## Machine learning

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Where do models come from?

Models have parameters:

- State space models: search problems have $\mathrm{Cost}(s, a)$, MDPs have $\mathrm{Reward}(s, a)$ and transitions $T(s, a, s')$, games have evaluation functions $\mathrm{Eval}(s)$
- Graphical models: Markov networks have factors $f_j(x_{i-1}, x_i)$, Bayesian networks have local conditional probability distributions $p(x_i \mid x_{i-1})$

Can only construct **rational agents (optimal policies)** with respect to **model with fixed parameters**.

$$\begin{tikzcd}\boxed{\text{training data}} \arrow[r, "\text{learning}"] & \boxed{\text{model}} \arrow[r, "\text{reasoning}"] & \boxed{\text{policy}}\end{tikzcd}$$

## Applications

*(Almost) everything.*

natural language processing
computer vision
robotics
information retrieval
medical diagnosis
computational biology
cognitive science
social science
fraud detection
spam recognition
speech recognition
handwriting recognition
finance
game playing
recommendation systems
computer security
computer architecture
programming languages
etc.

## Outline

- **Supervised learning**
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Outline

- Supervised learning
  - **Principles of learning and loss minimization**
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Application: spam classification

Input: $x =$ email message

```
From: pliang@cs.stanford.edu
Date: November 1, 2012
Subject: CS221 announcement

Hello students,
 There will be a review session...
```

```
From: a9k62n@hotmail.com
Date: November 1, 2012
Subject: URGENT

Dear Sir or maDam:
 my friend left sum of 10m dollars...
```

Output: $y \in \{\mathbf{spam}, \mathbf{not\text{-}spam}\}$

Objective: build predictor $f$ that maps input $x$ to (hopefully correct) prediction $y = f(x)$

## Supervised learning

Training data: examples of desired input-output behavior

$\text{Train} = \{(\text{"...10m dollars..."}, +1), (\text{"...CS221..."}, -1)\}$

Predictor: a function $f$ mapping input $x$ to prediction $y = f(x)$

$f(x) = +1$ if $x$ contains "10m dollars" else $-1$

$f(\text{"...10m dollars..."}) = +1$

Learning algorithm: takes training data and creates a predictor

**This is what we're going to build!**

$$\boxed{\text{Training data}} \longrightarrow \boxed{\text{Learning algorithm}} \longrightarrow \boxed{\text{Predictor}}$$

## Types of prediction problems

Classification: $y$ is yes/no (binary), one of $K$ labels (multiclass), subset of $K$ labels (multilabel)

Regression: $y$ is a real number, e.g., housing prices

Structured prediction: $y$ is a sentence, e.g., machine translation

Ranking: $y$ is an ordering (e.g., ranking web pages)

This lecture: focus on **binary classification** and **regression**.

## Rote learning algorithm

Idea: memorize the training data and regurgitate.

**Algorithm: rote learning**

Let $X$ be set of inputs seen in $\text{Train}$. Return predictor:

$$f(x) = \begin{cases} \arg\max_y [\#\ \text{times}(x, y) \in \text{Train}] & \text{if } x \in X \\ \text{random guess} & \text{otherwise} \end{cases}$$

Implementation: hash $x$ (linear in # examples), constant time prediction!

- Pros: simple, works well when lots of examples compared to number of possible inputs, can "learn" anything
- Cons: doesn't **generalize** at all to unseen examples (overfitting)!

## Majority algorithm

Notes

Idea: always predict the most frequent output based on training data.

**Algorithm: majority**

Let $y^*$ be the most frequent output in $\text{Train}$

Return predictor: $f(x) = y^*$ (don't even need to look at $x$!)

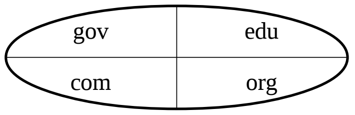
- Pros: simple, provides a useful baseline
- Cons: not very accurate

## Abstraction + rote learning

Idea: map each input $x$ onto abstract input $\alpha(x)$; do rote learning.

Example: $\alpha(x) =$ last three characters of $x$:
$\alpha(\text{"abc@hotmail.com"}) = \alpha(\text{"xyz@gmail.com"}) = \text{"com"}$

$\alpha$ partitions input space, e.g.:



Coarse $\alpha(x) = 1$
few parameters
low training accuracy
better generalization

Fine $\alpha(x) = x$
many parameters
high training accuracy
worse generalization

## Evaluation of predictors

Question: how good is a predictor $f$?

On a single example $(x, y)$, might penalize for each mistake:

$\text{Loss}(x, y, f) = [f(x) \neq y]$:
whether $f$ erred on $x$

| $y \backslash f(x)$ | $+1$ | $-1$ |
|---|---|---|
| $+1$ | 0 | 1 |
| $-1$ | 1 | 0 |

Terminology: average loss $=$ error $= 1 -$ accuracy

Predictor $f$ has high utility if:

- ~~$f$ has high accuracy over training examples~~
- $f$ has high accuracy over **future examples**

Key challenge: don't know future examples, we cannot evaluate our true utility function (in contrast with policy evaluation)!

## Evaluation of predictors

- Split examples into $\mathsf{Train}$ and $\mathsf{Test}$ either randomly or based on time if examples are time-stamped (training examples happened before test examples)

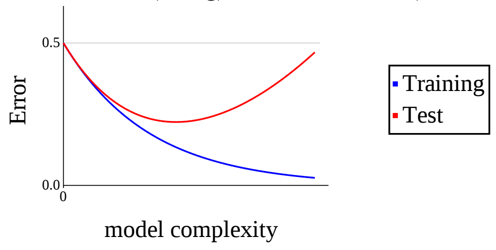
| Train | Test |
|---|---|

- Run learning algorithm on $\mathsf{Train}$, report accuracy on $\mathsf{Test}$ (provides estimate of accuracy on future examples).

