Machine Learning – Lecture 19

Repetition

31.01.2019

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

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)

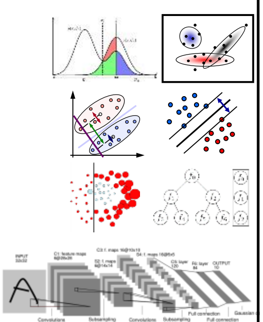
- Today, I'll summarize the most important points from the lecture.
  - It is an opportunity for you to ask questions…
  - …or get additional explanations about certain topics.
  - *So, please do ask.*
- Today's slides are intended as an index for the lecture.
  - But they are not complete, won't be sufficient as only tool.
  - Also look at the exercises – they often explain algorithms in detail.

Announcements (3)

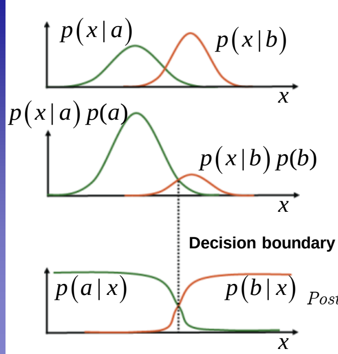
- Seminar in the summer semester
  - Current topics in Computer Vision and Machine Learning
  - Quick poll: Who is interested?

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



Likelihood

$Likelihood \times Prior$

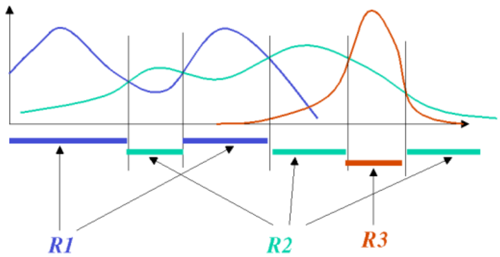
$$Posterior = \frac{Likelihood \times Prior}{NormalizationFactor}$$

Recap: Bayes Decision Theory

- Optimal decision rule
  - Decide for C_1 if
$$p(\mathcal{C}_1|x) > p(\mathcal{C}_2|x)$$
  - This is equivalent to
$$p(x|\mathcal{C}_1)p(\mathcal{C}_1) > p(x|\mathcal{C}_2)p(\mathcal{C}_2)$$
  - Which is again equivalent to (Likelihood-Ratio test)
$$\frac{p(x|\mathcal{C}_1)}{p(x|\mathcal{C}_2)} > \underbrace{\frac{p(\mathcal{C}_2)}{p(\mathcal{C}_1)}}_{\text{Decision threshold } \theta}$$

Recap: Bayes Decision Theory

- Decision regions: $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \ldots$



Recap: Classifying with Loss Functions

- In general, we can formalize this by introducing a loss matrix L_{kj}

$$L_{kj} = loss\ for\ decision\ \mathcal{C}_j\ if\ truth\ is\ \mathcal{C}_k.$$

- Example: cancer diagnosis

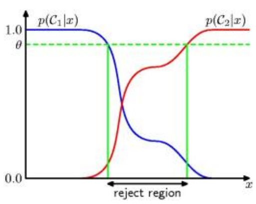
$$L_{cancer\ diagnosis} = \begin{array}{cc} & \text{Decision} \\ & \begin{array}{cc}\text{cancer} & \text{normal}\end{array} \\ \text{Truth} \begin{array}{c}\text{cancer} \\ \text{normal}\end{array} & \begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix} \end{array}$$

Recap: Minimizing the Expected Loss

see Exercise 1.3

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

Recap: The Reject Option

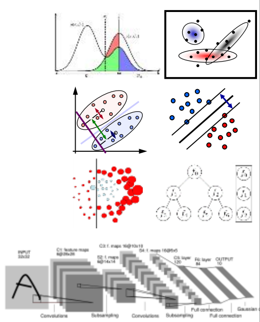


- Classification errors arise from regions where the largest posterior probability $p(\mathcal{C}_k|\mathbf{x})$ is significantly less than 1.
  - These are the regions where we are relatively uncertain about class membership.
  - For some applications, it may be better to reject the automatic decision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

Image source: C.M. Bishop, 2006

Course Outline

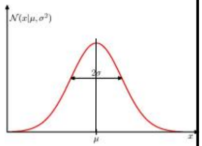
- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
  - Mixture Models and EM
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural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



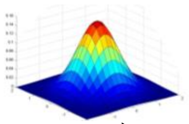
Recap: Gaussian (or Normal) Distribution

- One-dimensional case
  - Mean μ
  - Variance σ^2

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$



- Multi-dimensional case
  - Mean μ
  - Covariance Σ



$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

Recap: Maximum Likelihood Approach

see Exercise 1.4

- Computation of the likelihood
  - Single data point: $p(x_n|\theta)$
  - Assumption: all data points $X = \{x_1, \ldots, x_n\}$e 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 θ (Learning)
  - Maximize the likelihood (= minimize the negative log-likelihood)

  $\Rightarrow$ Take the derivative and set it to zero.

$$\frac{\partial}{\partial\theta}E(\theta) = -\sum_{n=1}^{N}\frac{\frac{\partial}{\partial\theta}p(x_n|\theta)}{p(x_n|\theta)} \overset{!}{=} 0$$

Recap: Bayesian Learning Approach

- Bayesian view:
  - Consider the parameter vector θ as a random variable.
  - When estimating the parameters, what we compute is

$$p(x|X) = \int p(x,\theta|X)d\theta$$

Assumption: given θ, this doesn't depend on X anymore

$$p(x,\theta|X) = p(x|\theta,\cancel{X})p(\theta|X)$$

$$p(x|X) = \int \underbrace{p(x|\theta)}p(\theta|X)d\theta$$

This is entirely determined by the parameter θ (i.e. by the parametric form of the pdf).

Recap: Bayesian Learning Approach

- Discussion

Likelihood of the parametric form θ given the data set X.

Estimate for x based on parametric form θ

Prior for the parameters θ

$$p(x|X) = \int \frac{p(x|\theta)L(\theta)p(\theta)}{\underbrace{\int L(\theta)p(\theta)d\theta}}d\theta$$

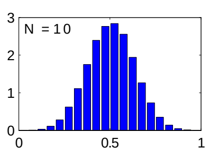
Normalization: integrate over all possible values of θ

  - The more uncertain we are about θ, the more we average over all possible parameter values.

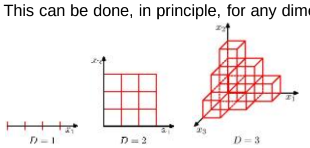
Recap: Histograms

- Basic idea:
  - Partition the data space into distinct bins with widths Δ_i and count the number of observations, n_i, in each bin.

$$p_i = \frac{n_i}{N\Delta_i}$$



  - Often, the same width is used for all bins, $\Delta_i = \Delta$.
  - This can be done, in principle, for any dimensionality D…



…but the required number of bins grows exponentially with D!

Recap: Kernel Density Estimation

see Exercise 1.5

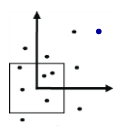
- Approximation formula:

$$p(\mathbf{x}) \approx \frac{K}{NV}$$

fixed V determine K → Kernel Methods

fixed K determine V → K-Nearest Neighbor

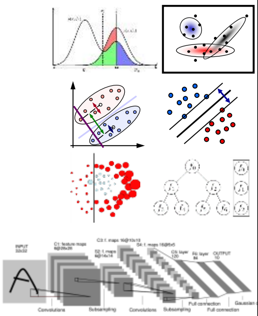
- Kernel methods
  - Place a *kernel window* k at location $\mathbf{x}$ and count how many data points fall inside it.



