Announcements

- Today, I’ll summarize the most important points from the lecture.
  - It is an opportunity for you to ask questions…
  - …or get additional explanations about certain topics.
  - So, please do ask.
- Today’s slides are intended as an index for the lecture.
  - But they are not complete, won’t be sufficient as only tool.
  - Also look at the exercises - they often explain algorithms in detail.
- Exam procedure
  - Closed-book exam, the core exam time will be 2h.
  - We will send around an announcement with the exact starting times and places by email.

This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes
- Approximate Inference
  - Sampling Approaches
  - MCMC
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.

Recap: Regression

- Learning to predict a continuous function value
  - Given: training set $X = \{x_1, \ldots, x_N\}$ with target values $T = \{t_1, \ldots, t_N\}$.
  - Learn a continuous function $g(x)$ to predict the function value for a new input $x$.

- Define an error function $E(w)$ to optimize
  - E.g., sum-of-squares error
  - $E(w) = \frac{1}{2} \sum_{n=1}^{N} (g(x_n, w) - t_n)^2$
  - Procedure: Take the derivative and set it to zero
  - $\frac{\partial E(w)}{\partial w_j} = \sum_{n=1}^{N} (g(x_n, w) - t_n) \frac{\partial g(x_n, w)}{\partial w_j} = 0$

Recap: Least-Squares Regression

- Setup
  - Step 1: Define $\hat{x}_i = \begin{pmatrix} x_i \\ 1 \end{pmatrix}$, $\hat{w} = \begin{pmatrix} w \\ w_0 \end{pmatrix}$
  - Step 2: Rewrite $\hat{x}_i^T \hat{w} = t_i$, $\forall i = 1, \ldots, n$
  - Step 3: Matrix-vector notation
  - $\hat{X}^T \hat{w} = t$ with $\hat{X} = [\hat{x}_1 \ldots \hat{x}_N]^T$
  - $t = [t_1 \ldots t_N]^T$
  - Step 4: Find least-squares solution
  - $\|\hat{X}^T \hat{w} - t\|^2 \rightarrow \text{min}$
  - Solution: $\hat{w} = (\hat{X}^T \hat{X})^{-1} \hat{X} t$
This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes
- Approximate Inference
  - Sampling Approaches
  - MCMC
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.

Recap: Regularization

- Problem: Overfitting
  - Many parameters & little data ⇒ tendency to overfit to the noise
  - Side effect: The coefficient values get very large.
- Workaround: Regularization
  - Penalize large coefficient values

Recap: Probabilistic Regression

- First assumption:
  - Our target function values $t_i$ are generated by adding noise to the ideal function estimate:
  - Target function value
  - Regression function
  - Input value
  - Weights or parameters
  - Noise

- Second assumption:
  - The noise is Gaussian distributed.

Recap: Maximum Likelihood Regression

\[ \nabla_w \log p(t|X, w, \beta) = -\beta \sum_{i=1}^{N} (t_i - w^T \phi(x_i)) \phi(x_i) \]

- Setting the gradient to zero:

Recap: Role of the Precision Parameter

- Also use ML to determine the precision parameter $\beta$:
  - Gradient w.r.t. $\beta$:
  \[ \nabla_{\beta} \log p(t|X, w, \beta) = -\frac{1}{\beta_{\text{ML}}} \sum_{i=1}^{N} (t_i - w^T \phi(x_i))^2 + \frac{N}{2} \frac{1}{\beta^2} \]

⇒ The inverse of the noise precision is given by the residual variance of the target values around the regression function.
Recap: Predictive Distribution

- Having determined the parameters \( w \) and \( \beta \), we can now make predictions for new values of \( x \).

\[
p(t|X, w_{ML}, \beta_{ML}) = \mathcal{N}(t|\mu(X, w_{ML}), \sigma^{-2}_{ML})
\]

- This means
  - Rather than giving a point estimate, we can now also give an estimate of the estimation uncertainty.

Recap: Maximum-A-Posteriori Estimation

- Introduce a prior distribution over the coefficients \( w \).
  - For simplicity, assume a zero-mean Gaussian distribution

\[
p(w|\alpha) = \mathcal{N}(w|0, \alpha^{-1}I) = \left(\frac{\alpha}{2\pi}\right)^{M+1/2} \exp\left\{-\frac{\alpha}{2} w^T w\right\}
\]
  - New hyperparameter \( \alpha \) controls the distribution of model parameters.
  - Express the posterior distribution over \( w \).
    - Using Bayes’ theorem:

\[
p(w|X, t, \beta, \alpha) \propto p(t|X, w)p(w|\alpha)
\]
  - We can now determine \( w \) by maximizing the posterior.
  - This technique is called maximum-a-posteriori (MAP).

Recap: MAP Solution

- Minimize the negative logarithm

\[
\begin{align*}
-\log p(w|X, t, \beta, \alpha) &\propto -\log p(t|X, w, \beta) - \log p(w|\alpha) \\
&= \frac{\beta}{2} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2 + \text{const}
\end{align*}
\]

- The MAP solution is therefore

\[
\arg \min_w \frac{\beta}{2} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2 + \frac{\alpha}{2} w^T w
\]

\[
\Rightarrow \text{Maximizing the posterior distribution is equivalent to minimizing the regularized sum-of-squares error (with } \lambda = \frac{\alpha}{\beta} \).
\]

Recap: MAP Solution (2)

\[
\nabla_w \log p(w|X, t, \beta, \alpha) = -\beta \sum_{n=1}^{N} (t_n - w^T \phi(x_n)) \phi(x_n) + \alpha w
\]

- Setting the gradient to zero:

\[
0 = -\beta \sum_{n=1}^{N} (t_n - w^T \phi(x_n)) \phi(x_n) + \alpha w
\]

\[
\Rightarrow \sum_{n=1}^{N} t_n \phi(x_n) = \left[ \sum_{n=1}^{N} \phi(x_n) \phi(x_n)^T \right] w + \frac{\alpha}{\beta} w
\]

\[
\Rightarrow \Phi t = \left( \Phi \Phi^T + \frac{\alpha}{\beta} I \right) w \\
\Rightarrow t = \Phi \Phi^T + \frac{\alpha}{\beta} \Phi^T w
\]

Effect of regularization:

- Keeps the inverse well-conditioned

Recap: Bayesian Curve Fitting

- Given
  - Training data points: \( X = [x_1, \ldots, x_n] \in \mathbb{R}^{d \times n} \)
  - Associated function values: \( t = [t_1, \ldots, t_n]^T \)
  - Our goal is to predict the value of \( t \) for a new point \( x \).

- Evaluate the predictive distribution

\[
p(t|x, X, t) = \int p(t|x, w)p(w|X, t)dw
\]

- Noise distribution - again assume a Gaussian here

\[
p(t|x, w) = \mathcal{N}(t|\mu(x, w), \sigma^{-2})
\]

- Assume that parameters \( \alpha \) and \( \beta \) are fixed and known for now.

