Lecture 4: Machine Learning 3

Roadmap

Backpropagation

K-means

Generalization

Best practices

Summary of Machine Learning

Motivation: regression with four-layer neural networks

Loss on one example:

$$\text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w}) = (\mathbf{w} \cdot \sigma(\mathbf{V}_3 \sigma(\mathbf{V}_2 \sigma(\mathbf{V}_1 \phi(x)))) - y)^2$$

(Stochastic) gradient descent:

$$\mathbf{V}_1 \leftarrow \mathbf{V}_1 - \eta \nabla_{\mathbf{V}_1} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

$$\mathbf{V}_2 \leftarrow \mathbf{V}_2 - \eta \nabla_{\mathbf{V}_2} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

$$\mathbf{V}_3 \leftarrow \mathbf{V}_3 - \eta \nabla_{\mathbf{V}_3} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

How to get the gradient without doing manual work?

Computation graphs

$$\text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w}) = (\mathbf{w} \cdot \sigma(\mathbf{V}_3 \sigma(\mathbf{V}_2 \sigma(\mathbf{V}_1 \phi(x)))) - y)^2$$

Definition: computation graph

A directed acyclic graph whose root node represents the final mathematical expression and each node represents intermediate subexpressions.

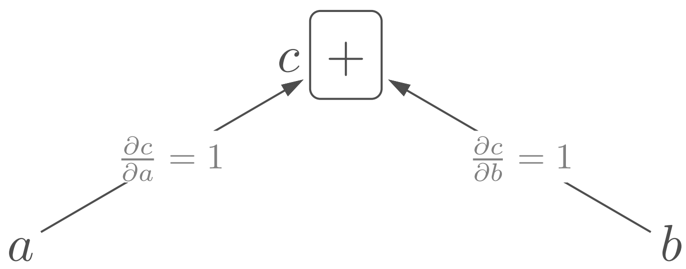
Upshot: compute gradients via general **backpropagation** algorithm

Purposes:

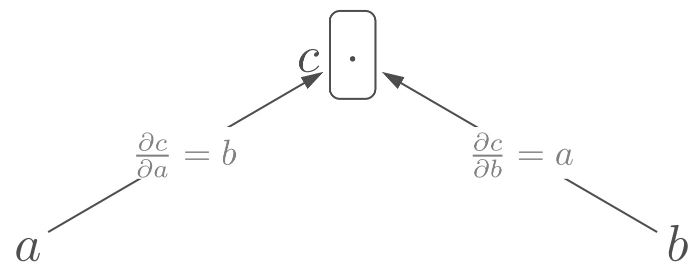
- Automatically compute gradients (how TensorFlow and PyTorch work)
- Gain insight into modular structure of gradient computations

