Advanced Machine Learning Lecture 21

Repetition

06.02.2017

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

leibe@vision.rwth-aachen.de

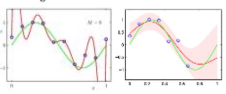
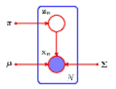
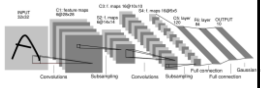
Announcements

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

This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \to \mathbb{R}$

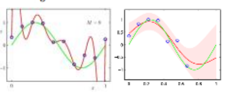
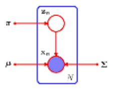
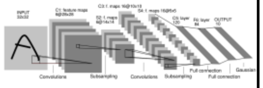
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes
- Approximate Inference
  - Sampling Approaches
  - MCMC
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs,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

This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \to \mathbb{R}$

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

Recap: Regression

- Learning to predict a continuous function value
  - Given: training set $\mathbf{X} = \{x_1, \ldots, x_N\}$ with target values $\mathbf{T} = \{t_1, \ldots, t_N\}$.

  $\Rightarrow$ Learn a continuous function $y(x)$ to predict the function value for a new input x.
- Define an error function $E(\mathbf{w})$ to optimize
  - E.g., sum-of-squares error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2$$

  - Procedure: Take the derivative and set it to zero

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}\frac{\partial y(x_n,\mathbf{w})}{\partial w_j} \overset{!}{=} 0$$

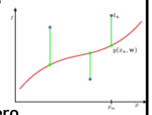


Image source: C.M. Bishop, 2006

Recap: Least-Squares Regression

$$\mathbf{x}_i^T\mathbf{w} + w_0 = t_i, \quad \forall i = 1,\ldots,n$$

- Setup
  - Step 1: Define $\tilde{\mathbf{x}}_i = \begin{pmatrix}\mathbf{x}_i \\ 1\end{pmatrix}, \quad \tilde{\mathbf{w}} = \begin{pmatrix}\mathbf{w} \\ w_0\end{pmatrix}$
  - Step 2: Rewrite $\tilde{\mathbf{x}}_i^T\tilde{\mathbf{w}} = t_i, \quad \forall i = 1,\ldots,n$
  - Step 3: Matrix-vector notation

$$\widetilde{\mathbf{X}}^T\tilde{\mathbf{w}} = \mathbf{t} \quad \text{with} \quad \begin{aligned}\widetilde{\mathbf{X}} &= [\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_n] \\ \mathbf{t} &= [t_1,\ldots,t_n]^T\end{aligned}$$

  - Step 4: Find least-squares solution

$$\|\widetilde{\mathbf{X}}^T\tilde{\mathbf{w}} - \mathbf{t}\|^2 \to \min$$

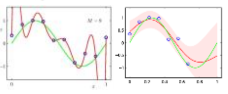
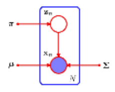
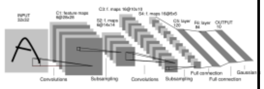
  - Solution: $\tilde{\mathbf{w}} = (\widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^T)^{-1}\widetilde{\mathbf{X}}\mathbf{t}$

Slide credit: Bernt Schiele

This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \to \mathbb{R}$

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

Recap: Regularization

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

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

  - Here we've simply added a quadratic regularizer, which is simple to optimize

$$\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w} = \cancel{w_0^2} + w_1^2 + \ldots + w_M^2$$

  - The resulting form of the problem is called Ridge Regression.
  - (*Note*: w_0 is often omitted from the regularizer.)

Recap: Probabilistic Regression

- First assumption:
  - Our target function values t are generated by adding noise to the ideal function estimate:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon$$

Target function value

Noise

Regression function

Input value

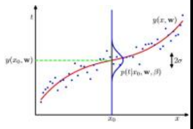
Weights or parameters

- Second assumption:
  - The noise is Gaussian distributed.

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

Mean

Variance (β precision)



Slide adapted from Bernt Schiele

Recap: Probabilistic Regression

- Given
  - Training data points: $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$
  - Associated function values: $\mathbf{t} = [t_1, \ldots, t_n]^T$
- Conditional likelihood (assuming i.i.d. data)

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N}\mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) = \prod_{n=1}^{N}\mathcal{N}(t_n|\underbrace{\mathbf{w}^T\phi(\mathbf{x}_n)}, \beta^{-1})$$

⇒ Maximize w.r.t. $\mathbf{w}$, β

Generalized linear regression function

Slide adapted from Bernt Schiele

Recap: Maximum Likelihood Regression

$$\nabla_{\mathbf{w}} \log p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = -\beta\sum_{n=1}^{N}(t_n - \mathbf{w}^T\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$

- Setting the gradient to zero:

$$0 = -\beta\sum_{n=1}^{N}(t_n - \mathbf{w}^T\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$

$$\Leftrightarrow \sum_{n=1}^{N} t_n\phi(\mathbf{x}_n) = \left[\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T\right]\mathbf{w}$$

$$\Leftrightarrow \mathbf{\Phi}\mathbf{t} = \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{w} \qquad \mathbf{\Phi} = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)]$$

$$\Leftrightarrow \mathbf{w}_{\text{ML}} = (\mathbf{\Phi}\mathbf{\Phi}^T)^{-1}\mathbf{\Phi}\mathbf{t}$$

Same as in least-squares regression!

⇒ *Least-squares regression is equivalent to Maximum Likelihood under the assumption of Gaussian noise.*

Slide adapted from Bernt Schiele

Recap: Role of the Precision Parameter

- Also use ML to determine the precision parameter β:

$$\log p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = -\frac{\beta}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)$$

- Gradient w.r.t. β:

$$\nabla_\beta \log p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = -\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 + \frac{N}{2}\frac{1}{\beta}$$

$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2$$

⇒ *The inverse of the noise precision is given by the residual variance of the target values around the regression function.*

Recap: Predictive Distribution

- Having determined the parameters $\mathbf{w}$ and β, we can now make predictions for new values of $\mathbf{x}$.

$$p(t|\mathbf{X}, \mathbf{w}_{\mathrm{ML}}, \beta_{\mathrm{ML}}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{\mathrm{ML}}), \beta_{\mathrm{ML}}^{-1})$$

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

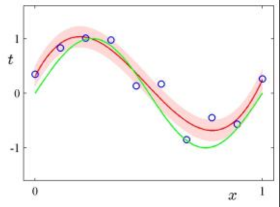


Image source: C.M. Bishop, 2006

Recap: Maximum-A-Posteriori Estimation

- Introduce a prior distribution over the coefficients $\mathbf{w}$.
  - For simplicity, assume a zero-mean Gaussian distribution

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right\}$$

  - New hyperparameter α controls the distribution of model parameters.

- Express the posterior distribution over $\mathbf{w}$.
  - Using Bayes' theorem:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \beta, \alpha) \propto p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha)$$

  - We can now determine $\mathbf{w}$ by maximizing the posterior.
  - This technique is called maximum-a-posteriori (MAP).