Recap: Bayesian Curve Fitting

- Under those assumptions, the posterior distribution is a Gaussian and can be evaluated analytically:

\[
p(t|x, X, t) = \mathcal{N}(t|m(x), \sigma^2(x))
\]

- where the mean and variance are given by

\[
m(x) = \beta \phi(x)^T S^{-1} \sum_{n=1}^{N} \phi(x_n) m_n
\]

\[
\sigma^2(x) = \beta^{-1} + \phi(x)^T S \phi(x)
\]

- and \( S \) is the regularized covariance matrix

\[
S^{-1} = \alpha I + \beta \sum_{n=1}^{N} \phi(x_n) \phi(x_n)^T
\]
Recap: Loss Functions for Regression

- **Optimal prediction**
  - Minimize the expected loss
  - Under squared loss, the optimal regression function is the mean $E[t|x]$ of the posterior $p(t|x)$ ("mean prediction").
  - For generalized linear regression function and squared loss:

$$y(x) = \int tN(t|w^T\phi(x), \beta^{-1})dt = w^T\phi(x)$$

Recap: Linear Basis Function Models

- Generally, we consider models of the following form
  - $y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T\phi(x)$
  - where $\phi_j(x)$ are known as basis functions.
  - In the simplest case, we use linear basis functions: $\phi_j(x) = x_j$.

- Other popular basis functions:
  - Polynomial
  - Gaussian
  - Sigmoid

Recap: Regularized Least-Squares

- Consider more general regularization functions
  - "L_q norms": $\frac{1}{2} \sum_{n=1}^{N} (t_n - w^T\phi(x_n))^2 + \lambda \sum_{j=1}^{M} |w_j|^q$

- **Effect:** Sparsity for $q \leq 1$.
  - Minimization tends to set many coefficients to zero

Recap: The Lasso

- $L_1$ regularization ("The Lasso")
  - $w = \arg \min_w \frac{1}{2} \sum_{n=1}^{N} (t_n - w^T\phi(x_n))^2 + \lambda \sum_{j=1}^{M} |w_j|$
  - The solution will be sparse (only few coefficients non-zero)
  - The $L_1$ penalty makes the problem non-linear.
  - There is no closed-form solution.

- Interpretation as Bayes Estimation
  - We can think of $|w_j|$ as the log-prior density for $w_j$.

- Prior for Lasso ($q = 1$):
  - Laplacian distribution
  - $p(w) = \frac{1}{2\tau} \exp \left(-\frac{|w|}{\tau}\right)$ with $\tau = \frac{1}{\lambda}$
This Lecture: Advanced Machine Learning

Regression Approaches
- Linear Regression
- Regularization (Ridge, Lasso)
- Kernels (Kernel Ridge Regression)
- Gaussian Processes

Approximate Inference
- Sampling Approaches
- MCMC

Deep Learning
- Linear Discriminants
- Neural Networks
- Backpropagation & Optimization
- CNNs, ResNets, RNNs, Deep RL, etc.

Recap: Kernel Ridge Regression
- Dual definition
  - Instead of working with $w$, substitute $w = \Phi^T a$ into $J(w)$ and write the result using the kernel matrix $K = \Phi \Phi^T$: $J(a) = \frac{1}{2} a^T K a - a^T K t + \frac{1}{2} t^T t + \lambda a^T a$.
  - Solving for $a$, we obtain $a = (K + \lambda I)^{-1} t$.
  - Prediction for a new input $x$:
    - Writing $k(x)$ for the vector with elements $k_i(x) = k(x, x_i)$.
    - $y(x) = w^T \phi(x) = a^T \phi(x) = k(x)^T (K + \lambda I)^{-1} t$.

Recap: Properties of Kernels
- Theorem
  - Let $k: X \times X \to \mathbb{R}$ be a positive definite kernel function. Then there exists a Hilbert Space $\mathcal{H}$ and a mapping $\phi: X \to \mathcal{H}$ such that $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$.
    - where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$.

Recap: The “Kernel Trick”
- How to kernelize an algorithm
  - Write the algorithm only in terms of inner products.
  - Replace all inner products by kernel function evaluations.

Recap: How to Check if a Function is a Kernel
- Problem:
  - Checking if a given $k: X \times X \to \mathbb{R}$ fulfills the conditions for a kernel is difficult.
  - We need to prove or disprove $\sum_{i,j=1}^{n} t_i k(x_i, x_j) t_j \geq 0$ for any set $x_1, \ldots, x_n \in X$ and any $t \in \mathbb{R}^n$ for any $n \in \mathbb{N}$.

- Workaround:
  - It is easy to construct functions $k$ that are positive definite kernels.
Recap: Gaussian Process

- **Gaussian distribution**
  - Probability distribution over scalars / vectors.
- **Gaussian process** (generalization of Gaussian distrib.)
  - Describes properties of functions.
  - Function: Think of a function as a long vector where each entry specifies the function value \( f(x_i) \) at a particular point \( x_i \).
  - Issue: How to deal with infinite number of points?
    - If you ask only for properties of the function at a finite number of points…
    - Then inference in Gaussian Process gives you the same answer if you ignore the infinitely many other points.
- **Definition**
  - A Gaussian process (GP) is a collection of random variables any finite number of which has a joint Gaussian distribution.

Recap: Gaussian Process

- **A Gaussian process is completely defined by**
  - Mean function \( m(x) \) and \( m(x) = \mathbb{E}[f(x)] \)
  - Covariance function \( k(x, x') \)
    \[ k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \]
  - We write the Gaussian process (GP)
    \[ f(x) \sim GP(m(x), k(x, x')) \]

Recap: GPs Define Prior over Functions

- **Distribution over functions:**
  - Specification of covariance function implies distribution over functions.
  - I.e. we can draw samples from the distribution of functions evaluated at a (finite) number of points.
- **Procedure**
  - We choose a number of input points \( X \).
  - We write the corresponding covariance matrix (e.g. using SE) element-wise:
    \[ K(X, X) \]
  - Then we generate a random Gaussian vector with this covariance matrix:
    \[ f \sim \mathcal{N}(0, K(X, X)) \]

Recap: Prediction with Noise-free Observations

- **Assume our observations are noise-free:**
  \[ \{(x_n, y_n) \mid n = 1, \ldots, N \} \]
  - Joint distribution of the training outputs \( f \) and test outputs \( f^* \) according to the prior:
    \[ \begin{bmatrix} f \\ f^* \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K(X, X) & K(X, X^*) \\ K(X^*, X) & K(X^*, X^*) \end{bmatrix} \right) \]
  - Calculation of posterior corresponds to conditioning the joint Gaussian prior distribution on the observations:
    \[ f_* | X, f \sim \mathcal{N}(\hat{f}, \text{cov}[f_*]) \]
    \[ \hat{f} = K(X, X)K(X, X^*)^{-1}f \]
    \[ \text{cov}[f_*] = K(X, X^*) - K(X, X)K(X, X^*)^{-1}K(X, X^*) \]

Recap: Prediction with Noisy Observations

- **Joint distribution of the observed values and the test locations under the prior:**
  \[ \begin{bmatrix} t \\ f_* \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} 0 \\ K(X, X) + \sigma_n^2I \\ K(X, X) \\ K(X, X) \end{bmatrix}, \begin{bmatrix} K(X, X) + \sigma_n^2I & K(X, X) \\ K(X, X) & K(X, X) \end{bmatrix} \right) \]
  - Calculation of posterior corresponds to conditioning the joint Gaussian prior distribution on the observations:
    - with:
      \[ f_* | X, f, t \sim \mathcal{N}(\hat{f}, \text{cov}[f_*]) \]
      \[ \hat{f} = K(X, X)K(X, X^*)^{-1}f \]
      \[ \text{cov}[f_*] = K(X, X^*) - K(X, X)K(X, X^*)^{-1}K(X, X^*) \]
  - This is the key result that defines Gaussian process regression!
    - Predictive distribution is Gaussian whose mean and variance depend on test points \( X \) and on the kernel \( k(x, x') \), evaluated on \( X \).