Functions as boxes

$$c = a + b$$



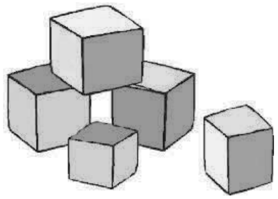
$$c = a \cdot b$$



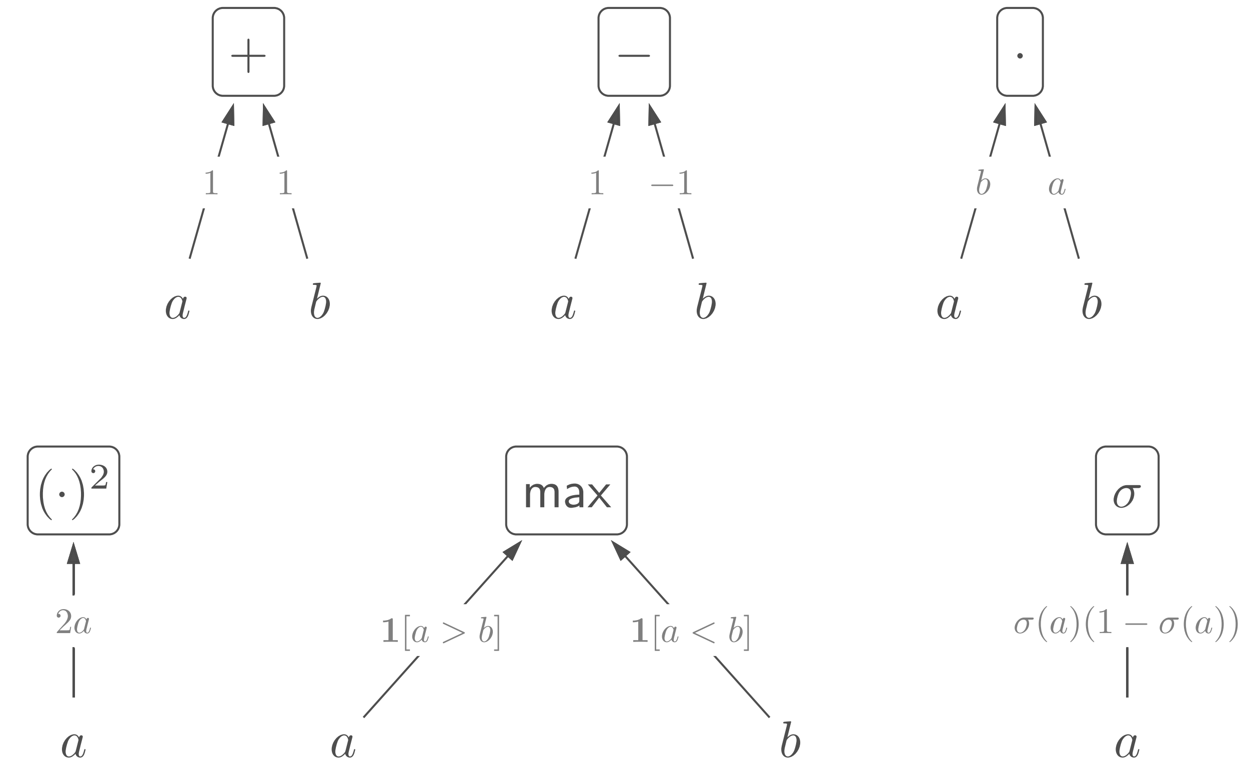
$$(a + \epsilon) + b = c + 1\epsilon$$
$$a + (b + \epsilon) = c + 1\epsilon$$

$$(a + \epsilon)b = c + b\epsilon$$
$$a(b + \epsilon) = c + a\epsilon$$

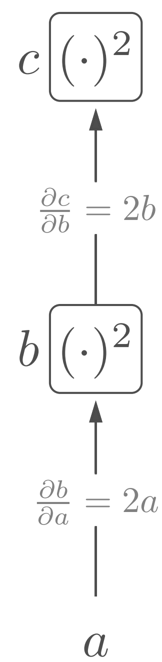
Gradients: how much does c change if a or b changes?



Basic building blocks



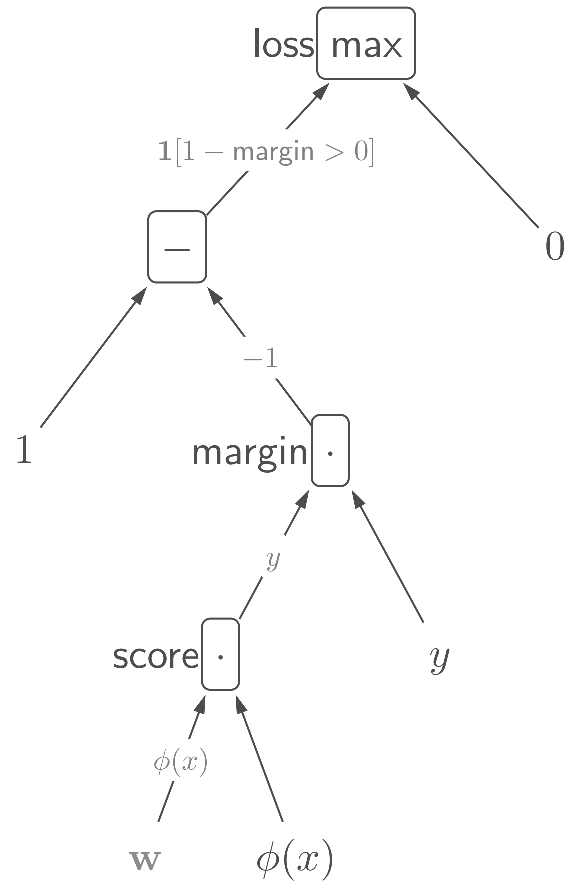
Function composition



Chain rule:

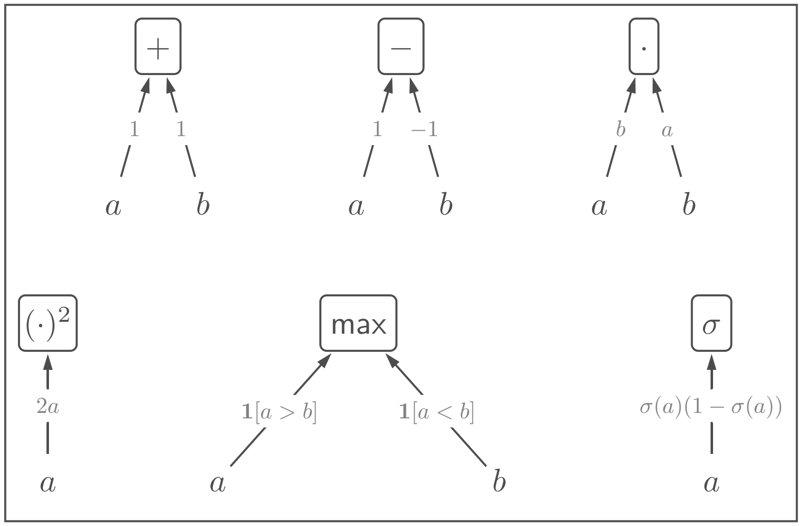
$$\frac{\partial c}{\partial a} = \frac{\partial c}{\partial b}\frac{\partial b}{\partial a} = (2b)(2a) = (2a^2)(2a) = 4a^3$$

Linear classification with hinge loss

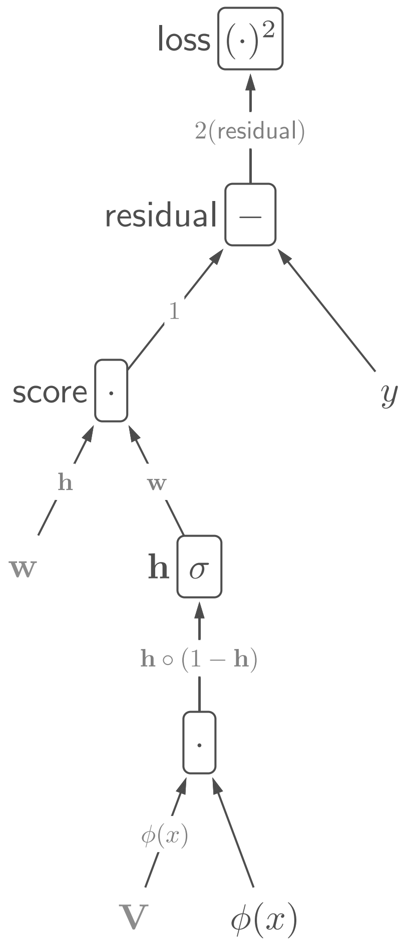


$$\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \mathbf{w} \cdot \phi(x) y, 0\}$$

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = -\mathbf{1}[\text{margin} < 1] \phi(x) y$$



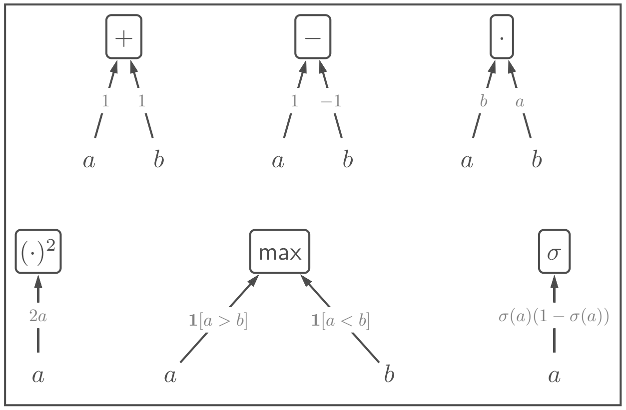
Two-layer neural networks



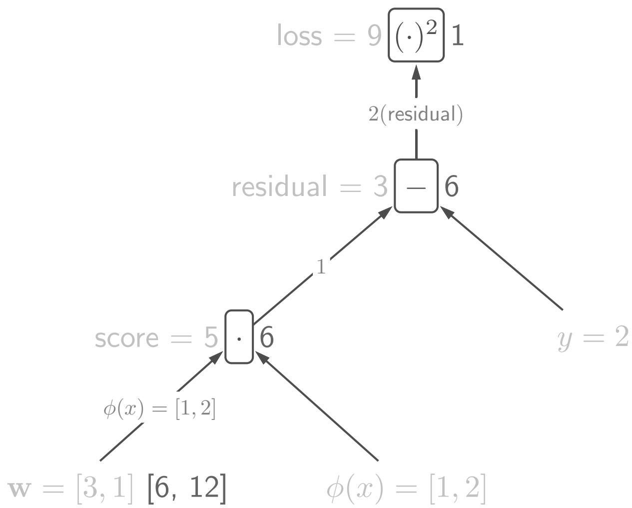
$$\text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = (\mathbf{w} \cdot \sigma(\mathbf{V}\phi(x)) - y)^2$$

