# HMM (continued), Deep Learning

Sriram Sankararaman

The instructor gratefully acknowledges Fei Sha, Ameet Talwalkar, Eric Eaton, Andrew Moore and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

## Administrivia

- Additional details on final exam will be posted soon.
- Thank you for your patience.

## Outline

## Hidden Markov model

**Hidden states**: $\{1, \ldots, K\}$
**Observed states**: $\{1, \ldots, L\}$
**Initial probability**

$$\pi_i = P(X_1 = i)$$

**Transition probability**

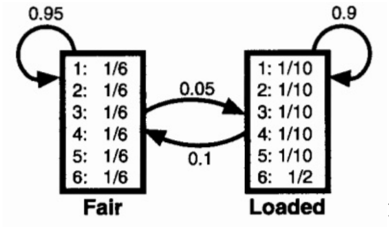
$$q_{ij} = P(X_{t+1} = i | X_t = j)$$

**Emission probability**

$$e_i(b) = P(Y_t = b | X_t = i)$$

## HMM: Example

**The occasionally dishonest casino**



[1]

- Hidden State: Is the casino using the fair or unfair die?
- Observed State: Roll of die ($\{1, \ldots, 6\}$)

[1] Durbin et al. Biological Sequence Analysis

## Querying an HMM

We observe $(Y_1, \ldots, Y_5) = (2, 6, 6, 1, 3)$. Can we infer for which of the throws the casino used the unfair die ?

## HMM

**The joint probability of the sequence of hidden states and the observed states**

$$\begin{aligned}
P(\boldsymbol{y},\boldsymbol{x}) &= P(y_{1:T}, x_{1:T}) \\
&= P(y_{1:T}|x_{1:T})P(x_{1:T}) \\
&= \prod_{t=1}^{T} P(y_t|x_t) \underbrace{P(x_{1:T})}_{\text{Markov chain}} \\
&= \prod_{t=1}^{T} e_{x_t}(y_t)\pi_{x_1} \prod_{t=1}^{T-1} q_{x_{t+1}x_t}
\end{aligned}$$

Easy to compute but not very useful because we don't know the hidden states.

## HMM: The most probable path problem

Given a sequence of observations $(y_1, \ldots, y_T)$, what is the most probable sequence of hidden states $(x_1, \ldots, x_T)$?

$$\arg\max_{x_{1:T}} P(y_{1:T}, x_{1:T})$$

- Often called the decoding problem.
- One way to solve this problem is to search over all possible values of $(x_{1:T})$.
- There are exponentially many of them (O$(K^T)$)
- Fortunately, it turns out there is an efficient dynamic programming algorithm to solve this problem.

## The Viterbi algorithm

### A recursive algorithm

- Suppose we have computed the the probability $v_t(k)$ for all values of $t \in \{1, \ldots, T\}$ and $k \in \{1, \ldots, K\}$:
  The probability of the most probable path (MPP) for observations $(y_1, \ldots, y_t)$ (so that path has length $t$) that ends up in state $k$.

$$v_t(k) = \max_{x_{1:t-1}} P(y_{1:t}, x_{1:t-1}, x_t = k)$$

Why is this a useful quantity?

- If we look at $v_T(k)$, it tells us the probability of the MPP of length $T$ that ends in state $k$.
- The answer to the MPP problem then is $\max_k v_T(k)$!

## The Viterbi algorithm

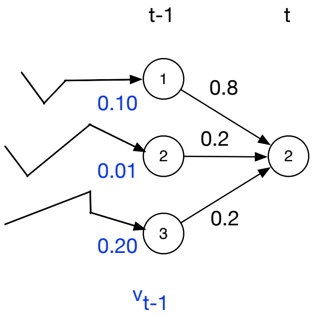
**Can we compute $v_t(k)$ efficiently?**

Assume we have computed $v_{t-1}(l)$.

We have computed the probability of the MPP of length $t-1$ that ends in state $l$ for all values of $l$

How do we use this to compute the probability of MPP of length $t$ that ends in state $k$?

