p(x_i) - \log p_\theta(y_i, x_i).$$

§2.3.2 The Optimizer

Deep learning relies on iterative optimization: start from an initial guess, and refine until we are satisfied with the answer.

Most techniques rely on gradient based optimization:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; x_i, y_i).$$

§3 Lecture 3: 1/26/2022

§3.1 True Risk and Empircal Risk

Definition 3.1 (Risk)**.** $R(\theta) := \mathbb{E}[\ell(\theta; x, y)]$.

Definition 3.2 (Empirical Risk)**.** $\widehat{R}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x, y)$.

The empirical risk looks like a Monte Carlo estimate of the true risk, so shouldn't we have $\widehat{R}(\theta) \approx R(\theta)$? The issue is that we are using the training dataset to learn θ, so we can't reuse the same data to get an estimate of the risk.

- When empirical risk is low but the true risk is high, we are overfitting.
  - This can happen if the dataset is too small or the model is too powerful.
  -
- When empirical risk is high and the true risk is high, we are underfitting.
  - This happens when the model is too weak or the optimization doesn't work well.

"High" is up to the practitioner.

The **model class** is used to describe the set of possible functions that the chosen model can represent via different parameter settings. This could be:

- The set of all linear functions.
- the set of all neural network functios with a certain network architecture,
- etc.

The **capacity** of a model is a measure of how many different functions it can represent.

§3.2 Training and Validation Set

To estimate $R(\theta)$, we split the dataset into two parts, one for learning θ and one for estimating $R(\theta)$.

The first part is called the **training set**, which is used to learn θ. The loss on the training set can inform us whether the empirical risk is high. So, we can use this to make sure the otimization is working.

The second part is called the **validation set**, which is used to diagnose overfitting. The loss on the validation set should be an accurate estimate of the true risk, so we can compare losses on the two sets.

§3.3 Machine Learning Workflow

1. Learn θ on the training set.
   - If the training loss is now low enough, we are underfitting: increase model capacity, improve optimizer, ...
   - Afterwards, go back to step 1.
2. Measure loss on the validation set.
   - If the training loss is much smaller than the validation loss, we are overfitting: decrease model capacity, collect more data, ...
3. If not overfitting or underfitting, done!

§3.4 Regularization

Underfitting is not as much as a concern as overfitting. Some techniques to handle overfitting are:

- Making the network smaller? But we like big models.
- Collect more data? This is fine, but not always possible.
- Add more inductive biases - we can do this via regularization.

A **regularizer** is anything we add to the loss function and/or optimization that does not depend on the data. WE add it to encode some prior belief about what a good model looks like, so it is a form of inductive bias.

Many regularization techniques can be interpreted from a Bayesian perspective as switching from MLE to maximum a posteriori(MAP) estimation.

$$\text{MLE:} \quad \operatorname*{argmax}_{\theta} \sum_{i=1}^{N} \log p_\theta(y_i|x_i)$$

$$\text{MAP:} \quad \operatorname*{argmax}_{\theta} \sum_{i=1}^{N} \log p(y_i|x_i,\theta) + \log p(\theta).$$

For ℓ_2 regularization, we chose $p(\theta) = \mathcal{N}(\theta; 0, \sigma^2 I)$, giving

$$-\log p(\theta) = \sum_{i=1}^{D} \frac{\theta_i^2}{2\sigma^2} + C = \lambda \|\theta\|_2^2, \qquad \lambda = \frac{1}{2\sigma^2}.$$

§3.5 Bias-Variance Tradeoff

$$\mathbb{E}[(f_{\theta(\mathcal{D})}(x') - y)^2] = (\overline{f}(x') - f(x'))^2 + E[(f_{\theta(\mathcal{D})}(x') - \overline{f}(x'))^2] + \sigma^2.$$

- The first term is the Bias^2: how wrong is the model on expectation, regardless of the dataset it is trained on?
- The second term is Variance: regardless of the true function f, how much does the model change based on the training dataset?
- The last term is irreducible error: the noise in the process itself.