Recap: MAP Solution

- Minimize the negative logarithm

$$-\log p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \beta, \alpha) \propto -\log p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) - \log p(\mathbf{w}|\alpha)$$

$$-\log p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \mathrm{const}$$

$$-\log p(\mathbf{w}|\alpha) = \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + \mathrm{const}$$

- The MAP solution is therefore

$$\arg\min_{\mathbf{w}} \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$

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

Recap: MAP Solution (2)

$$\nabla_{\mathbf{w}} \log p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \beta, \alpha) = -\beta\sum_{n=1}^{N}(t_n - \mathbf{w}^T\phi(\mathbf{x}_n))\phi(\mathbf{x}_n) + \alpha\mathbf{w}$$

- Setting the gradient to zero:

$$0 = -\beta\sum_{n=1}^{N}(t_n - \mathbf{w}^T\phi(\mathbf{x}_n))\phi(\mathbf{x}_n) + \alpha\mathbf{w}$$

$$\Leftrightarrow \sum_{n=1}^{N} t_n\phi(\mathbf{x}_n) = \left[\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T\right]\mathbf{w} + \frac{\alpha}{\beta}\mathbf{w}$$

$$\Leftrightarrow \boldsymbol{\Phi}\mathbf{t} = \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \frac{\alpha}{\beta}\mathbf{I}\right)\mathbf{w} \qquad \boldsymbol{\Phi} = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)]$$

$$\Leftrightarrow \mathbf{w}_{\mathrm{MAP}} = \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \frac{\alpha}{\beta}\mathbf{I}\right)^{-1}\boldsymbol{\Phi}\mathbf{t}$$

Effect of regularization: Keeps the inverse well-conditioned

Recap: Bayesian Curve Fitting

- Given
  - Training data points: $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$
  - Associated function values: $\mathbf{t} = [t_1, \ldots, t_n]^T$
  - Our goal is to predict the value of t for a new point $\mathbf{x}$.

- Evaluate the predictive distribution

$$p(t|x, \mathbf{X}, \mathbf{t}) = \int \underbrace{p(t|x, \mathbf{w})}\underbrace{p(\mathbf{w}|\mathbf{X}, \mathbf{t})}d\mathbf{w}$$

What we just computed for MAP

  - Noise distribition - again assume a Gaussian here

$$p(t|x, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

  - Assume that parameters α and β are fixed and known for now.

Recap: Bayesian Curve Fitting

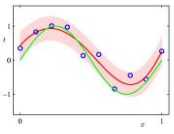
- Under those assumptions, the posterior distribution is a Gaussian and can be evaluated analytically:

$$p(t|x, \mathbf{X}, \mathbf{t}) = \mathcal{N}(t|m(x), s^2(x))$$

  - where the mean and variance are given by

$$m(x) = \beta\phi(x)^T\mathbf{S}\sum_{n=1}^{N}\phi(\mathbf{x}_n)t_n$$

$$s(x)^2 = \beta^{-1} + \phi(x)^T\mathbf{S}\phi(x)$$

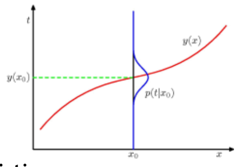


  - and S is the regularized covariance matrix

$$\mathbf{S}^{-1} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T$$

Image source: C.M. Bishop, 2006

Recap: Loss Functions for Regression



Mean prediction

- Optimal prediction
  - Minimize the expected loss

$$\mathbb{E}[L] = \iint L(t, y(\mathbf{x}))p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t$$

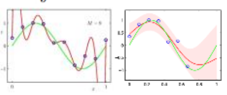
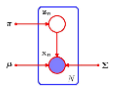
  - Under squared loss, the optimal regression function is the mean $\mathbb{E}[t|\mathbf{x}]$ of the posterior $p(t|\mathbf{x})$ ("mean prediction").
  - For generalized linear regression function and squared loss:

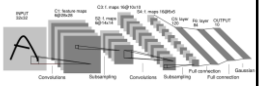
$$y(\mathbf{x}) = \int t\mathcal{N}(t|\mathbf{w}^T\phi(\mathbf{x}), \beta^{-1})\mathrm{d}t = \mathbf{w}^T\phi(\mathbf{x})$$

This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \to \mathbb{R}$

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



Recap: Loss Functions for Regression

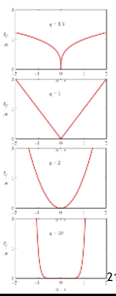
- The squared loss is not the only possible choice
  - Poor choice when conditional distribution $p(t|\mathbf{x})$ is multimodal.
- Simple generalization: Minkowski loss

$$L(t, y(\mathbf{x})) = |y(\mathbf{x}) - t|^q$$

  - Expectation

$$\mathbb{E}[L_q] = \iint |y(\mathbf{x}) - t|^q p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t$$

- Minimum of $\mathbb{E}[L_q]$ is given by
  - Conditional mean for $q = 2$,
  - Conditional median for $q = 1$,
  - Conditional mode for $q = 0$.

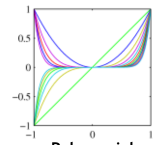


Recap: Linear Basis Function Models

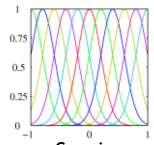
- Generally, we consider models of the following form

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$$

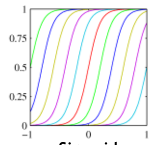
  - where $\phi_j(\mathbf{x})$ are known as *basis functions*.
  - In the simplest case, we use linear basis functions: $\phi_d(\mathbf{x}) = x_d$.
- Other popular basis functions



Polynomial



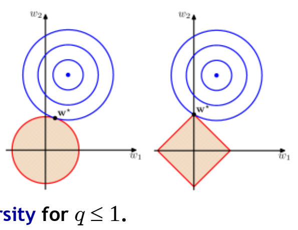
Gaussian



Sigmoid

Recap: Regularized Least-Squares

- Consider more general regularization functions
  - "L$_q$ norms": $\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$



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

Recap: The Lasso

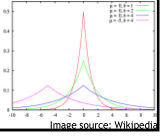
- L$_1$ regularization ("The Lasso")

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 + \lambda\sum_{j=1}^{M}|w_j|$$

  - The solution will be sparse (only few coefficients non-zero)
  - The L$_1$ penalty makes the problem non-linear.

