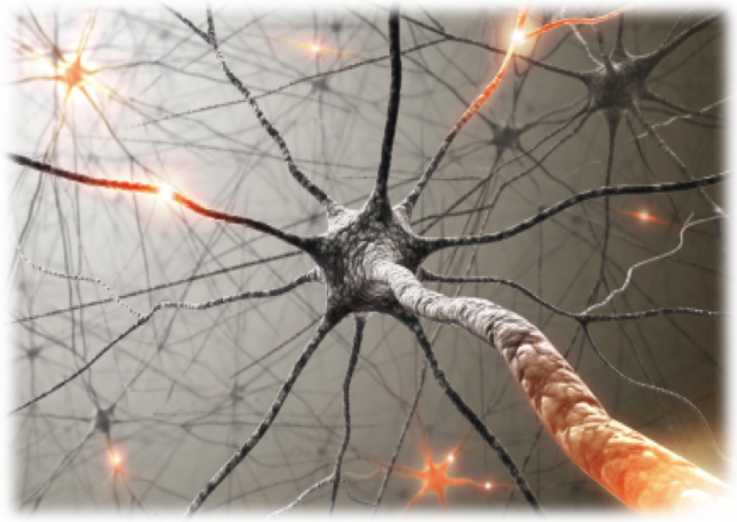
Machine learning: generalization



Minimizing training loss

Hypothesis class:

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$

Training objective (loss function):

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Optimization algorithm:

stochastic gradient descent

Is the training loss a good objective to optimize?

A strawman algorithm

Algorithm: rote learning

Training: just store $\mathcal{D}_{\text{train}}$.

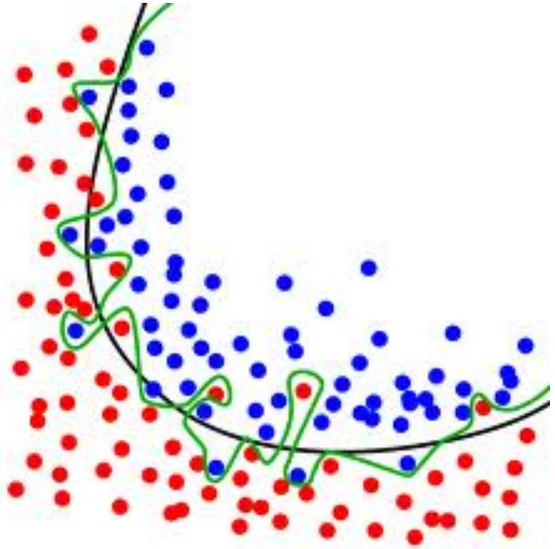
Predictor $f(x)$:

If $(x, y) \in \mathcal{D}_{\text{train}}$: return y.

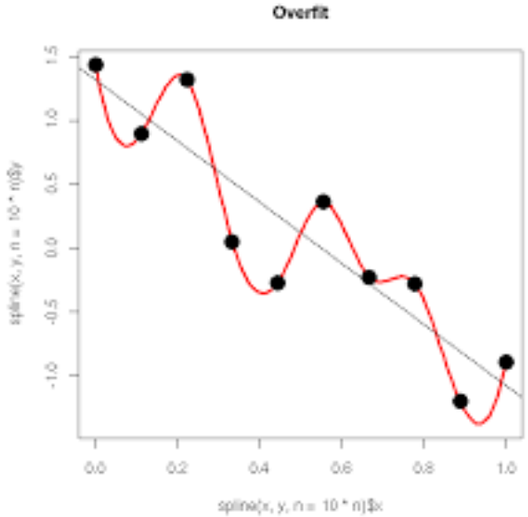
Else: **segfault**.

Minimizes the objective perfectly (zero), but clearly bad...

Overfitting pictures



Classification



Regression

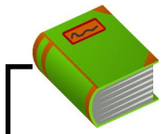
Evaluation

How good is the predictor f?

Key idea: the real learning objective

Our goal is to minimize **error on unseen future examples**.

Don't have unseen examples; next best thing:



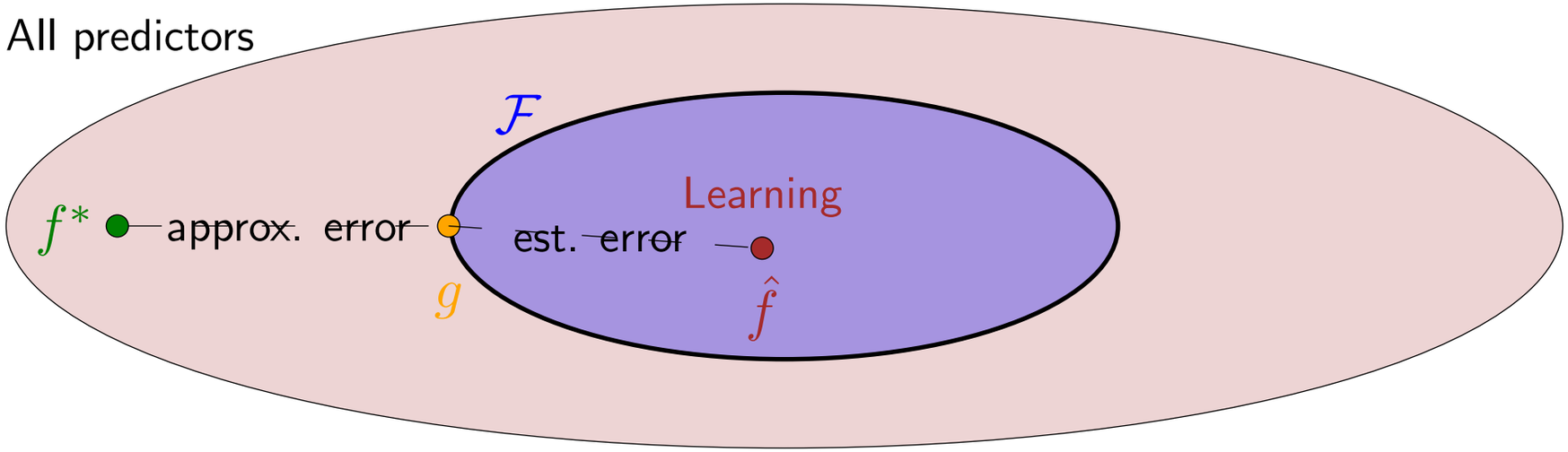
Definition: test set

Test set $\mathcal{D}_{\text{test}}$ contains examples not used for training.

Generalization

When will a learning algorithm **generalize** well?

Approximation and estimation error

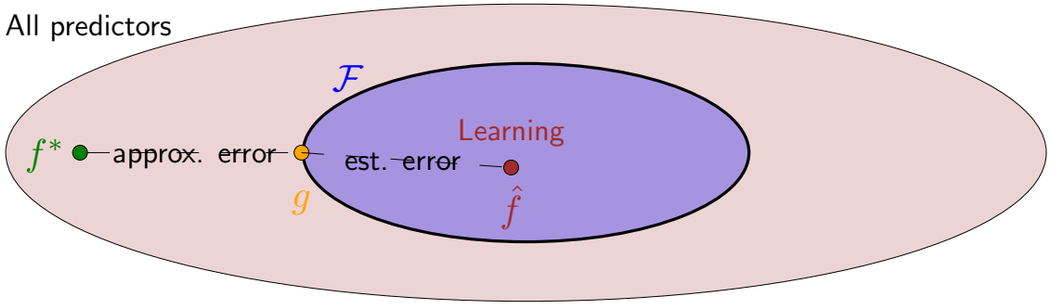


- Approximation error: how good is the hypothesis class?
- Estimation error: how good is the learned predictor **relative to** the potential of the hypothesis class?

$$\text{Err}(\hat{f}) - \text{Err}(f^*) = \underbrace{\text{Err}(\hat{f}) - \text{Err}(g)}_{\text{estimation}} + \underbrace{\text{Err}(g) - \text{Err}(f^*)}_{\text{approximation}}$$

- Here's a cartoon that can help you understand the balance between fitting and generalization. Out there somewhere, there is a magical predictor f^* that classifies everything perfectly. This predictor is unattainable; all we can hope to do is to use a combination of our domain knowledge and data to approximate that. The question is: how far are we away from f^*?
- Recall that our learning framework consists of (i) choosing a hypothesis class $\mathcal{F}$ (e.g., by defining the feature extractor) and then (ii) choosing a particular predictor $\hat{f}$ from $\mathcal{F}$.
- **Approximation error** is how far the entire hypothesis class is from the target predictor f^*. Larger hypothesis classes have lower approximation error. Let $g \in \mathcal{F}$ be the best predictor in the hypothesis class in the sense of minimizing test error $g = \arg\min_{f \in \mathcal{F}} \text{Err}(f)$. Here, distance is just the differences in test error: $\text{Err}(g) - \text{Err}(f^*)$.
- **Estimation error** is how good the predictor $\hat{f}$ returned by the learning algorithm is with respect to the best in the hypothesis class: $\text{Err}(\hat{f}) - \text{Err}(g)$. Larger hypothesis classes have higher estimation error because it's harder to find a good predictor based on limited data.
- We'd like both approximation and estimation errors to be small, but there's a tradeoff here.

Effect of hypothesis class size



As the hypothesis class size increases...

Approximation error decreases because:

taking min over larger set

Estimation error increases because:

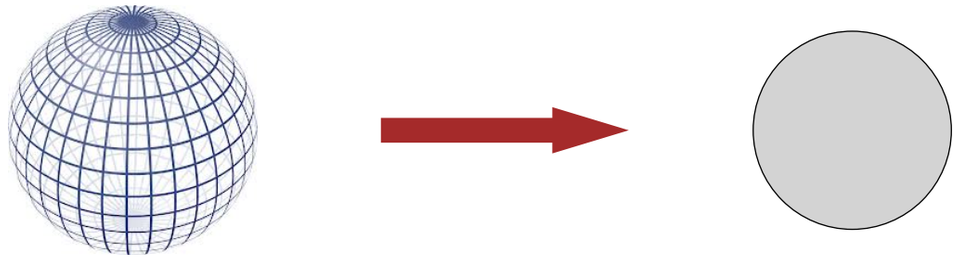
harder to estimate something more complex

How do we control the hypothesis class size?