- K-Nearest Neighbor
  - Increase the volume V until the K next data points are found.

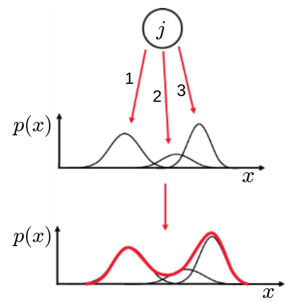
Course Outline

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



Recap: Mixture of Gaussians (MoG)

- "Generative model"



$p(j) = \pi_j$ "Weight" of mixture component

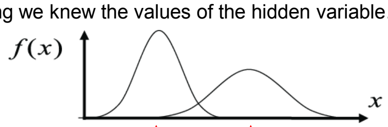
$p(x|\theta_j)$ Mixture component

Mixture density

$$p(x|\theta) = \sum_{j=1}^{M} p(x|\theta_j)p(j)$$

Recap: MoG – Iterative Strategy

- Assuming we knew the values of the hidden variable…



ML for Gaussian #1 ML for Gaussian #2

assumed known → 1 111 22 2 2 j

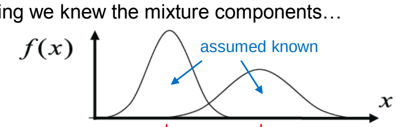
$h(j=1|x_n) =$ 1 111 00 0 0

$h(j=2|x_n) =$ 0 000 11 1 1

$$\mu_1 = \frac{\sum_{n=1}^{N} h(j=1|x_n)x_n}{\sum_{i=1}^{N} h(j=1|x_n)} \qquad \mu_2 = \frac{\sum_{n=1}^{N} h(j=2|x_n)x_n}{\sum_{i=1}^{N} h(j=2|x_n)}$$

Recap: MoG – Iterative Strategy

- Assuming we knew the mixture components…



$p(j=1|x)$ $p(j=2|x)$

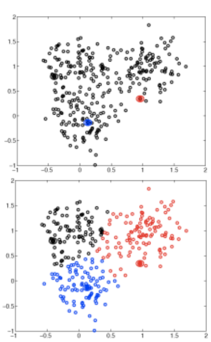
1 111 22 2 2 j

- Bayes decision rule: Decide $j = 1$ if

$$p(j=1|x_n) > p(j=2|x_n)$$

Recap: K-Means Clustering

- Iterative procedure
  1. Initialization: pick K arbitrary centroids (cluster means)
  2. Assign each sample to the closest centroid.
  3. Adjust the centroids to be the means of the samples assigned to them.
  4. Go to step 2 (until no change)
- Algorithm is guaranteed to converge after finite #iterations.
  - Local optimum
  - Final result depends on initialization.



Recap: EM Algorithm

see Exercise 1.6

- Expectation-Maximization (EM) Algorithm
  - E-Step: softly assign samples to mixture components

$$\gamma_j(\mathbf{x}_n) \leftarrow \frac{\pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{k=1}^{N} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \qquad \forall j = 1,\dots,K, \quad n = 1,\dots,N$$

  - M-Step: re-estimate the parameters (separately for each mixture component) based on the soft assignments

$$\hat{N}_j \leftarrow \sum_{n=1}^{N} \gamma_j(\mathbf{x}_n) = \text{soft number of samples labeled } j$$

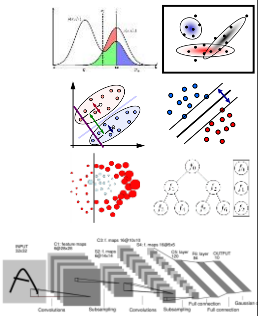
$$\hat{\pi}_j^{\text{new}} \leftarrow \frac{\hat{N}_j}{N}$$

$$\hat{\boldsymbol{\mu}}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)\mathbf{x}_n$$

$$\hat{\boldsymbol{\Sigma}}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_j^{\text{new}})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_j^{\text{new}})^{\text{T}}$$

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

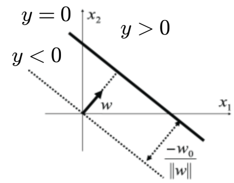


Recap: Linear Discriminant Functions

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

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

weight vector — "bias" (= threshold)



  - $\mathbf{w}$, 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.

Slide adapted from Bernt Schiele

Recap: Least-Squares Classification

see Exercise 2.1

- Simplest approach
  - Directly try to minimize the sum-of-squares error

$$E(\mathbf{w}) = \sum_{n=1}^{N} (y(\mathbf{x}_n; \mathbf{w}) - \mathbf{t}_n)^2$$

$$E_D(\widetilde{\mathbf{W}}) = \frac{1}{2}\mathrm{Tr}\left\{(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})^{\mathrm{T}}(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})\right\}$$

  - Setting the derivative to zero yields

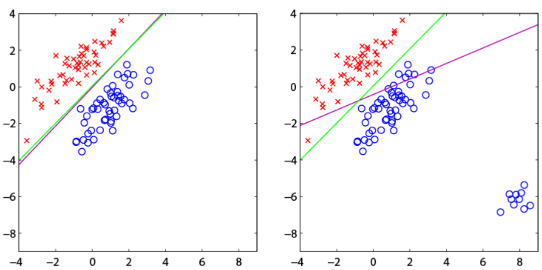
$$\widetilde{\mathbf{W}} = (\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{T} = \widetilde{\mathbf{X}}^{\dagger}\mathbf{T}$$

  - We then obtain the discriminant function as

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{x}} = \mathbf{T}^{\mathrm{T}}\left(\widetilde{\mathbf{X}}^{\dagger}\right)^{\mathrm{T}}\widetilde{\mathbf{x}}$$

  - ⇒ Exact, closed-form solution for the discriminant function parameters.

Recap: Problems with Least Squares



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

Image source: C.M. Bishop, 2006

Recap: Generalized Linear Models

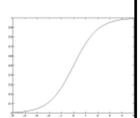
- Generalized linear model

$$y(\mathbf{x}) = g(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0)$$

  - $g(\cdot)$ is called an activation function and may be nonlinear.
  - The decision surfaces correspond to

$$y(\mathbf{x}) = const. \quad \Leftrightarrow \quad \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 = const.$$

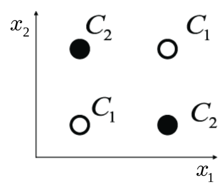
  - If g is monotonous (which is typically the case), the resulting decision boundaries are still linear functions of $\mathbf{x}$.
- Advantages of the non-linearity
  - Can be used to bound the influence of outliers and "too correct" data points.
  - When using a sigmoid for $g(\cdot)$, we can interpret the $y(\mathbf{x})$ as posterior probabilities.



$$g(a) \equiv \frac{1}{1 + \exp(-a)}$$

Recap: Linear Separability

- Up to now: restrictive assumption
  - Only consider linear decision boundaries
- Classical counterexample: XOR



Slide credit: Bernt Schiele

Recap: Extension to Nonlinear Basis Fcts.

- Generalization
  - Transform vector $\mathbf{x}$ with M nonlinear basis functions $\phi_j(\mathbf{x})$:

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{ki}\phi_j(\mathbf{x}) + w_{k0}$$

- Advantages
  - Transformation allows non-linear decision boundaries.
  - By choosing the right ϕ_j, every continuous function can (in principle) be approximated with arbitrary accuracy.
- Disadvatage
  - The error function can in general no longer be minimized in closed form.