  $\Rightarrow$ There is no closed-form solution.
- Interpretation as Bayes Estimation
  - We can think of $|w_j|^q$ as the log-prior density for w_j.
- Prior for Lasso $(q = 1)$:
  - Laplacian distribution

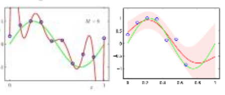
$$p(\mathbf{w}) = \frac{1}{2\tau}\exp\{-|\mathbf{w}|/\tau\} \quad \text{with} \quad \tau = \frac{1}{\lambda}$$



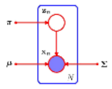
This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \rightarrow \mathbb{R}$

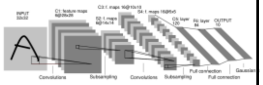
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes



- Approximate Inference
  - Sampling Approaches
  - MCMC



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



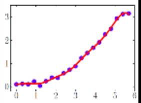
Recap: Kernel Ridge Regression

- Dual definition
  - Instead of working with $\mathbf{w}$, substitute $\mathbf{w} = \mathbf{\Phi}^T\mathbf{a}$ into $J(\mathbf{w})$ and write the result using the kernel matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}$$

  - Solving for $\mathbf{a}$, we obtain

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$



- Prediction for a new input $\mathbf{x}$:
  - Writing $\mathbf{k}(\mathbf{x})$ for the vector with elements $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\mathbf{\Phi}\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$

$\Rightarrow$ *The dual formulation allows the solution to be entirely expressed in terms of the kernel function $k(\mathbf{x},\mathbf{x}')$.*

Image source: Christoph Lampert

Recap: Properties of Kernels

- Theorem
  - *Let k: $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a positive definite kernel function. Then there exists a Hilbert Space* H *and a mapping $\varphi : \mathcal{X} \rightarrow \mathcal{H}$ such that*

$$k(x, x') = \langle(\phi(x), \phi(x')\rangle_{\mathcal{H}}$$

  - where $\langle . , . \rangle_{\mathcal{H}}$ *is the inner product in* H.
- Translation
  - Take *any* set $\mathcal{X}$ and *any* function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.
  - If k is a positive definite kernel, then we can use k to learn a classifier for the elements in $\mathcal{X}$!
- Note
  - $\mathcal{X}$ can be any set, e.g. $\mathcal{X}$ = *"all videos on YouTube"* or $\mathcal{X}$ = *"all permutations of* {1, . . . , *k*}*"*, or $\mathcal{X}$ = *"the internet"*.

Slide credit: Christoph Lampert

Recap: The "Kernel Trick"

Any algorithm that uses data only in the form of inner products can be *kernelized.*

- How to kernelize an algorithm
  - Write the algorithm only in terms of inner products.
  - Replace all inner products by kernel function evaluations.

$\Rightarrow$ The resulting algorithm will do the same as the linear version, but in the (hidden) feature space $\mathcal{H}$.
  - Caveat: working in $\mathcal{H}$ is not a guarantee for better performance. A good choice of k and model selection are important!

Slide credit: Christoph Lampert

Recap: How to Check if a Function is a Kernel

- Problem:
  - Checking if a given $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ fulfills the conditions for a kernel is difficult:
  - We need to prove or disprove

$$\sum_{i,j=1}^{n} t_i k(x_i, x_j) t_j \geq 0$$

for any set $x_1, \ldots, x_n \in \mathcal{X}$ and any $\mathbf{t} \in \mathbb{R}^n$ for any $n \in N$.

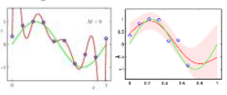
- Workaround:
  - It is easy to construct functions k that are positive definite kernels.

Slide credit: Christoph Lampert

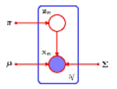
This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \rightarrow \mathbb{R}$

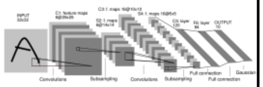
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes



- Approximate Inference
  - Sampling Approaches
  - MCMC



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



Recap: Gaussian Process

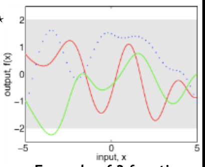
- Gaussian distribution
  - Probability distribution over scalars / vectors.
- Gaussian process (generalization of Gaussian distrib.)
  - Describes properties of functions.
  - Function: Think of a function as a long vector where each entry specifies the function value $f(\mathbf{x}_i)$ at a particular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

Recap: Gaussian Process

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

Recap: GPs Define Prior over Functions

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



Example of 3 functions sampled

Recap: Prediction with Noise-free Observations

- Assume our observations are noise-free:
  $$\{(\mathbf{x}_n, f_n) \mid n = 1, \ldots, N\}$$
  - Joint distribution of the training outputs $\mathbf{f}$ and test outputs $\mathbf{f}_\star$ according to the prior:
    $$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X_\star) \\ K(X_\star,X) & K(X_\star,X_\star) \end{bmatrix}\right)$$
  - Calculation of posterior corresponds to conditioning the joint Gaussian prior distribution on the observations:
    $$\mathbf{f}_\star | X_\star, X, \mathbf{f} \sim \mathcal{N}(\bar{\mathbf{f}}_\star, \mathrm{cov}[\mathbf{f}_\star]) \qquad \bar{\mathbf{f}}_\star = \mathbb{E}[\mathbf{f}_\star | X, X_\star, \mathbf{t}]$$
  - with:
    $$\bar{\mathbf{f}}_\star = K(X_\star, X)K(X,X)^{-1}\mathbf{f}$$
    $$\mathrm{cov}[\mathbf{f}_\star] = K(X_\star,X_\star) - K(X_\star,X)K(X,X)^{-1}K(X,X_\star)$$

Recap: Prediction with Noisy Observations

- Joint distribution of the observed values and the test locations under the prior:
  $$\begin{bmatrix} \mathbf{t} \\ \mathbf{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X)+\sigma_n^2 I & K(X,X_\star) \\ K(X_\star,X) & K(X_\star,X_\star) \end{bmatrix}\right)$$
  - Calculation of posterior corresponds to conditioning the joint Gaussian prior distribution on the observations:
    $$\mathbf{f}_\star | X_\star, X, \mathbf{t} \sim \mathcal{N}(\bar{\mathbf{f}}_\star, \mathrm{cov}[\mathbf{f}_\star]) \qquad \bar{\mathbf{f}}_\star = \mathbb{E}[\mathbf{f}_\star | X, X_\star, \mathbf{t}]$$
  - with:
    $$\bar{\mathbf{f}}_\star = K(X_\star, X)\left(K(X,X)+\sigma_n^2 I\right)^{-1}\mathbf{t}$$
    $$\mathrm{cov}[\mathbf{f}_\star] = K(X_\star,X_\star) - K(X_\star,X)\left(K(X,X)+\sigma_n^2 I\right)^{-1}K(X,X_\star)$$

$\Rightarrow$ This is the key result that defines Gaussian process regression!
  - Predictive distribution is Gaussian whose mean and variance depend on test points $X_\star$ and on the kernel $k(\mathbf{x},\mathbf{x}')$, evaluated on X.