Recap: GP Regression Algorithm

- **Very simple algorithm**
  \[ \begin{align*}
  \text{input: } X & \text{ (inputs), } y \text{ (targets), } k \text{ (covariance function), } \sigma_n^2 \text{ (noise level), } x_c \text{ (test input)} \\
  1: & \text{ L := cholesky}(K + \sigma_n^2I) \\
  2: & \alpha := L^{-1}y \\
  3: & f_* := k(X, x_c) \\
  4: & v := k(x_c, x_c) - \alpha^T \alpha \\
  5: & \text{log}p(t | X) = -\frac{1}{2}t^T(K + \sigma_n^2I)^{-1}t - \frac{1}{2} \text{log} | K + \sigma_n^2I | - \frac{N}{2} \text{log} 2\pi \\
  \end{align*} \]
Recap: Computational Complexity

- Complexity of GP model
  - Training effort: $O(N^3)$ through matrix inversion
  - Test effort: $O(N^2)$ through vector-matrix multiplication
- Complexity of basis function model
  - Training effort: $O(M^3)$
  - Test effort: $O(M^2)$
- Discussion
  - If the number of basis functions $M$ is smaller than the number of data points $N$, then the basis function model is more efficient.
  - However, advantage of GP viewpoint is that we can consider covariance functions that can only be expressed by an infinite number of basis functions.
  - Still, exact GP methods become infeasible for large training sets.

Recap: Bayesian Model Selection for GPs

- Goal
  - Determine/learn different parameters of Gaussian Processes
- Hierarchy of parameters
  - Lowest level
    - $w$ - e.g. parameters of a linear model.
  - Mid-level (hyperparameters)
    - $\theta$ - e.g. controlling prior distribution of $w$.
  - Top level
    - Typically discrete set of model structures $H_i$.
- Approach
  - Inference takes place one level at a time.

Recap: Model Selection at Lowest Level

- Posterior of the parameters $w$ is given by Bayes’ rule
  \[
p(w|X, \theta, H_i) = \frac{p(t|X, w, \theta, H_i)p(w|\theta, H_i)}{p(t|X, \theta, H_i)} \]
  \[
p(t|X, \theta, H_i) = \int p(t|X, w, H_i)p(w|\theta, H_i)dw
  \]
- with
  - $p(t|X, w, H_i)$ likelihood and
  - $p(w|\theta, H_i)$ prior parameters $w$,
  - Denominator (normalizing constant) is independent of the parameters and is called marginal likelihood.
  \[
p(t|X, \theta, H_i) = \int p(t|X, w, H_i)p(w|\theta, H_i)dw
  \]

Recap: Model Selection at Mid Level

- Posterior of parameters $\theta$ is again given by Bayes’ rule
  \[
p(\theta|t, X, H_i) = \frac{p(t|X, \theta, H_i)p(\theta|H_i)}{p(\theta|X, H_i)} \]
  \[
p(\theta|H_i) = \int p(t|X, \theta, H_i)p(\theta|H_i)d\theta
  \]
  - where
    - The marginal likelihood of the previous level $p(t|X, \theta, H_i)$ plays the role of the likelihood of this level.
    - $p(\theta|H_i)$ is the hyperprior (prior of the hyperparameters)
    - Denominator (normalizing constant) is given by:

Recap: Model Selection at Top Level

- At the top level, we calculate the posterior of the model
  \[
p(H_i|t, X) = \frac{p(t|X, H_i)p(H_i)}{p(t|X)}
  \]
  - where
    - Again, the denominator of the previous level $p(t|X, H_i)$ plays the role of the likelihood.
    - $p(H_i)$ is the prior of the model structure.
    - Denominator (normalizing constant) is given by:

Recap: Bayesian Model Selection

- Discussion
  - Marginal likelihood is main difference to non-Bayesian methods
    - $p(t|X, H_i) = \int p(t|X, \theta, H_i)p(\theta|H_i)d\theta$
    - It automatically incorporates a trade-off between the model fit and the model complexity:
      - A simple model can only account for a limited range of possible sets of target values - if a simple model fits well, it obtains a high marginal likelihood.
      - A complex model can account for a large range of possible sets of target values - therefore, it can never attain a very high marginal likelihood.

Image source: Rasmussen & Williams, 2006
Regression Approaches

In general, assume we are given the pdf $p(x) = \frac{1}{Z} f(z)$. The resulting:

Motivation: Avoid having to determine the constant $k$, but the distribution becomes correct in the limit $L \to \infty$.

Deep Learning

Approach

- Linear Discriminants
- Neural Networks
- Backpropagation & Optimization
- CNNs, ResNets, RNNs, Deep RL, etc.

Approximate Inference

- Sampling Approaches
- MCMC

Sampling idea

Assumptions

- Sampling directly from $p(x)$ is difficult.
- But we can easily evaluate $p(x)$ (up to some norm. factor $Z_x$):
  \[
p(x) = \frac{1}{Z_x} f(z)
  \]

Idea

- We need some simpler distribution $q(x)$ (called proposal distribution) from which we can draw samples.
- Choose a constant $k$ such that: $\forall z : kq(z) \geq p(z)$

Sampling procedure

1. Generate a number $z_s$ from $q(z)$.
2. Generate a number $u$, from the uniform distribution over $[0, kq(z_s)]$.
3. If $u > \bar{p}(z_s)$ reject sample, otherwise accept.

Recap: Rejection Sampling

Recap: Importance Sampling

Approach

- Approximate expectations directly (but does not enable to draw samples from $p(x)$ directly).

Goal:

\[ \mathbb{E}_q[f] = \int f(x)p(x)dx = \int f(x)p(x)q(x)dx \]

\[ \approx \frac{1}{L} \sum_{l=1}^{L} \frac{p(x_l)}{q(x_l)} f(x_l) \]

Importance weights

Recap: Sampling-Importance-Resampling

Motivation: Avoid having to determine the constant $k$.