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = 2(\text{residual})\mathbf{h}$$

$$\nabla_{\mathbf{V}} \text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = 2(\text{residual})\mathbf{w} \circ \mathbf{h} \circ (1-\mathbf{h})\phi(x)^\top$$



Backpropagation



$$\text{Loss}(x, y, \mathbf{w}) = (\mathbf{w} \cdot \phi(x) - y)^2$$

$$\mathbf{w} = [3,1], \phi(x) = [1,2], y = 2$$

backpropagation

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = [6, 12]$$

Definition: Forward/backward values

Forward: f_i is value for subexpression rooted at i

Backward: $g_i = \frac{\partial \text{loss}}{\partial f_i}$ is how f_i influences loss

Algorithm: backpropagation algorithm

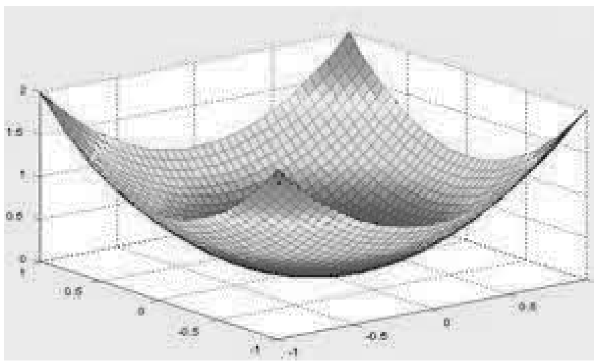
Forward pass: compute each f_i (from leaves to root)

Backward pass: compute each g_i (from root to leaves)

A note on optimization

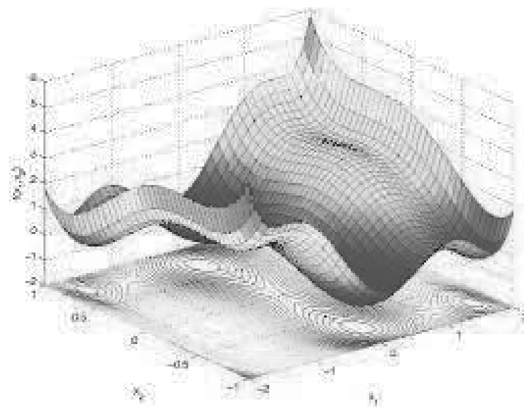
$$\min_{\mathbf{V},\mathbf{w}} \text{TrainLoss}(\mathbf{V},\mathbf{w})$$

Linear predictors



(convex)

Neural networks

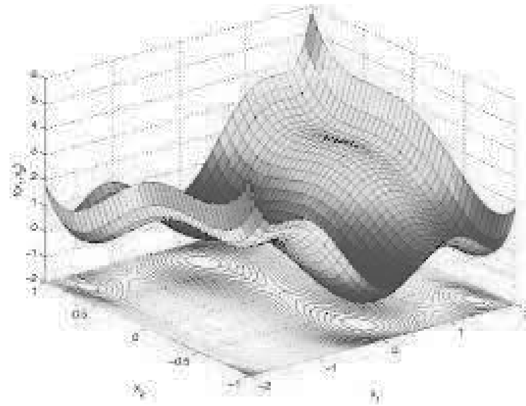


(non-convex)

Optimization of neural networks is in principle hard

How to train neural networks

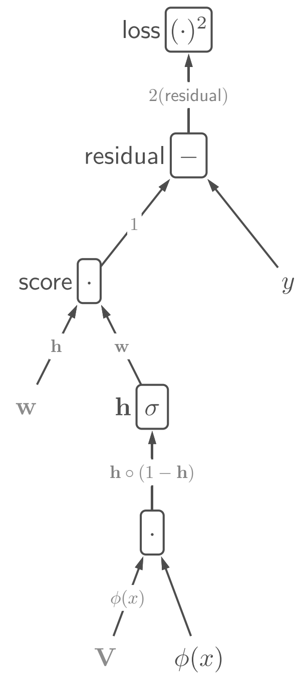
$$\text{score} = \mathbf{w} \cdot \sigma(\mathbf{V}\,\phi(x))$$



- Careful initialization (random noise, pre-training)
- Overparameterization (more hidden units than needed)
- Adaptive step sizes (AdaGrad, Adam)

> Don't let gradients vanish or explode!

Summary



- Computation graphs: visualize and understand gradients
- Backpropagation: general-purpose algorithm for computing gradients

Roadmap

Backpropagation

K-means

Generalization

Best practices

Summary of Machine Learning

[Brown et al, 1992]

Word clustering

Input: raw text (100 million words of news articles)...