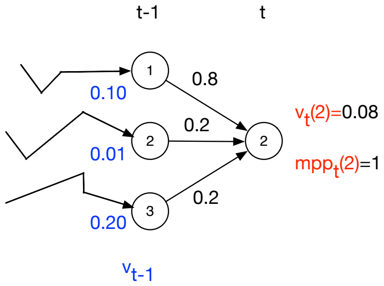
# Viterbi example



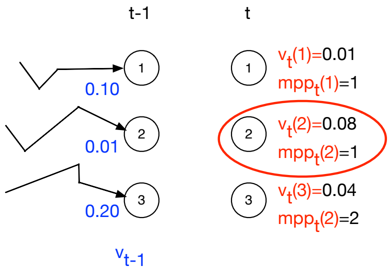
$v_t(2)$=max(0.8*0.10,
0.2*0.01,
0.20*0.2)

$mpp_t(2)$=argmax(0.8*0.10,
0.2*0.01,
0.20*0.2

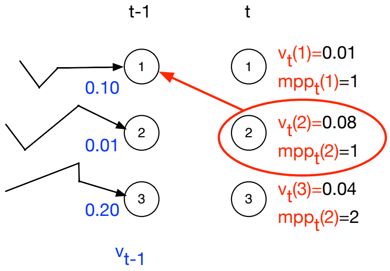
# Viterbi example



## Viterbi example



## Viterbi example



MPP for observations upto time $t$ (backwards):
$(2, 1, mpp_{t-1}(1), mpp_{t-2}(mpp_{t-1}(1)), \ldots)$

In other words, state $2$ at time $t$, state $1$ at time $t-1$,...

## The Viterbi algorithm

**Can we compute $v_t(k)$ efficiently?**

Begin with $t = 1$

$$\begin{aligned} v_1(k) &= P(y_1, x_1 = k) \\ &= P(y_1 | x_1 = k) P(x_1 = k) \\ &= e_k(y_1) \pi_k \end{aligned}$$

## The Viterbi algorithm

**Can we compute $v_t(k)$ efficiently?**

Assume we have computed $v_{t-1}(l)$.

We have computed the probability of the MPP of length $t-1$ that ends in state $l$ for all values of $l$

How do we use this to compute the probability of MPP of length $t$ that ends in state $k$?

## The Viterbi algorithm

### Can we compute $v_t(k)$ efficiently?

The most probable path with last two states $(l, k)$ is the most probable path with state $l$ at time $t-1$ followed by a transition from state $l$ to state $k$ and emitting the observation at time $t$.
What is the probability of this path?

$$\begin{aligned} & v_{t-1}(l)P(X_t = k | X_{t-1} = l)P(y_t | X_t = k) \\ & = v_{t-1}(l)q_{lk}e_t(y_t) \end{aligned}$$

So the most probable path that ends in state $k$ at time $t$ is obtained by maximizing over all possible states $l$ in the previous time $t-1$.

$$v_t(k) = \max_l v_{t-1}(l)q_{kl}e_t(y_t)$$

Also keep a pointer to the state that lead to the current state

$$mpp_t(k) = l^*$$

$$l^* = \arg\max_l v_{t-1}(l)q_{kl}e_t(y_t)$$

## The Viterbi algorithm

**Can we compute $v_t(k)$ efficiently?**

Continue till we compute $v_T(k), k \in \{1, \ldots, K\}$. Let:

$$k^* = \arg\max_k v_T(k)$$

To obtain the MPP, follow the pointers defined by $mpp_t(k)$.

## The Viterbi algorithm

**Can we compute $v_t(k)$ efficiently?**

$$v_t(k) = \max_l v_{t-1}(l) q_{kl} e_t(y_t)$$

- The cost of computing this is $\mathrm{O}(K)$ for a given $k$.
- The total cost of computing this is $\mathrm{O}(K^2)$ for all $k$.
- Total cost of computing $v_T(k)$ is $\mathrm{O}(TK^2)$.

## Other HMM computations

**Given a sequence of observations $(y_1, \ldots, y_T)$, what is the probability that state at time $t$ is $k$ ?**

$$P(X_t = k | y_{1:T})$$

**Given a sequence of observations $(y_1, \ldots, y_T)$, what is the probability of the observations ?**

$$P(y_{1:T})$$

Can also be computed efficiently using dynamic programming.

## Learning HMMs

**We assume the parameters are known. Can we learn parameters from data?**

Parameters of the HMM

$$\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{Q}, \boldsymbol{E})$$

Here $\boldsymbol{E}$ is the matrix of emission probabilities. $E_{kb} = e_k(b)$.

Given training data of observed states $(y_{1:T})$, find parameters $\boldsymbol{\theta}$ that maximizes the log likelihood.

## Learning HMMs

**We assume the parameters are known. Can we learn parameters from data?**

Our model contains observed and unobserved random variables and hence is incomplete.

