Will we get the right answer?
Empirical risk and true risk

zero-one loss: \( \sum_i \delta(f_\theta(x_i) \neq y_i) \)

Risk: probability you will get it wrong
expected value of our loss quantifies this
can be generalized to other losses
(e.g., NLL)

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

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

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

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

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)
Let’s analyze error!

Last time, we discussed classification

This time, we’ll focus on regression

All this stuff applies to classification too, it’s just simpler to derive for regression

$$\log p_\theta(y|x) = \mathcal{N}(f_\theta(x), \Sigma_\theta(x)) = -\frac{1}{2}(f_\theta(x) - y)\Sigma_\theta(x)^{-1}(f_\theta(x) - y) - \frac{1}{2} \log |\Sigma_\theta(x)| + \text{const}$$

If $\Sigma_\theta(x) = I$

$$= -\frac{1}{2}||f_\theta(x) - y||^2 + \text{const}$$
Let’s analyze error!

\[
\log p_\theta(y|x) = -\frac{1}{2}||f_\theta(x) - y||^2 + \text{const} \quad \text{if } \Sigma_\theta(x) = \mathbf{I}
\]

Also the same as the mean squared error (MSE) loss!

\[
\mathcal{L}(\theta, x, y) = -\frac{1}{2}||f_\theta(x) - y||^2
\]

\hspace{1cm} \text{a bit easier to analyze, but we can analyze other losses too}

**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)
Let’s analyze error!

\[ \mathcal{L}(\theta, x, y) = -\frac{1}{2} ||f_\theta(x) - y||^2 \]

Let’s try to understand **overfitting** and **underfitting** more formally.

**Question:** how does the error change for different training sets?

Why is this question important?

**overfitting**
- The training data is fitted well
- The true function is fitted poorly
- The learned function looks different each time!

**underfitting**
- The training data is fitted poorly
- The true function is fitted poorly
- The learned function looks similar, even if we pool together all the datasets!
Let’s analyze error!

\[ \mathcal{L}(\theta, x, y) = -\frac{1}{2} \| f_\theta(x) - y \|^2 \]

What is the expected error, given a distribution over datasets?

\[ \mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \]
\[ \sim p(x) \]
\[ p(\mathcal{D}) = \prod_{i=1}^{N} p(x_i)p(y_i|x_i) \]

\[ y \sim p(y|x) \]
\[ f_{\mathcal{D}}(x) \sim f(x) \]

\[ E_{\mathcal{D} \sim p(\mathcal{D})}[\| f_\theta(x) - y \|^2] = \sum_{\mathcal{D}} p(\mathcal{D})\| f_{\mathcal{D}}(x) - f(x) \|^2 \]

expected value of error w.r.t. data distribution

sum over all possible datasets
Let’s analyze error!

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

Why do we care about this quantity?

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

This is very important if we want to improve our algorithm!
Bias-variance tradeoff

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

Let \( \bar{f}(x) = E_{D \sim p(D)}[f_D(x)] \)

\[ = E_{D \sim p(D)} \left[ \| f_D(x) - \bar{f}(x) + \bar{f}(x) - f(x) \|^2 \right] \]

\[ = E_{D \sim p(D)} \left[ \| (f_D(x) - \bar{f}(x)) + (\bar{f}(x) - f(x)) \|^2 \right] \]

\[ = E_{D \sim p(D)} \left[ \| f_D(x) - \bar{f}(x) \|^2 \right] + E_{D \sim p(D)} \left[ \| \bar{f}(x) - f(x) \|^2 \right] \]

\[ = E_{D \sim p(D)} \left[ 2 \left( f_D(x) - \bar{f}(x) \right) (\bar{f}(x) - f(x)) \right] \]

\[ = 0 \]

\[ 2E \left[ (f_D(x) - \bar{f}(x)) \right] (f(x) - \bar{f}(x)) \]
Bias-variance tradeoff

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

\[ = E_{D \sim p(D)}[\|f_D(x) - \bar{f}(x)\|^2] + E_{D \sim p(D)}[\|\bar{f}(x) - f(x)\|^2] \]

Variance

Regardless of what the true function is, how much does our prediction change with dataset?

\[ \|\bar{f}(x) - f(x)\|^2 \]

Bias^2

This error doesn’t go away no matter how much data we have!
Bias-variance tradeoff

