Announcements

• Exams
  - Special oral exams (for exchange students):
    - We’re in the process of sending out the exam slots
    - You’ll receive an email with details tonight
    - Format: 30 minutes, 4 questions, 3 answers
  - Regular exams:
    - We will send out an email with the assignment to lecture halls
    - Format: 120min, closed book exam

Announcements (2)

• 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.

Announcements (3)

• Seminar in the summer semester
  - Current topics in Computer Vision and Machine Learning
  - Quick poll: Who is interested?

Course Outline

• Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM

• Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting

• Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks

Recap: Bayes Decision Theory

\[
p(x|a) p(a) \quad p(x|b) p(b) \quad \frac{\text{Likelihood} \times \text{Prior}}{\text{Normalization Factor}} \quad \text{Decision boundary} \]

\[
p(x|a) \quad p(x|b) \quad \begin{cases} p(a|x) & \text{if } a \text{ is true} \\ p(b|x) & \text{if } b \text{ is true} \end{cases} \quad \begin{cases} \text{Likelihood} & \text{if } a \text{ is true} \\ \text{Likelihood} \times \text{Prior} & \text{if } b \text{ is true} \end{cases} \]

Slide credit: Bernt Schiele
Image source: C.M. Bishop, 2006
Recap: Bayes Decision Theory

- Optimal decision rule
  - Decide for $C_1$ if
    $$p(C_1|x) > p(C_2|x)$$
  - This is equivalent to
    $$p(x|C_1)p(C_1) > p(x|C_2)p(C_2)$$
  - Which is again equivalent to (Likelihood-Ratio test)
    $$\frac{p(x|C_1)}{p(x|C_2)} > \frac{p(C_2)}{p(C_1)}$$

Decision threshold $\theta$

Recap: Minimizing the Expected Loss

- Optimal solution minimizes the loss.
  - But: loss function depends on the true class, which is unknown.
- Solution: Minimize the expected loss
  $$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(x,C_k) \, dx$$
  - This can be done by choosing the regions $\mathcal{R}_j$ such that
    $$\mathbb{E}[L] = \sum_k L_{kj} p(C_k|x)$$
    which is easy to do once we know the posterior class probabilities $p(C_k|x)$

Recap: Classifying with Loss Functions

- In general, we can formalize this by introducing a loss matrix $L_{kj}$
  $$L_{kj} = \text{loss for decision } C_j \text{ if truth is } C_k.$$
- Example: cancer diagnosis
  \[
  L_{\text{cancer diagnosis}} = \begin{pmatrix}
  0 & 1000 \\
  1 & 0
  \end{pmatrix}
  \]

Recap: The Reject Option

- Classification errors arise from regions where the largest posterior probability $p(C_k|x)$ is significantly less than 1.
  - These are the regions where we are relatively uncertain about class membership.
  - For some applications, it may be better to reject the automatic decision entirely in such a case and e.g. consult a human expert.

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks
Recap: Gaussian (or Normal) Distribution

- One-dimensional case
  - Mean \( \mu \)
  - Variance \( \sigma^2 \)
  \[
  N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi \sigma^2}} \exp\left\{ -\frac{(x - \mu)^2}{2\sigma^2} \right\}
  \]
- Multi-dimensional case
  - Mean \( \mu \)
  - Covariance \( \Sigma \)
  \[
  N(x; \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left\{ -\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu) \right\}
  \]

Recap: Maximum Likelihood Approach

- Computation of the likelihood
  - Single data point: \( p(x_n|\theta) \)
  - Assumption: all data points \( X = \{x_1, \ldots, x_n\} \) are independent
  \[
  L(\theta) = p(X|\theta) = \prod_{n=1}^{N} p(x_n|\theta)
  \]
  - Log-likelihood
  \[
  E(\theta) = -\ln L(\theta) = -\sum_{n=1}^{N} \ln p(x_n|\theta)
  \]
- Estimation of the parameters \( \theta \) (Learning)
  - Maximize the likelihood (= minimize the negative log-likelihood)
  \[
  \frac{\partial}{\partial \theta} E(\theta) = -\sum_{n=1}^{N} \frac{\partial}{\partial \theta} \ln p(x_n|\theta) = 0
  \]

Recap: Bayesian Learning Approach

- Bayesian view:
  - Consider the parameter vector \( \theta \) as a random variable.
  - When estimating the parameters, what we compute is
    \[
    p(x|X) = \int p(x, \theta|X) d\theta
    \]
    \[
    p(x, \theta|X) = p(x|\theta, X)p(\theta|X)
    \]
    \[
    p(x|X) = \int p(x|\theta)p(\theta|X)d\theta
    \]
    This is entirely determined by the parameter \( \theta \)
    (i.e. by the parametric form of the pdf).