- Observed: $\mathcal{D} = y_{1:T}$
- Unobserved (hidden): $x_{1:T}$

$$\begin{aligned}
\widehat{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \log P(y_{1:T}|\boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \log \sum_{x_{1:T}} P(y_{1:T}, x_{1:T}|\boldsymbol{\theta})
\end{aligned}$$

The objective function $\ell(\boldsymbol{\theta})$ is called the incomplete log likelihood.

We can optimize this function using the EM algorithm (we won't get into the details in this course).

## Summary

**HMM**

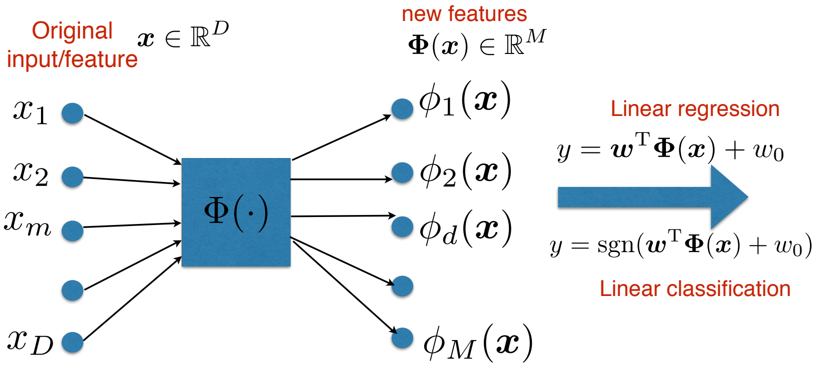
- Allows us to model dependencies.
- Can perform computations efficiently using dynamic programming.
- Can learn the parameters using the EM algorithm.
- Is widely used.

## Outline

## Basic idea

**Use nonlinear basis functions**

**Transform the input feature with nonlinear function**



Original input/feature
$\boldsymbol{x} \in \mathbb{R}^D$
new features
$\boldsymbol{\Phi}(\boldsymbol{x}) \in \mathbb{R}^M$
$x_1$
$x_2$
$x_m$
$x_D$
$\Phi(\cdot)$
$\phi_1(\boldsymbol{x})$
$\phi_2(\boldsymbol{x})$
$\phi_d(\boldsymbol{x})$
$\phi_M(\boldsymbol{x})$
Linear regression
$y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) + w_0$
$y = \mathrm{sgn}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) + w_0)$
Linear classification

## Nonlinear basis as two-layer network

### Layered architecture of “neurons”

Input layer: features

hidden layer: nonlinear transformation

Output layer: targets

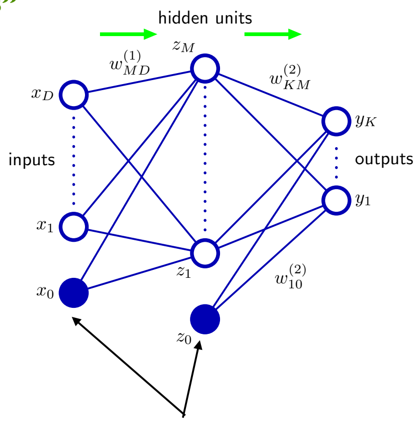
### Feedforward computation

hidden layer output:

$$z_j = h(a_j) = h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)$$

Output layer output

$$y_k = g\left(\sum_{j=0}^{M} w_{kj}^{(2)} z_j\right)$$

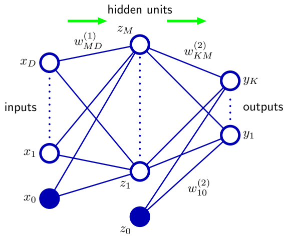


we often set these two have a constant value of 1, thus “bias”

## Neural networks are very powerful

### Sufficient

Universal approximator: with sufficient number of nonlinear hidden units, linear output unit can approximate any continuous functions



### Transfer function for the neurons

sigmoid function

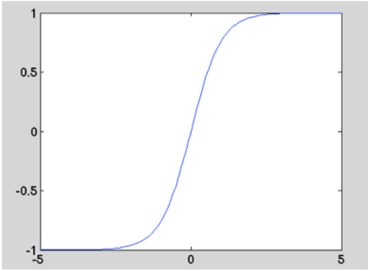
$$h(z) = \frac{1}{1+e^{-z}}$$

tanh function:

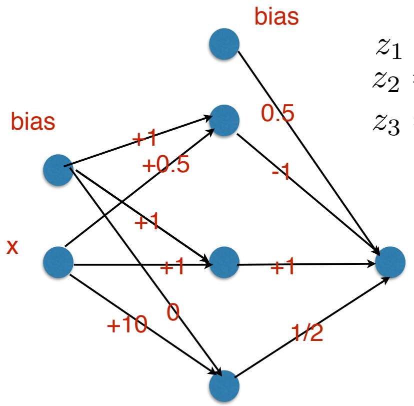
$$h(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

piecewise linear

$$h(z) = \max(0, z)$$



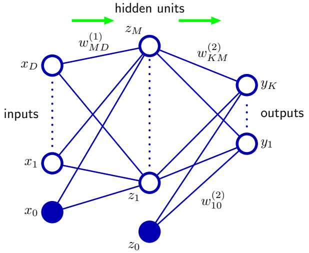
## Ex: computing highly nonlinear function



$$z_1 = h(0.5x + 1)$$
$$z_2 = h(x + 1)$$
$$z_3 = h(10x)$$

$$y = -z_1 + z_2 + 0.5 * z3 + 0.5$$

## Choice of output nodes



### Regression

Linear output

$$y_k = \sum_k w^{(2)}_{kj} h \left( \sum_i w^{(1)}_{ji} x_i \right)$$

### Classification

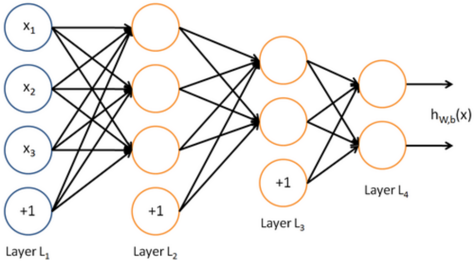
sigmoid (for binary classification)

$$y = \sigma \left( \sum_k w^{(2)}_{kj} h \left( \sum_i w^{(1)}_{ji} x_i \right) \right)$$

## Can have multiple (ie, deep) layers

Implements highly complicated nonlinear mapping

$$\boldsymbol{y} = f(\boldsymbol{x})$$



## How to learn the parameters?

### Choose the right loss function

Regression: least-square loss

$$\min \sum_n (f(\boldsymbol{x}_n) - y_n)^2$$

Classification: cross-entropy loss

$$\min \; - \sum_n \sum_k y_{nk} \log f_k(\boldsymbol{x}_n)$$

### Very hard optimization problem

Stochastic gradient descent is commonly used

Many optimization tricks are applied

## Stochastic gradient descent

### High-level idea

Randomly pick a data point ($x_n$, $y_n$)

Compute the gradient using only this data point, for example,

$$g = \frac{\partial [f(\boldsymbol{x}_n) - y_n)^2]}{\partial \boldsymbol{w}}$$

Update the parameter right away

$$\boldsymbol{w} = \boldsymbol{w} - \eta g$$

Iterate the process until some stop criteria

There are many possible improvements to this simple procedure
(in practice, this procedure works pretty well in many cases, though!)

## Several common tricks

### Initialization is very important

We are solving a very difficult optimization problem.

There are several heuristics on how to select your starting points wisely.

### Learning rate decay

Step size can be big in the begin but should be tuned down later, for example

$$\eta = \eta - t\delta_\eta$$

As the iteration t goes up, the learning grate becomes smaller.

### Minibatch

Use small batch of data points (instead just one) to estimate gradients more robustly.

### Momentum

Remembering the good direction in previous iterations that you have changed the parameters

....

## Heavy tuning

### In practice

- Many tricks require experimenting, and tweaking to obtain the best results
- Additionally, other hyperparameters need to be tuned too
  - Number of hidden layers?
  - Number of hidden units in each layers?
  - ...

### But all those pay off

- Deep neural networks attains the best results in automatic speech recognition.
- Deep neural networks attains the best results in image  recognition.
- Deep neural networks attains the best results in recognizing faces
- Deep neural networks attains the best results in recognizing poses.

## How to compute the gradient?

### Even for very complicated nonlinear functions

Computing the gradient is surprisingly simple to implement

The idea behind it is called error back propagation.

It employs the simple chain-rule for taking derivative.