  $\Rightarrow$ Minimization with Gradient Descent

Recap: Probabilistic Discriminative Models

- Consider models of the form

$$p(\mathcal{C}_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma(\mathrm{w}^T\boldsymbol{\phi})$$

with $$p(\mathcal{C}_2|\boldsymbol{\phi}) = 1 - p(\mathcal{C}_1|\boldsymbol{\phi})$$

- 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 $p(\phi|\mathcal{C}_k)$ and $p(\mathcal{C}_k)$.

Recap: Logistic Regression

- Let's consider a data set $\{\phi_n, t_n\}$ with $n = 1,\ldots,N$, where $\boldsymbol{\phi}_n = \boldsymbol{\phi}(\mathbf{x}_n)$ and $t_n \in \{0,1\}$ $\mathbf{t} = (t_1,\ldots,t_N)^T$
- With $y_n = p(\mathcal{C}_1|\phi_n)$, we can write the likelihood as

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

- Define the error function as the negative log-likelihood

$$\begin{aligned} E(\mathbf{w}) &= -\ln p(\mathbf{t}|\mathbf{w}) \\ &= -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \end{aligned}$$

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

Recap: Iterative Methods for Estimation

- Gradient Descent (1st order)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\, \nabla E(\mathbf{w})|_{\mathbf{w}^{(\tau)}}$$

  - Simple and general
  - Relatively slow to converge, has problems with some functions
- Newton-Raphson (2nd order)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\, \mathbf{H}^{-1}\nabla E(\mathbf{w})|_{\mathbf{w}^{(\tau)}}$$

  where $\mathbf{H} = \nabla\nabla E(\mathbf{w})$ is the Hessian matrix, i.e. the matrix of second derivatives.
  - Local quadratic approximation to the target function
  - Faster convergence

Recap: Iteratively Reweighted Least Squares

- Update equations

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T(\mathbf{y}-\mathbf{t}) \\ &= (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\left\{\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi}\mathbf{w}^{(\tau)} - \mathbf{\Phi}^T(\mathbf{y}-\mathbf{t})\right\} \\ &= (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{R}\mathbf{z} \end{aligned}$$

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

- Very similar form to pseudo-inverse (normal equations)
  - But now with non-constant weighing matrix $\mathbf{R}$ (depends on $\mathbf{w}$).
  - 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.

$$\mathbf{y}(\mathbf{x};\mathbf{w}) = \begin{bmatrix} P(y=1|\mathbf{x};\mathbf{w}) \\ P(y=2|\mathbf{x};\mathbf{w}) \\ \vdots \\ P(y=K|\mathbf{x};\mathbf{w}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K}\exp(\mathbf{w}_j^\top\mathbf{x})}\begin{bmatrix} \exp(\mathbf{w}_1^\top\mathbf{x}) \\ \exp(\mathbf{w}_2^\top\mathbf{x}) \\ \vdots \\ \exp(\mathbf{w}_K^\top\mathbf{x}) \end{bmatrix}$$

  - 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

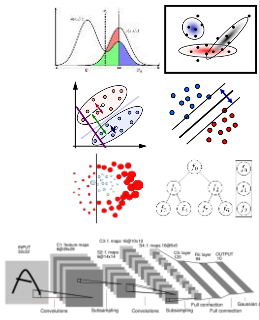
$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$
$$= -\sum_{n=1}^{N}\sum_{k=0}^{1}\{\mathbb{I}(t_n = k)\ln P(y_n = k|\mathbf{x}_n;\mathbf{w})\}$$

- Softmax regression
  - Generalization to K classes using indicator functions.

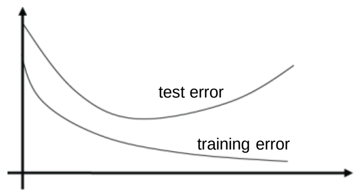
$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\left\{\mathbb{I}(t_n = k)\ln\frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K}\exp(\mathbf{w}_j^\top \mathbf{x})}\right\}$$
$$\nabla_{\mathbf{w}_k}E(\mathbf{w}) = -\sum_{n=1}^{N}\left[\mathbb{I}(t_n = k)\ln P(y_n = k|\mathbf{x}_n;\mathbf{w})\right]$$

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



Recap: Generalization and Overfitting



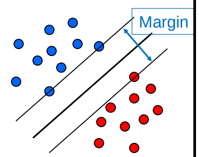
- Goal: predict class labels of new observations
  - Train classification model on limited training set.
  - The further we optimize the model parameters, the more the training error will decrease.
  - However, at some point the test error will go up again.
  - ⇒ *Overfitting to the training set!*

Image source: B. Schiele

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

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$$



- Formulation as a convex optimization problem
  - Find the hyperplane satisfying

$$\arg\min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2$$

under the constraints

$$t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

based on training data points $\mathbf{x}_n$ and target values $t_n \in \{-1, 1\}$

Recap: SVM – Primal Formulation

- Lagrangian primal form

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}a_n\left\{t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1\right\}$$
$$= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}a_n\left\{t_n y(\mathbf{x}_n) - 1\right\}$$

- The solution of L_p needs to fulfill the KKT conditions
  - Necessary and sufficient conditions

$$a_n \geq 0$$
$$t_n y(\mathbf{x}_n) - 1 \geq 0$$
$$a_n\left\{t_n y(\mathbf{x}_n) - 1\right\} = 0$$

KKT:

$$\lambda \geq 0$$
$$f(\mathbf{x}) \geq 0$$
$$\lambda f(\mathbf{x}) = 0$$

Recap: SVM – Solution

- Solution for the hyperplane
  - Computed as a linear combination of the training examples

$$\mathbf{w} = \sum_{n=1}^{N}a_n t_n \mathbf{x}_n$$

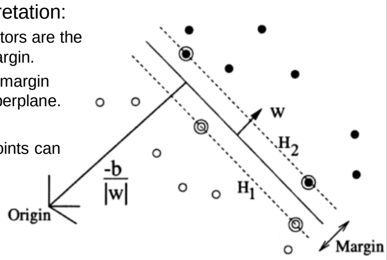
  - Sparse solution: $a_n \neq 0$ only for some points, the support vectors
  - ⇒ Only the SVs actually influence the decision boundary!
  - Compute b by averaging over all support vectors:

$$b = \frac{1}{N_{\mathcal{S}}}\sum_{n\in\mathcal{S}}\left(t_n - \sum_{m\in\mathcal{S}}a_m t_m \mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n\right)$$

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

see Exercise 2.2

- Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n)$$

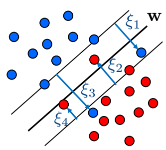
under the conditions

$$a_n \geq 0 \quad \forall n$$
$$\sum_{n=1}^{N} a_n t_n = 0$$

- Comparison
  - L_d is equivalent to the primal form L_p, but only depends on a_n.
  - L_p scales with $\mathcal{O}(D^3)$.
  - L_d scales with $\mathcal{O}(N^3)$ – in practice between $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$.

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.
  - $\xi_n = |t_n - y(\mathbf{x}_n)|$ for all other points.



Point on decision boundary: $\xi_n = 1$

Misclassified point: $\xi_n > 1$

- We do not have to set the slack variables ourselves!

$\Rightarrow$ They are jointly optimized together with $\mathbf{w}$.

Recap: SVM – New Dual Formulation

see Exercise 2.2

- New SVM Dual: Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n)$$

under the conditions

$$0 \leq a_n \leq C$$
$$\sum_{n=1}^{N} a_n t_n = 0$$

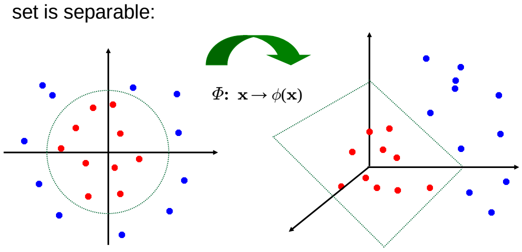
This is all that changed!