Two stages

- Draw $L$ samples $x^{(1)}, ..., x^{(L)}$ from $q(x)$.
- Construct weights using importance weighting

\[ w_l = \frac{p(x_l)}{\sum_m p(x_m)} \]

and draw a second set of samples $x^{(1)*}, ..., x^{(L)*}$ with probabilities given by the weights $w^{(1)*}, ..., w^{(L)*}$.

Result

The resulting $L$ samples are only approximately distributed according to $p(x)$, but the distribution becomes correct in the limit $L \to \infty$.
Overview

- Allows to sample from a large class of distributions.
- Scales well with the dimensionality of the sample space.

Properties

- For homogeneous Markov chain, distribution
- Sufficient (but not necessary) condition to ensure that a
- The new candidate sample
- \( p \)

Approach

- At each time step, we generate a candidate sample from the proposal distribution and accept the sample according to a criterion.
- Different variants of MCMC for different criteria.

Recap: MCMC - Markov Chain Monte Carlo

Recap: Detailed Balance

- Detailed balance means
- If we pick a state from the target distribution \( p(x) \) and make a transition under \( T \) to another state, it is just as likely that we will pick \( z \) and go from \( x_i \) to \( x_j \) than that we will pick \( x_j \) and go from \( x_i \) to \( x_j \).

- It can easily be seen that a transition probability that satisfies detailed balance w.r.t. a particular distribution will leave that distribution invariant, because

\[
\sum_{a'} p'(a')T(a',a) = \sum_{a'} p'(a)T(a,a') = p'(a) \sum_{a'} p(a') = p'(a)
\]

Recap: MCMC - Metropolis Algorithm

Recap: Markov Chains - Properties

- Invariant distribution
- A distribution is said to be invariant (or stationary) w.r.t. a Markov chain if each step in the chain leaves that distribution invariant.
- Transition probabilities:
- For homogeneous Markov chain, distribution \( p(x) \) is invariant if:

\[
p'(x) = \sum_{a'} T(a',x)p'(x')
\]

- Detailed balance
- Sufficient (but not necessary) condition to ensure that a distribution is invariant:

\[
p'(x)/T(a,x') = p'(x')/T(x',a)
\]

- A Markov chain which respects detailed balance is reversible.

Recap: Gibbs Sampling

- Approach
- MCMC-algorithm that is simple and widely applicable.
- May be seen as a special case of Metropolis-Hastings.

- Idea
- Sample variable-wise: replace \( x_i \) by a value drawn from the distribution \( p(z_i|x_{\neg i}) \).
- This means we update one coordinate at a time.
- Repeat procedure either by cycling through all variables or by choosing the next variable.

- Properties
- The algorithm always accepts!
- Completely parameter free.
- Can also be applied to subsets of variables.

This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes

- Approximate Inference
  - Sampling Approaches
  - MCMC

- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.
Recap: Linear Discriminant Functions

- **Basic idea**
  - Directly encode decision boundary
  - Minimize misclassification probability directly.

- **Linear discriminant functions**
  \[ y(x) = w^T x + w_0 \]
  
  - Weight vector "bias" (\( w, w_0 \) define a hyperplane in \( R^d \)).
  - If a data set can be perfectly classified by a linear discriminant, then we call it linearly separable.

Recap: Generalized Linear Discriminants

- **Extension with non-linear basis functions**
  - Transform vector \( x \) with \( M \) nonlinear basis functions \( \phi_j(x) \):
    \[ y_k(x) = g \left( \sum_{j=1}^{M} w_{kj} \phi_j(x) + w_{0k} \right) \]
  - Basis functions \( \phi_j(x) \) allow non-linear decision boundaries.
  - Activation function \( g(\cdot) \) bounds the influence of outliers.
  - Disadvantage: minimization no longer in closed form.

- **Notation**
  \[ y_k(x) = g \left( \sum_{j=0}^{M} w_{kj} \phi_j(x) \right) \] with \( \phi_0(x) = 1 \)

Recap: Gradient Descent

- **Iterative minimization**
  - Start with an initial guess for the parameter values \( w_0^{(0)} \).
  - Move towards a (local) minimum by following the gradient.

- **Basic strategies**
  - "Batch learning"
    \[ w_{kj}^{(r+1)} = w_{kj}^{(r)} - \eta \frac{\partial E(w)}{\partial w_{kj}} \mid_{w=w^{(r)}} \]
  - "Sequential updating"
    \[ w_{kj}^{(r+1)} = w_{kj}^{(r)} - \eta \frac{\partial E_n(w)}{\partial w_{kj}} \mid_{w=w^{(r)}} \]

  where \( E(w) = \sum_{n=1}^{N} E_n(w) \)

Recap: Probabilistic Discriminative Models

- **Consider models of the form**
  \[ p(C_1|\phi) = y(\phi) = \sigma(w^T \phi) \]
  
  with \[ p(C_2|\phi) = 1 - p(C_1|\phi) \]

  - This model is called logistic regression.

- **Properties**
  - Probabilistic interpretation
  - But discriminative method: only focus on decision hyperplane
  - Advantageous for high-dimensional spaces, requires less parameters than explicitly modeling \( p(\phi|C_1) \) and \( p(C_1) \).

Recap: Logistic Sigmoid

- **Properties**
  - **Definition:**
    \[ \sigma(a) = \frac{1}{1 + \exp(-a)} \]

  - **Inverse:**
    \[ a = \ln \left( \frac{\sigma}{1 - \sigma} \right) \]

  - **Symmetry property:**
    \[ \sigma(-a) = 1 - \sigma(a) \]

  - **Derivative:**
    \[ \frac{da}{da} = \sigma(1 - \sigma) \]
Let’s consider a data set \( \{ \phi_n, t_n \} \) with \( n = 1, \ldots, N \), where \( \phi_n = \phi(x_n) \) and \( t_n \in \{0, 1\} \), \( t = (t_1, \ldots, t_N)^T \).

With \( y_n = p(C_1 | \phi_n) \), we can write the likelihood as

\[
p(t | w) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}
\]

- Define the error function as the negative log-likelihood

\[
E(w) = -\ln p(t | w)
\]

\[
= - \sum_{n=1}^{N} \left( t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right)
\]

This is the so-called cross-entropy error function.

---

Recap: Logistic Regression

- Let’s consider a data set \( \{ \phi_n, t_n \} \) with \( n = 1, \ldots, N \), where \( \phi_n = \phi(x_n) \) and \( t_n \in \{0, 1\} \), \( t = (t_1, \ldots, t_N)^T \).
- With \( y_n = p(C_1 | \phi_n) \), we can write the likelihood as

\[
p(t | w) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}
\]

- Define the error function as the negative log-likelihood

\[
E(w) = -\ln p(t | w)
\]

\[
= - \sum_{n=1}^{N} \left( t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right)
\]

This is the so-called cross-entropy error function.