## Training error and test error

As model complexity increases, usually:

- Training error decreases
- Test error decreases (fitting) and then increases (overfitting)



Key question: how can a learning algorithm **generalize**?

## Nearest neighbors

Idea: find most **similar** input, and regurgitate its output.

How to measure similarity?

**Definition: Distance function**
A distance function $\mathsf{Dist}(x, x') \geq 0$ measures how different $x'$ is from $x$.

Example: $\mathsf{Dist}(x, x') = [\# \text{ words in exactly one of } x \text{ and } x']$

$$\mathsf{Dist}(\text{“make 10m dollars”}, \text{“make 20m dollars”}) = 2$$

$$\mathsf{Dist}(\text{“make 10m dollars”}, \text{“make a movie”}) = 4$$

## Nearest neighbors

**Algorithm: nearest neighbors**
Return predictor that takes output of closest example:

```
$f(x) = \{$
    $(x^*, y^*) \leftarrow \arg\min_{(x', y') \in \mathsf{Train}} \mathsf{Dist}(x, x')$
    return $y^*$
$\}$
```

Implementation: data structures k-d trees or approximate hashing

- Pros: simple, works when have a lot of data, can "learn" almost anything, very useful in practice
- Cons: **generalizes** only a little bit better than rote learning

## Features

Objective: Given input $x$, extract (feature, value) pairs which might be related to $y$.

$$\text{“abc@gmail.com”} \xrightarrow{\text{feature extraction}}$$

| | |
|---|---|
| length | : 13 |
| contains(@) | : 1 |
| endsWith(.com) | : 1 |
| endsWith(.org) | : 0 |
| ... | |

For notation: number the features $1, \dots, d$, represent key-value map as vector (e.g., [13, 1, 1, 0])

**Definition: Feature vector**
For each input $x$, have feature vector $\phi(x) = (\phi_1(x), \dots, \phi_d(x))$.
Think of $\phi(x) \in \mathbb{R}^d$ as a point in a high-dimensional space.

## Feature engineering

Arguably the most important part of machine learning!

Examples of features:

- Natural language: words, parts-of-speech, capitalization pattern
- Computer vision: HOG, SIFT, image transformations, smoothing, histograms
- In general: use domain knowledge about problem

Intuition: define many features (akin to multiple incomparable/overlapping abstractions)

## Weight vector

Weight vector: for each feature $j$, have weight $w_j$ representing contribution of feature to prediction

| | |
|---|---|
| length | :-1.2 |
| contains(@) | :3 |
| endsWith(.com) | :2 |
| endsWith(.org) | :1 |
| ... | |

## Linear predictors

Feature vector $\phi(x) \in \mathbb{R}^d$

| | |
|---|---|
| length | :13 |
| contains(@) | :1 |
| endsWith(.com) | :1 |
| endsWith(.org) | :0 |

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

| | |
|---|---|
| length | :-0.4 |
| contains(@) | :5 |
| endsWith(.com) | :4 |
| endsWith(.org) | :1 |

Take weighted combination of features:

$$\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{d} w_j \phi(x)_j$$

Example: $-0.4(13) + 5(1) + 4(1) + 1(0) = 3.8$

**Definition: Linear predictors**

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

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

## Two perspectives on features $\phi(x)$

Ensemble perspective: each feature $\phi_j(x)$ is a weak predictor based on partial view of $x$; prediction is weighted combination of $\phi_j(x)$

Useful for designing features: what parts of $x$ are relevant for predicting $y$?

Geometric perspective: $\phi(x)$ is a high-dimensional point

Useful for designing algorithms: how to separate positive and negative points

## How to get the weight vector?

Notes

Learning algorithm sets weights $\mathbf{w}$ based on training data.

**Loss minimization framework**:

**Objective (version 1)**

Set weights to minimize training error:

$$\min_{\mathbf{w}} \sum_{(x,y)\in\text{Train}} \text{Loss}(x, y, \mathbf{w})$$

Loss functions:

- Regression: $L_2$ (least squares), $L_1$ (least absolute deviations)
- Classification: zero-one (minimize # mistakes), perceptron, hinge (SVM), logistic

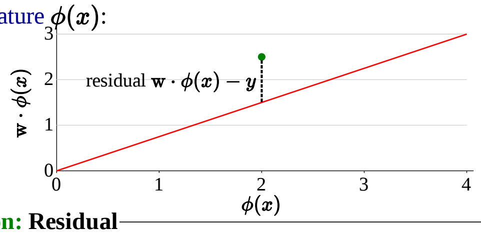
Many popular algorithms fall into this framework.

## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - **Linear regression**
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Linear regression

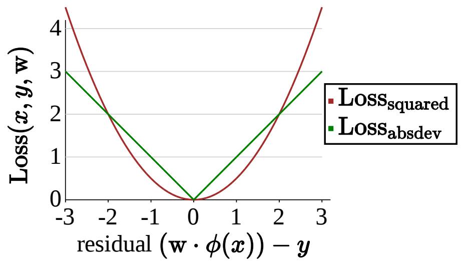
If one feature $\phi(x)$:



**Definition: Residual**

The **residual** of an example $(x, y)$ with respect to weights $\mathbf{w}$ is $(\mathbf{w} \cdot \phi(x)) - y$, the amount by which model prediction $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$ overshoots $y$. Regression losses depend on the residual.