- 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
  - $\phi(\mathbf{x})$ only appears in the form of dot products $\phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{y})$:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) + b$$
$$= \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}) + b$$

  - Define a so-called kernel function $k(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{y})$.
  - Now, in place of the dot product, use the kernel instead:

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}) + b$$

  - The kernel function *implicitly* maps the data to the higher-dimensional space (without having to compute $\phi(\mathbf{x})$ explicitly)!

Recap: Kernels Fulfilling Mercer's Condition

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

Recap: Kernels Fulfilling Mercer's Condition

- Polynomial kernel
$$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}^{\mathrm{T}}\mathbf{y}+1)^p$$
- Radial Basis Function kernel
$$k(\mathbf{x},\mathbf{y}) = \exp\left\{-\frac{(\mathbf{x}-\mathbf{y})^2}{2\sigma^2}\right\}$$ e.g. Gaussian
- Hyperbolic tangent kernel
$$k(\mathbf{x},\mathbf{y}) = \cancel{\tanh(\kappa\mathbf{x}^{\mathrm{T}}\mathbf{y}+\delta)}$$ e.g. Sigmoid

Actually, that was wrong in the original SVM paper...
  - And many, many more, including kernels on graphs, strings, and symbolic data…

Recap: Nonlinear SVM – Dual Formulation

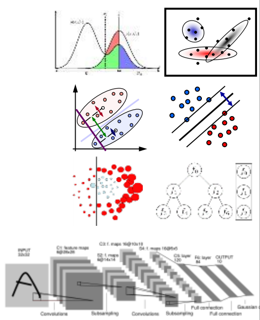
see Exercise 2.2

- SVM Dual: Maximize
$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_m,\mathbf{x}_n)$$

under the conditions
$$0 \cdot a_n \cdot C$$
$$\sum_{n=1}^{N} a_n t_n = 0$$
- Classify new data points using
$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n,\mathbf{x}) + b$$

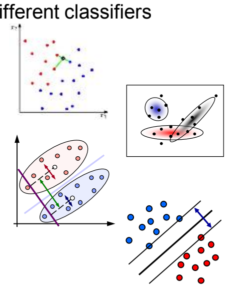
Course Outline

- 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: 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

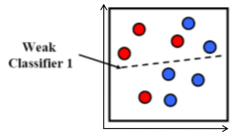
- Model Averaging
  - Suppose we have H different models $h = 1,\ldots,H$ with prior probabilities $p(h)$.
  - Construct the marginal distribution over the data set
$$p(\mathbf{X}) = \sum_{h=1}^{H} p(\mathbf{X}|h)p(h)$$
- Average error of committee
$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$$
  - 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: AdaBoost – "Adaptive Boosting"

- 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
  - $h_m(\mathbf{x})$: "weak" or base classifier
    - Condition: <50% training error over any distribution
  - $H(\mathbf{x})$: "strong" or final classifier
- AdaBoost:
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:

$$H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$$

Recap: AdaBoost – Intuition



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

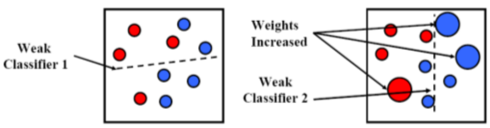
Each weak classifier splits the training examples with at least 50% accuracy.

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

Slide credit: Kristen Grauman

Figure adapted from Freund & Schapire

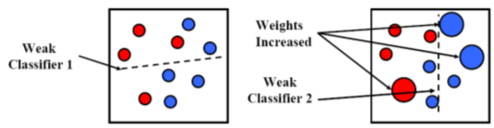
Recap: AdaBoost – Intuition

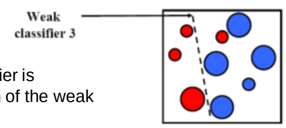


Slide credit: Kristen Grauman

Figure adapted from Freund & Schapire

Recap: AdaBoost – Intuition





Final classifier is combination of the weak classifiers

Slide credit: Kristen Grauman

Figure adapted from Freund & Schapire

Recap: AdaBoost – Algorithm

see Exercise 3.1

1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1,\ldots,N$.
2. For $m = 1,\ldots,M$ iterations
   a) Train a new weak classifier $h_m(\mathbf{x})$ using the current weighting coefficients $\mathbf{W}^{(m)}$ by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n) \qquad I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}$$

   b) Estimate the weighted error of this classifier on $\mathbf{X}$:

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

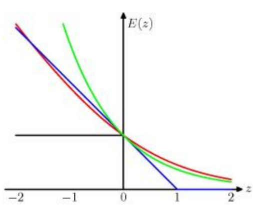
   c) Calculate a weighting coefficient for $h_m(\mathbf{x})$:

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}$$

   d) Update the weighting coefficients:

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{\alpha_m I(h_m(\mathbf{x}_n) \neq t_n)\right\}$$

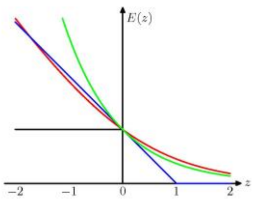
Recap: Comparing Error Functions



- 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!

Image source: Bishop, 2006

Recap: Comparing Error Functions



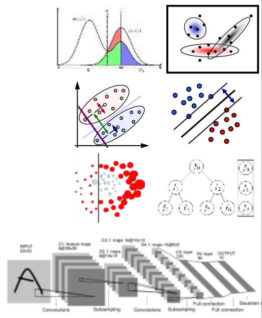
- Ideal misclassification error function
- "Hinge error" used in SVMs
- Exponential error function
- "Cross-entropy error" $E = -\sum\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$
  - Similar to exponential error for z>0.
  - Only grows linearly with large negative values of z.
  - ⇒ Make AdaBoost more robust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

Image source: Bishop, 2006

Machine Learning Winter '18 — 61

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

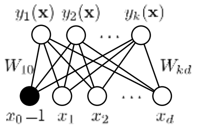


B. Leibe

Machine Learning Winter '18 — 62

Recap: Perceptrons

- One output node per class



Output layer
Weights
Input layer

- Outputs
  - Linear outputs

$$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki} x_i$$

With output nonlinearity

$$y_k(\mathbf{x}) = g\left(\sum_{i=0}^{d} W_{ki} x_i\right)$$

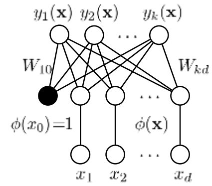
⇒ Can be used to do multidimensional linear regression or multiclass classification.

Slide adapted from Stefan Roth — B. Leibe

Machine Learning Winter '18 — 63

Recap: Non-Linear Basis Functions

- Straightforward generalization



Output layer
Weights
Feature layer
Mapping (fixed)
Input layer

- Outputs
  - Linear outputs

$$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki} \phi(x_i)$$

with output nonlinearity

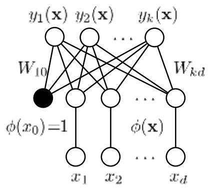
$$y_k(\mathbf{x}) = g\left(\sum_{i=0}^{d} W_{ki} \phi(x_i)\right)$$

B. Leibe

Machine Learning Winter '18 — 64

Recap: Non-Linear Basis Functions

- Straightforward generalization



Output layer
Weights
Feature layer
Mapping (fixed)
Input layer

- Remarks
  - Perceptrons are generalized linear discriminants!
  - Everything we know about the latter can also be applied here.
  - Note: feature functions $\phi(\mathbf{x})$ are kept fixed, not learned!