---

Recap: Iteratively Reweighted Least Squares

- Result of applying Newton-Raphson to logistic regression

\[
w^{(r+1)} = w^{(r)} - (\Phi^T \Phi)^{-1} \Phi^T (y - t)
\]

\[
= (\Phi^T \Phi)^{-1} \left\{ \Phi^T \Phi w^{(r)} - \Phi^T (y - t) \right\}
\]

\[
= (\Phi^T \Phi)^{-1} \Phi^T R z
\]

with \( z = \Phi w^{(r)} - R^{-1} (y - t) \)

- Very similar form to pseudo-inverse (normal equations)
  - But now with non-constant weighting matrix \( R \) (depends on \( w \)).
  - Need to apply normal equations iteratively.
  - \( \Rightarrow \) Iteratively Reweighted Least-Squares (IRLS)

---

Recap: Softmax Regression

- Multi-class generalization of logistic regression
  - In logistic regression, we assumed binary labels \( t_n \in \{0, 1\} \).
  - Softmax generalizes this to \( K \) values in 1-of-\( K \) notation.

\[
y(x; w) = \begin{pmatrix}
P(y = 1 | x; w) \\
P(y = 2 | x; w) \\
\vdots \\
P(y = K | x; w)
\end{pmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(w_j^T x)} \begin{pmatrix}
\exp(w_1^T x) \\
\exp(w_2^T x) \\
\vdots \\
\exp(w_K^T x)
\end{pmatrix}
\]

This uses the softmax function

\[
\frac{\exp(a_k)}{\sum_j \exp(a_j)}
\]

- Note: the resulting distribution is normalized.

---

Recap: Softmax Regression Cost Function

- Logistic regression
  - Alternative way of writing the cost function

\[
E(w) = - \sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right\}
\]

\[
= - \sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right\}
\]

- Softmax regression
  - Generalization to \( K \) classes using indicator functions.

\[
E(w) = - \sum_{n=1}^{N} \sum_{k=1}^{K} \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(w_k^T x)}{\sum_{j=1}^{K} \exp(w_j^T x)} \right\}
\]

\[
\nabla_w E(w) = - \sum_{n=1}^{N} \left[ \mathbb{I}(t_n = k) \ln \frac{\exp(w_k^T x)}{\sum_{j=1}^{K} \exp(w_j^T x)} \right]
\]

This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes

- Approximate Inference
  - Sampling Approaches
  - MCMC

- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.
Recap: Perceptrons
- One output node per class
  \[ y_k(x) = w_k(x) \]
  \[ W_0 \] \[ \ldots \] \[ W_d \]
  \[ x_0 = 1 \] \[ x_1 \] \[ \ldots \] \[ x_d \]
  Output layer
  Weights
  Input layer
- Outputs
  - Linear outputs
  - With output nonlinearity
  \[ y_k(x) = \sum_{i=0}^d W_{ki} x_i \]
  \[ y_k(x) = g \left( \sum_{i=0}^d W_{ki} x_i \right) \]
  \[ \Rightarrow \text{Can be used to do multidimensional linear regression or multiclass classification.} \]

Recap: Non-Linear Basis Functions
- Straightforward generalization
  \[ y_k(x) = \sum_{i=0}^d W_{ki} \phi(x_i) \]
  \[ \phi(x_i) = \cdots \]
  \[ x_0 = 1 \] \[ x_1 \] \[ \ldots \] \[ x_d \]
  Output layer
  Weights
  Feature layer
  Mapping (fixed)
  Input layer
- Outputs
  - Linear outputs
  - With output nonlinearity
  \[ y_k(x) = \sum_{i=0}^d W_{ki} \phi(x_i) \]

Recap: Perceptron Learning
- Process the training cases in some permutation
  - If the output unit is correct, leave the weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a one, subtract the input vector from the weight vector.
- Translation
  \[ w_{kj}^{(r+1)} = w_{kj}^{(r)} - \eta \left( y_k(x_n; \mathbf{w}) - t_{kn} \right) \phi_j(x_n) \]
  \[ \Rightarrow \text{Perceptron Learning corresponds to 1-st order (stochastic)} \]
  \[ \Rightarrow \text{Gradient Descent of a quadratic error function!} \]

Recap: Loss Functions
- We can now also apply other loss functions
  - L_1 loss
    \[ L(t, y(x)) = \sum_n (y_n - t_n)^2 \]
    \[ \Rightarrow \text{Least-squares regression} \]
  - L_1 loss:
    \[ L(t, y(x)) = \sum_n |y_n - t_n| \]
    \[ \Rightarrow \text{Median regression} \]
  - Cross-entropy loss
    \[ L(t, y(x)) = -\sum_n \left( t_n \ln y_n + (1 - t_n) \ln (1 - y_n) \right) \]
    \[ \Rightarrow \text{Logistic regression} \]
  - Hinge loss
    \[ L(t, y(x)) = \sum_n \max(0, 1 - t_n y_n) \]
    \[ \Rightarrow \text{SVM classification} \]
  - Softmax loss
    \[ L(t, y(x)) = -\sum_n \sum_k \left( \frac{1}{k = 0} \ln \left( \frac{\exp(\sum_{j=0}^d W_{kj}^0 x_j)}{\sum_{j=0}^d \exp(\sum_{j=0}^d W_{kj}^0 x_j)} \right) \right) \]
    \[ \Rightarrow \text{Multi-class probabilistic classification} \]

Recap: Multi-Layer Perceptrons
- Adding more layers
  \[ y_k(x) = g^{(2)} \left( \sum_{i=0}^h W_{ki}^{(2)} y^{(1)} \right) \]
  \[ x_0 = 1 \] \[ x_1 \] \[ \ldots \] \[ x_d \]
  Output layer
  Hidden layer
  Input layer
- Output
  \[ y_k(x) = g^{(2)} \left( \sum_{i=0}^h W_{ki}^{(2)} y^{(1)} \right) \]
  \[ \Rightarrow \text{Direct generalization of one-layer Perceptrons} \]
This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes
- Approximate Inference
  - Sampling Approaches
  - MCMC
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.

Recap: Learning with Hidden Units

- How can we train multi-layer networks efficiently?
  - Need an efficient way of adapting all weights, not just the last layer.
- Idea: Gradient Descent
  - Set up an error function
    \[ E(W) = \sum_{i} L(t_i, y_i(W)) + \lambda \Omega(W) \]
  - With a loss \( L(.) \) and a regularizer \( \Omega(.) \).
  - E.g., \( L(t, y(W)) = \sum_{i} (y_i(W) - t_i)^2 \) L_2 loss
  - \( \Omega(W) = ||W||_F^2 \) L_2 regularizer ("weight decay")
  \[ \Rightarrow \text{Update each weight } W_{ij} \text{ in the direction of the gradient } \frac{dE}{dW_{ij}} \]

Recap: Gradient Descent

- Two main steps
  1. Computing the gradients for each weight
  2. Adjusting the weights in the direction of the gradient
- We consider those two steps separately
  - Computing the gradients: Backpropagation
  - Adjusting the weights: Optimization techniques

Recap: Backpropagation Algorithm

- Core steps
  1. Convert the discrepancy between each output and its target value into an error derivative.
  \[ \frac{\partial E}{\partial y_j} = -(t_j - y_j) \]
  2. Compute error derivatives in each hidden layer from error derivatives in the layer above.
  3. Use error derivatives w.r.t. activities to get error derivatives w.r.t. the incoming weights