Recap: GP Regression Algorithm

- Very simple algorithm

```
   input: $X$ (inputs), $\mathbf{y}$ (targets), $k$ (covariance function), $\sigma_n^2$ (noise level),
                                                                       $\mathbf{x}_*$ (test input)
2: $L := \mathrm{cholesky}(K + \sigma_n^2 I)$
   $\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$
                                                  } predictive mean eq. (2.25)
4: $\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$
   $\mathbf{v} := L \backslash \mathbf{k}_*$
                                                  } predictive variance eq. (2.26)
6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$
   $\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$    eq. (2.30)
8: return: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)
```

  - Based on the following equations (Matrix inv. $\leftrightarrow$ Cholesky fact.)
    $$\bar{f}_\star = \mathbf{k}_\star^T \left(K + \sigma_n^2 I\right)^{-1}\mathbf{t}$$
    $$\mathrm{cov}[f_\star] = k(\mathbf{x}_\star, \mathbf{x}_\star) - \mathbf{k}_\star^T\left(K + \sigma_n^2 I\right)^{-1}\mathbf{k}_\star$$
    $$\log p(\mathbf{t}|X) = -\frac{1}{2}\mathbf{t}^T\left(K + \sigma_n^2 I\right)^{-1}\mathbf{t} - \frac{1}{2}\log|K + \sigma_n^2 I| - \frac{N}{2}\log 2\pi$$

Recap: Computational Complexity

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

Recap: Bayesian Model Selection for GPs

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

Recap: Model Selection at Lowest Level

- Posterior of the parameters $\mathbf{w}$ is given by Bayes' rule

$$\begin{aligned} p(\mathbf{w}|\mathbf{t}, X, \theta, \mathcal{H}_i) &= \frac{p(\mathbf{t}|X, \mathbf{w}, \theta, \mathcal{H}_i)p(\mathbf{w}|\theta, X, \mathcal{H}_i)}{p(\mathbf{t}|X, \theta, \mathcal{H}_i)} \\ &= \frac{p(\mathbf{t}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\theta, \mathcal{H}_i)}{p(\mathbf{t}|X, \theta, \mathcal{H}_i)} \end{aligned}$$

- with
  - $p(\mathbf{t}|X,\mathbf{w},\mathcal{H}_i)$ likelihood and
  - $p(\mathbf{w}|\theta,\mathcal{H}_i)$ prior parameters $\mathbf{w}$,
  - Denominator (normalizing constant) is independent of the parameters and is called marginal likelihood.

$$p(\mathbf{t}|X, \theta, \mathcal{H}_i) = \int p(\mathbf{t}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\theta, \mathcal{H}_i)d\mathbf{w}$$

Recap: Model Selection at Mid Level

- Posterior of parameters θ is again given by Bayes' rule

$$\begin{aligned} p(\theta|\mathbf{t}, X, \mathcal{H}_i) &= \frac{p(\mathbf{t}|X, \theta, \mathcal{H}_i)p(\theta|X, \mathcal{H}_i)}{p(\mathbf{t}|X, \mathcal{H}_i)} \\ &= \frac{p(\mathbf{t}|X, \theta, \mathcal{H}_i)p(\theta|\mathcal{H}_i)}{p(\mathbf{t}|X, \mathcal{H}_i)} \end{aligned}$$

- where
  - The marginal likelihood of the previous level $p(\mathbf{t}|X,\theta,\mathcal{H}_i)$ plays the role of the likelihood of this level.
  - $p(\theta|\mathcal{H}_i)$ is the hyperprior (prior of the hyperparameters)
  - Denominator (normalizing constant) is given by:

$$p(\mathbf{t}|X, \mathcal{H}_i) = \int p(\mathbf{t}|X, \theta, \mathcal{H}_i)p(\theta|\mathcal{H}_i)d\theta$$

Recap: Model Selection at Top Level

- At the top level, we calculate the posterior of the model

$$p(\mathcal{H}_i|\mathbf{t}, X) = \frac{p(\mathbf{t}|X, \mathcal{H}_i)p(\mathcal{H}_i)}{p(\mathbf{t}|X)}$$

- where
  - Again, the denominator of the previous level $p(\mathbf{t}|X,\mathcal{H}_i)$ plays the role of the likelihood.
  - $p(\mathcal{H}_i)$ is the prior of the model structure.
  - Denominator (normalizing constant) is given by:

$$p(\mathbf{t}|X) = \sum_i p(\mathbf{t}|X, \mathcal{H}_i)p(\mathcal{H}_i)$$

Recap: Bayesian Model Selection

- Discussion
  - Marginal likelihood is main difference to non-Bayesian methods

$$p(\mathbf{t}|X, \mathcal{H}_i) = \int p(\mathbf{t}|X, \theta, \mathcal{H}_i)p(\theta|\mathcal{H}_i)d\theta$$

  - It automatically incorporates a trade-off between the model fit and the model complexity:
    - A simple model can only account for a limited range of possible sets of target values - if a simple model fits well, it obtains a high marginal likelihood.
    - A complex model can account for a large range of possible sets of target values - therefore, it can never attain a very high marginal likelihood.

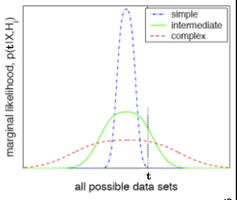
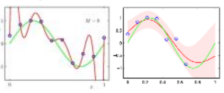
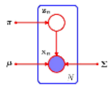
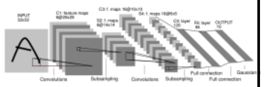


Image source: Rasmussen & Williams, 2006

This Lecture: *Advanced Machine Learning*

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

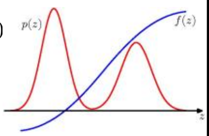
$$f : \mathcal{X} \rightarrow \mathbb{R}$$

Recap: Sampling Idea

- Objective:
  - Evaluate expectation of a function $f(\mathbf{z})$ w.r.t. a probability distribution $p(\mathbf{z})$.

$$\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z}$$



- Sampling idea
  - Draw L independent samples $\mathbf{z}^{(l)}$ with l = 1,…,L from $p(\mathbf{z})$.
  - This allows the expectation to be approximated by a finite sum

$$\hat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^l)$$

  - As long as the samples $\mathbf{z}^{(l)}$ are drawn independently from $p(\mathbf{z})$, then $\mathbb{E}[\hat{f}] = \mathbb{E}[f]$

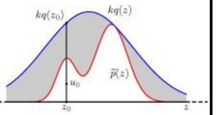
⇒ Unbiased estimate, independent of the dimension of $\mathbf{z}$!

Recap: Rejection Sampling

- Assumptions
  - Sampling directly from $p(\mathbf{z})$ is difficult.
  - But we can easily evaluate $p(\mathbf{z})$ (up to some norm. factor Z_p):

$$p(\mathbf{z}) = \frac{1}{Z_p}\tilde{p}(\mathbf{z})$$

- Idea
  - We need some simpler distribution $q(\mathbf{z})$ (called proposal distribution) from which we can draw samples.
  - Choose a constant k such that: $\forall z : kq(z) \geq \tilde{p}(z)$
- Sampling procedure
  - Generate a number z_0 from $q(z)$.
  - Generate a number u_0 from the uniform distribution over $[0,kq(z_0)]$.
  - If $u_0 > \tilde{p}(z_0)$ reject sample, otherwise accept.