B. Leibe

Machine Learning Winter '18 — 65

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

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta\left(y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}\right)\phi_j(\mathbf{x}_n)$$

  - This is the Delta rule a.k.a. LMS rule!
  - ⇒ Perceptron Learning corresponds to 1st-order (stochastic) Gradient Descent of a quadratic error function!

Slide adapted from Geoff Hinton — B. Leibe

Machine Learning Winter '18 — 66

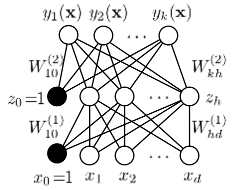
Recap: Loss Functions

- We can now also apply other loss functions
  - L_2 loss — $\Rightarrow$ Least-squares regression
    $L(t, y(\mathbf{x})) = \sum_n (y(\mathbf{x}_n) - t_n)^2$
  - L_1 loss: — $\Rightarrow$ Median regression
    $L(t, y(\mathbf{x})) = \sum_n |y(\mathbf{x}_n) - t_n|$
  - Cross-entropy loss — $\Rightarrow$ Logistic regression
    $L(t, y(\mathbf{x})) = -\sum_n \{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$
  - Hinge loss — $\Rightarrow$ SVM classification
    $L(t, y(\mathbf{x})) = \sum_n [1 - t_n y(\mathbf{x}_n)]_+$
  - Softmax loss — $\Rightarrow$ Multi-class probabilistic classification
    $L(t, y(\mathbf{x})) = -\sum_n \sum_k \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))} \right\}$

Machine Learning Winter '18 — B. Leibe — 67

Recap: Multi-Layer Perceptrons

- Adding more layers



- Output

$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

Slide adapted from Stefan Roth — Machine Learning Winter '18 — B. Leibe — 68

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

$$E(\mathbf{W}) = \sum_n L(t_n, y(\mathbf{x}_n; \mathbf{W})) + \lambda\Omega(\mathbf{W})$$

with a loss $L(\cdot)$ and a regularizer $\Omega(\cdot)$.

  - E.g., $L(t, y(\mathbf{x}; \mathbf{W})) = \sum_n (y(\mathbf{x}_n; \mathbf{W}) - t_n)^2$ — L_2 loss
    $\Omega(\mathbf{W}) = ||\mathbf{W}||_F^2$ — L_2 regularizer ("weight decay")

$\Rightarrow$ Update each weight $W_{ij}^{(k)}$ in the direction of the gradient $\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(k)}}$

Machine Learning Winter '18 — B. Leibe — 69

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

Machine Learning Winter '18 — B. Leibe — 70

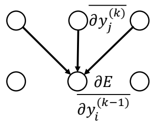
Recap: Backpropagation Algorithm

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

$$E = \frac{1}{2}\sum_{j \in output}(t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$

  2. Compute error derivatives in each hidden layer from error derivatives in the layer above.

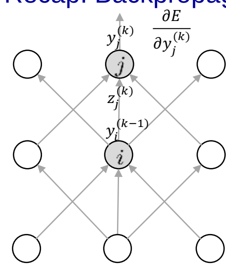


  3. Use error derivatives *w.r.t.* activities to get error derivatives *w.r.t.* the incoming weights

$$\frac{\partial E}{\partial y_j^{(k)}} \longrightarrow \frac{\partial E}{\partial w_{ji}^{(k-1)}}$$

Slide adapted from Geoff Hinton — Machine Learning Winter '18 — B. Leibe — 71

Recap: Backpropagation Algorithm



$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}}\frac{\partial E}{\partial y_j^{(k)}} = \frac{\partial g\left(z_j^{(k)}\right)}{\partial z_j^{(k)}}\frac{\partial E}{\partial y_j^{(k)}}$$

$$\frac{\partial E}{\partial y_i^{(k-1)}} = \sum_j \frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}}\frac{\partial E}{\partial z_j^{(k)}} = \sum_j w_{ji}^{(k-1)}\frac{\partial E}{\partial z_j^{(k)}}$$

$$\frac{\partial E}{\partial w_{ji}^{(k-1)}} = \frac{\partial z_j^{(k)}}{\partial w_{ji}^{(k-1)}}\frac{\partial E}{\partial z_j^{(k)}} = y_i^{(k-1)}\frac{\partial E}{\partial z_j^{(k)}}$$

- Efficient propagation scheme
  - $y_i^{(k-1)}$ is already known from forward pass! (Dynamic Programming)
  - $\Rightarrow$ Propagate back the gradient from layer k and multiply with $y_i^{(k-1)}$.

Slide adapted from Geoff Hinton — Machine Learning Winter '18 — B. Leibe — 72

Recap: MLP Backpropagation Algorithm

- Forward Pass

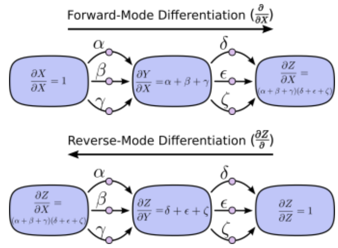
$$\mathbf{y}^{(0)} = \mathbf{x}$$
for $k = 1, ..., l$ do
$$\mathbf{z}^{(k)} = \mathbf{W}^{(k)}\mathbf{y}^{(k-1)}$$
$$\mathbf{y}^{(k)} = g_k(\mathbf{z}^{(k)})$$
endfor
$$\mathbf{y} = \mathbf{y}^{(l)}$$
$$E = L(\mathbf{t}, \mathbf{y}) + \lambda\Omega(\mathbf{W})$$

- Backward Pass

$$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}} = \frac{\partial}{\partial \mathbf{y}} L(\mathbf{t}, \mathbf{y}) + \lambda \frac{\partial}{\partial \mathbf{y}} \Omega$$
for $k = l, l\text{-}1, ..., 1$ do
$$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{z}^{(k)}} = \mathbf{h} \odot g'(\mathbf{y}^{(k)})$$
$$\frac{\partial E}{\partial \mathbf{W}^{(k)}} = \mathbf{h}\mathbf{y}^{(k-1)\top} + \lambda \frac{\partial \Omega}{\partial \mathbf{W}^{(k)}}$$
$$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}^{(k-1)}} = \mathbf{W}^{(k)\top}\mathbf{h}$$
endfor

- Notes
  - For efficiency, an entire batch of data $\mathbf{X}$ is processed at once.
  - $\odot$ denotes the element-wise product

Recap: Computational Graphs



Apply operator $\frac{\partial}{\partial X}$ to every node.

Apply operator $\frac{\partial Z}{\partial}$ to every node.

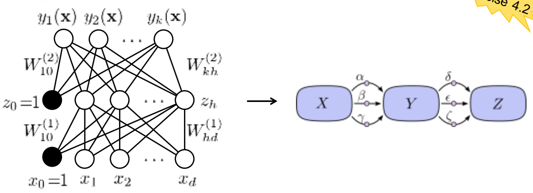
- Forward differentiation needs one pass per node. Reverse-mode differentiation can compute all derivatives in one single pass.

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

Recap: Automatic Differentiation

- Approach for obtaining the gradients

see Exercise 4.2

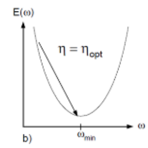


- Convert the network into a computational graph.
- Each new layer/module just needs to specify how it affects the forward and backward passes.
- Apply reverse-mode differentiation.