Recap: MLP Backpropagation Algorithm

- Forward Pass
  \[ y^{(0)} = x \]
  for \( k = 1, \ldots, l \) do
  \[ y^{(k)} = W^{(k)} y^{(k-1)} \]
  \[ y^{(k)} = g_k(u^{(k)}) \]
  endfor
  \[ y = y^{(l)} \]
  \[ E = L(t, y) + \lambda \Omega(W) \]
- Backward Pass
  \[ h \leftarrow \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} L(t, y) + \lambda \frac{\partial}{\partial y} \Omega \]
  for \( k = l, l-1, \ldots, 1 \) do
  \[ h \leftarrow \frac{\partial E}{\partial W^{(k)}} = h \odot g'(y^{(k)}) \]
  \[ \frac{\partial E}{\partial W^{(k)}} = h y^{(k-1)T} + \lambda \frac{\partial}{\partial W^{(k)}} \Omega \]
  \[ h \leftarrow \frac{\partial E}{\partial y^{(k-1)}} = W^{(k)^T} h \]
  endfor
- Notes
  - For efficiency, an entire batch of data \( X \) is processed at once.
  - \( \odot \) denotes the element-wise product
Recap: Computational Graphs

- **Forward Mode Differentiation**: $\frac{\partial y}{\partial x}$
  - Apply operator to every node.

- **Reverse Mode Differentiation**: $\frac{\partial x}{\partial f}$
  - Apply operator to every node.

- Forward differentiation needs one pass per node. Reverse-mode differentiation can compute all derivatives in one single pass.

$\Rightarrow$ Speed-up in $O(\#\text{inputs})$ compared to forward differentiation!

Recap: Automatic Differentiation

- **Approach for obtaining the gradients**
  - Convert the network into a computational graph.
  - Each new layer/module just needs to specify how it affects the forward and backward passes.
  - Apply reverse-mode differentiation.

$\Rightarrow$ Very general algorithm, used in today’s Deep Learning packages.

Recap: Data Augmentation

- **Effect**
  - Much larger training set
  - Robustness against expected variations

- **During testing**
  - When cropping was used during training, need to again apply crops to get same image size.
  - Beneficial to also apply flipping during test.
  - Applying several ColorPCA variations can bring another ~1% improvement, but at a significantly increased runtime.

Recap: Normalizing the Inputs

- **Convergence is fastest if**
  - The mean of each input variable over the training set is zero.
  - The inputs are scaled such that all have the same covariance.
  - Input variables are uncorrelated if possible.

- **Advisable normalization steps for MLPs**
  - Normalize all inputs that an input unit sees to zero-mean, unit covariance.
  - If possible, try to decorrelate them using PCA (also known as Karhunen-Loève expansion).

Recap: Choosing the Right Learning Rate

- **Convergence of Gradient Descent**
  - Simple 1D example
    $$W^{(r+1)} = W^{(r)} - \eta \frac{dE(W)}{dW}$$
  - What is the optimal learning rate $\eta_{opt}$?
    $$\eta_{opt} = \frac{d^2E(W^{(r)})}{dW^2}$$
  - Advanced optimization techniques try to approximate the Hessian by a simplified form.
    - If we exceed the optimal learning rate, bad things happen!
Recap: Advanced Optimization Techniques

- **Momentum**
  - Instead of using the gradient to change the position of the weight "particle", use it to change the velocity.
  - Effect: dampen oscillations in directions of high curvature
  - Nesterov-Momentum: Small variation in the implementation

- **RMS-Prop**
  - Separate learning rate for each weight: Divide the gradient by a running average of its recent magnitude.

- **AdaGrad**
- **AdaDelta**
- **Adam**

Some more recent techniques, work better for some problems. Try them.

Recap: Patience

- Saddle points dominate in high-dimensional spaces!

⇒ Learning often doesn’t get stuck, you just may have to wait...

Recap: Reducing the Learning Rate

- Final improvement step after convergence is reached
  - Reduce learning rate by a factor of 10.
  - Continue training for a few epochs.
  - Do this 1-3 times, then stop training.

- Effect
  - Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different minibatches.

- Be careful: Do not turn down the learning rate too soon!
  - Further progress will be much slower after that.

Recap: Glorot Initialization

- Variance of neuron activations
  - Suppose we have an input $X$ with $n$ components and a linear neuron with random weights $W$ that spits out a number $Y$.
  - We want the variance of the input and output of a unit to be the same, therefore $\text{Var}(W)$ should be 1. This means
  - Or for the backpropagated gradient
  - As a compromise, Glorot & Bengio propose to use
  - Randomly sample the weights with this variance. That’s it.

Recap: Batch Normalization

- Motivation
  - Optimization works best if all inputs of a layer are normalized.

- Idea
  - Introduce intermediate layer that centers the activations of the previous layer per minibatch.
  - I.e., perform transformations on all activations and undo those transformations when backpropagating gradients

- Effect
  - Much improved convergence
Recap: Dropout

[Srivastava, Hinton '12]

- Idea
  - Randomly switch off units during training.
  - Change network architecture for each data point, effectively training many different variants of the network.
  - When applying the trained network, multiply activations with the probability that the unit was set to zero.

⇒ Greatly improved performance

Recap: ImageNet Challenge 2012

- ImageNet
  - ~14M labeled internet images
  - 20k classes
  - Human labels via Amazon Mechanical Turk

- Challenge (ILSVRC)
  - 1.2 million training images
  - 1000 classes
  - Goal: Predict ground-truth class within top-5 responses
  - Currently one of the top benchmarks in Computer Vision

Recap: Convolutional Neural Networks

- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant features
  - Classification layer at the end

Recap: CNN Structure

- Feed-forward feature extraction
  1. Convolve input with learned filters
  2. Non-linearity
  3. Spatial pooling
  4. (Normalization)

- Supervised training of convolutional filters by back-propagating classification error

Recap: Intuition of CNNs

- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels

- Learn multiple filters
  - E.g. 1000×1000 image 100 filters
  - Only 10k parameters

- Result: Response map
  - Size: 1000×1000×100
  - Only memory, not params!
Recap: Convolution Layers

- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth
  - Form a single \([1 \times 1 \times \text{depth}]\) depth column in output volume.

Recap: Activation Maps

- Each activation map is a depth slice through the output volume.

Recap: Pooling Layers

- **Effect:**
  - Make the representation smaller without losing too much information
  - Achieve robustness to translations

Recap: AlexNet (2012)

- **Similar framework as LeNet, but**
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data \(10^6\) images instead of \(10^3\)
  - GPU implementation
  - Better regularization and up-to-date tricks for training (Dropout)

Recap: VGGNet (2014/15)

- **Main ideas**
  - Deeper network
  - Stacked convolutional layers with smaller filters (+ nonlinearity)
  - Detailed evaluation of all components

- **Results**
  - Improved ILSVRC top-5 error rate to 6.7%.
Recap: Residual Networks

- Core component
  - Skip connections bypassing each layer
  - Better propagation of gradients to the deeper layers
  - This makes it possible to train (much) deeper networks.