## Regression loss functions



$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = \tfrac{1}{2}(\mathbf{w} \cdot \phi(x) - y)^2$$

$$\text{Loss}_{\text{absdev}}(x, y, \mathbf{w}) = |\mathbf{w} \cdot \phi(x) - y|$$

## Which loss to use?

Assume one feature with value 1 ($\phi(x) = 1$ for all $x$).

For least squares ($L_2$) regression:

$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = \tfrac{1}{2}(\mathbf{w} - y)^2$$

$\mathbf{w}$ that minimizes training loss is **mean** $y$

For least absolute deviation ($L_1$) regression:

$$\text{Loss}_{\text{absdev}}(x, y, \mathbf{w}) = |\mathbf{w} - y|$$

$\mathbf{w}$ that minimizes training loss is **median** $y$

Pros/cons:

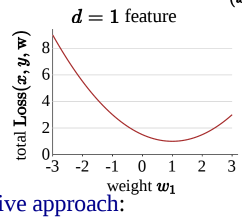
- $L_2$: penalizes outliers more (try to make every example happy); popular, easier to optimize
- $L_1$: more robust to outliers

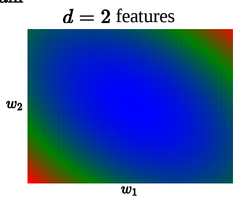
## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - **Stochastic gradient descent**
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Optimization problem

$$\text{Objective: } \min_{\mathbf{w}} \sum_{(x,y) \in \text{Train}} \text{Loss}(x, y, \mathbf{w})$$





Iterative approach:

- Start with a guess for $\mathbf{w}$ (e.g., $\mathbf{w} = 0$)
- Change $\mathbf{w}$ to decrease the loss using the gradient: $\mathbf{w} \leftarrow \mathbf{w} - \eta_t \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$

## Gradient for least squares regression

Consider one feature, one example, squared loss.

Objective function:

$$\text{Loss}(x, y, \mathbf{w}) = \tfrac{1}{2}(\mathbf{w} \cdot \phi(x) - y)^2$$

Gradient (use chain rule):

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

Update of weights:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta_t \underbrace{(\mathbf{w} \cdot \phi(x) - y)}_{\text{prediction} - \text{target}} \phi(x)$$

Notes

## Stochastic gradient descent

Objective:

$$\min_{\mathbf{w}} \sum_{(x,y)\in\text{Train}} \text{Loss}(x, y, \mathbf{w})$$

Strategy: go through training examples and adjust weights (using gradient) to decrease loss

**Algorithm: stochastic gradient descent (SGD)**

```
$\mathbf{w} \leftarrow (0, \dots, 0)$
For $t = 1, 2, \dots, T$:
    Choose an example $(x, y) \in \text{Train}$
    $\mathbf{w} \leftarrow \mathbf{w} - \eta_t \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$
```

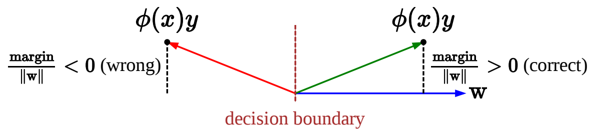
Step size: $\eta_t = \frac{1}{t^\alpha}$ for $\alpha \in [0, 1]$, (update less over time)

## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - **Linear classification**
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Linear classification

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

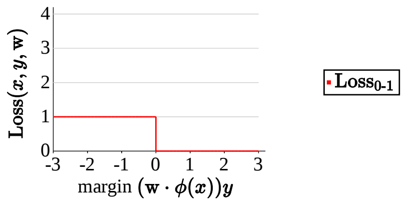

$\phi(x)y$
$\phi(x)y$
$\frac{\text{margin}}{\|\mathbf{w}\|} < 0$ (wrong)
$\frac{\text{margin}}{\|\mathbf{w}\|} > 0$ (correct)
$\mathbf{w}$
decision boundary

**Definition: Margin**

The **margin** of an example $x$ with respect to weights $\mathbf{w}$ is $(\mathbf{w} \cdot \phi(x))y$. The margin is positive (prediction and $y$ have the same sign) iff the example is classified correctly. Classification losses depend on the margin.

## Classification: zero-one loss

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = [(\mathbf{w} \cdot \phi(x))y < 0]$$



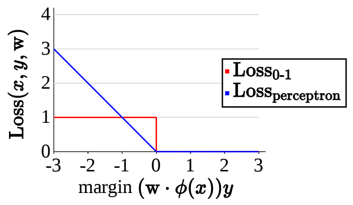
Problems:

- Gradient of $\text{Loss}_{0\text{-}1}$ is 0 everywhere, SGD not applicable
- $\text{Loss}_{0\text{-}1}$ is insensitive to how badly model messed up

Notes

## Classification: perceptron loss

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



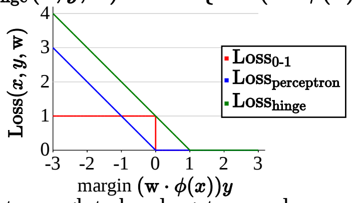
Perceptron algorithm is SGD on perceptron loss:

- Update weights only when make mistake: $\mathbf{w} \leftarrow \mathbf{w} + \eta_t \phi(x)y$
- If barely classify correctly (0.01 margin), zero loss; not robust...

Notes

## Hinge loss

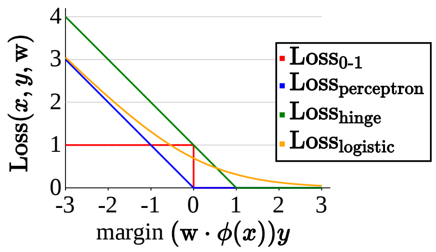
$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$



- Intuition: not enough to barely get example correct, want $\text{margin} \geq 1$
- Update weights when margin $< 1$: $\mathbf{w} \leftarrow \mathbf{w} + \eta_t \phi(x)y$
- Corresponds to online learning of support vector machines (SVMs).