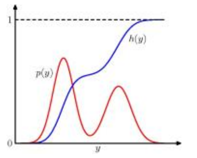
Recap: Sampling from a pdf

- In general, assume we are given the pdf $p(\mathbf{x})$ and the corresponding cumulative distribution:

$$F(x) = \int_{-\infty}^{x} p(z)dz$$

- To draw samples from this pdf, we can invert the cumulative distribution function:

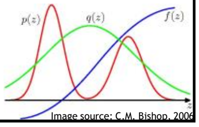
$$u \sim Uniform(0,1) \Rightarrow F^{-1}(u) \sim p(x)$$



Recap: Importance Sampling

- Approach
  - Approximate expectations directly (but does not enable to draw samples from $p(\mathbf{z})$ directly).
  - Goal: $\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z}$
- Idea
  - Use a proposal distribution $q(\mathbf{z})$ from which it is easy to sample.
  - Express expectations in the form of a finite sum over samples $\{\mathbf{z}^{(l)}\}$ drawn from $q(\mathbf{z})$.

$$\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} = \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z}$$
$$\simeq \frac{1}{L}\sum_{l=1}^{L}\underbrace{\frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})}}_{\text{Importance weights}} f(\mathbf{z}^{(l)})$$



Recap: Sampling-Importance-Resampling

- Motivation: Avoid having to determine the constant k.
- Two stages
  - Draw L samples $\mathbf{z}^{(1)}$,…, $\mathbf{z}^{(L)}$ from $q(\mathbf{z})$.
  - Construct weights using importance weighting

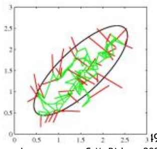
$$w_l = \frac{\tilde{r}_l}{\sum_m \tilde{r}_m} = \frac{\frac{\tilde{p}(\mathbf{z}^{(l)})}{\tilde{q}(\mathbf{z}^{(l)})}}{\sum_m \frac{\tilde{p}(\mathbf{z}^{(m)})}{\tilde{q}(\mathbf{z}^{(m)})}}$$

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

- Result
  - The resulting L samples are only approximately distributed according to $p(\mathbf{z})$, but the distribution becomes correct in the limit $L \rightarrow \infty$.

Recap: MCMC – Markov Chain Monte Carlo

- Overview
  - Allows to sample from a large class of distributions.
  - Scales well with the dimensionality of the sample space.
- Idea
  - We maintain a record of the current state $\mathbf{z}^{(\tau)}$
  - The proposal distribution depends on the current state: $q(\mathbf{z}|\mathbf{z}^{(\tau)})$
  - The sequence of samples forms a Markov chain $\mathbf{z}^{(1)}$, $\mathbf{z}^{(2)}$,…
- Approach
  - At each time step, we generate a candidate sample from the proposal distribution and accept the sample according to a criterion.
  - Different variants of MCMC for different criteria.



Recap: Markov Chains – Properties

- Invariant distribution
  - A distribution is said to be invariant (or stationary) w.r.t. a Markov chain if each step in the chain leaves that distribution invariant.
  - Transition probabilities:
  $$T\left(\mathbf{z}^{(m)}, \mathbf{z}^{(m+1)}\right) = p\left(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)}\right)$$
  - For homogeneous Markov chain, distribution $p^*(\mathbf{z})$ is invariant if:
  $$p^\star(\mathbf{z}) = \sum_{\mathbf{z}'} T(\mathbf{z}', \mathbf{z})\, p^\star(\mathbf{z}')$$
- Detailed balance
  - Sufficient (but not necessary) condition to ensure that a distribution is invariant:
  $$p^\star(\mathbf{z}) T(\mathbf{z}, \mathbf{z}') = p^\star(\mathbf{z}') T(\mathbf{z}', \mathbf{z})$$
  - A Markov chain which respects *detailed balance* is reversible.

Recap: Detailed Balance

- Detailed balance means
  - If we pick a state from the target distribution $p(\mathbf{z})$ and make a transition under T to another state, it is just as likely that we will pick $\mathbf{z}_A$ and go from $\mathbf{z}_A$ to $\mathbf{z}_B$ than that we will pick $\mathbf{z}_B$ and go from $\mathbf{z}_B$ to $\mathbf{z}_A$.
  - It can easily be seen that a transition probability that satisfies detailed balance w.r.t. a particular distribution will leave that distribution invariant, because
  $$\begin{aligned}\sum_{\mathbf{z}'} p^\star(\mathbf{z}') T(\mathbf{z}', \mathbf{z}) &= \sum_{\mathbf{z}'} p^\star(\mathbf{z}) T(\mathbf{z}, \mathbf{z}') \\ &= p^\star(\mathbf{z}) \sum_{\mathbf{z}'} p(\mathbf{z}'|\mathbf{z}) = p^\star(\mathbf{z})\end{aligned}$$

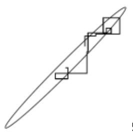
Recap: MCMC – Metropolis Algorithm

- Metropolis algorithm [Metropolis et al., 1953]
  - Proposal distribution is symmetric: $q(\mathbf{z}_A|\mathbf{z}_B) = q(\mathbf{z}_B|\mathbf{z}_A)$
  - The new candidate sample $\mathbf{z}^*$ is accepted with probability
  $$A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\tilde{p}(\mathbf{z}^\star)}{\tilde{p}(\mathbf{z}^{(\tau)})}\right)$$

  $\Rightarrow$ New candidate samples always accepted if $\tilde{p}(\mathbf{z}^\star) \geq \tilde{p}(\mathbf{z}^{(\tau)})$.
  - The algorithm sometimes accepts a state with lower probability.
- Metropolis-Hastings algorithm
  - Generalization: Proposal distribution not necessarily symmetric.
  - The new candidate sample $\mathbf{z}^*$ is accepted with probability
  $$A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\tilde{p}(\mathbf{z}^\star) q_k(\mathbf{z}^{(\tau)}|\mathbf{z}^\star)}{\tilde{p}(\mathbf{z}^{(\tau)}) q_k(\mathbf{z}^\star|\mathbf{z}^{(\tau)})}\right)$$
  - where k labels the members of the set of considered transitions.

Recap: Gibbs Sampling

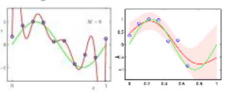
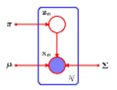
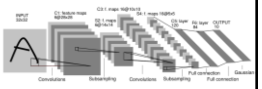
- Approach
  - MCMC-algorithm that is simple and widely applicable.
  - May be seen as a special case of Metropolis-Hastings.
- Idea
  - Sample variable-wise: replace $\mathbf{z}_i$ by a value drawn from the distribution $p(z_i|\mathbf{z}_{\setminus i})$.
    - This means we update one coordinate at a time.
  - Repeat procedure either by cycling through all variables or by choosing the next variable.
- Properties
  - The algorithm always accepts!
  - Completely parameter free.
  - Can also be applied to subsets of variables.