Recap: Transfer Learning with CNNs

1. Train on ImageNet
2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier
   I.e., replace the Softmax layer at the end
3. If you have a medium sized dataset, "finetune" instead: use the old weights as initialization, train the full network or only some of the higher layers.

Recap: Visualizing CNNs

DeconvNet
ConvNet

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Recap: R-CNN for Object Detection

- One network, four losses
  - Remove dependence on external region proposal algorithm.
  - Instead, infer region proposals from same CNN.
  - Feature sharing
  - Joint training
  - Object detection in a single pass becomes possible.
Recap: Fully Convolutional Networks
- CNN
- FCN
- Intuition
  - Think of FCNs as performing a sliding-window classification, producing a heatmap of output scores for each class

Recap: Image Segmentation Networks
- Encoder-Decoder Architecture
  - Problem: FCN output has low resolution
  - Solution: perform upsampling to get back to desired resolution
  - Use skip connections to preserve higher-resolution information

This Lecture: Advanced Machine Learning
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes
- Approximate Inference
  - Sampling Approaches
  - MCMC
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.

Recap: Neural Probabilistic Language Model
- Core idea
  - Learn a shared distributed encoding (word embedding) for the words in the vocabulary.

Recap: word2vec
- Goal
  - Make it possible to learn high-quality word embeddings from huge data sets (billions of words in training set).
- Approach
  - Define two alternative learning tasks for learning the embedding:
    - "Continuous Bag of Words" (CBOW)
    - "Skip-gram"
  - Designed to require fewer parameters.

Recap: word2vec CBOW Model
- Continuous BOW Model
  - Remove the non-linearity from the hidden layer
  - Share the projection layer for all words (their vectors are averaged)
  - Bag-of-Words model (order of the words does not matter anymore)
Recap: word2vec Skip-Gram Model

- Continuous Skip-Gram Model
  - Similar structure to CBOW
  - Instead of predicting the current word, predict words within a certain range of the current word.
  - Give less weight to the more distant words
- Implementation
  - Randomly choose a number \( R \in [1, C] \).
  - Use \( R \) words from history and \( R \) words from the future of the current word as correct labels.
  \( \Rightarrow R + R \) word classifications for each input.

Problems with 100k-1M outputs

- Weight matrix gets huge!
  - Example: CBOW model
  - One-hot encoding for inputs
  \( \Rightarrow \) Input-hidden connections are just vector lookups.
  - This is not the case for the hidden-output connections!
  - State \( h \) is not one-hot, and vocabulary size is 1M.
  \( \Rightarrow W_{V \times 1} \) has 300 \( \times \) 1M entries
- Softmax gets expensive!
  - Need to compute normalization over 100k-1M outputs

Recap: Hierarchical Softmax

- Idea
  - Organize words in binary search tree, words are at leaves
  - Factorize probability of word \( w_i \) as a product of node probabilities along the path.
  - Learn a linear decision function \( y = v_{nj}(w, j) \) at each node to decide whether to proceed with left or right child node.
  \( \Rightarrow \) Decision based on output vector of hidden units directly.

Recap: Recurrent Neural Networks

- Up to now
  - Simple neural network structure: 1-to-1 mapping of inputs to outputs
  - Recurrent Neural Networks
  \( \Rightarrow \) Generalize this to arbitrary mappings

Recap: Recurrent Neural Networks (RNNs)

- RNNs are regular NNs whose hidden units have additional connections over time.
  - You can unroll them to create a network that extends over time.
  - When you do this, keep in mind that the weights for the hidden are shared between temporal layers.
- RNNs are very powerful
  - With enough neurons and time, they can compute anything that can be computed by your computer.

Recap: Backpropagation Through Time (BPTT)

- Configuration
  \( h_t = \sigma (W_{xh} x_t + W_{hh} h_{t-1} + b) \)
  \( y_t = \text{softmax} (W_{h y} h_t) \)
- Backpropagated gradient
  - For weight \( w_{ij} \):
    \( \frac{\partial E_i}{\partial w_{ij}} = \sum_{1 \leq t \leq T} \left( \frac{\partial E_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial w_{ij}} \right) \)
Recap: Backpropagation Through Time (BPTT)

• Analyzing the terms
  - For weight $w_{ij}$: $\frac{\partial E_i}{\partial w_{ij}} = \sum_{1 \leq t \leq T} \left( \frac{\partial E_i}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_{ij}} \right)$
  - This is the "immediate" partial derivative (with $h_{t-1}$ as constant)

Recap: Exploding / Vanishing Gradient Problem

• BPTT equations:
  - $E = \sum_{1 \leq t \leq T} E_t$
  - $\frac{\partial E_i}{\partial w_{ij}} = \sum_{1 \leq t \leq T} \left( \frac{\partial E_i}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_{ij}} \right)$
  - $\frac{\partial h_t}{\partial h_{t-1}} = \prod_{t > k} W_{hh} \text{diag}(\sigma'(h_{t-1}))$
  - Remaining issue: how to set the initial state $h_0$?
    ➞ Learn this together with all the other parameters.

Recap: Long Short-Term Memory (LSTM)

• Trick to handle exploding gradients
  - If the gradient is larger than a threshold, clip it to that threshold.
  - This makes a big difference in RNNs

Recap: Gradient Clipping

• Trick to handle exploding gradients
  - If the gradient is larger than a threshold, clip it to that threshold.
Recap: Elements of LSTMs

- **Forget gate layer**
  - Look at $h_{t-1}$ and $x_t$ and output a number between 0 and 1 for each dimension in the cell state $C_t$.
  - 0: completely keep this, 1: completely delete this.

- **Update gate layer**
  - Decide what information to store in the cell state.
  - Sigmoid network (input gate layer) decides which values are updated.
  - Tanh layer creates a vector of new candidate values that could be added to the state.

- **Output gate layer**
  - Output is a filtered version of our gate state.
  - First, apply sigmoid layer to decide what parts of the cell state to output.
  - Then, pass the cell state through a tanh (to push the values to be between -1 and 1) and multiply it with the output of the sigmoid gate.

Recap: Gated Recurrent Units (GRU)

- Simpler model than LSTM
  - Combines the forget and input gates into a single update gate $z_t$.
  - Similar definition for a reset gate $r_t$, but with different weights.
  - In both cases, merge the cell state and hidden state.

- **Empirical results**
  - Both LSTM and GRU can learn much longer-term dependencies than regular RNNs.
  - GRU performance similar to LSTM (no clear winner yet), but fewer parameters.

Recap: Reinforcement Learning

- **Motivation**
  - General purpose framework for decision making.
  - Basis: Agent with the capability to interact with its environment.
  - Each action influences the agent’s future state.
  - Success is measured by a scalar reward signal.
  - Goal: select actions to maximize future rewards.

  - Formalized as a partially observable Markov decision process (POMDP).