## Logistic regression

$$\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-(\mathbf{w}\cdot\phi(x))y})$$



- Intuition: even if example correct, want large margin

## Logistic regression

Probabilistic interpretation:

- Two assignments $y \in \{-1, +1\}$
- Non-negative $\text{Weight}(y) = e^{\text{margin}/2} = e^{\mathbf{w}\cdot\phi(x)y/2}$
- Normalize to get distribution:
  $p_{\mathbf{w}}(y \mid x) = \frac{\text{Weight}(y)}{\text{Weight}(-1)+\text{Weight}(1)} = \frac{1}{1+e^{-\mathbf{w}\cdot\phi(x)y}}$

Optimization:

- Goal: maximize probability of correct classification $p_{\mathbf{w}}(y \mid x)$
- Same: minimize $\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-\mathbf{w}\cdot\phi(x)y})$
- Update weights (always): $\mathbf{w} \leftarrow \mathbf{w} + \eta_t (1 - p_{\mathbf{w}}(y \mid x))\phi(x)y$

## Summary

Linear models: prediction governed by $\mathbf{w}\cdot\phi(x)$

Loss functions: capture various desiderata (e.g., robustness) for both regression and binary classification (can be generalized to many other problems)

Objective function: minimize loss over training data

Strategy: take stochastic gradient steps on $\mathbf{w}$ to decrease loss

## The entire pipeline

Features $\phi(x)$ + training examples

**Learning**: minimize training loss

Input $x \Rightarrow$ Weights $\mathbf{w}$ (defines predictor $f_{\mathbf{w}}$) $\Rightarrow f_{\mathbf{w}}(x)$

## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - **Linearity, non-linearity, and kernels**
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Linearity

Linear predictors:

- Regression: $f_{\mathbf{w}}(x) = \mathbf{w}\cdot\phi(x)$
- Binary classification: $f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w}\cdot\phi(x))$

Linear in what?

- Prediction is linear in $\mathbf{w}$
- Prediction is **not** linear in $x$ (doesn't even make sense)
- Prediction is linear in $\phi(x)$ (can define however we want)

[Examples]

## Kernels

Observation: all updates are of form $\mathbf{w} \leftarrow \mathbf{w} - (\text{number})\phi(x)$

Implication: Final $\mathbf{w}$ is some linear combination of training examples: $\mathbf{w} = \sum_{(x,y)\in\text{Train}} \alpha_{x,y}\phi(x)$, where coefficients $\alpha_{x,y}$ specifies contribution of example $(x, y)$.

Key identity: $\mathbf{w} \cdot \phi(x') = \sum_{(x,y)\in\text{Train}} \alpha_{x,y} \underbrace{(\phi(x) \cdot \phi(x'))}_{=K(x,x')}$

Algorithms only need a black box that computes **kernel function** $K(x, x')$ (captures **similarity** between $x$ and $x'$), don't have to explicitly create $\phi(x)$.

## Kernels

**Linear kernel** (assume $x \in \mathbb{R}^d$):

$$K(x, x') = x \cdot x'$$

Corresponds to $\phi(x) = x$.

**Polynomial kernel** (assume $x \in \mathbb{R}^d$):

$$K(x, x') = (1 + x \cdot x')^r$$

If $r = 2, d = 2$, corresponds to:

$$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

In general, $\phi(x)$ is $\binom{d}{r}$ dimensions (huge!), but computing $K(x, x')$ only takes $O(d)$ time. Algorithms can take $O(|\text{Train}|^2)$ time.

## More examples of kernels

Radial basis function kernel: $K(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$

(similar effect to nearest neighbors)

String and tree kernels: count number of common substrings/subtrees (applications in computational biology and NLP)

## Kernels: summary

Modifying $\phi(x)$ induces rich non-linear decision boundaries in $x$

$$\boxed{K(x, x') = \phi(x) \cdot \phi(x')}$$

Think in terms of similarity between inputs rather than features of input

$K(x, x')$ is easy to compute when $\phi(x)$ is high-dimensional or infinite

Applicable to any linear model (regression, classification losses)

Store $\alpha_{(x,y)}$ instead of $\mathbf{w}$ (pay $O(|\text{Train}|)$ rather than $O(d)$)

## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - **Complexity control via regularization**
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Regularization

**Definition: Regularizer**
A regularizer prevents the weights from being too big (complex).
Commonly used $L_2$ regularizer (squared length of weight vector):

$$\text{Penalty}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\sum_{j=1}^d w_j^2$$

Objective:

$$\min_{\mathbf{w}} \underbrace{\sum_{(x,y)\in\text{Train}} \text{Loss}(x, y, f_{\mathbf{w}})}_{\text{fit data}} + \underbrace{\lambda \cdot \text{Penalty}(\mathbf{w})}_{\text{prefer simpler model}}$$

As regularization $\lambda$ increases, shrink weights $\mathbf{w}$ towards zero.
Weight update:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta_t \Big(\underbrace{\nabla_{\mathbf{w}}\text{Loss}(x, y, f_{\mathbf{w}})}_{\text{e.g., } =(\mathbf{w}\cdot\phi(x)-y)\phi(x)} + \underbrace{\frac{\lambda}{|\text{Train}|}\nabla_{\mathbf{w}}\text{Penalty}(\mathbf{w})}_{\text{e.g., } =\frac{\lambda}{|\text{Train}|}\mathbf{w}}\Big)$$

## Hyperparameters

Parameters: weights $\mathbf{w}$ set by learning algorithm

Hyperparameters: properties of the learning algorithm (features, regularization parameter $\lambda$, number of iterations $T$, step size $\eta_t$) - how to set them?