This Lecture: *Advanced Machine Learning*

$$f : \mathcal{X} \to \mathbb{R}$$

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

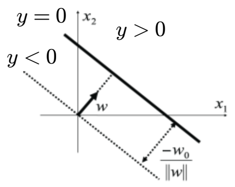




Recap: Linear Discriminant Functions

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

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

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.

Recap: Generalized Linear Discriminants

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

$$y_k(\mathbf{x}) = g\left(\sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}\right)$$

  - Basis functions $\phi_j(\mathbf{x})$ allow non-linear decision boundaries.
  - Activation function $g(\,\cdot\,)$ bounds the influence of outliers.
  - Disadvantage: minimization no longer in closed form.
- Notation

$$y_k(\mathbf{x}) = g\left(\sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})\right) \qquad \text{with } \phi_0(\mathbf{x}) = 1$$

Recap: Gradient Descent

- Iterative minimization
  - Start with an initial guess for the parameter values $w_{kj}^{(0)}$.
  - Move towards a (local) minimum by following the gradient.
- Basic strategies
  - "Batch learning" $$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left.\frac{\partial E(\mathbf{w})}{\partial w_{kj}}\right|_{\mathbf{w}^{(\tau)}}$$
  - "Sequential updating" $$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left.\frac{\partial E_n(\mathbf{w})}{\partial w_{kj}}\right|_{\mathbf{w}^{(\tau)}}$$

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

Recap: Gradient Descent

- Example: Quadratic error function

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

- Sequential updating leads to delta rule (=LMS rule)

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

  - where $$\delta_{kn} = y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}$$

$\Rightarrow$ Simply feed back the input data point, weighted by the classification error.

Recap: Probabilistic Discriminative Models

- Consider models of the form

$$\begin{aligned} p(\mathcal{C}_1|\boldsymbol{\phi}) &= y(\boldsymbol{\phi}) = \sigma(\mathrm{w}^T\boldsymbol{\phi}) \\ \text{with} \quad p(\mathcal{C}_2|\boldsymbol{\phi}) &= 1 - p(\mathcal{C}_1|\boldsymbol{\phi}) \end{aligned}$$

- 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 Sigmoid

- Properties
  - Definition: $$\sigma(a) = \frac{1}{1 + \exp(-a)}$$
  - Inverse: $$a = \ln\left(\frac{\sigma}{1-\sigma}\right)$$ "logit" function
  - Symmetry property: $$\sigma(-a) = 1 - \sigma(a)$$
  - Derivative: $$\frac{d\sigma}{da} = \sigma(1-\sigma)$$

Recap: Logistic Regression

- Let's consider a data set $\{\phi_n, t_n\}$ with $n = 1,\ldots,N$, where $\phi_n = \phi(\mathbf{x}_n)$ 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: Gradient of the Error Function

- Gradient for logistic regression
$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\phi_n$$
- This is the same result as for the Delta (=LMS) rule
$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta(y_k(\mathbf{x}_n;\mathbf{w}) - t_{kn})\phi_j(\mathbf{x}_n)$$
- We can use this to derive a sequential estimation algorithm.
  - However, this will be quite slow…
  - More efficient to use 2nd-order Newton-Raphson ⇒ IRLS

Recap: Iteratively Reweighted Least Squares

- Result of applying Newton-Raphson to logistic regression
$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T(\mathbf{y}-\mathbf{t}) \\ &= (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\left\{\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi}\mathbf{w}^{(\tau)} - \mathbf{\Phi}^T(\mathbf{y}-\mathbf{t})\right\} \\ &= (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{R}\mathbf{z} \end{aligned}$$
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.
  - ⇒ 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.

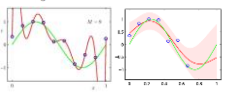
Recap: Softmax Regression Cost Function

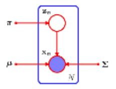
- Logistic regression
  - Alternative way of writing the cost function
$$\begin{aligned} E(\mathbf{w}) &= -\sum_{n=1}^{N} \{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \\ &= -\sum_{n=1}^{N}\sum_{k=0}^{1} \{\mathbb{I}(t_n = k)\ln P(y_n = k|\mathbf{x}_n;\mathbf{w})\} \end{aligned}$$
- Softmax regression
  - Generalization to K classes using indicator functions.
$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} \left\{\mathbb{I}(t_n = k)\ln \frac{\exp(\mathbf{w}_k^\top\mathbf{x})}{\sum_{j=1}^{K}\exp(\mathbf{w}_j^\top\mathbf{x})}\right\}$$
$$\nabla_{\mathbf{w}_k} E(\mathbf{w}) = -\sum_{n=1}^{N} [\mathbb{I}(t_n = k)\ln P(y_n = k|\mathbf{x}_n;\mathbf{w})]$$

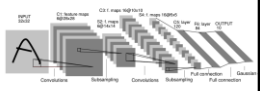
This Lecture: *Advanced Machine Learning*

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

$$f : \mathcal{X} \rightarrow \mathbb{R}$$

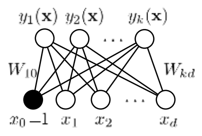






Recap: Perceptrons

- One output node per class



$y_1(\mathbf{x})$ $y_2(\mathbf{x})$ $y_K(\mathbf{x})$
W_{10}
W_{Kd}
$x_0=1$ x_1 x_2 x_d

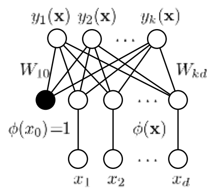
Output layer
Weights
Input layer

- 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.

Recap: Non-Linear Basis Functions

- Straightforward generalization

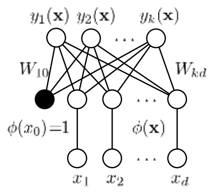


Output layer
Weights
Feature layer
Mapping (fixed)
Input layer

- Outputs
  - Linear outputs $$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki}\phi(x_i)$$
  - with output nonlinearity $$y_k(\mathbf{x}) = g\left(\sum_{i=0}^{d} W_{ki}\phi(x_i)\right)$$

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!

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!

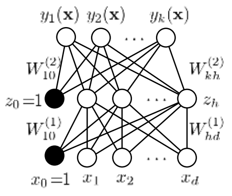
  ⇒ Perceptron Learning corresponds to 1st-order (stochastic) Gradient Descent of a quadratic error function!