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

let \( \bar{f}(x) = E_{D \sim p(D)}[f_D(x)] \)

\[ = E_{D \sim p(D)} \left[ \| f_D(x) - \bar{f}(x) \|^2 \right] + E_{D \sim p(D)} \left[ \| \bar{f}(x) - f(x) \|^2 \right] \]

\[ = \text{Variance} + \text{Bias}^2 \]

If \textbf{variance} is too high, we have too little data/too complex a function class/etc. \( \Rightarrow \) this is \textbf{overfitting}

If \textbf{bias} is too high, we have an insufficiently complex function class \( \Rightarrow \) this is \textbf{underfitting}

How do we \textbf{regulate} the bias-variance tradeoff?
Regularization
How to regulate bias/variance?

Get more data
addresses variance
has no effect on bias

Change your model class  e.g., 12\textsuperscript{th} degree polynomials to linear functions

Can we “smoothly” restrict the model class?

Can we construct a “continuous knob” for complexity?
Regularization

Regularization: something we add to the loss function to reduce variance

Bayesian interpretation: could be regarded as a prior on parameters (but this is not the only interpretation!)

High level intuition:

When we have **high variance**, it’s because the data doesn’t give enough information to identify parameters

If there is not enough information in the **data**, can we give **more information** through the loss function?

If we provide enough **information** to **disambiguate** between (almost) equally good models, we can pick the best one

---

all of these solutions have zero training error
The Bayesian perspective

Regularization: something we add to the loss function to reduce variance

Bayesian interpretation: could be regarded as a prior on parameters (but this is not the only interpretation!)

Question: Given \( \mathcal{D} \), what is the most likely \( \theta \)? \( \mathcal{D} = \{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \} \)

\[
p(\theta | \mathcal{D}) = \frac{p(\theta, \mathcal{D})}{p(\mathcal{D})} \propto p(\theta, \mathcal{D}) = p(\mathcal{D} | \theta)p(\theta)
\]

Can we pick a prior that makes the smoother function more likely?

we’ve seen this part before!

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

remember: this is just shorthand for \( p(y_i | x_i, \theta) \)
The Bayesian perspective

Regularization: something we add to the loss function to reduce variance

Bayesian interpretation: could be regarded as a prior on parameters (but this is not the only interpretation!)

Question: Given \( \mathcal{D} \), what is the most likely \( \theta \)?

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

\[
p(\theta | \mathcal{D}) = \frac{p(\theta, \mathcal{D})}{p(\mathcal{D})} \propto p(\theta, \mathcal{D}) = p(\mathcal{D} | \theta) p(\theta)
\]

New loss function: we choose this bit

\[
\mathcal{L}(\theta) = - \left( \sum_{i=1}^{N} \log p(y_i | x_i, \theta) \right) - \log p(\theta)
\]
Example: regularized linear regression

Can we pick a prior that makes the smoother function more likely?

\[ \mathcal{L}(\theta) = -\left( \sum_{i=1}^{N} \log p(y_i | x_i, \theta) \right) - \log p(\theta) \]

what kind of distribution assigns higher probabilities to small numbers?

Simple idea: \( p(\theta) = \mathcal{N}(0, \sigma^2) \)

example: linear regression with polynomial features

\[ f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \ldots \]

this kind of thing typically requires large coefficients

if we only allow small coefficients, best fit might be more like this
Example: regularized linear regression

Can we pick a prior that makes the smoother function more likely?

\[
\mathcal{L}(\theta) = -\left( \sum_{i=1}^{N} \log p(y_i|x_i, \theta) \right) - \log p(\theta)
\]

what kind of distribution assigns higher probabilities to small numbers?

Simple idea: \( p(\theta) = \mathcal{N}(0, \sigma^2) \)

\[
\mathcal{L}(\theta) = \left( \sum_{i=1}^{N} \| f_\theta(x_i) - y_i \|^2 \right) + \lambda \| \theta \|^2
\]

if \( \Sigma_\theta(x) = I \)

(but we don’t care, we’ll just select it directly)
Example: regularized logistic regression

what we wanted

what we got

P(blue) = P(red) = 0.5

larger P(red)

larger P(blue)

technically every point is classified correctly
Example: regularized logistic regression

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

Simple idea: \( p(\theta) = \mathcal{N}(0, \sigma^2) \)

Larger \( \theta \) values \( \Rightarrow \) sharper probabilities

Solution: prefer smaller \( \theta \)!
Example: regularized logistic regression

\[ \mathcal{L}(\theta) = - \left( \sum_{i=1}^{N} \log p(y_i | x_i, \theta) \right) + \lambda \|\theta\|^2 \]

same prior, but now for a classification problem
this is sometimes called weight decay

Other examples of regularizers (we’ll discuss some of these later):

\[ \lambda \sum_{i=1}^{D} |\theta_i| \]
creates a preference for zeroing out dimensions!