Output:

Cluster 1: Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays
Cluster 2: June March July April January December October November September August
Cluster 3: water gas coal liquid acid sand carbon steam shale iron
Cluster 4: great big vast sudden mere sheer gigantic lifelong scant colossal
Cluster 5: man woman boy girl lawyer doctor guy farmer teacher citizen
Cluster 6: American Indian European Japanese German African Catholic Israeli Italian Arab
Cluster 7: pressure temperature permeability density porosity stress velocity viscosity gravity tension
Cluster 8: mother wife father son husband brother daughter sister boss uncle
Cluster 9: machine device controller processor CPU printer spindle subsystem compiler plotter
Cluster 10: John George James Bob Robert Paul William Jim David Mike
Cluster 11: anyone someone anybody somebody
Cluster 12: feet miles pounds degrees inches barrels tons acres meters bytes
Cluster 13: director chief professor commissioner commander treasurer founder superintendent dean custodian
Cluster 14: had hadn't hath would've could've should've must've might've
Cluster 15: head body hands eyes voice arm seat eye hair mouth

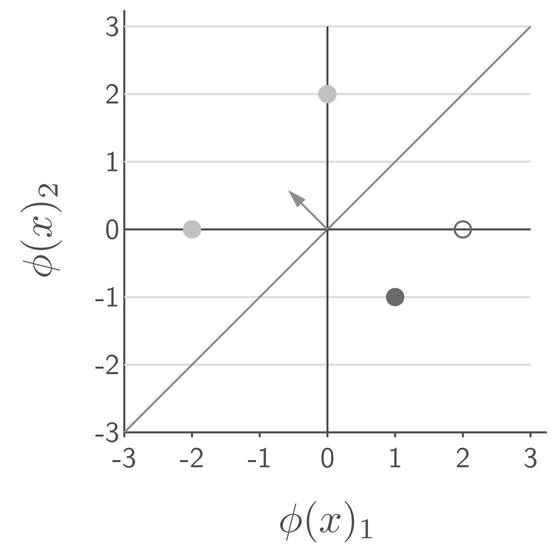
Classification (supervised learning)

training data $\mathcal{D}_{\text{train}}$

$\phi(x)_1$	$\phi(x)_2$	y
0	2	1
-2	0	1
1	-1	-1

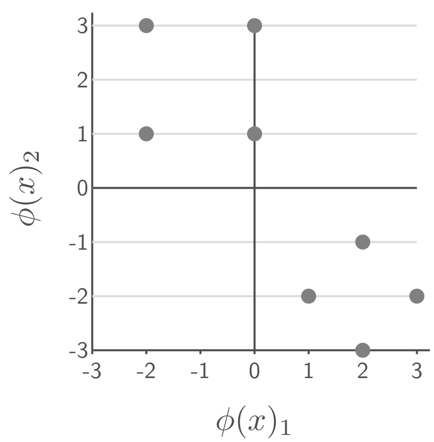
$\xrightarrow{\textbf{learning algorithm}}$ $\boxed{f}$ **classifier**

[2, 0] $\downarrow$ f $\downarrow$ -1



Labeled data is expensive to obtain

Clustering (unsupervised learning)



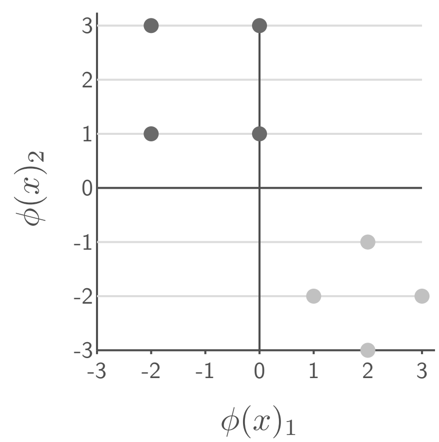
training data $\mathcal{D}_{\text{train}}$

$\phi(x)_1$	$\phi(x)_2$
-2	1
0	1
-2	3
0	3
2	-1
1	-2
2	-3
3	-2

learning algorithm $\longrightarrow$

assignments

z
1
1
1
1
2
2
2
2



Intuition: Want to assign nearby points to same cluster

$\boxed{\text{Unlabeled data is very cheap to obtain}}$

Clustering task

Definition: clustering

Input: training points

$$\mathcal{D}_{\text{train}} = [x_1, \dots, x_n]$$

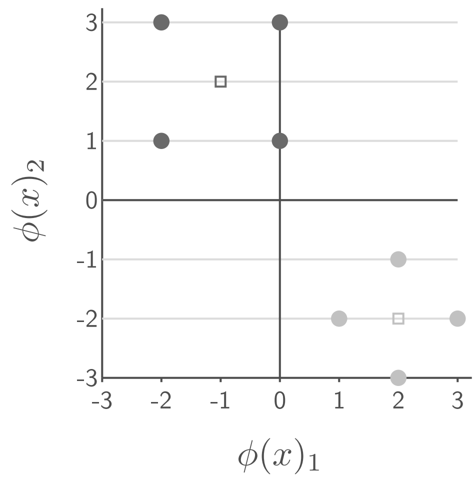
Output: assignment of each point to a cluster