Recap: Kernel Density Estimation

- Approximation formula:
  \[
  p(x) \approx K \frac{V}{N(N-1)}
  \]
  - Kernel methods
  - Place a kernel window \( k \) at location \( x \) and count how many data points fall inside it.
  - K-Nearest Neighbor
  - Increase the volume \( V \) until the \( K \) next data points are found.
Recap: Mixture of Gaussians (MoG)

• “Generative model”

\[ p(j) = \pi_j \]

“Weight” of mixture component

\[ p(x) = \sum_{j=1}^{M} p(x|\theta_j)p(j) \]

Mixture density

Recap: MoG – Iterative Strategy

• Assuming we knew the values of the hidden variable...

\[ f(x) \]

ML for Gaussian #1

assumed known

ML for Gaussian #2

\[ h(j = 1|x_n) = 1 \quad 111 \quad 22 \quad 2 \quad j \]

\[ h(j = 2|x_n) = 0 \quad 000 \quad 11 \quad 1 \quad 1 \]

\[ \mu_1 = \frac{\sum_{n=1}^{N} h(j = 1|x_n)x_n}{\sum_{n=1}^{N} h(j = 1|x_n)} \quad \mu_2 = \frac{\sum_{n=1}^{N} h(j = 2|x_n)x_n}{\sum_{n=1}^{N} h(j = 2|x_n)} \]

Recap: MoG – Iterative Strategy

• Assuming we knew the mixture components...

\[ f(x) \]

assumed known

\[ p(j = 1|x) \quad 1 \quad 111 \quad 22 \quad 2 \quad j \]

\[ p(j = 2|x) \quad 1 \quad 111 \quad 22 \quad 2 \quad j \]

• Bayes decision rule: Decide \( j = 1 \) if

\[ p(j = 1|x_n) > p(j = 2|x_n) \]

Recap: EM Algorithm

• Expectation-Maximization (EM) Algorithm

  - E-Step: softly assign samples to mixture components
    \[ \gamma_j(x_n) = \frac{\pi_j N(x_n; \mu_j, \Sigma_j)}{\sum_{j=1}^{K} \pi_j N(x_n; \mu_j, \Sigma_j)} \quad \forall j = 1, \ldots, K, \quad n = 1, \ldots, N \]

  - M-Step: re-estimate the parameters (separately for each mixture component) based on the soft assignments
    \[ \hat{N}_j = \sum_{n=1}^{N} \gamma_j(x_n) \quad N \]
    \[ \hat{\mu}_j = \frac{1}{N_j} \sum_{n=1}^{N} \gamma_j(x_n) x_n \]
    \[ \hat{\Sigma}_j = \frac{1}{N_j} \sum_{n=1}^{N} \gamma_j(x_n) (x_n - \hat{\mu}_j) (x_n - \hat{\mu}_j)^T \]

Slide adapted from Bernt Schiele
Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks

Recap: Linear Discriminant Functions

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

• Linear discriminant functions
  - \( w, w_0 \) define a hyperplane in \( \mathbb{R}^D \).
  - If a data set can be perfectly classified by a linear discriminant, then we call it linearly separable.

Recap: Least-Squares Classification

- Simplest approach
  - Directly try to minimize the sum-of-squares error
    \[
    E_D(W) = \frac{1}{2} \text{Tr} \left\{ (XW - T)^T(XW - T) \right\}
    \]
  - Setting the derivative to zero yields
    \[
    \hat{W} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T \hat{T}
    \]
  - We then obtain the discriminant function as
    \[
    y(x) = \hat{W}^T \hat{x} = \hat{T}^T (\hat{X}^T \hat{x})
    \]
  - \( y(x) \) as posterior probabilities.

Recap: Problems with Least Squares

- Least-squares is very sensitive to outliers!
  - The error function penalizes predictions that are "too correct".

