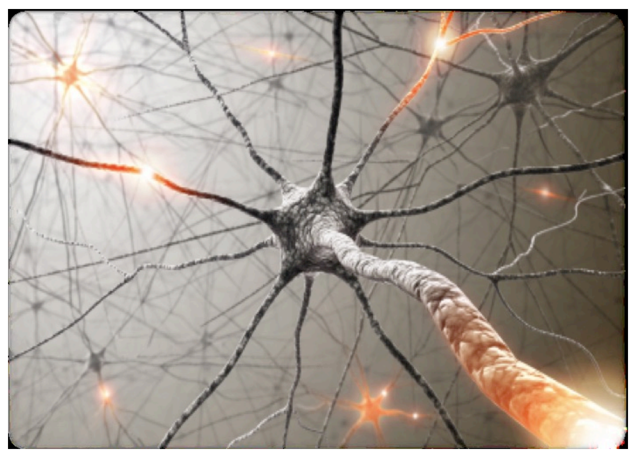
02 Linear Predictor



Question

- What's the true objective of machine learning?
  - Minimize error on the training set
  - Minimize training error with regularization
  - Minimize error on unseen future examples
  - Learn about machines

Review

Feature extractor ϕ

abc@gmail.com $\xrightarrow[\text{arbitrary!}]{\text{feature extractor}}$

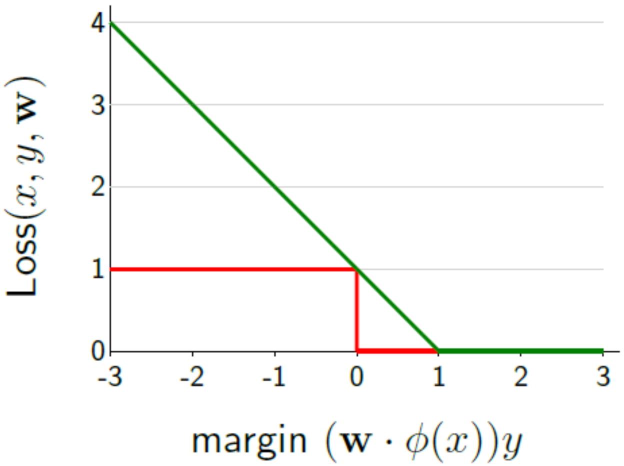
Length > 10	:1
fracOfAlpha	:0.85
contains_@	: 1
endsWith_com	: 1
endsWith_org	: 0

Predict score:

- Linear predictor: $\text{score} = \mathbf{w} \cdot \phi(x)$
- Neural network: $\text{score} = \sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x))$

Review

Loss function $\text{Loss}(x, y, \mathbf{w})$ ：



(for binary classification)

Optimization algorithm ：stochastic gradient descent

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

Training error

- Loss minimization:

$$\min_{\mathbf{w}} \mathrm{TrainLoss}(\mathbf{w})$$

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

- Is the training loss a good objective to optimize?

A strawman algorithm

Algorithm: strawman

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

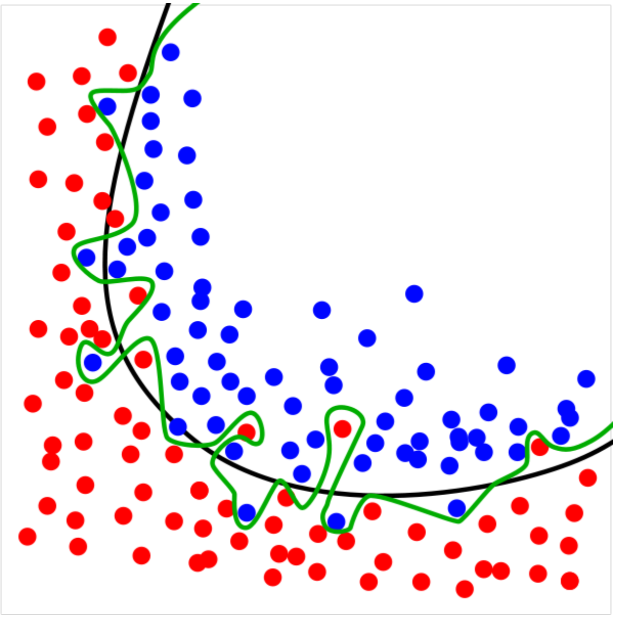
Predictor f(x):