$$\mathbf{z} = [z_1, \dots, z_n] \text{ where } z_i \in \{1, \dots, K\}$$

Centroids

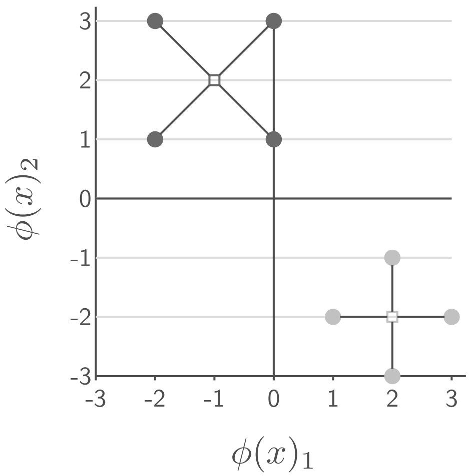
Each cluster $k = 1, \dots, K$ is represented by a **centroid** $\mu_k \in \mathbb{R}^d$

$$\boldsymbol{\mu} = [\mu_1, \dots, \mu_K]$$



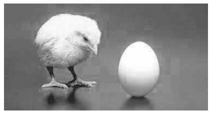
Intuition: want each point $\phi(x_i)$ to be close to its assigned centroid μ_{z_i}

K-means objective

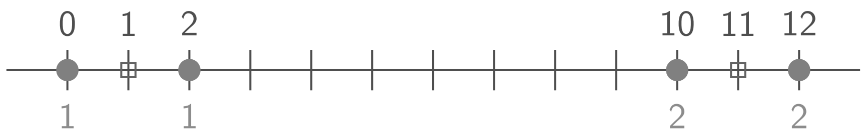


$$\text{Loss}_{\text{kmeans}}(\mathbf{z}, \boldsymbol{\mu}) = \sum_{i=1}^{n} \|\phi(x_i) - \mu_{z_i}\|^2$$

$$\min_{\mathbf{z}} \min_{\boldsymbol{\mu}} \text{Loss}_{\text{kmeans}}(\mathbf{z}, \mu)$$



Alternating minimization from optimum



If know centroids $\mu_1 = 1$, $\mu_2 = 11$:

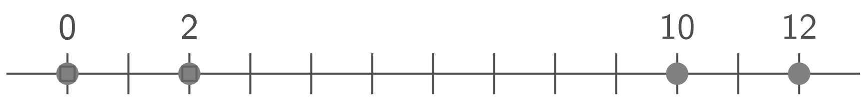
$$z_1 = \arg\min\{(0-1)^2, (0-11)^2\} = 1$$
$$z_2 = \arg\min\{(2-1)^2, (2-11)^2\} = 1$$
$$z_3 = \arg\min\{(10-1)^2, (10-11)^2\} = 2$$
$$z_4 = \arg\min\{(12-1)^2, (12-11)^2\} = 2$$

If know assignments $z_1 = z_2 = 1$, $z_3 = z_4 = 2$:

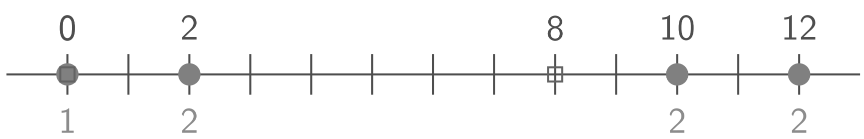
$$\mu_1 = \arg\min_\mu (0-\mu)^2 + (2-\mu)^2 = 1$$
$$\mu_2 = \arg\min_\mu (10-\mu)^2 + (12-\mu)^2 = 11$$

Alternating minimization from random initialization

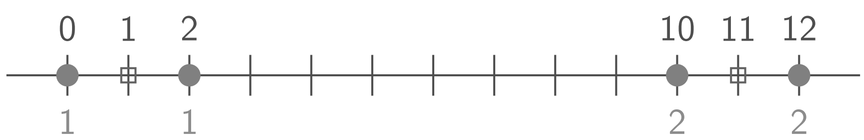
Initialize $\boldsymbol{\mu}$:



Iteration 1:



Iteration 2:



Converged.