Recap: Reward vs. Return

- **Objective of learning**
  - We seek to maximize the expected return $G_t$ as some function of the reward sequence $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
  - Standard choice: expected discounted return

\[
G_t = R_{t+1} + y R_{t+2} + y^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} y^k R_{t+k+1}
\]

where $0 \leq y \leq 1$ is called the discount rate.

- **Difficulty**
  - We don’t know which past actions caused the reward.

  \[
  \Rightarrow \text{Temporal credit assignment problem}
  \]
Definition

A policy determines the agent’s behavior

- Map from state to action: $a = \pi(s)$

Two types of policies

- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \Pr(A_t = a|S_t = s)$

Note

- $\pi(a|s)$ denotes the probability of taking action $a$ when in state $s$. $\pi_i$ denotes the probability of taking action $a_i$ under policy $\pi$.

Recap: Exploration-Exploitation Trade-off

- Example: N-armed bandit problem
  - Suppose we have the choice between $N$ actions $a_1, \ldots, a_N$.
  - If we knew their value functions $q_i(s, a_i)$, it would be trivial to choose the best.
  - However, we only have estimates based on our previous actions and their returns.

- We can now
  - Explore our current knowledge
    - And choose the greedy action that has the highest value based on our current estimate.
  - Explore to gain additional knowledge
    - And choose a non-greedy action to improve our estimate of that action’s value.

Recap: TD-Learning

- Policy evaluation (the prediction problem)
  - For a given policy $\pi$, compute the state-value function $v_\pi$. $v_\pi(s)$ is the expected return when starting in $s$ and following $\pi$ thereafter.

  - For the optimal state-value function $v_*$:
    $$ v_*(s) = \max_{a \in A(s)} q_*(s, a) $$

    - $v_*$ is the unique solution to this system of nonlinear equations.

  - For the optimal action-value function $q_*$:
    $$ q_*(s, a) = \sum_{s'} p(s'|s, a) [r + \gamma v_*(s')] $$

    - $q_*$ is the unique solution to this system of nonlinear equations.

- Temporal Difference Learning - TD(0)
  - Directly perform an update using the estimate $V(S_{t+1})$.
    $$ V(S_t) \rightleftharpoons V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] $$

    - Target: an estimate of the return (here: TD(0))

Recap: SARSA - On-Policy TD Control

- Idea
  - Turn the TD idea into a control method by always updating the policy to be greedy w.r.t. the current estimate

- Procedure
  - Estimate $q_\pi(s, a)$ for the current policy $\pi$ and for all states $s$ and actions $a$.
  - TD(0) update equation
    $$ q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha [R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)] $$

    - This rule is applied after every transition from a nonterminal state $S_t$.
    - It uses every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.
    - Hence, the name SARSA.
Recap: Q-Learning - Off-Policy TD Control

- **Idea**
  - Directly approximate the optimal action-value function $q_a$, independent of the policy being followed.

- **Procedure**
  - TD(0) update equation
    
    $$Q(S_t, A_t) = Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$
  - Dramatically simplifies the analysis of the algorithm.
  - All that is required for correct convergence is that all pairs continue to be updated.

Recap: Deep Q-Networks (DQN)

- **Adaptation: Experience Replay**
  - To remove correlations, build a dataset from agent's own experience
    
    $$S_1, S_2, S_3, S_4 \rightarrow S, a, r, s'$$
    
    $$S_1, S_2, (s_1, r_1, s_2) \rightarrow S, a, r, s'$$
  - Perform minibatch updates to samples of experience drawn at random from the pool of stored samples
    - $(s, a, r, s') \sim D(t)$ where $D = \{(s_t, a_t, r_{t+1}, s_{t+1})\}$ is the dataset
  - Advantages
    - Each experience sample is used in many updates (more efficient)
    - Avoids correlation effects when learning from consecutive samples
    - Avoids feedback loops from on-policy learning

Recap: Deep Policy Gradients (DPG)

- **Idea**
  - Optimal Q-values should obey Bellman equation
    
    $$Q(s, a) = \mathbb{E}_r [r + \gamma \max_a Q(s', a') | s, a]$$
  - Treat the right-hand side $r + \gamma \max_a Q(s', a')$ as a target
  - Minimize MSE loss by stochastic gradient descent
    
    $$L(w) = (r + \gamma \max_a Q(s', a') - Q(s, a))^2$$
  - This converges to $Q$, using a lookup table representation.
  - Unfortunately, it diverges using neural networks due to:
    - Correlations between samples
    - Non-stationary targets

Recap: Deep Q-Learning

- **Idea**
  - **Q-values** should obey Bellman equation
    
    $$Q(s, a) = \mathbb{E}_r [r + \gamma \max_a Q(s', a') | s, a]$$
  - Treat the right-hand side as a target
  - Minimize MSE loss by stochastic gradient descent
    
    $$L(w) = (r + \gamma \max_a Q(s', a') - Q(s, a))^2$$
  - This converges to $Q$, using a lookup table representation.
  - Unfortunately, it diverges using neural networks due to:
    - Correlations between samples
    - Non-stationary targets

Recap: Deep Q-Networks (DQN)

- **Adaptation: Experience Replay**
  - To remove correlations, build a dataset from agent's own experience
    
    $$S_1, S_2, S_3, S_4 \rightarrow S, a, r, s'$$
    
    $$S_1, S_2, (s_1, r_1, s_2) \rightarrow S, a, r, s'$$
  - Sample from the dataset and apply an update
    
    $$L(w) = (r + \gamma \max_a Q(s', a') - Q(s, a))^2$$
  - To deal with non-stationary parameters $w^-$, are held fixed.
    - Only update the target network parameters every $c$ steps.
    - I.e., clone the network $\phi$ to generate a target network $\phi'$.
    - Then, this reduces oscillations to make learning more stable.

Recap: Policy Gradients

- **How to make high-value actions more likely**
  - The gradient of a stochastic policy $\pi(s, a)$ is given by
    
    $$\frac{\partial L(u)}{\partial u} = \mathbb{E}_a \left[ \frac{\partial \log \pi(s, a)}{\partial u} Q_a(s, a) \right]$$
    
    $$= \mathbb{E}_a \left[ \frac{\partial \log \pi(s, a)}{\partial u} Q_a(s, a) \right]$$
  - The gradient of a deterministic policy $a = \pi(s)$ is given by
    
    $$\frac{\partial L(u)}{\partial u} = \mathbb{E}_a \left[ \frac{\partial Q_a(s, a)}{\partial u} \frac{\partial a}{\partial u} \right]$$
  - If $a$ is continuous and $Q$ is differentiable.
Any Questions?

So what can you do with all of this?

Robust Object Detection & Tracking

Applications for Driver Assistance Systems

Mobile Tracking in Densely Populated Settings

Articulated Multi-Person Tracking

- Multi-Person tracking
  - Recover trajectories and solve data association
- Articulated Tracking
  - Estimate detailed body pose for each tracked person

Semantic 2D-3D Scene Segmentation
Integrated 3D Point Cloud Labels

Any More Questions?

Good luck for the exam!