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

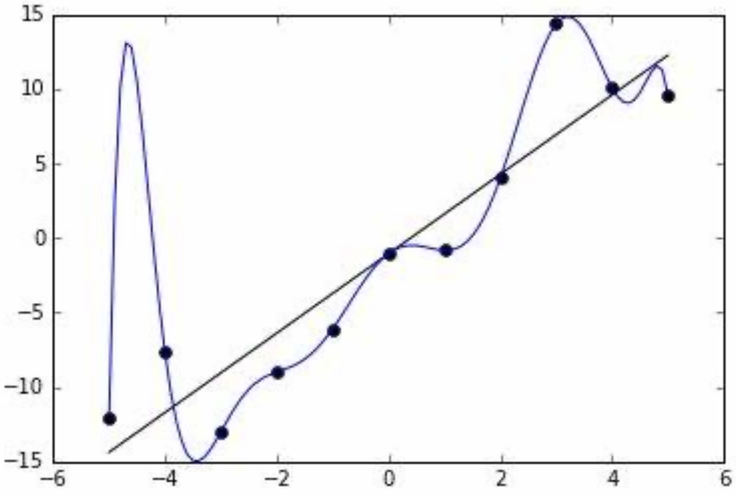
Else: segfault

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

Overfitting pictures



Classification



Regression

Evaluation

$$\mathcal{D}_{\text{train}} \longrightarrow \boxed{\text{Learner}} \longrightarrow f$$

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

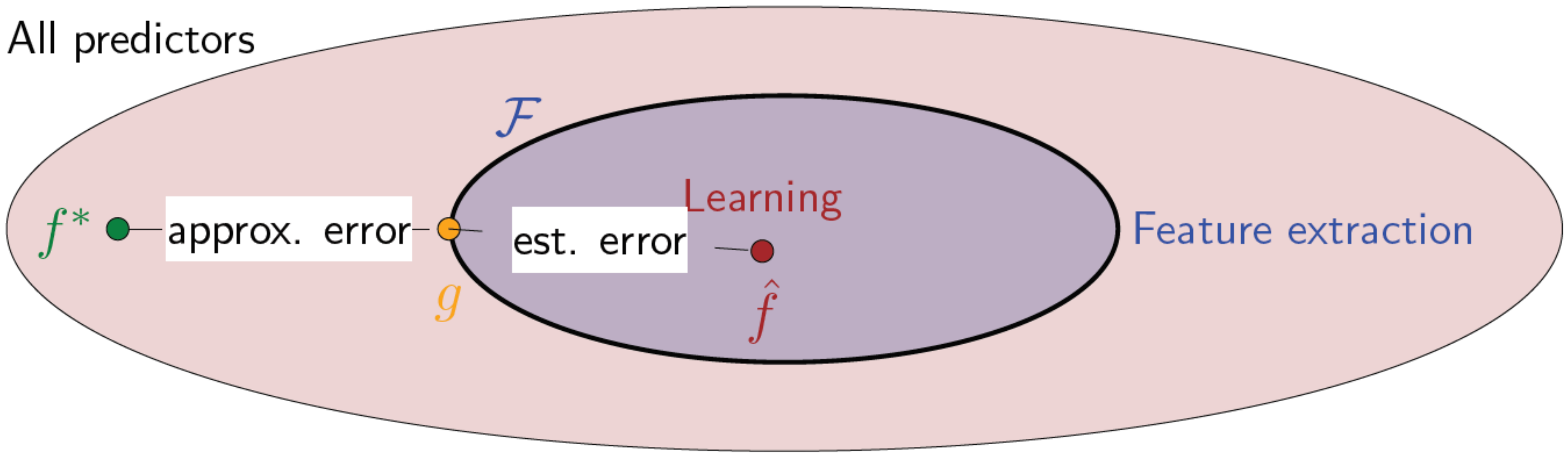
Test set $\mathcal{D}_{\text{test}}$ contains examples not used for training.

Generalization

- When will a learning algorithm generalize well?

$$\boxed{\mathcal{D}_{\text{train}}} \longrightarrow \boxed{\mathcal{D}_{\text{test}}}$$