K-means algorithm

Algorithm: K-means

Initialize $\boldsymbol{\mu} = [\mu_1, \dots, \mu_K]$ randomly.

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

Step 1: set assignments $\mathbf{z}$ given $\boldsymbol{\mu}$

For each point $i = 1, \dots, n$:

$$z_i \leftarrow \arg \min_{k=1,\dots,K} \|\phi(x_i) - \mu_k\|^2$$

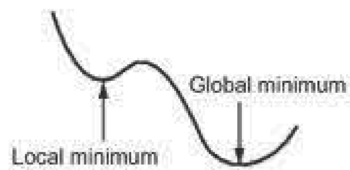
Step 2: set centroids $\boldsymbol{\mu}$ given $\mathbf{z}$

For each cluster $k = 1, \dots, K$:

$$\mu_k \leftarrow \frac{1}{|\{i : z_i = k\}|} \sum_{i : z_i = k} \phi(x_i)$$

Local minima

K-means is guaranteed to converge to a local minimum, but is not guaranteed to find the global minimum.



[demo: getting stuck in local optima, seed $= 100$]

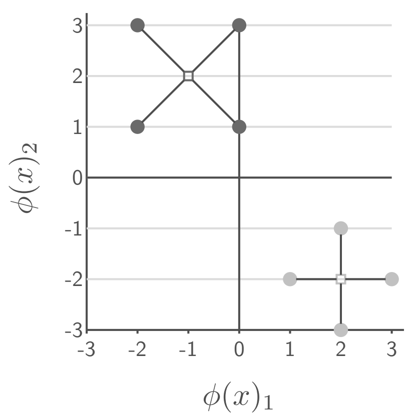
Solutions:

- Run multiple times from different random initializations
- Initialize with a heuristic (K-means++)

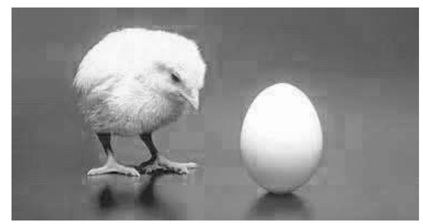
Summary

Clustering: discover structure in unlabeled data

K-means objective:



K-means algorithm:

assignments $\mathbf{z}$  centroids $\boldsymbol{\mu}$

Unsupervised learning use cases:

- Data exploration and discovery
- Providing representations to downstream supervised learning

Roadmap

Backpropagation

K-means

Generalization

Best practices

Summary of Machine Learning

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 $\mathcal{D}_{\text{train}}$.

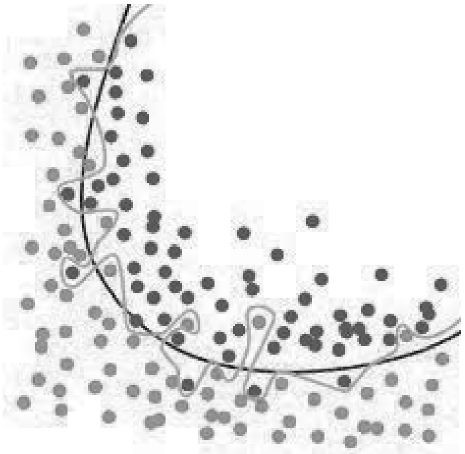
Predictor $f(x)$:

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

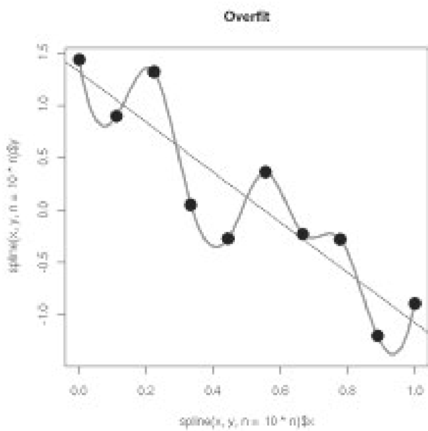
  Else: **segfault**.