Recap: Generalized Linear Models

- Generalized linear model
  \[
  y(x) = g(w^T x + w_0)
  \]
  - \( g(\cdot) \) is called an activation function and may be nonlinear.
  - The decision surfaces correspond to
    \[
    y(x) = \text{const} \iff w^T x + w_0 = \text{const}
    \]
  - If \( g \) is monotonous (which is typically the case), the resulting decision boundaries are still linear functions of \( x \).

- Advantages of the non-linearity
  - Can be used to bound the influence of outliers and "too correct" data points.
  - When using a sigmoid for \( g(\cdot) \), we can interpret the \( y(x) \) as posterior probabilities.
Recap: Extension to Nonlinear Basis Fcts.

- **Generalization**: Transform vector $x$ with $M$ nonlinear basis functions $\phi_j(x)$:
  \[
y_\theta(x) = \sum_{j=1}^{M} w_{kj} \phi_j(x) + w_{k0}
  \]

- **Advantages**
  - Transformation allows non-linear decision boundaries.
  - By choosing the right $\phi_j$, every continuous function can (in principle) be approximated with arbitrary accuracy.

- **Disadvantage**
  - The error function can in general no longer be minimized in closed form.

  ⇒ Minimization with Gradient Descent

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

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

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

Recap: Probabilistic Discriminative Models

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

  - 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 $p(\phi | C_i)$ and $p(C_i)$.

Recap: Iterative Methods for Estimation

- **Gradient Descent (1st order)**
  \[
  w^{(r+1)} = w^{(r)} - \eta \nabla E(w)|_{w^{(r)}}
  \]

  ⇒ Simple and general
  - Relatively slow to converge, has problems with some functions

- **Newton-Raphson (2nd order)**
  \[
  w^{(r+1)} = w^{(r)} - \eta H^{-1} \nabla E(w)|_{w^{(r)}}
  \]

  where $H = \nabla \nabla E(w)$; the Hessian matrix, i.e. the matrix of second derivatives.

- Local quadratic approximation to the target function
- Faster convergence

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.

  \[
y(x; w) = \begin{bmatrix}
P(y = 1; x; w) \\
P(y = 2; x; w) \\
\vdots \\
P(y = K; x; w)
\end{bmatrix}
= \frac{1}{\sum_{k=1}^{K} \exp(w_{k}^T x)} \begin{bmatrix}
\exp(w_{1}^T x) \\
\exp(w_{2}^T x) \\
\vdots \\
\exp(w_{K}^T x)
\end{bmatrix}
\]

  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
  \[
  E(w) = -\sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right\}
  = -\sum_{n=1}^{N} \sum_{k=0}^{1} \left\{ I(t_n = k) \ln P(y_n = k|x_n; w) \right\}
  \]
- Softmax regression
  - Generalization to \( K \) classes using indicator functions.
  \[
  E(w) = -\sum_{n=1}^{N} \sum_{k=0}^{K} \left\{ I(t_n = k) \ln \frac{\exp(w_k^T x_n)}{\sum_{j=0}^{K} \exp(w_j^T x_n)} \right\}
  \]
- \( \nabla_w E(w) = -\sum_{n=1}^{N} \left\{ I(t_n = k) \ln P(y_n = k|x_n; w) \right\} \)

Recap: Generalization and Overfitting

- Goal: predict class labels of new observations
  - Train classification model on limited training set.
  - The further we optimize the model parameters, the more the training error will decrease.
  - However, at some point the test error will go up again.
  \( \Rightarrow \) Overfitting to the training set!

Recap: Support Vector Machine (SVM)

- Basic idea
  - The SVM tries to find a classifier which maximizes the margin between pos. and neg. data points.
  - Up to now: consider linear classifiers
  \[
  w^T x + b = 0
  \]
- Formulation as a convex optimization problem
  - Find the hyperplane satisfying
  \[
  \arg\min_{w, b} \frac{1}{2} \|w\|^2
  \]
  under the constraints
  \[
  t_n (w^T x_n + b) \geq 1 \quad \forall n
  \]
  based on training data points \( x_n \) and target values \( t_n \in \{-1, 1\} \)