Choose hyperparameters to minimize $\mathsf{Train}$ error? **No** - solution would be to include all features, set $\lambda = 0$, $T \to \infty$.

Choose hyperparameters to minimize $\mathsf{Test}$ error? **No** - choosing based on $\mathsf{Test}$ makes it an unreliable estimate of error!

## Cross-validation

Partition training data $\mathsf{Train}$ into $K$ folds:

| $\mathsf{Train}_1$ | $\mathsf{Train}_2$ | $\mathsf{Train}_3$ | $\mathsf{Train}_4$ | $\mathsf{Train}_5$ |
|---|---|---|---|---|

**Algorithm: cross-validation**

```
For each hyperparameter value (say, $\lambda = 0.1, 1, 10, \dots$):
    For each $k = 1, \dots, K$:
        Run learning algorithm on $\mathsf{Train} - \mathsf{Train}_k$
        Compute error on $\mathsf{Train}_k$ (validation set)
    Let $\mathsf{Error}(\lambda)$ be error averaged over $K$ folds
Choose hyperparameter $\lambda$ with minimum $\mathsf{Error}(\lambda)$
```

## Other classifiers

Naive Bayes: linear classifier, independently estimate weights in closed form (probabilistic interpretation as generative model)

Neural networks: cascade of logistic regressions; maps raw data to internal representation to output; requires less feature engineering

Decision trees: partition input space (learning abstraction functions); yields interpretable rules

## Summary

Learning algorithm: want to fit (small loss) but not overfit (small model complexity)

Features: represent inputs as feature vectors (important, use domain knowledge)

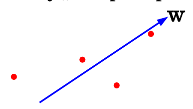
Linear predictors: weighted combination of features $\mathbf{w} \cdot \phi(x)$; remember linear in weights, not features (e.g., kernels)

## Outline

- Supplementary learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - **Maximum likelihood for Bayesian networks**
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## From predictors to distributions

- So far, focused on prediction (regression and binary classification): predictor $f_{\mathbf{w}}$ maps input $x$ to output $y$



Goal: estimate weights $\mathbf{w}$ given training data

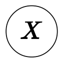
- Now, focus on learning Bayesian networks

$$\begin{tikzcd} X_1 \arrow[r] & X_2 \arrow[r] & X_3 \arrow[r] & X_4 \arrow[r] & X_5 \end{tikzcd}$$

Goal: estimate local conditional probability distributions $p(x_i \mid x_{\text{Parents}(i)})$ given training data

## Example: one variable

One variable $X$ representing the rating of a movie $\{1, 2, 3, 4, 5\}$



$$\mathbb{P}(X = x) = p(x)$$

Parameters: $\theta = (p(1), p(2), p(3), p(4), p(5))$

Training data: $\mathbf{Train}$ is multi-set of example assignments to $X$ (user ratings)

Example: $\mathbf{Train} = \{1, 3, 4, 4, 4, 4, 4, 5, 5, 5\}$

Learning:

$$\mathbf{Train} \quad \Rightarrow \quad \theta$$

## Example: one variable

Learning:

$$\mathbf{Train} \quad \Rightarrow \quad \theta$$

Intuition: $p(x) \propto$ frequency of $x$ in $\mathbf{Train}$

Example:

$$\mathbf{Train} = \{1, 3, 4, 4, 4, 4, 4, 5, 5, 5\}$$

$$\Big\downarrow$$

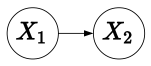
$\theta$:

| $x$ | $p(x)$ |
|---|---|
| 1 | 0.1 |
| 2 | 0 |
| 3 | 0.1 |
| 4 | 0.5 |
| 5 | 0.3 |

## Example: two variables

Variables:

- Genre $X_1 \in \{\text{drama}, \text{comedy}\}$
- Rating $X_2 \in \{1, 2, 3, 4, 5\}$



$$\mathbb{P}(X_1 = x_1, X_2 = x_2) = p_1(x_1)p_2(x_2 \mid x_1)$$

$\mathbf{Train} = \{(\text{d}, 4), (\text{d}, 4), (\text{d}, 5), (\text{c}, 1), (\text{c}, 5)\}$

What are parameters $\theta = (p_1, p_2)$?

## Example: two variables

$\mathbf{Train} = \{(\text{d}, 4), (\text{d}, 4), (\text{d}, 5), (\text{c}, 1), (\text{c}, 5)\}$

Intuitive strategy:

- Estimate each local conditional distribution separately ($p_1$ and $p_2$)
- For each value of conditioned variable (e.g., $x_1$), estimate distribution over values of unconditioned variable (e.g., $x_2$)

$\theta$:

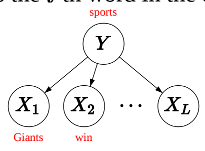
| $x_1$ | $p_1(x_1)$ |
|---|---|
| d | 3/5 |
| c | 2/5 |

| $x_1$ | $x_2$ | $p_2(x_2 \mid x_1)$ |
|---|---|---|
| d | 4 | 2/3 |
| d | 5 | 1/3 |
| c | 1 | 1/2 |
| c | 5 | 1/2 |

## Example: Naive Bayes

Variables:

- $Y \in \{\text{sports}, \text{politics}, \cdots\}$: possible document classes
- $X_1, \ldots, X_L$: $X_i$ is the $i$-th word in the document



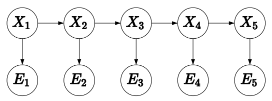
$$\mathbb{P}(Y = y, X_1 = x_1, \ldots, X_L = x_L) = p_{\text{class}}(y) \prod_{j=1}^{d} p_{\text{word}}(x_j \mid y)$$