### Implemented in many sophisticated packages

Theano

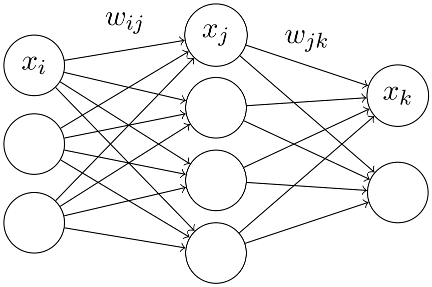
cuDNN

…

## Derivation of the error-backpropagation

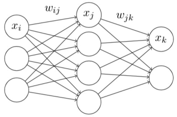
- Calculate the feed-forward signals from the input to the output.
- Calculate output error $E$ based on the predictions $x_k$ and the target $t_k$.
- Backpropagate the error by weighting it by the gradients of the associated activation functions and the weights in previous layers.
- Calculating the gradients $\frac{\partial E}{\partial w}$ for the parameters based on the backpropagated error signal and the feedforward signals from the inputs.
- Update the parameters using the calculated gradients $w \leftarrow w - \eta \frac{\partial E}{\partial w}$.

## Illustrative example



- $w_{ij}$: weights connecting node $i$ in layer $(\ell - 1)$ to node $j$ in layer $\ell$.
- $b_j$: bias for node $j$.
- $z_j$: input to node $j$ (where $z_j = b_j + \sum_i x_i w_{ij}$).
- $g_j$: activation function for node $j$ (applied to $z_j$).
- $x_j = g_j(z_j)$: ouput/activation of node $j$.
- $t_k$: target value for node $k$ in the output layer.

## Illustrative example (cont'd)



- Network output

$$x_k = g_k(b_k + \sum_j g_j(b_j + \sum_i x_i w_{ij}) w_{jk})$$

- Let's assume that the error function is the sum of the squared difference between the target values $t_k$ and the network output $\mathrm{x}_k$

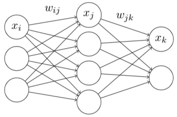
$$E = \frac{1}{2} \sum_{k \in K} (x_k - t_k)^2$$

- Gradients for the output layer

$$\frac{\partial E}{\partial w_{jk}} = (x_k - t_k) \frac{\partial}{\partial w_{jk}} (x_k - t_k) = (x_k - t_k) g_k'(z_k) \frac{\partial}{\partial w_{jk}} z_k$$
$$= (x_k - t_k) g_k'(z_k) x_j = \delta_k x_j \qquad \left( \begin{array}{ll} x_k & = g_k(z_k) \\ z_k & = b_k + w_{jk} x_j + \sum_{j' \neq j} w_{j'k} x_{j'} \end{array} \right)$$

where $\delta_k$ is the output error through the top activation layer.

## Illustrative example (cont'd)



- Gradients for the hidden layer

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k\in K}(x_k - t_k)\frac{\partial}{\partial w_{ij}}(x_k - t_k) = \sum_{k\in K}(x_k - t_k)g_k'(z_k)\frac{\partial}{\partial w_{ij}}z_k \quad \begin{pmatrix} x_k & = g_k(z_k) \\ z_k & = b_k + w_{jk}x_j + \sum_{j'\neq j} w_{j'k}x_{j'} \end{pmatrix}$$

$$= \sum_{k\in K}(x_k - t_k)g_k'(z_k)w_{jk}g_j'(z_j)x_i \quad \begin{pmatrix} x_j & = g_j(z_j) \\ z_j & = b_j + w_{ij}x_i + \sum_{i'\neq i} w_{i'j}x_{i'} \end{pmatrix}$$

$$= x_i g_j'(z_j)\sum_{k\in K}(x_k - t_k)g_k'(z_k)w_{jk} = \delta_j x_i \quad \left( \delta_j = g_j'(z_j)\sum_{k\in K}\delta_k w_{jk} \right)$$

where we substituted $z_k = b_k + \sum_j g_j(b_i + \sum_i x_i w_{ij})w_{jk}$
and $\frac{z_k}{w_{ij}} = \frac{z_k}{x_j}\frac{x_j}{w_{ij}} = w_{jk}g_j'(z_j)x_i$.

- The gradients with respect to the biases are respectively:

$$\frac{E}{\partial b_k} = \delta_k, \ \frac{E}{\partial b_i} = \delta_j$$

## Basic idea behind DNNs

**Architecturally, a big neural networks (with a lot of variants)**

- in depth: 4-5 layers are commonly (Google LeNet uses more than 20)
- in width: the number of hidden units in each layer can be a few thousands
- the number of parameters: hundreds of millions, even billions

**Algorithmically, many new things**

- Pre-training: do not do error-backprogation right away
- Layer-wise greedy: train one layer at a time
- ...