Recap: SVM – Primal Formulation

- Lagrangian primal form
  \[
  L_p = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n (w^T x_n + b) - 1 \right\}
  = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n y_n(x_n) - 1 \right\}
  \]
- The solution of \( L_p \) needs to fulfill the KKT conditions
  - Necessary and sufficient conditions
  \[
  \begin{align*}
  a_n &\geq 0 \\
  t_n y_n(x_n) - 1 &\geq 0 \\
  a_n \left\{ t_n y_n(x_n) - 1 \right\} & = 0 \\
  \end{align*}
  \]
  - KKT:
  \[
  \begin{align*}
  \lambda &\geq 0 \\
  f(x) &\geq 0 \\
  \Lambda f(x) & = 0 \\
  \end{align*}
  \]

Recap: SVM – Solution

- Solution for the hyperplane
  - Computed as a linear combination of the training examples
  \[
  w = \sum_{n=1}^{N} a_n t_n x_n
  \]
  - Sparse solution: \( a_n = 0 \) only for some points, the support vectors
  \( \Rightarrow \) Only the SVs actually influence the decision boundary!
  - Compute \( b \) by averaging over all support vectors:
  \[
  b = \frac{1}{N_S} \sum_{n \in S} \left( t_n - \sum_{m \in S} a_m t_m x_m^T x_n \right)
  \]

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks
- Perceptual and Sensory Augmented Computing
Recap: SVM – Support Vectors

- The training points for which \( a_n > 0 \) are called “support vectors”.
- Graphical interpretation:
  - The support vectors are the points on the margin.
  - They define the margin and thus the hyperplane.
  - \( \Rightarrow \) All other data points can be discarded!

Recap: SVM – Dual Formulation

- Maximize
  \[
  \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m)
  \]
  under the conditions
  \[
  a_n \geq 0 \quad \forall n \quad \text{and} \quad \sum_{n=1}^{N} a_n t_n = 0
  \]
- Comparison
  - \( L_d \) is equivalent to the primal form \( L_p \), but only depends on \( a_n \).
  - \( L_p \) scales with \( O(D) \).
  - \( L_d \) scales with \( O(N^2) \) – in practice between \( O(N) \) and \( O(N^2) \).

Recap: SVM for Non-Separable Data

- Slack variables
  - One slack variable \( \xi_n \geq 0 \) for each training data point.
- Interpretation
  - \( \xi_n = 0 \) for points that are on the correct side of the margin.
  - \( \xi_n = y_n - y(x_n) \) for all other points.
  - \( \Rightarrow \) We do not have to set the slack variables ourselves!
  - They are jointly optimized together with \( w \).

Recap: SVM – New Dual Formulation

- New SVM Dual: Maximize
  \[
  L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m)
  \]
  under the conditions
  \[
  a_n \geq C \quad \forall n \quad \text{and} \quad \sum_{n=1}^{N} a_n t_n = 0
  \]
- This is again a quadratic programming problem
  \( \Rightarrow \) Solve as before...

Recap: Nonlinear SVMs

- General idea: The original input space can be mapped to some higher-dimensional feature space where the training set is separable:

Recap: The Kernel Trick

- Important observation
  - \( \phi(x) \) only appears in the form of dot products \( \phi(x)^T \phi(y) \):
    \[
    y(x) = w^T \phi(x) + b = \sum_{n=1}^{N} a_n t_n \phi(x_n)^T \phi(x) + b
    \]
  - Define a so-called kernel function \( k(x,y) = \phi(x)^T \phi(y) \).
  - Now, in place of the dot product, use the kernel instead:
    \[
    \tilde{y}(x) = \sum_{n=1}^{N} a_n t_n k(x_n,x) + b
    \]
  - The kernel function implicitly maps the data to the higher-dimensional space (without having to compute \( \phi(x) \) explicitly)!
Recap: Kernels Fulfilling Mercer’s Condition

- Polynomial kernel
  \[ k(x, y) = (x^T y + 1)^p \]
- Radial Basis Function kernel
  \[ k(x, y) = \exp \left\{ -\frac{(x - y)^2}{2\sigma^2} \right\} \]  
  e.g. Gaussian
- Hyperbolic tangent kernel
  \[ k(x, y) = \tanh(\kappa x^T y + \delta) \]  
  e.g. Sigmoid
  
  And many, many more, including kernels on graphs, strings, and symbolic data...

Recap: Kernels Fulfilling Mercer’s Condition

- Polynomial kernel
  \[ k(x, y) = (x^T y + 1)^p \]
- Radial Basis Function kernel
  \[ k(x, y) = \exp \left\{ -\frac{(x - y)^2}{2\sigma^2} \right\} \]  
  e.g. Gaussian
- Hyperbolic tangent kernel
  \[ k(x, y) = \tanh(\kappa x^T y + \delta) \]  
  e.g. Sigmoid
  
  Actually, that was wrong in the original SVM paper...