Recap: Loss Functions

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

Recap: Multi-Layer Perceptrons

- Adding more layers



$y_1(\mathbf{x})$ $y_2(\mathbf{x})$ $y_K(\mathbf{x})$
$W_{10}^{(2)}$
$W_{Kh}^{(2)}$
$z_0=1$
z_h
$W_{10}^{(1)}$
$W_{hd}^{(1)}$
$x_0=1$ x_1 x_2 x_d

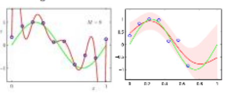
Output layer
Hidden layer
Input layer

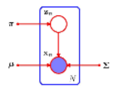
- Output $$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

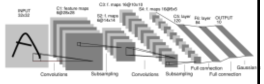
This Lecture: *Advanced Machine Learning*

$$f : \mathcal{X} \rightarrow \mathbb{R}$$

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







Recap: Learning with Hidden Units

- How can we train multi-layer networks efficiently?
  - Need an efficient way of adapting all weights, not just the last layer.
- Idea: Gradient Descent
  - Set up an error function

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

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”)

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

Recap: Gradient Descent

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

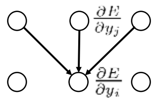
Recap: Backpropagation Algorithm

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



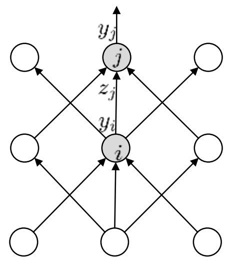
$\frac{\partial E}{\partial y_j}$
$\frac{\partial E}{\partial y_i}$

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

$$\frac{\partial E}{\partial y_j} \longrightarrow \frac{\partial E}{\partial w_{ik}}$$

Slide adapted from Geoff Hinton

Recap: Backpropagation Algorithm



$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial z_j}{\partial y_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

- Efficient propagation scheme
  - y_i is already known from forward pass! (Dynamic Programming)

⇒ Propagate back the gradient from layer j and multiply with y_i.

Slide adapted from Geoff Hinton

Recap: MLP Backpropagation Algorithm

- Forward Pass

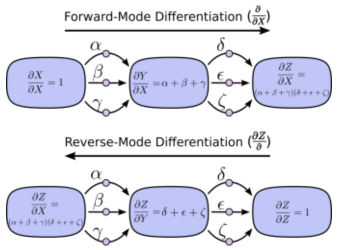
```
$\mathbf{y}^{(0)} = \mathbf{x}$
for $k = 1, ..., l$ do
    $\mathbf{z}^{(k)} = \mathbf{W}^{(k)}\mathbf{y}^{(k-1)}$
    $\mathbf{y}^{(k)} = g_k(\mathbf{z}^{(k)})$
endfor
$\mathbf{y} = \mathbf{y}^{(l)}$
$E = L(\mathbf{t}, \mathbf{y}) + \lambda\Omega(\mathbf{W})$
```

- 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


Forward-Mode Differentiation ($\frac{\partial}{\partial X}$)
$\frac{\partial X}{\partial X}=1$
$\frac{\partial Y}{\partial X}=\alpha+\beta+\gamma$
$\frac{\partial Z}{\partial X}=(\alpha+\beta+\gamma)(\delta+\epsilon+\zeta)$
Reverse-Mode Differentiation ($\frac{\partial Z}{\partial}$)
$\frac{\partial Z}{\partial X}=(\alpha+\beta+\gamma)(\delta+\epsilon+\zeta)$
$\frac{\partial Z}{\partial Y}=\delta+\epsilon+\zeta$
$\frac{\partial Z}{\partial Z}=1$

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

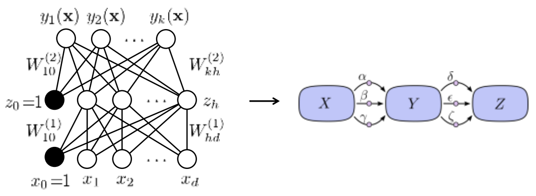
Apply operator $\frac{\partial Z}{\partial}$ to every node.

- 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


$y_1(\mathbf{x})$ $y_2(\mathbf{x})$ $y_k(\mathbf{x})$
$W_{10}^{(2)}$
$W_{kh}^{(2)}$
$z_0=1$
z_h
$W_{10}^{(1)}$
$W_{hd}^{(1)}$
$x_0=1$ x_1 x_2 x_d
X Y Z
α β γ δ ϵ ζ

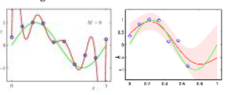
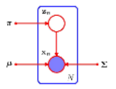
- Convert the network into a computational graph.
- Each new layer/module just needs to specify how it affects the forward and backward passes.
- Apply reverse-mode differentiation.

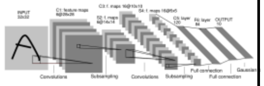
$\Rightarrow$ Very general algorithm, used in today's Deep Learning packages

This Lecture: *Advanced Machine Learning*

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

$$f : \mathcal{X} \to \mathbb{R}$$



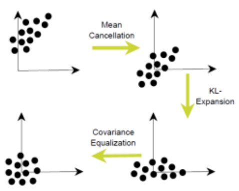
Recap: Data Augmentation

- 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)
  - 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: Choosing the Right Learning Rate

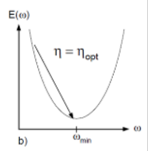
- Convergence of Gradient Descent
  - Simple 1D example

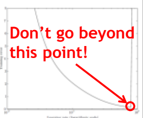
$$W^{(\tau-1)} = W^{(\tau)} - \eta\frac{\mathrm{d}E(W)}{\mathrm{d}W}$$

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

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

  - 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.

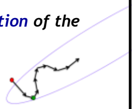
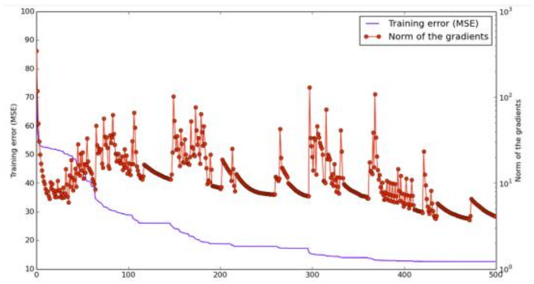


Image source: Geoff Hinton

Recap: Patience

- Saddle points dominate in high-dimensional spaces!

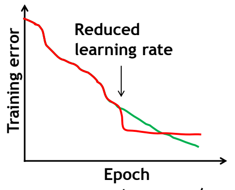


$\Rightarrow$ Learning often doesn't get stuck, you just may have to wait...

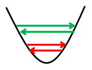
Image source: Yoshua Bengio

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.

Slide adapted from Geoff Hinton

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

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

  - Or for the backpropagated gradient

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

  - As a compromise, Glorot & Bengio propose to use

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

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

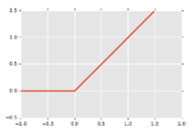
Recap: He Initialization [He et al., '15]

- 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

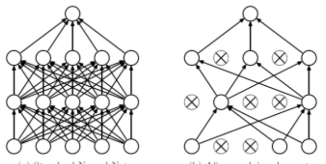
$$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}}}$$

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
  - Much improved convergence

Recap: Dropout

[Srivastava, Hinton ’12]



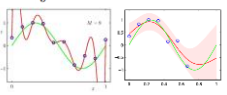
(a) Standard Neural Net (b) After applying dropout.

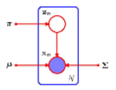
- Idea
  - Randomly switch off units during training.
  - Change network architecture for each data point, effectively training many different variants of the network.
  - When applying the trained network, multiply activations with the probability that the unit was set to zero.