Dropout: a special type of regularizer for neural networks

Gradient penalty: a special type of regularizer for GANs

...lots of other choices
Other perspectives

**Regularization:** something we add to the loss function to reduce variance

**Bayesian perspective:** the regularizer is prior knowledge about parameters

**Numerical perspective:** the regularizer makes underdetermined problems well-determined

**Optimization perspective:** the regularizer makes the loss landscape easier to search
  
  paradoxically, regularizers can sometimes reduce **underfitting** if it was due to poor optimization!

  especially common with GANs

In machine learning, any “heuristic” term added to the loss that doesn’t depend on data is generally called a regularizer

\[\mathcal{L} = -\lambda ||\theta||^2 + \text{const}\]

“hyperparameter”

Regularizers introduce **hyperparameters** that we have to select in order for them to work well
Training sets and test sets
Some questions...

How do we **know** if we are overfitting or underfitting?

How do we select which algorithm to use?

How do we select **hyperparameters**?

**One idea:** choose whatever makes the **loss** low

\[ \mathcal{L}(\theta) = 0 \]

Can’t diagnose **overfitting** by looking at the training loss!
The machine learning workflow

the dataset

training set

use this for training

\[ \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \]

validation set

reserve this for...

...selecting hyperparameters
...adding/removing features
...tweaking your model class

1. Train \( \theta \) with \( \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \)
   if \( \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \) not low enough
   you are **underfitting**
   decrease regularization
   improve your optimizer

2. Look at \( \mathcal{L}(\theta, \mathcal{D}_{\text{val}}) \)
   if \( \mathcal{L}(\theta, \mathcal{D}_{\text{val}}) \gg \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \)
   you are **overfitting**
   increase regularization

\[ \mathcal{L}(\theta, \mathcal{D}_{\text{val}}) \]
The machine learning workflow

**the dataset**

**used to select...**

$\theta$ (via optimization)

optimizer hyperparameters (e.g., learning rate $\alpha$)

**used to select...**

model class (e.g., logistic regression vs. neural net)

regularizer hyperparameters

which features to use
Learning curves

Question: can we stop here?
How do we know when to stop?
The final exam

We followed the recipe, **now what?**

How good is our final classifier?

\[ \mathcal{L}(\theta, \mathcal{D}_{\text{val}}) \]

That’s *no good* – we already used the validation set to pick hyperparameters!

What if we reserve another set for a **final exam** (a kind of... validation validation set!)

1. Train \( \theta \) with \( \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \)
   
   if \( \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \) not low enough
   
   you are **underfitting**
   
   decrease regularization
   
   improve your optimizer

2. Look at \( \mathcal{L}(\theta, \mathcal{D}_{\text{val}}) \)
   
   if \( \mathcal{L}(\theta, \mathcal{D}_{\text{val}}) \gg \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \)
   
   you are **overfitting**
   
   increase regularization
The machine learning workflow

- **the dataset**
- **training set**
- **validation set**
- **test set**

**used to select...**

- $\theta$ (via optimization)
- optimizer hyperparameters (e.g., learning rate $\alpha$)

**used to select...**

- model class (e.g., logistic regression vs. neural net)
- regularizer hyperparameters
- which features to use

**Used only to report final performance**
Summary and takeaways

➢ Where do errors come from?
  ▪ Variance: too much capacity, not enough information in the data to find the right parameters
  ▪ Bias: too little capacity, not enough representational power to represent the true function
  ▪ Error = Variance + Bias^2
  ▪ Overfitting = too much variance
  ▪ Underfitting = too much bias

➢ How can we trade off bias and variance?
  ▪ Select your model class carefully
  ▪ Select your features carefully
  ▪ Regularization: stuff we add to the loss to reduce variance

➢ How do we select hyperparameters?
  ▪ Training/validation split
  ▪ Training set is for optimization (learning)
  ▪ Validation set is for selecting hyperparameters
  ▪ Test set is for reporting final results and nothing else!