Recap: Nonlinear SVM – Dual Formulation

- SVM Dual: Maximize
  \[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_m, x_n) \]
  under the conditions
  \[ \sum_{n=1}^{N} a_n t_n = 0 \]
  \[ 0 \leq a_n \leq C \]

- Classify new data points using
  \[ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b \]

Recap: Nonlinear SVM – Dual Formulation

- SVM Dual: Maximize
  \[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_m, x_n) \]
  under the conditions
  \[ \sum_{n=1}^{N} a_n t_n = 0 \]
  \[ 0 \leq a_n \leq C \]

- Classify new data points using
  \[ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b \]

Recap: Classifier Combination

- We’ve seen already a variety of different classifiers
  - \( k \)-NN
  - Bayes classifiers
  - Fisher’s Linear Discriminant
  - SVMs

- Each of them has their strengths and weaknesses...
  - Can we improve performance by combining them?

Recap: Classifier Combination

- We’ve seen already a variety of different classifiers
  - \( k \)-NN
  - Bayes classifiers
  - Fisher’s Linear Discriminant
  - SVMs

- Each of them has their strengths and weaknesses...
  - Can we improve performance by combining them?

Recap: Bayesian Model Averaging

- Model Averaging
  - Suppose we have \( H \) different models \( h = 1, \ldots, H \) with prior probabilities \( p(h) \).
  - Construct the marginal distribution over the data set
    \[ p(X) = \sum_{h=1}^{H} p(X|h)p(h) \]

- Average error of committee
  \[ E_{COM} = \frac{1}{M} E_{AV} \]
  - This suggests that the average error of a model can be reduced by a factor of \( M \) simply by averaging \( M \) versions of the model!
  - Unfortunately, this assumes that the errors are all uncorrelated. In practice, they will typically be highly correlated.

Recap: Bayesian Model Averaging

- Model Averaging
  - Suppose we have \( H \) different models \( h = 1, \ldots, H \) with prior probabilities \( p(h) \).
  - Construct the marginal distribution over the data set
    \[ p(X) = \sum_{h=1}^{H} p(X|h)p(h) \]

- Average error of committee
  \[ E_{COM} = \frac{1}{M} E_{AV} \]
  - This suggests that the average error of a model can be reduced by a factor of \( M \) simply by averaging \( M \) versions of the model!
  - Unfortunately, this assumes that the errors are all uncorrelated. In practice, they will typically be highly correlated.

- **Main idea** [Freund & Schapire, 1996]
  - Instead of resampling, reweight misclassified training examples.
    - Increase the chance of being selected in a sampled training set.
    - Or increase the misclassification cost when training on the full set.

- **Components**
  - \( h_m(x) \): “weak” or base classifier
    - Condition: <50% training error over any distribution
  - \( H(x) \): “strong” or final classifier

- **AdaBoost**
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:
    \[
    H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)
    \]

Recap: AdaBoost – Intuition

- **Intuition**

  Consider a 2D feature space with positive and negative examples.

  Each weak classifier splits the training examples with at least 50% accuracy.

  Examples misclassified by a previous weak learner are given more emphasis at future rounds.

Recap: AdaBoost – Algorithm

1. **Initialization**: Set \( w(1) = \frac{1}{N} \) for \( n = 1, \ldots, N \).
2. For \( m = 1, \ldots, M \) iterations
   a) Train a new weak classifier \( h_m(x) \) using the current weighting coefficients \( W^{(m)} \) by minimizing the weighted error function
      \[
      e_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)
      \]
   b) Estimate the weighted error of this classifier on \( X \):
      \[
      e_m = \frac{1}{N} \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)
      \]
   c) Calculate a weighting coefficient for \( h_m(x) \):
      \[
      \alpha_m = \ln \left( \frac{1 - e_m}{e_m} \right)
      \]
   d) Update the weighting coefficients:
      \[
      w_n^{(m+1)} = w_n^{(m)} \exp \left( \alpha_m I(h_m(x_n) \neq t_n) \right)
      \]