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

Recap: Choosing the Right Learning Rate

- Convergence of Gradient Descent
  - Simple 1D example

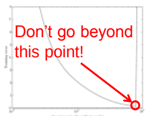


$$W^{(\tau-1)} = W^{(\tau)} - \eta \frac{\mathrm{d}E(W)}{\mathrm{d}W}$$

  - What is the optimal learning rate η_{opt}?
  - If E is quadratic, the optimal learning rate is given by the inverse of the Hessian

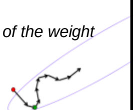
$$\eta_{\text{opt}} = \left(\frac{\mathrm{d}^2 E(W^{(\tau)})}{\mathrm{d}W^2}\right)^{-1}$$

  - Advanced optimization techniques try to approximate the Hessian by a simplified form.
  - *If we exceed the optimal learning rate, bad things happen!*



Recap: 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

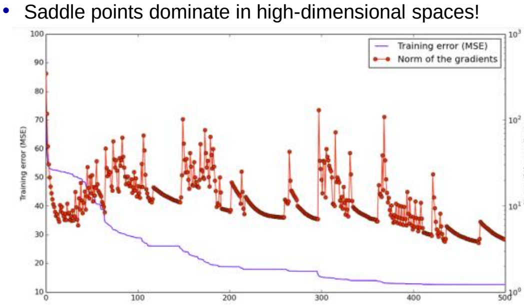


  - Nesterov-Momentum: Small variation in the implementation
- RMS-Prop
  - *Separate learning rate for each weight: Divide the gradient by a running average of its recent magnitude.*
- AdaGrad
- AdaDelta
- Adam

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

Recap: Patience

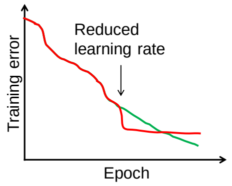
- Saddle points dominate in high-dimensional spaces!



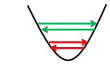
$\Rightarrow$ Learning often doesn't get stuck, you just may have to wait...

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.

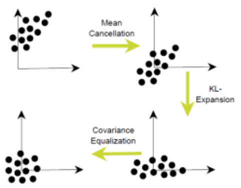
Recap: Data Augmentation

- 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.

Augmented training data
(from one original image)

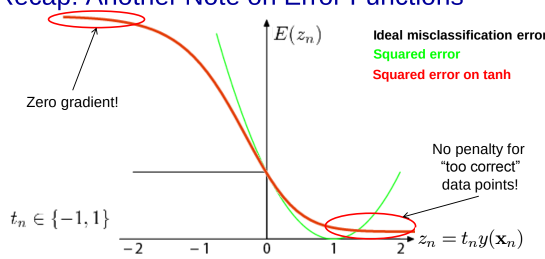
Recap: Normalizing the Inputs

- 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-Loeve 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!

  $\Rightarrow$ Do not use L_2 loss with sigmoid outputs (instead: cross-entropy)!

Recap: Commonly Used Nonlinearities

- Sigmoid

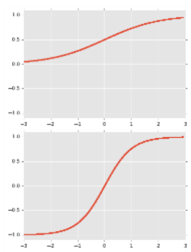
$$g(a) = \sigma(a) = \frac{1}{1+\exp\{-a\}}$$

- Hyperbolic tangent

$$g(a) = tanh(a) = 2\sigma(2a) - 1$$

- Softmax

$$g(\mathbf{a}) = \frac{\exp\{-a_i\}}{\sum_j \exp\{-a_j\}}$$



Recap: Commonly Used Nonlinearities (2)

- Rectified linear unit (ReLU)

$$g(a) = \max\{0, a\}$$

- Leaky ReLU

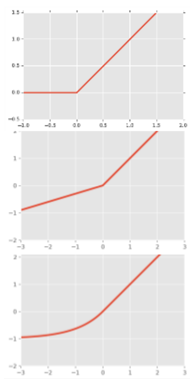
$$g(a) = \max\{\beta a, a\} \qquad \beta \in [0.01, 0.3]$$

  - 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 [Glorot & Bengio, '10]

- 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 $n \, \mathrm{Var}(W_i)$ should be 1. This means

$$\mathrm{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

  - Or for the backpropagated gradient

$$\mathrm{Var}(W_i) = \frac{1}{n_{\text{out}}}$$

  - As a compromise, Glorot & Bengio propose to use

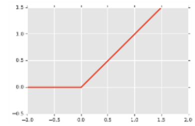
$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

$\Rightarrow$ Randomly sample the weights with this variance. That's it.

Recap: He Initialization [He et al., '15]

- 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

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$

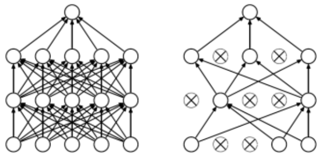
- Same basic idea: Output should have the input variance
  - However, the Glorot derivation was based on tanh units, linearity assumption around zero does not hold for ReLU.
  - He et al. made the derivations, proposed to use instead

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}$$

Recap: Batch Normalization [Ioffe & Szegedy '14]

- 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 [Srivastava, Hinton '12]



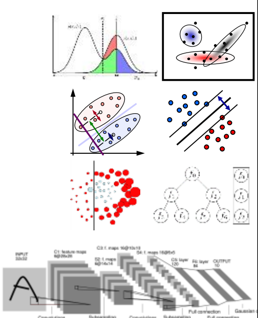
(a) Standard Neural Net (b) After applying dropout.

- 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.

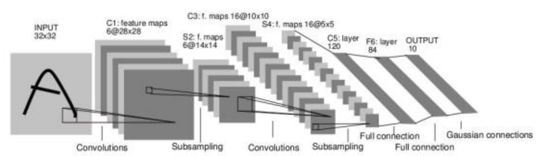
$\Rightarrow$ Improved performance

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



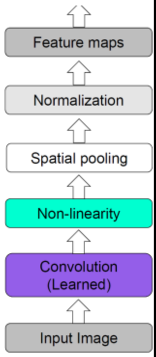
- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant features
  - Classification layer at the end

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86(11): 2278–2324, 1998.

Slide credit: Svetlana Lazebnik

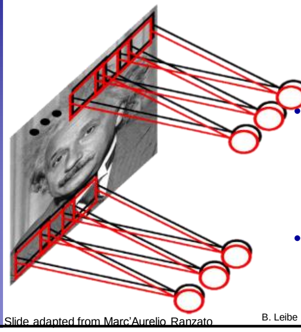
Recap: CNN Structure

- 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

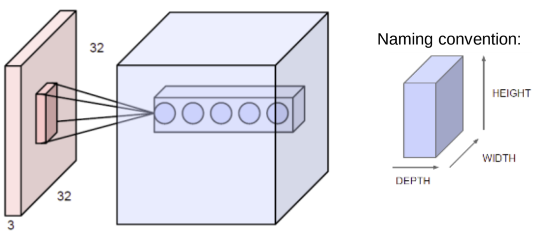
see Exercise 5.1



- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels
- Learn *multiple* filters
  - E.g. 1000×1000 image
    100 filters
    10×10 filter size

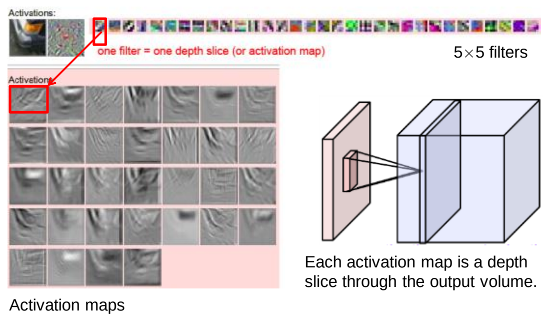
  $\Rightarrow$ only 10k parameters
- Result: Response map
  - size: 1000×1000×100
  - Only memory, not params!

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 [1×1×depth] depth column in output volume.