Parameters: $\theta = (p_{\text{class}}, p_{\text{word}})$
$\mathbf{Train}$ is a set of full assignments to $(Y, X_1, \ldots, X_L)$

## Example: Hidden Markov models (HMMs)

Variables:

- $X_1, \ldots, X_T$ (e.g., part-of-speech tags, actual positions)
- $E_1, \ldots, E_T$ (e.g., words, sensor readings)



$$\mathbb{P}(X_{1:T} = x_{1:T}, E_{1:T} = e_{1:T}) = \prod_{t=1}^{T} p_{\text{transition}}(x_t \mid x_{t-1}) p_{\text{emission}}(e_t \mid x_t)$$

Parameters: $\theta = (p_{\text{transition}}, p_{\text{emission}})$

$\mathbf{Train}$ is a set of full assignments to $(X_{1:T}, E_{1:T})$

## General case

Bayesian network: variables $X_1, \dots, X_n$

Parameters: collection of distributions $\theta = \{p_d : d \in D\}$ (e.g., $D = \{\text{class}, \text{word}\}$)

Each variable $X_i$ is generated from distribution $p_{d_i}$:

$$\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) = \prod_{i=1}^{n} p_{d_i}(x_i \mid x_{\text{Parents}(i)})$$

Training data: $\text{Train}$ set of assignments $x = (x_1, \dots, x_n)$

Learning:

$$\text{Train} \quad \Rightarrow \quad \theta$$

## General case: learning algorithm

Input: training examples $\text{Train}$ of full assignments

Output: parameters $\theta = \{p_d : d \in D\}$

**Algorithm: maximum likelihood for Bayesian networks**

```
For each distribution $d \in D$:
**Count**:
    For each $x \in \text{Train}$:
        For each variable $x_i$ generated from $d_i = d$:
            Increment count for partial assignment $(x_{\text{Parents}(i)}, x_i)$ for $d$
**Normalize**:
    For each partial assignment $x_{\text{Parents}(i)}$:
        Set $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$
```

## Maximum likelihood

Recall loss minimzation framework:

$$\min_{\mathbf{w}} \sum_{(x,y) \in \text{Train}} \text{Loss}(x, y, \mathbf{w})$$

Maximum likelihood framework:

$$\max_{\theta} \prod_{x \in \text{Train}} \mathbb{P}_\theta(X = x)$$

Algorithm on previous slide exactly computes maximum likelihood parameters (closed form solution).

## Problem with maximum likelihood

Scenario 1: Suppose you have a coin with an unknown probability of heads $p(\text{H})$. You flip it 100 times, resulting in 23 heads, 77 tails. What is estimate of $p(\text{H})$?

Maximum likelihood estimate: $p(\text{H}) = 0.23 \quad p(\text{T}) = 0.77$

Scenario 2: Suppose you flip a coin once and get heads. What is estimate of $p(\text{H})$?

Maximum likelihood estimate: $p(\text{H}) = 1 \quad p(\text{T}) = 0$

Intuition: This is a bad estimate; real $p(\text{H})$ is closer to half

When have less data, maximum likelihood not reliable, want a more reasonable estimate...

## Regularization: Laplace smoothing

Maximum likelihood:

$p(\text{H}) = \frac{1}{1} \quad p(\text{T}) = \frac{0}{1}$

Maximum likelihood with Laplace smoothing:

$p(\text{H}) = \frac{1+1}{1+2} = \frac{2}{3} \quad p(\text{T}) = \frac{0+1}{1+2} = \frac{1}{3}$

**Laplace smoothing**

For each distribution $d$ and partial assignment $(x_{\text{Parents}(i)}, x_i)$, add $\lambda$ to $\text{count}_d(x_{\text{Parents}(i)}, x_i)$

Interpretation: hallucinate $\lambda$ occurrences of each partial assignment

Larger $\lambda$ means more smoothing $\Rightarrow$ probabilities closer to uniform.

Analogous to regularization for learning predictors.

## Example: two variables

$\text{Train} = \{(\text{d}, 4), (\text{d}, 4), (\text{d}, 5), (\text{c}, 1), (\text{c}, 5)\}$

$\theta$:

| $x_1$ | $p_1(x_1)$ |
|---|---|
| d | 4/7 |
| c | 3/7 |

| $x_1$ | $x_2$ | $p_2(x_2 \mid x_1)$ |
|---|---|---|
| d | 1 | 1/8 |
| d | 2 | 1/8 |
| d | 3 | 1/8 |
| d | 4 | 3/8 |
| d | 5 | 2/8 |
| c | 1 | 2/7 |
| c | 2 | 1/7 |
| c | 3 | 1/7 |
| c | 4 | 1/7 |
| c | 5 | 2/7 |

## The entire pipeline

(Bayesian network without parameters) + training examples

$\Downarrow$

**Learning**: maximum likelihood (with Laplace smoothing)

$\Downarrow$

Query $A \mid B \Rightarrow$ $\boxed{\text{Parameters } \boldsymbol{\theta} \text{ (defines Bayesian network)}}$ $\Rightarrow \mathbb{P}_{\theta}(A \mid B)$
(use inference algorithm)

## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- **Unsupervised learning**
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Supervision?

