Lecture 3: Error Analysis

CS 182 Spring 2021 – Taught by Sergey Levine

Notes by Michael Zhu

Empirical Risk vs True Risk

Risk: probability that your output is wrong

This is quantified by **expected value of loss** under the distribution that your data comes from

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

NOT THE SAME AS TRAINING LOSS. During training, we can't sample $x \sim p(x)$. We just have dataset D and can't generate new samples during training.

$$\text{Empirical risk (from training)} = \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i, \theta)$$

Empirical risk minimization

Supervised learning is usually *empirical* risk minimization.

Question: Is this the same as *true* risk minimization?

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

Not always. Since we are selecting θ based on the empirical risk, the θ we get from training will be **biased** to the empirical risk. This creates a potential issue where the empirical risk is no longer a good approximation of true risk. It possible that we end up with a low empirical risk but a high true risk after training (aka overfitting).

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

- training data fits well
- true function fits poorly
- learned function is very different for different training sets, even if the training sets come from the same distribution

Potential causes:

- can happen if dataset is to small
- can happen if the model is too high capacity (i.e. there are many possible function approximations that can match the data)

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

- traing data fits poorly
- true function fits poorly
- learned function stays the same for different training sets, even if you increase dataset size

Potential causes:

- can happen if the model is too low capacity (i.e. there are no function approximations that match the data well)
- can happen if optmizer is not configured well enough

Mathematical Derivation of Bias and Variance

$$\text{Reminder: } p(D) = \prod_i p(x_i)p(y_i|x_i)$$

Consider the **expected error** of the algorithm with respect to the **distribution of datasets**.

$$E_{D\sim p(D)}[||f_D(x) - f(y)||^2]$$

Where $f_D(x)$ is the function found by the learning algorithm for dataset D and $f(y)$ is the true function.

$$E_{D\sim p(D)}[||f_D(x) - f(y)||^2] = \sum_D p(D)||f_D(x) - f(y)||^2$$

Why is this value useful?

We want to understand how well our **algorithm** does independently of the **particular (random) choice of dataset**.

Note: this is a theoretical exercise, it's not practical to compute this in the real world

$$\text{Let } \bar{f}(x) := E_{D\sim p(D)}[f_D(x)] \quad \text{the average function learned by our algorithm}$$

$$\begin{gathered}
E_{D\sim p(D)}[||f_D(x) - f(y)||^2] \\
= E_{D\sim p(D)}[||f_D(x) - \bar{f}(x) + \bar{f}(x) - f(y)||^2] \\
= E_{D\sim p(D)}[||(f_D(x) - \bar{f}(x)) + (\bar{f}(x) - f(y))||^2] \\
= E_{D\sim p(D)}[||f_D(x) - \bar{f}(x)||^2] + E_{D\sim p(D)}[||\bar{f}(x) - f(y)||^2] + E_{D\sim p(D)}[2(f_D(x) - \bar{f}(x))^T(\bar{f}(x) - f(y))] \\
= E_{D\sim p(D)}[||f_D(x) - \bar{f}(x)||^2] + E_{D\sim p(D)}[||\bar{f}(x) - f(y)||^2] + E_{D\sim p(D)}[2(0)(\bar{f}(x) - f(y))] \\
= E_{D\sim p(D)}[||f_D(x) - \bar{f}(x)||^2] + E_{D\sim p(D)}[||\bar{f}(x) - f(y)||^2] + 0 \\
= E_{D\sim p(D)}[||f_D(x) - \bar{f}(x)||^2] + E_{D\sim p(D)}[||\bar{f}(x) - f(y)||^2]
\end{gathered}$$

$$\begin{gathered}
= E_{D\sim p(D)}[||f_D(x) - \bar{f}(x)||^2] + ||\bar{f}(x) - f(y)||^2 \\
\text{Expected Error} = \text{Variance} + \text{Bias}^2
\end{gathered}$$

$$\begin{gathered}
E_{D\sim p(D)}[||f_D(x) - \bar{f}(x)||^2] \rightarrow \text{Variance} \\
||\bar{f}(x) - f(y)||^2 \rightarrow \text{Bias}^2
\end{gathered}$$

Expected Error = Variance + Bias^2

Variance:

How much does the algorithm's predicted function change with the dataset (difference from the average function)?