Recap: Activation Maps



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

Recap: Pooling Layers

Single depth slice (axes x, y)

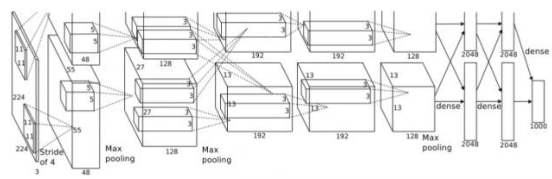
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2 →

6	8
3	4

- 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)

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.

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%.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Mainly used

Image source: Simonyan & Zisserman

Recap: GoogLeNet (2014)

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

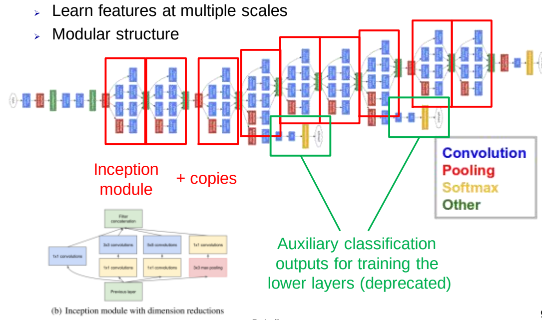


Image source: Szegedy et al.

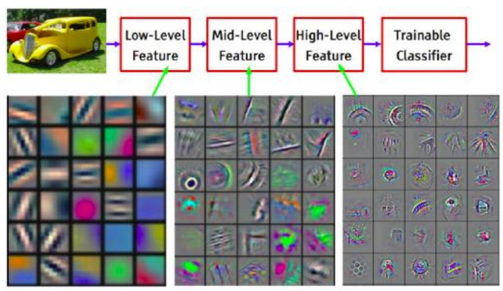
Discussion

- GoogLeNet
  - 12× fewer parameters than AlexNet
  - ⇒ ~5M parameters
  - *Where does the main reduction come from?*
  - ⇒ From throwing away the fully connected (FC) layers.
- Effect
  - After last pooling layer, volume is of size [7×7×1024]
  - 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 [1×1×1024].
  - ⇒ Performance actually improves by 0.6% compared to when using FC layers (less overfitting?)

Slide credit: Andrej Karpathy

Image source: Szegedy et al.

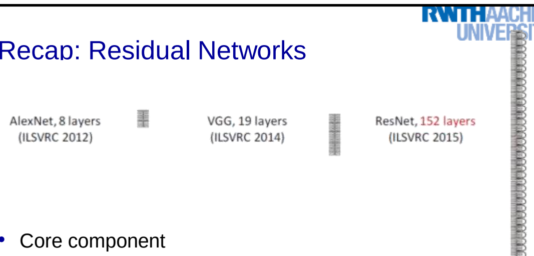
Recap: Visualizing CNNs



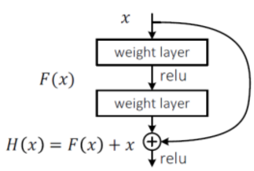
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Yann LeCun

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: 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.

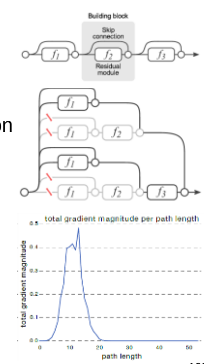
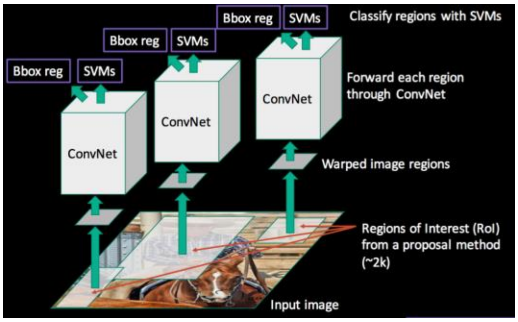


Image source: Veit et al. 2016

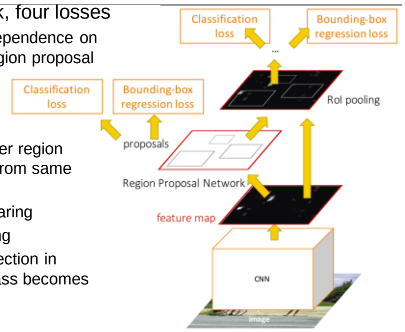
Recap: R-CNN for Object Detection



Slide credit: Ross Girshick

Recap: Faster R-CNN for Object Detection

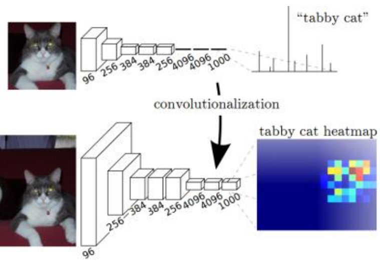
- One network, four losses
  - Remove dependence on external region proposal algorithm.
  - Instead, infer region proposals from same CNN.
  - Feature sharing
  - Joint training
  - ⇒ Object detection in a single pass becomes possible.



Slide credit: Ross Girshick

Recap: Fully Convolutional Networks

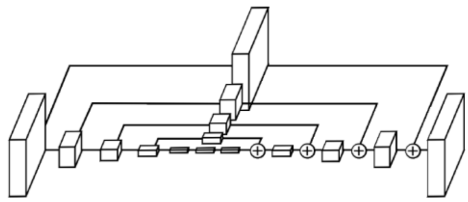
- CNN
- FCN



- Intuition
  - Think of FCNs as performing a sliding-window classification, producing a heatmap of output scores for each class

Image source: Long, Shelhamer, Darrell

Recap: Semantic Image Segmentation

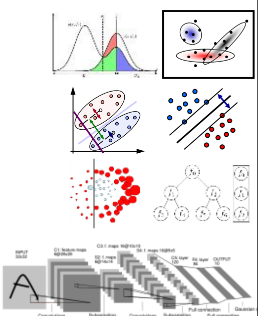


- Encoder-Decoder Architecture
  - Problem: FCN output has low resolution
  - Solution: perform upsampling to get back to desired resolution
  - Use skip connections to preserve higher-resolution information

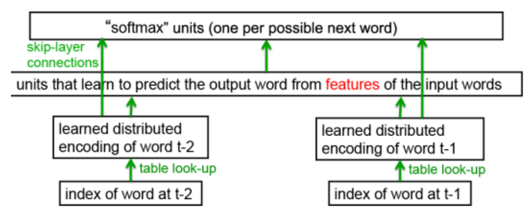
Image source: Newell et al.

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: Neural Probabilistic Language Model



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

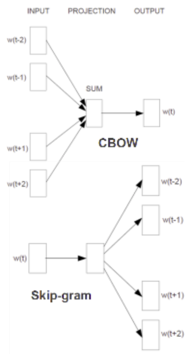
Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A Neural Probabilistic Language Model, In JMLR, Vol. 3, pp. 1137-1155, 2003.

Slide adapted from Geoff Hinton

Image source: Geoff Hinton

Recap: word2vec

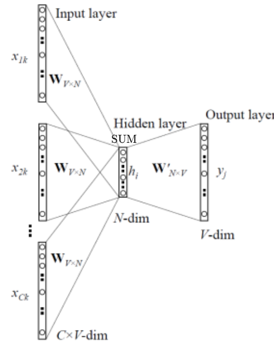
- Goal
  - Make it possible to learn high-quality word embeddings from huge data sets (billions of words in training set).
- Approach
  - Define two alternative learning tasks for learning the embedding:
    - "Continuous Bag of Words" (CBOW)
    - "Skip-gram"
  - Designed to require fewer parameters.