Recap: Comparing Error Functions

- **Ideal misclassification error function**
- **“Hinge error” used in SVMs**
- **Exponential error function**
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple AdaBoost scheme.
  - Disadvantage: exponential penalty for large negative values!
  - Less robust to outliers or misclassified data points!
Recap: Comparing Error Functions

- Ideal misclassification error function
- “Hinge error” used in SVMs
- Exponential error function
- “Cross-entropy error” \( E = - \sum (t_n \ln y_n + (1 - t_n) \ln(1 - y_n)) \)
  - Similar to exponential error for \( z > 0 \)
  - Only grows linearly with large negative values of \( z \)
  - Make AdaBoost more robust by switching to “GentleBoost”

Recap: Perceptrons

- One output node per class

\[
\begin{align*}
y_1(x) & \quad y_2(x) & \cdots & \quad y_k(x) \\
W_{1,1} & & \cdots & & W_{k,1} \\
x_0 & \quad x_1 & \cdots & \quad x_d
\end{align*}
\]

Output layer
Weights
Input layer

- Outputs
  - Linear outputs
  \[
y_k(x) = \sum_{i=0}^{d} W_{k,i} x_i
\]
  - Can be used to do multidimensional linear regression or multiclass classification.

Recap: Non-Linear Basis Functions

- Straightforward generalization

\[
\begin{align*}
y_1(x) & \quad y_2(x) & \cdots & \quad y_k(x) \\
W_{1,1} & & \cdots & & W_{k,1} \\
\phi(x_0) & \quad \phi(x_1) & \cdots & \quad \phi(x_d)
\end{align*}
\]

Output layer
Weights
Feature layer
Mapping (fixed)
Input layer

- Outputs
  - Linear outputs with output nonlinearity
  \[
y_k(x) = \sum_{i=0}^{d} W_{k,i} \phi(x_i)
\]

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
  \[
  w_{kj}^{(r+1)} = w_{kj}^{(r)} - \eta \left( y_k(X_n; w) - t_{kn} \right) \phi_j(X_n)
  \]
  - This is the Delta rule a.k.a. LMS rule!
  \[
  \Rightarrow \text{Perceptron Learning corresponds to 1st-order (stochastic) Gradient Descent of a quadratic error function!}
  \]
Recap: Loss Functions

- We can now also apply other loss functions
  - $L_2$ loss: $L(t, y(x)) = \sum_n (y(x_n) - t_n)^2$
  - $L_1$ loss: $L(t, y(x)) = \sum_n |y(x_n) - t_n|$
  - Cross-entropy loss: $L(t, y(x)) = -\sum_n t_n \ln y_n + (1 - t_n) \ln (1 - y_n)$
  - Hinge loss: $L(t, y(x)) = \sum_n [1 - t_n y(x_n)]^+$
  - Softmax loss: $L(t, y(x)) = -\sum_n \sum_k \left\{ \delta(t_n = k) \ln \frac{e^{y_k(x_n)}}{\sum_j e^{y_j(x_n)}} \right\}$

Recap: Multi-Layer Perceptrons

- Adding more layers
  - $y_k(x) = \left( \sum_{i=0}^n W^{(k)}_{ji} y_i(x) \right)$ Output layer
  - $x_0 = 1, x_1, x_2, \ldots, x_d$ Input layer

- Output
  - $y_k(x) = y^{(2)}(\sum_{i=0}^n W^{(2)}_{ji} y^{(1)}(\sum_{j=0}^n W^{(1)}_{ij} x_j))$

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: 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
    $$E(W) = \sum_n L(t_n, y(x_n; W)) + \lambda \Omega(W)$$
    with a loss $L(\cdot)$ and a regularizer $\Omega(\cdot)$.
  - E.g., $L(t_n, y(x_n; W)) = \sum_n (y(x_n; W) - t_n)^2$ $L_2$ loss
    $$\Omega(W) = ||W||^2_p \quad \text{L}_p \text{ regularizer} \quad \text{("weight decay")}$$
  - Update each weight $W^{(k)}_{ij}$ in the direction of the gradient

Recap: Backpropagation Algorithm

- Core steps
  1. Convert the discrepancy between each output and its target value into an error derivate.
  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

- Efficient propagation scheme
  - $y^{(k-1)}_i$ is already known from forward pass! (Dynamic Programming)
  - Propagate back the gradient from layer $k$ and multiply with $y^{(k-1)}_i$
Recap: MLP Backpropagation Algorithm