If **variance** is too high $\rightarrow$ **overfitting**

Bias^2:

How far off is the algorithm's average function to the true function?

If **bias** is too high $\rightarrow$ **underfitting**

Key Question: how to regulate the tradeoff between variance and bias?

Usually when you decrease one the other increases

Regularization

Bayesian interpretation: a prior belief about our parameters

When we have **high variance**, it's because the data **doesn't give enough information** to pick one of the many possible functions that are **equally good** based on empirical risk

We can provide **more information** to our model by adding regularization to the loss function.

Question: Given dataset D, what is the most likely θ?

$$p(\theta|D) = \frac{p(\theta, D)}{p(D)} \propto p(\theta, D) = p(D|\theta)p(\theta)$$

$$\text{Remember: } p(D|\theta) = \prod_i p(x_i)p_\theta(y_i|x_i)$$

So what is $p(\theta)$? It's the probability of θ before we've seen the dataset D (aka the *prior*)

$$\text{Old loss func: } L(\theta) = -\sum_i \log p_\theta(y_i|x_i)$$

$$\text{New loss func: } L(\theta) = -\sum_i \log p_\theta(y_i|x_i) + \log p(\theta)$$

We add regularization to the loss function in the form of the *prior* $\log p(\theta)$

How do we pick $p(\theta)$? To keep things simple, we can represent it as a **normal distribution** with a **mean of zero**.

$$p(\theta) = N(0, \sigma^2)$$

$$\text{If we set } \log p(\theta) = \lambda||\theta||^2, \text{ we'll find that } \lambda = 1/2\sigma^2$$

So λ is **one divided by variance times half**. You can derive this from the formula of $\log p(\theta)$ when $p(\theta) = N(0, \sigma^2)$

In practice, λ is a **hyperparameter** that we can pick to set the **variance of the normal distribution** of $p(\theta)$ based on our **prior knowledge/bias** that we want to impose on the loss function

This is also known as "**L2 regularization**" and "**weight decay**", because it generally forces θ to take on smaller values which makes the learned function **smoother**

$$L(\theta) = -\sum_i \log p_\theta(y_i|x_i) + \lambda||\theta||^2$$

Other types of regularizers

- $\lambda|\theta|$ taking the **absolute value** of θ instead of its square
- also known as "**L1 regularization**"
- creates a preference for **weight sparsity**, meaning that it will try to zero out the less useful dimensions

Dropout: a special regularizer for neural networks where you randomly remove nodes during training

Gradient penalty: a special regularizer for GANs

And many more ...

Machine Learning Workflow

Define $L(\theta, D_{train})$ as loss on training set

Define $L(\theta, D_{val})$ as loss on validation set

Note that $L(\theta, D_{train})$ only gives us information on **bias (underfitting)** and not **variance (overfitting)**, which is why we need $L(\theta, D_{val})$ to observe **variance**

1. Split dataset into training, validation, and test sets
   - **training set** $\rightarrow$ used to select θ and optimizer hyperparameters (i.e. learning rate α)
   - **validation set** $\rightarrow$ used to select model class (i.e. neural net architecture) and regularizer hyperparameters (i.e. λ)
   - **test set** $\rightarrow$ used for final evaluation
2. Train θ with $L(\theta, D_{train})$
   - if $L(\theta, D_{train})$ is low $\rightarrow$ **underfitting** $\rightarrow$ decrease regularization, improve optimizer, etc
3. Observe $L(\theta, D_{val})$
   - if $L(\theta, D_{val}) >> L(\theta, D_{train})$ $\rightarrow$ **overfitting** $\rightarrow$ increase regularization, tweak model class, etc.
4. Repeat steps 2 and 3 until you reach an acceptable loss or error (Reminder: error = variance + bias^2)
5. Report **final performance** based on **test set**
   - Think about why we can't use training set or validation set to evaluate final performance (they're **unbiased estimators** now since we've used them for training and hyperparameter optimization!)