Recap: word2vec CBOW Model

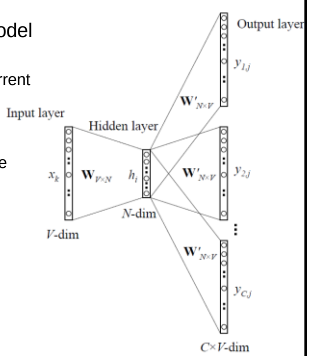
- Continuous BOW Model
  - Remove the non-linearity from the hidden layer
  - Share the projection layer for all words (their vectors are averaged)

  $\Rightarrow$ 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

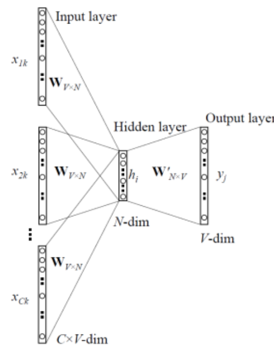


Recap: Problems with 100k-1M outputs

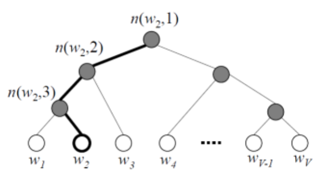
- Weight matrix gets huge!
  - Example: CBOW model
  - One-hot encoding for inputs

  $\Rightarrow$ 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.

  $\Rightarrow$ $\mathbf{W}'_{N\times V}$ has 300×1M entries
- Softmax gets expensive!
  - Need to compute normalization over 100k-1M outputs



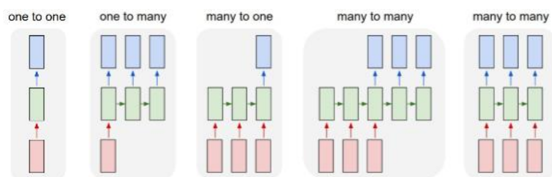
Recap: Hierarchical Softmax



- 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 $y = v_{n(w,j)} \cdot h$ at each node to decide whether to proceed with left or right child node.

  $\Rightarrow$ Decision based on output vector of hidden units directly.

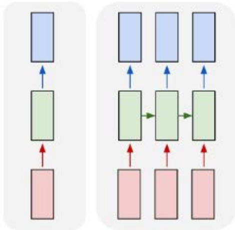
Recap: Recurrent Neural Networks



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

Recap: Recurrent Neural Networks (RNNs)

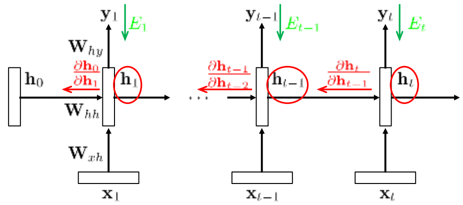
- RNNs are regular NNs whose hidden units have additional connections over time.
  - You can unroll them to create a network that extends over time.
  - When you do this, keep in mind that the weights for the hidden are shared between temporal layers.



- RNNs are very powerful
  - With enough neurons and time, they can compute anything that can be computed by your computer.

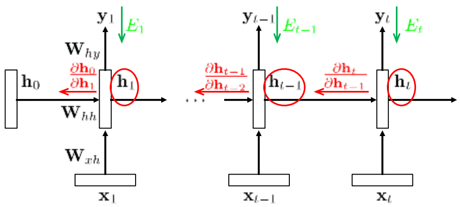
Image source: Andrej Karpathy

Recap: Backpropagation Through Time (BPTT)



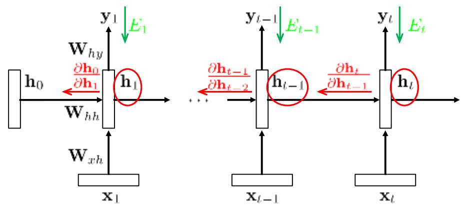
- Configuration
  $$\mathbf{h}_t = \sigma\left(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + b\right)$$
  $$\hat{\mathbf{y}}_t = \operatorname{softmax}\left(\mathbf{W}_{hy}\mathbf{h}_t\right)$$
- Backpropagated gradient
  - For weight w_{ij}:
  $$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$

Recap: Backpropagation Through Time (BPTT)



- Analyzing the terms
  - For weight w_{ij}:
  $$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$
  - This is the "immediate" partial derivative (with $\mathbf{h}_{k-1}$ as constant)

Recap: Backpropagation Through Time (BPTT)



- Analyzing the terms
  - For weight w_{ij}:
  $$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$
  - Propagation term:
  $$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k}\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}$$

Recap: Backpropagation Through Time (BPTT)

see Exercise 6.1

- Summary
  - Backpropagation equations
  $$E = \sum_{1 \le t \le T} E_t$$
  $$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$
  $$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k}\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{t \ge i > k}\mathbf{W}_{hh}^\top diag\left(\sigma'(\mathbf{h}_{i-1})\right)$$
  - Remaining issue: how to set the initial state $\mathbf{h}_0$?

  $\Rightarrow$ Learn this together with all the other parameters.

Recap: Exploding / Vanishing Gradient Problem

- BPTT equations:
  $$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$
  $$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k}\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{t \ge i > k}\mathbf{W}_{hh}^\top diag\left(\sigma'(\mathbf{h}_{i-1})\right)$$
  $$= \left(\mathbf{W}_{hh}^\top\right)^l$$
  (if t goes to infinity and $l = t - k$.)

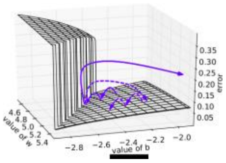
  $\Rightarrow$ We are effectively taking the weight matrix to a high power.
  - The result will depend on the eigenvalues of $\mathbf{W}_{hh}$.
    - Largest eigenvalue > 1 $\Rightarrow$ Gradients *may* explode.
    - Largest eigenvalue < 1 $\Rightarrow$ Gradients *will* vanish.
    - This is very bad...

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

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if



  - This makes a big difference in RNNs

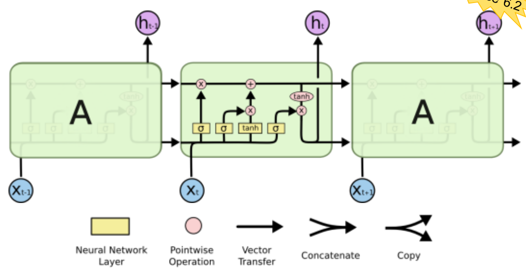
Machine Learning Winter '18

Slide adapted from Richard Socher

B. Leibe

Recap: Long Short-Term Memory

see Exercise 6.2



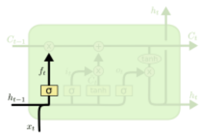
- LSTMs
  - Inspired by the design of memory cells
  - Each module has 4 layers, interacting in a special way.

Machine Learning Winter '18

Image source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

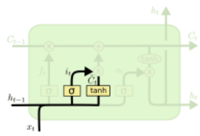
Recap: Elements of LSTMs

- Forget gate layer
  - Look at $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ and output a number between 0 and 1 for each dimension in the cell state $\mathbf{C}_{t-1}$.
    - 0: completely delete this,
    - 1: completely keep this.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

- 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.



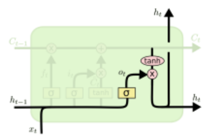
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Machine Learning Winter '18

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recap: Elements of LSTMs

- 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.



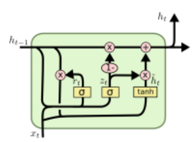
$$o_t = \sigma\left(W_o \, [h_{t-1}, x_t] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Machine Learning Winter '18

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- 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.

Machine Learning Winter '18

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Any More Questions?

Good luck for the exam!

Machine Learning Winter '18