Strategy 1: dimensionality

$$\mathbf{w} \in \mathbb{R}^d$$

Reduce the dimensionality d (number of features):



Controlling the dimensionality

Manual feature (template) selection:

- Add feature templates if they help
- Remove feature templates if they don't help

Automatic feature selection (beyond the scope of this class):

- Forward selection
- Boosting
- L_1 regularization

It's the number of features that matters

Strategy 2: norm

$$\mathbf{w} \in \mathbb{R}^d$$

Reduce the norm (length) $\|\mathbf{w}\|$:

Controlling the norm

Regularized objective:

$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) + \lambda \mathbf{w})$$

Same as gradient descent, except shrink the weights towards zero by λ.

Controlling the norm: early stopping

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Idea: simply make T smaller

Intuition: if have fewer updates, then $\|\mathbf{w}\|$ can't get too big.

Lesson: try to minimize the training error, but don't try too hard.

Summary

Not the real objective: training loss

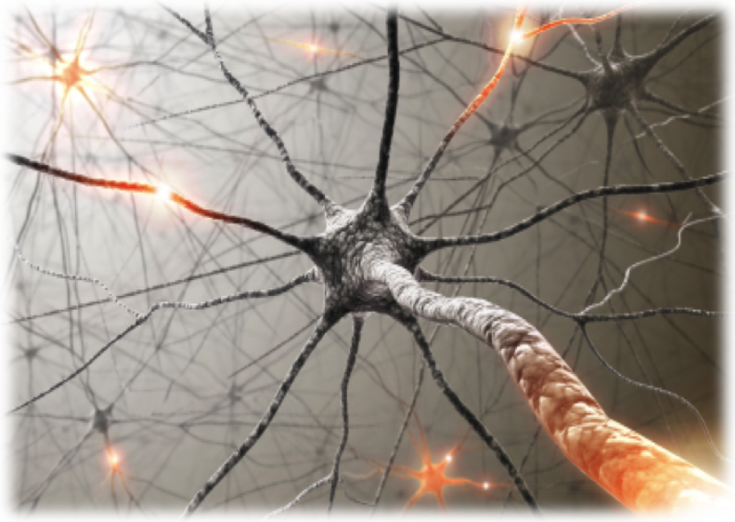
Real objective: loss on unseen future examples

Semi-real objective: test loss

Key idea: keep it simple

Try to minimize training error, but keep the hypothesis class small.

Machine learning: best practices



Choose your own adventure

Hypothesis class:

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$$

Feature extractor ϕ: linear, quadratic

Architecture: number of layers, number of hidden units

Training objective:

$$\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w}) + \text{Reg}(\mathbf{w})$$

Loss function: hinge, logistic

Regularization: none, L2

Optimization algorithm:

Algorithm: stochastic gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

  For $(x, y) \in \mathcal{D}_{\text{train}}$:

    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{V}, \mathbf{w})$

Number of epochs

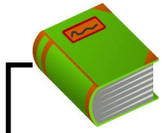
Step size: constant, decreasing, adaptive

Initialization: amount of noise, pre-training

Batch size

Dropout

Hyperparameters



Definition: hyperparameters

Design decisions (hypothesis class, training objective, optimization algorithm) that need to be made before running the learning algorithm.

How do we choose hyperparameters?

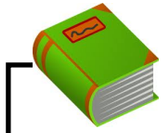
Choose hyperparameters to minimize $\mathcal{D}_{\text{train}}$ error?

No - optimum would be to include all features, no regularization, train forever

Choose hyperparameters to minimize $\mathcal{D}_{\text{test}}$ error?

No - choosing based on $\mathcal{D}_{\text{test}}$ makes it an unreliable estimate of error!

Validation set



Definition: validation set

A **validation set** is taken out of the training set and used to optimize hyperparameters.

For each setting of hyperparameters, train on $\mathcal{D}_{\text{train}} \setminus \mathcal{D}_{\text{val}}$, evaluate on $\mathcal{D}_{\text{val}}$

Model development strategy

Algorithm: Model development strategy

- Split data into train, validation, test
- Look at data to get intuition
- Repeat:
  - Implement model/feature, adjust hyperparameters
  - Run learning algorithm
  - Sanity check train and validation error rates
  - Look at weights and prediction errors
- Evaluate on test set to get final error rates

Tips

Start simple:

- Run on small subsets of your data or synthetic data
- Start with a simple baseline model
- Sanity check: can you overfit 5 examples

Log everything:

- Track training loss and validation loss over time
- Record hyperparameters, statistics of data, model, and predictions
- Organize experiments (each run goes in a separate folder)

Report your results:

- Run each experiment multiple times with different random seeds
- Compute multiple metrics (e.g., error rates for minority groups)

Summary

Don't look at the test set!

Understand the data!

Start simple!

Practice!