- **Forward Pass**
  \[
  y^{(0)} = x
  \]
  for \( k = 1, \ldots, l \) do
  \[
  z^{(k)} = W^{(k)}y^{(k-1)}
  \]
  \[
  y^{(k)} = g_k(z^{(k)})
  \]
  endfor
  \[
  E = L(t, y) + \Omega(W)
  \]
- **Backward Pass**
  \[
  h_i = \frac{\partial E}{\partial y_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial y_i} + \lambda \frac{\partial \Omega}{\partial y_i}
  \]
  for \( k = l, l-1, \ldots, 1 \) do
  \[
  h = \frac{\partial E}{\partial z^{(k)}} = h \circ g'(z^{(k)})
  \]
  \[
  \frac{\partial E}{\partial W^{(k)}} = h y^{(k-1)\top} + \lambda \frac{\partial \Omega}{\partial W^{(k)}}
  \]
  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**
  \[
  \frac{\partial E}{\partial W} = \frac{\partial E}{\partial Y} \cdot \frac{\partial Y}{\partial W}
  \]
- **Reverse Mode Differentiation**
  \[
  \frac{\partial E}{\partial W} = \frac{\partial E}{\partial Z} \cdot \frac{\partial Z}{\partial W}
  \]

- For 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.
  - Very general algorithm, used in today’s Deep Learning packages.

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: Choosing the Right Learning Rate

- **Convergence of Gradient Descent**
  - Simple 1D example
  \[
  W^{(r+1)} = W^{(r)} - \eta \cdot \frac{dE(W)}{dW}
  \]
  - What is the optimal learning rate \( \eta_{\text{opt}} \)?
  - If \( E \) is quadratic, the optimal learning rate is given by the inverse of the Hessian
  \[
  \eta_{\text{opt}} = \left( \frac{\partial^2 E(W^{(r)})}{dW^2} \right)^{-1}
  \]
  - Advanced optimization techniques try to approximate the Hessian by a simplified form.
  - If we exceed the optimal learning rate, bad things happen!

Recap: Patience

- **Saddle points dominate in high-dimensional spaces!**
  \[\Rightarrow\] 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: 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 only, not for CNNs)
  - 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: Another Note on Error Functions

- Squared error on sigmoid/tanh output function
  - Avoids penalizing “too correct” data points.
  - But: zero gradient for confidently incorrect classifications!
  - ⇒ Do not use $L_2$ loss with sigmoid outputs (instead: cross-entropy)!

Recap: Commonly Used Nonlinearities

- Sigmoid
  \[ g(a) = \sigma(a) = \frac{1}{1 + \exp(-a)} \]

- Hyperbolic tangent
  \[ g(a) = \tanh(a) = 2\sigma(2a) - 1 \]

- Softmax
  \[ g(a) = \frac{\exp(-a_i)}{\sum_j \exp(-a_j)} \]

Recap: Commonly Used Nonlinearities (2)

- Rectified linear unit (ReLU)
  \[ g(a) = \max(0, a) \]

- Leaky ReLU
  \[ g(a) = \max(\beta a, a), \quad \beta \in [0.01, 0.3] \]
  - Avoids stuck-at-zero units
  - Weaker offset bias

- ELU
  \[ g(a) = \begin{cases} a, & a \geq 0 \\ e^a - 1, & a < 0 \end{cases} \]
  - No offset bias anymore
  - BUT: need to store activations
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 spills out a number $Y$.
  - We want the variance of the input and output of a unit to be the same, therefore $n \text{Var}(W_i)$ should be 1. This means
    \[ \text{Var}(W_i) = \frac{1}{n} \]
  - Or for the backpropagated gradient
    \[ \text{Var}(W_i) = \frac{1}{n^{\text{out}}} \]
  - As a compromise, Glorot & Bengio propose to use
    \[ \text{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}} \]
  - Randomly sample the weights with this variance. That’s it.

Recap: He Initialization

- Extension of Glorot Initialization to ReLU units
  - Use Rectified Linear Units (ReLU)
    \[ g(a) = \max(0, a) \]
  - Effect: gradient is propagated with a constant factor
    \[ \frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases} \]
  - Same basic idea: Output should have the input variance
  - However, the Glorot derivation was based on tanh units, linearity assumption around zero does not hold for ReLU.
  - He et al. made the derivations, proposed to use instead
    \[ \text{Var}(W) = \frac{2}{n_{\text{in}}} \]

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
  - (Typically) much improved convergence

Recap: Dropout

- 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.
  - Improved performance

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation

- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting

- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks

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
  → E.g. 1000 x 1000 image 100 filters
  → 10 x 10 filter size
  → Only 10k parameters
→ Result: Response map
  → size: 1000 x 1000 x 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 x 1 x depth] depth column in output volume.