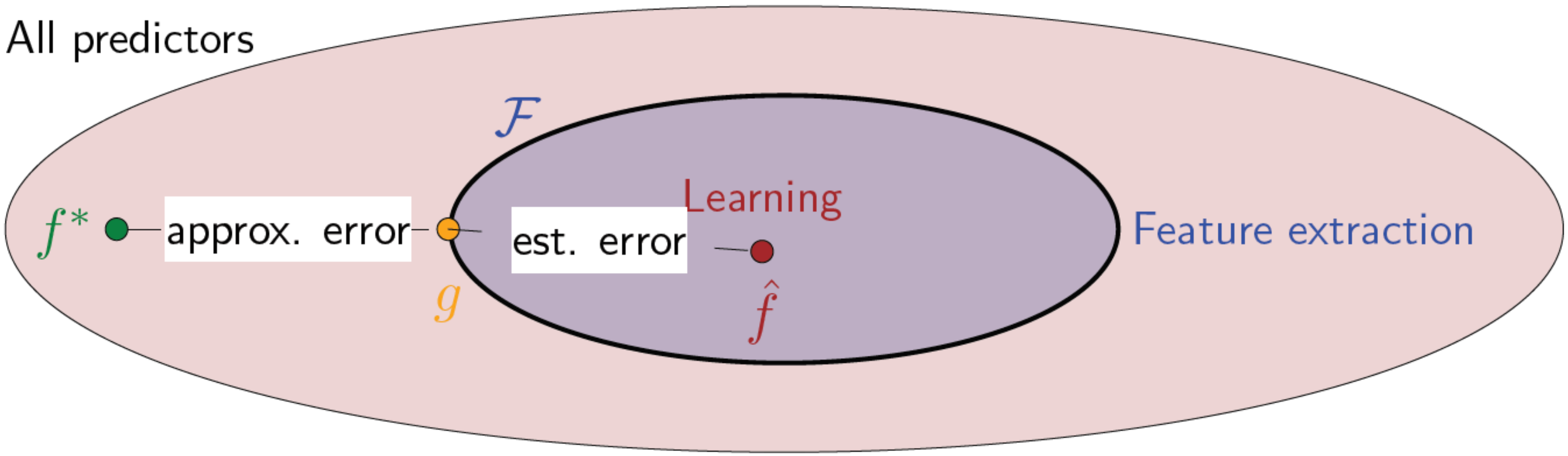
Approximation and estimation error



- Approximation error: how good is the hypothesis class?
- Estimation error: how good is the learned predictor with respect to the hypothesis class?

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

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

Estimation error analogy

Scenario 1: ask few people around

Is your name Joe?

Scenario 2: email all of HUNU

Is your name Joe?

people = hypotheses, questions = examples

Controlling size of hypothesis class

Linear predictors are specified by weight vector $\mathbf{w} \in \mathbb{R}^d$

Keeping the dimensionality d small:

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

Controlling the dimensionality

Manual feature (template) selection:

- Add features if they help
- Remove features if they don't help

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

- Forward selection
- Boosting
- L_1 regularization

Controlling the norm: regularization

Regularized objective:

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

Algorithm: gradient descent (with regularization)

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.

Note: SVM = hinge loss + regularization

Controlling the norm: early stopping

Algorithm: gradient descent

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

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

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

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 so far

Not the real objective: training loss

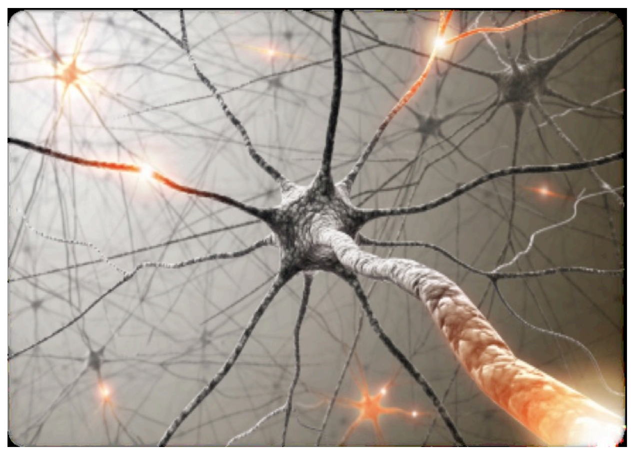
Real objective: loss on unseen future examples

Semi-real objective: test loss

Key idea: minimize training loss

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

Machine learning: best practices



Choose your own adventure

Hypothesis class:

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

Feature extractor φ: linear, quadratic

Architecture: number of layers, number of hidden units

Training objective:

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

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 $(x, y) \in D_{\text{train}}$:

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

Number of epochs

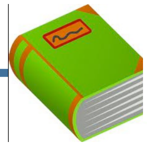
Step size: constant, decreasing, adaptive

Initialization: amount of noise, pre-training

Batch size

Dropout

Hyper-parameters



Definition: Hyper-parameters

Properties of the learning algorithm (features, regularization parameter λ, number of iterations T, step size η, etc.).

How do we choose hyper-parameters?

Choose hyper-parameters to minimize $\mathcal{D}_{\text{train}}$ error? **No** - solution would be to include all features, set $\lambda = 0, T \rightarrow \infty$.

Choose hyper-parameters to minimize $\mathcal{D}_{\text{test}}$ error? **No** - choosing based on $\mathcal{D}_{\text{test}}$ makes it an unreliable estimate of error!

Validation

Problem: can't use test set!

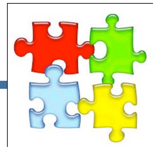
Solution: randomly take out 10-50% of training data and use it instead of the test set to estimate test error.

$\mathcal{D}_{\text{train}} \setminus \mathcal{D}_{\text{val}}$	$\mathcal{D}_{\text{val}}$		$\mathcal{D}_{\text{test}}$

Definition: validation set

A **validation (development) set** is taken out of the training data which acts as a surrogate for the test set.

Development cycle



Problem: simplified name-entity recognition

Input: a string x (e.g., President [Barack Obama] in)

Output: y, whether x contains a person or not (e.g., +1)

Algorithm: recipe for success

- Split data into train, validation, test
- Look at data to get intuition
- Repeat:
  - Implement feature / tune hyper-parameters
  - Run learning algorithm
  - Sanity check train and validation error rates, weights
  - Look at errors to brainstorm improvements
- Run on test set to get final error rates