Supervised learning:

- Prediction: $\textbf{Train}$ contains input-output pairs $(x, y)$
- Fully-labeled data is very **expensive** to obtain, sometimes don't know what "correct labels" are (get 10000 labeled examples)

Unsupervised learning:

- Clustering: $\textbf{Train}$ only contains inputs $x$
- Unlabeled data is much **cheaper** to obtain (get 100 million unlabeled examples)

[Brown et al, 1992]

## Word clustering using HMMs

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

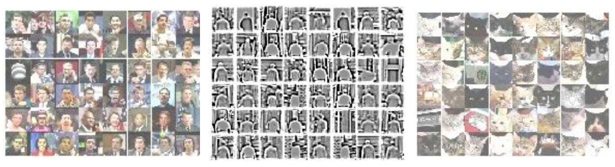
Impact: used in many state-of-the-art NLP systems

[Le et al, 2012]

## Feature learning using neural networks

Input: 10 million images (sampled frames from YouTube)

Output:

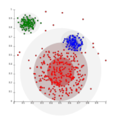


Impact: state-of-the-art results on object recognition (22,000 categories)

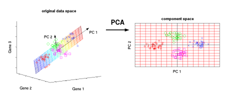
Key: data has lots of rich **latent** structures; want methods to discover this **structure** automatically

## Types of unsupervised learning

Clustering (e.g., K-means):



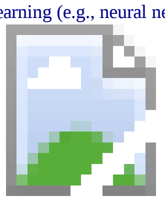
Dimensionality reduction (e.g., PCA):



Latent-variable models (e.g., HMMs):

$$\begin{tikzcd}
X_1 \arrow[r] \arrow[d] & X_2 \arrow[r] \arrow[d] & X_3 \arrow[r] \arrow[d] & X_4 \arrow[r] \arrow[d] & X_5 \arrow[d] \\
E_1 & E_2 & E_3 & E_4 & E_5
\end{tikzcd}$$

Feature learning (e.g., neural networks):



## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - **K-means clustering**
  - Latent-variable models and hard EM
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

## Clustering

**Clustering task**

Input: training set of input points $\text{Train} = \{x_1, \cdots, x_n\}$

Output: assignment of each input into a cluster $z_i \in \{1, \dots, K\}$

Desiderata: Want similar points to be put in same cluster, dissimilar points to put in different clusters

[Demo]

## K-means model

Setup:

- Each cluster $k = 1, \dots, K$ is represented by a **center** point $\mu_k \in \mathbb{R}^d$ (think of it as a prototype)
- Intuition: encode each point $\phi(x_i)$ by its cluster center $\mu_{z_i}$, pay for deviation

Variables:

- Cluster assignments $z = (z_1, \dots, z_n)$
- Cluster centers $\mu = (\mu_1, \dots, \mu_K)$

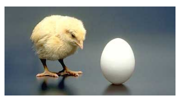
Loss function based on **reconstruction**:

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

## K-means algorithm

Goal:

$$\min_z \min_\mu \text{Loss}_{\text{reconstruct}}(z, \mu)$$



Strategy: alternating minimization / coordinate-wise descent

- E-step: if know cluster centers $\mu$, can find best $z$
- M-step: if know cluster assignments $z$, can find best cluster centers $\mu$

## K-means algorithm (E-step)

Goal: given cluster centers $\mu_1, \dots, \mu_K$, assign each point to the best cluster.

Solution:

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

Assign $i$ to cluster with closest center:

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

## K-means algorithm (M-step)

Goal: given cluster assignments $z_1, \dots, z_n$, find the best cluster centers $\mu_1, \dots, \mu_K$.

Solution:

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

Set center $\mu_k$ to average of points assigned to cluster $k$:

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

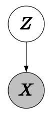
## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - **Latent-variable models and hard EM**
- Reinforcement learning
  - Q-learning
  - Exploration/exploitation

Notes

## Learning latent-variable models

Given Bayesian network with unknown parameters:



$$\mathbb{P}_\theta(Z = z, X = x)$$

- Observed variables: $X$
- Latent variables: $Z$
- Parameters: $\theta$

Optimization problem:

$$\max_z \max_\theta \mathbb{P}_\theta(X = x, Z = z)$$

Notes

## Expectation maximization (EM)

Objective: $\max_z \max_\theta \mathbb{P}_\theta(X = x, Z = z)$

E-step:
- Find latent variables with highest probability: $z \leftarrow \arg\max_{z'} \mathbb{P}_\theta(X = x, Z = z')$
- MAP inference: max variable elimination

M-step:
- Find the maximum likelihood parameters: $\theta \leftarrow \arg\max_{\theta'} \mathbb{P}_{\theta'}(X = x, Z = z)$
- Supervised learning: count and normalize

## Unsupervised learning summary

latent variables $z$

parameters $\theta$

Properties:

- Strategy: turn one hard problem into two easy problems
- Warning: not guaranteed to converge to global optimum (same issue with ICM, Gibbs sampling)

## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- **Reinforcement learning**
  - Q-learning
  - Exploration/exploitation

[Francis wyffels]

## Crawling robot

Goal: maximize distance travelled by robot

Markov decision process (MDP)?

- States: positions (4 possibilities) for each of 2 servos
- Actions: choose a servo, move it up/down
- Transitions: move into new position (**unknown dynamics**)
- Rewards: distance travelled (**unknown dynamics**)

## From MDPs to reinforcement learning

**Markov decision process (offline)**

States: $\mathsf{States}$

Actions: $\mathsf{Actions}(s)$ for each state $s$

Transitions: $T(s, a, s')$

Rewards: $\mathsf{Reward}(s, a)$

**Reinforcement learning (online)**

States: $\mathsf{States}$

Actions: $\mathsf{Actions}(s)$ for each state $s$

**Samples of transitions or rewards by acting!**

## Example

- States: board positions
- Actions: $\{\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}\}$ (that stay on board)
- Rewards: points for entering square
- Discount $\gamma = 0.95$
- Terminal states: squares with non-zero reward

| | | | |
|---|---|---|---|
| **0** | **0** | **-1** | **5** |
| **0** | **0** | **0** | **0** |
| **1** | **0** | **-1** | **10** |