**Computing**

- Heavy computing: in both speed in computation and coping with a lot of data
- Ex: fast Graphics Processing Unit (GPUs) are almost indispensable

## Good references

- Easy to find as DNNs are very popular these days
- Many, many online video tutorials
- Good open-source packages: Theano, Caffe, MatConvNet, TensorFlow, etc
- Examples:
  - Wikipedia entry on "Deep Learning" `http://en.wikipedia.org/wiki/Deep_learning` provides a decent portal to many things including deep belief networks, convolution nets
  - A collection of tutorials and codes for implementing them in Python `http://www.deeplearning.net/tutorial/`

## Summary of the course

### Types of learning problems

- Supervised, unsupervised and reinforcement
- Labeled vs unlabeled data
- Labeled: Supervised learning. Type of label: classification (categorical) vs regression (quantitative)
- Unlabeled: Unsupervised learning.

## Summary of the course

**Supervised learning**

- Model/hypotheses
- Loss function
- Regularizer
- Algorithm to solve optimization problem

## Summary of the course

### Supervised learning

- Key goal is to pick hypothesis $h$ that minimizes risk for some loss function:

$$\mathcal{R}[h(\boldsymbol{x})] = \sum_{\boldsymbol{x},y} \ell(h(\boldsymbol{x}), y) p(\boldsymbol{x}, y)$$

Difficulty: we don't know the data generating distribution $p(\boldsymbol{x}, y)$.

- Instead pick $h$ that minimizes empirical risk (a.k.a training error)

$$\mathcal{R}^{\text{EMP}}[h(\boldsymbol{x})] = \frac{1}{N} \sum_{n} \ell(h(\boldsymbol{x}_n), y_n)$$

## Summary of the course

**Supervised learning**

- Setup: given a training dataset $\{\boldsymbol{x}_n, y_n\}_{n=1}^{N}$, learn a function $h(\boldsymbol{x})$ to predict $y$ given $\boldsymbol{x}$.
    - Choose hypothesis space/models.
    - Define a loss function.
    - Define a cost function (typically loss function evaluated over the training data + regularizer).
    - Algorithm to solve optimization problem.

## Summary of the course

### Supervised learning

- Hypotheses
  - Decision trees, Nearest neighbors
  - Linear models: $h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$
  - How we can convert linear to non-linear models: $h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})$ and use kernel trick to keep computations efficient.
  - Neural Networks: that jointly learn $\boldsymbol{\phi}$ and $\boldsymbol{w}$.
  - Ensembles as a way to combine classifiers.
- Loss functions
  - Squared loss: least squares for regression
  - 0-1 loss for binary classification and surrogates for 0-1 loss.
  - Logistic loss: logistic regression
  - Exponential loss: AdaBoost
  - Hinge loss: SVM
- Main principles
  - Many of these learning algorithms can be thought of as solving the problem of finding "good" parameters for some probabilistic model.
  - Principles for finding good parameters: Maximum likelihood, MAP, regularize likelihood

## Summary of the course

**Supervised learning**

- Optimization
  - Convex vs non-convex optimization problems
  - Methods: gradient descent (batch vs stochastic)
  - Constrained optimization. Lagrange function. Primal vs dual formulations.
- Concepts
  - Training error vs generalization error
  - Overfitting vs underfitting
  - The role of inductive bias
- Practical issues
  - How to tune hyperparameters, how to estimate generalization error
  - Importance of train-validation-test setup and cross-validation

## Unsupervised learning

- Finding structure in data.
- Dimensionality reduction, Clustering and mixture models, Modeling dependencies
- PCA
  - Linear Dimensionality reduction
  - Finds projections that maximize variance, minimize reconstruction error
  - Obtained by computing the top eigenvectors
- Clustering
  - K-means. Requires solving a non-convex problem
  - Can be viewed as a probabilistic model with hidden variable (GMM)
  - EM algorithm: iterative algorithm to estimate MLE
- Hidden Markov Models (HMM)
  - Model dependency among observations
  - Use dynamic programming to efficiently query the HMM

- Additional details on final exam will be posted soon.
- Will hold office hours next Tuesday 4-5pm over zoom.
- Thank you for your participation!