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

Overfitting pictures



Classification



Regression

Evaluation

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

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

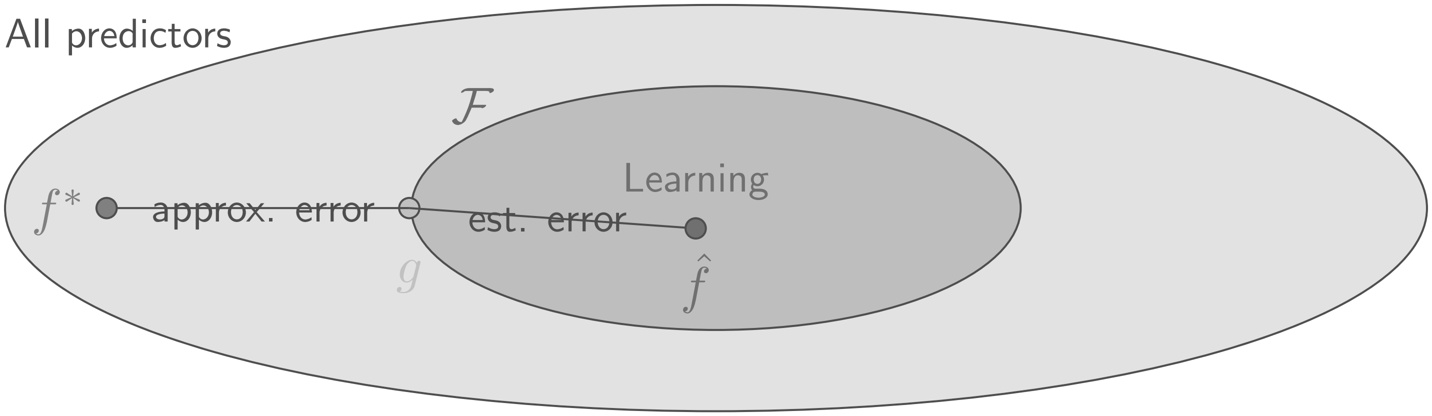
Test set $\mathcal{D}_{\text{test}}$ contains examples not used for training.

Generalization

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

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

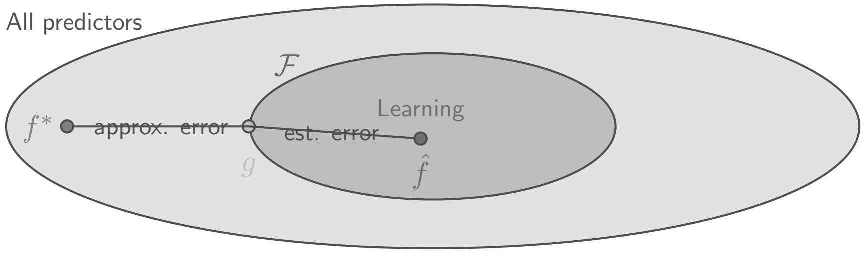
Approximation and estimation error



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

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

Effect of hypothesis class size



As the hypothesis class size increases...

Approximation error decreases because:

taking min over larger set

Estimation error increases because:

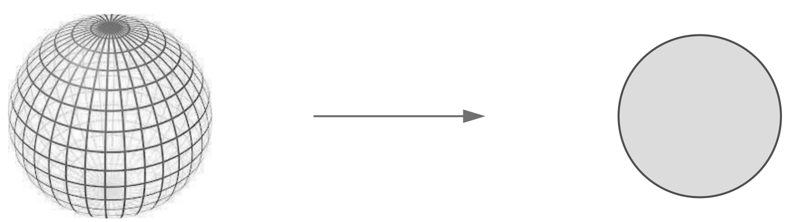
harder to estimate something more complex

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:

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

Algorithm: gradient descent

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

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

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

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 $t = 1, \ldots, T$:
$\quad \mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

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

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.

Roadmap

Backpropagation

K-means

Generalization

Best practices

Summary of Machine Learning

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 $\mathbf{w} = [0, \dots, 0]$
For $t = 1, \dots, T$:
  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 $\mathcal{D}_{\text{train}} \backslash \mathcal{D}_{\text{val}}$, 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

$\mathcal{D}_{\text{train}} \backslash \mathcal{D}_{\text{val}}$ | $\mathcal{D}_{\text{val}}$ | $\mathcal{D}_{\text{test}}$

Don't look at the test set!

Understand the data!

Start simple!

Practice!

Roadmap

Backpropagation

K-means

Generalization

Best practices

Summary of Machine Learning

Machine Learning Summary

- Feature extraction (think hypothesis classes) [modeling]
- Prediction (linear, neural network, k-means) [modeling]
- Loss functions (evaluate errors) [modeling]
- Optimization (stochastic gradient, alternating minimization) [learning]
- Generalization (think development cycle) [modeling]
- We are not covering some other important aspects, e.g., fairness, privacy, interpretability

Machine learning

Key idea: learning

Programs should improve with experience.

So far: reflex-based models

Next time: state-based models

Homework

due: next week

作业 2-周2-Learning
PyTorch with Examples