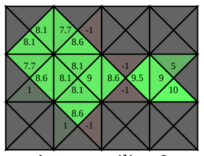
Average utility: 0.62

## Solving MDP via modified value iteration

| | | | |
|---|---|---|---|
| **0** | **0** | **-1** | **5** |
| **0** | **0** | **0** | **0** |
| **1** | **0** | **-1** | **10** |

Average utility: 0.82

Board



Average utility: 0

$Q(s, a)$ by solving MDP

- $Q(s, a)$ is maximum expected utility if take action $a$ in state $s$
- Given $Q$, optimal policy is $\pi_{\text{opt}}(s) = \arg\max_a Q(s, a)$

$$Q(s, a) = \mathsf{Reward}(s, a) + \gamma \sum_{s'} T(s, a, s') V(s'), \text{ where}$$

$$V(s') = \max_{a'} Q(s', a') \text{ is max. expected utility starting in state } s'$$

## Outline

- Supervised learning
  - Principles of learning and loss minimization
  - Linear regression
  - Stochastic gradient descent
  - Linear classification
  - Linearity, non-linearity, and kernels
  - Complexity control via regularization
  - Maximum likelihood for Bayesian networks
- Unsupervised learning
  - K-means clustering
  - Latent-variable models and hard EM
- Reinforcement learning
  - **Q-learning**
  - Exploration/exploitation

## Q-learning

MDP:

$$Q(s, a) = \mathsf{Reward}(s, a) + \gamma \sum_{s'} T(s, a, s') V(s')$$

Reinforcement learning (Q-learning):

In state $s$, took action $a$, got reward $r$, ended up in state $s'$:

Think regression:

$$\text{input } x = (s, a) \quad \Rightarrow \quad \text{output } y = r + \gamma \hat{V}(s')$$

Stochastic gradient update with step size $\eta_t$: [compare]

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) - \eta_t [\underbrace{\hat{Q}(s, a)}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}(s'))}_{\text{target}}]$$

## Outline

## Generating samples from a following a policy

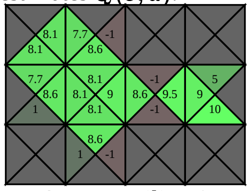
Where do samples $(s, a, r, s')$ come from? Agent obtains them by executing some policy $\pi_{\text{act}}$ (unlike supervised learning, agent gets to determine data).

**Q-learning algorithm**

```
Loop:
    Choose action $a = \pi_{\text{act}}(s)$.
    Execute action $a$, observe reward $r$ and new state $s'$.
    Update $\hat{Q}(s, a)$ using $(s, a, r, s')$ (might affect $\pi_{\text{act}}$).
    Set $s$ to $s'$.
```

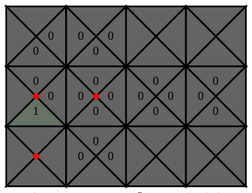
## Generating samples from a policy

What policy $\pi_{\text{act}}(s)$ to follow?
Attempt 1: Set $\pi_{\text{act}}(s) = \arg\max_a \hat{Q}(s, a)$ based on current estimates $\hat{Q}(s, a)$.



Average utility: 0

True $Q(s, a)$



Average utility: 0.99

Q-learning with current optimal policy

Problem: $\hat{Q}(s, a)$ estimates are inaccurate, **too greedy**!

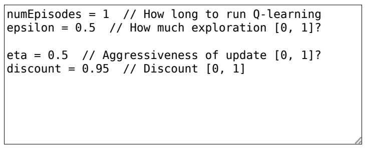
## $\epsilon$-greedy, exploration/exploitation tradeoff

Intuition: need to balance **exploration** and **exploitation**

$\epsilon$-greedy policy:

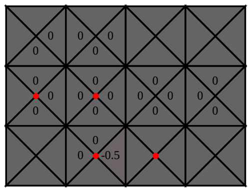
$$\pi_{\text{act}}(s) = \begin{cases} \arg\max_a \hat{Q}(s, a) & \text{probability } 1 - \epsilon, \\ \text{uniform over Actions}(s) & \text{probability } \epsilon. \end{cases}$$

Press ctrl-enter to run.



```
numEpisodes = 1  // How long to run Q-learning
epsilon = 0.5  // How much exploration [0, 1]?

eta = 0.5  // Aggressiveness of update [0, 1]?
discount = 0.95  // Discount [0, 1]
```



Average utility: -0.9

## Function approximation

Stochastic gradient update:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) - \eta[\underbrace{\hat{Q}(s, a)}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}(s'))}_{\text{target}}]$$

This is **rote learning**: every $\hat{Q}(s, a)$ has different value; doesn't generalize to unseen states/actions.

Linear regression model: define **features** $\phi(s, a)$ and set $\hat{Q}(s, a) = \mathbf{w} \cdot \phi(s, a)$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}(s, a)}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}(s'))}_{\text{target}}]\phi(s, a)$$

## Supervision summary

Supervised learning
input-output pairs $(x, y)$

**Reinforcement learning**
state-action-rewards-state $(s, a, r, s')$
new: actions determine data

Unsupervised learning
inputs $x$

$\downarrow$ Less supervision

## Summary

- Learning: training data $\Rightarrow$ model $\Rightarrow$ predictions
- Real goal: loss on future inputs; can't even evaluate!
- Objective function: loss minimization/maximum likelihood on training data + regularization/smoothing to mitigate overfitting
- Features: encode domain knowledge, arbitrary non-linear properties of inputs
- Algorithms:
    - stochastic gradient descent (supervised/reinforcement learning)
    - count+normalize (maximum likelihood)
    - alternating minimization (unsupervised learning)