Recap: Activation Maps

Activation maps

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 (10^9 images instead of 10^6)
  → 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%.

<table>
<thead>
<tr>
<th>VGGNet Configuration</th>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width (filters)</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Height (units)</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Depth (layers)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Recap: GoogLeNet (2014)

- Ideas:
  - Learn features at multiple scales
  - Modular structure

- Effect
  - After last pooling layer, volume is of size $[7 \times 7 \times 1024]$
  - Normally you would place the first 4096-D FC layer here (Many million params).
  - Instead: use Average pooling in each depth slice:
    - Reduces the output to $[1 \times 1 \times 1024]$.
  - Performance actually improves by 0.6% compared to when using FC layers (less overfitting?)

Discussion

- GoogLeNet
  - 12× fewer parameters than AlexNet
    - ~5M parameters
  - Where does the main reduction come from?
    - From throwing away the fully connected (FC) layers.

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: Visualizing CNNs

Recap: Analysis of ResNets

- The effective paths in ResNets are relatively shallow
  - Effectively only 5-17 active modules
- This explains the resilience to deletion
  - Deleting any single layer only affects a subset of paths (and the shorter ones less than the longer ones).
- New interpretation of ResNets
  - ResNets work by creating an ensemble of relatively shallow paths
  - Making ResNets deeper increases the size of this ensemble
  - Excluding longer paths from training does not negatively affect the results.
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: Faster 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: Semantic Image Segmentation

- 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

Recap: Neural Probabilistic Language Model

- Core idea
  - Learn a shared distributed encoding (word embedding) for the words in the vocabulary.

Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks
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

Recap: Problems with 100k-1M outputs

- Weight matrix gets huge!
  - Example: CBOW model
  - One-hot encoding for inputs
  - 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.
  - $W_{N \times V}$ 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 $w_0$ as a product of node probabilities along the path.
  - Learn a linear decision function $y = v_y(x) \cdot h$ at each node to decide whether to proceed with left or right child node.
  - 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
  - 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
  \[ h_t = \sigma \left( W_{xh} x_t + W_{hh} h_{t-1} + b \right) \]
  \[ y_t = \text{softmax} \left( W_{yh} h_t \right) \]
- Backpropagated gradient
  - For weight \( w_{ij} \):
    \[ \frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k \leq t} \left( \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w_{ij}} \right) \]

Recap: Exploding / Vanishing Gradient Problem

- BPTT equations:
  \[ \frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k \leq t} \left( \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w_{ij}} \right) \]
  \[ \frac{\partial h_t}{\partial h_k} = \prod_{t \geq k+1} \frac{\partial h_t}{\partial h_{k+1}} = \prod_{t \geq k} W_{hh} \text{diag}(\sigma'(h_{t-1})) \]
  \[ = (W_{hh})^t \]

\( \Rightarrow \) We are effectively taking the weight matrix to a high power.
  - The result will depend on the eigenvalues of \( W_{hh} \).
    - Largest eigenvalue \( > 1 \) \( \Rightarrow \) Gradients may explode.
    - Largest eigenvalue \( < 1 \) \( \Rightarrow \) Gradients will vanish.
  - This is very bad...
Recap: Gradient Clipping

- Trick to handle exploding gradients
  - If the gradient is larger than a threshold, clip it to that threshold.

  Algorithm 1 Pseudo-code:

  \[
  \text{if } |g| \geq \text{threshold} \text{ then}
  \text{clip } g \text{ to threshold}
  \text{end if}
  \]

  - This makes a big difference in RNNs

Recap: Long Short-Term Memory

- LSTMs
  - Inspired by the design of memory cells
  - Each module has 4 layers, interacting in a special way.

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 delete this.
  - 1: completely keep this.

- Update gate layer
  - Decide what information to store in the cell state.
  - Sigmoid network (input gate layer) decides which values are updated.
  - \( \text{tanh} \) layer creates a vector of new candidate values.

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.

Any More Questions?

Good luck for the exam!