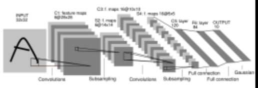
  $\Rightarrow$ Greatly improved performance

This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \rightarrow \mathbb{R}$







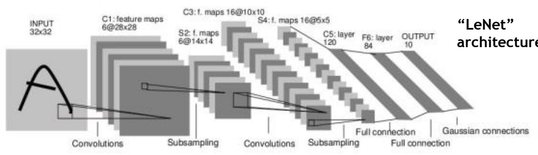
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes
- Approximate Inference
  - Sampling Approaches
  - MCMC
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs,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

Recap: ImageNet Challenge 2012

[Deng et al., CVPR’09]

- ImageNet
  - ~14M labeled internet images
  - 20k classes
  - Human labels via Amazon Mechanical Turk
- Challenge (ILSVRC)
  - 1.2 million training images
  - 1000 classes
  - Goal: Predict ground-truth class within top-5 responses
  - Currently one of the top benchmarks in Computer Vision

Recap: Convolutional Neural Networks



“LeNet” architecture

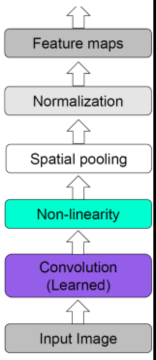
- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant features
  - Classification layer at the end

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

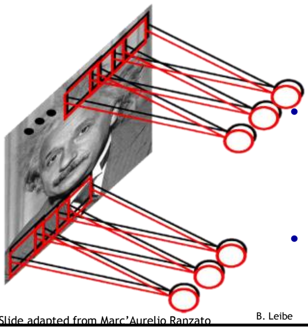
Recap: CNN Structure

- 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



Slide credit: Svetlana Lazebnik

Recap: Intuition of CNNs



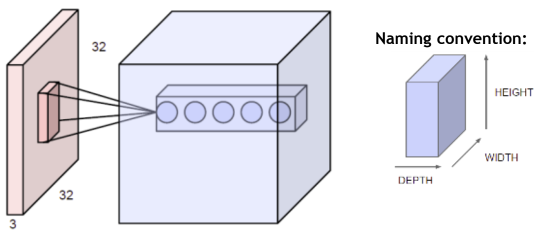
- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels
- Learn *multiple* filters
  - E.g. 1000×1000 image
    100 filters
    10×10 filter size

  $\Rightarrow$ only 10k parameters
- Result: Response map
  - size: $1000\times1000\times100$
  - Only memory, not params!

Slide adapted from Marc’Aurelio Ranzato

Image source: Yann LeCun

Recap: Convolution Layers

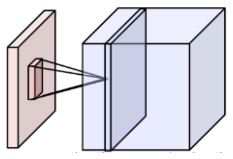


- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth
  - Form a single [1×1×depth] depth column in output volume.

Recap: Activation Maps

Activations:
one filter = one depth slice (or activation map)
Activations:

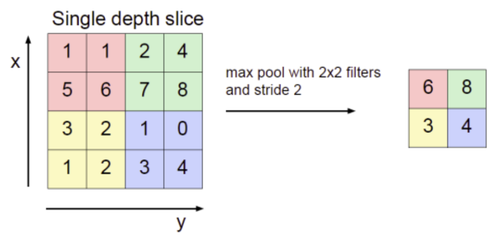
5×5 filters



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

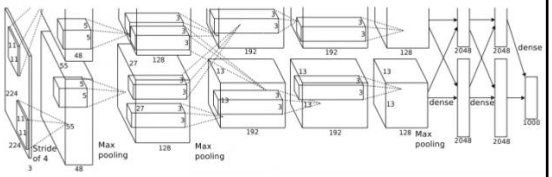
Activation maps

Recap: Pooling Layers



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

Recap: AlexNet (2012)



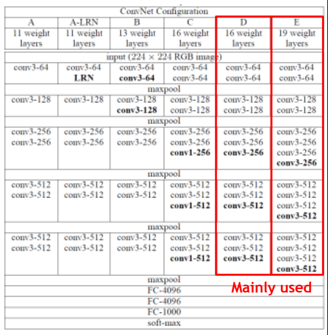
- Similar framework as LeNet, but
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data (10^6 images instead of 10^3)
  - GPU implementation
  - Better regularization and up-to-date tricks for training (Dropout)

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

Image source: A. Krizhevsky, I. Sutskever and G.E. Hinton, 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

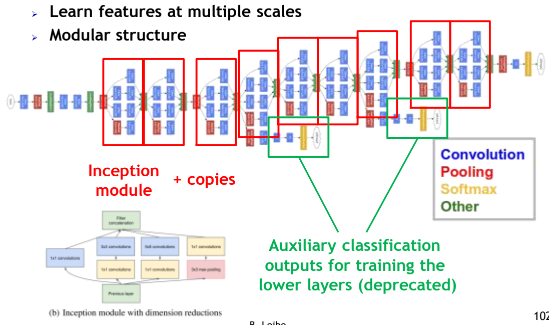
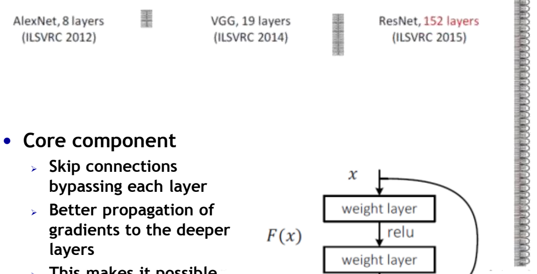
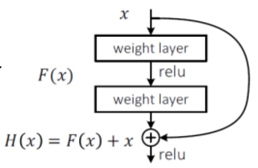

Inception module
+ copies
Convolution
Pooling
Softmax
Other
Auxiliary classification outputs for training the lower layers (deprecated)
(b) Inception module with dimension reductions

Image source: Szegedy et al.

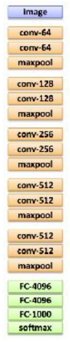
Recap: Residual Networks



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

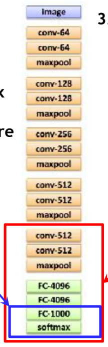


Recap: Transfer Learning with CNNs



1. Train on ImageNet
2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

I.e., replace the Softmax layer at the end



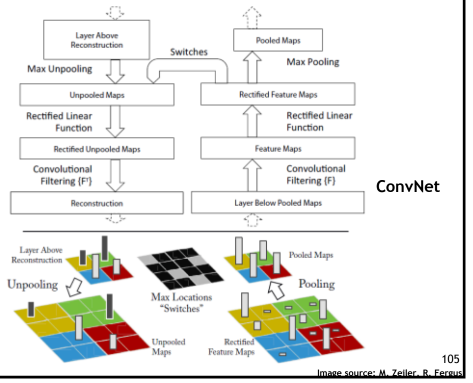
3. If you have a medium sized dataset, "finetune" instead: use the old weights as initialization, train the full network or only some of the higher layers.

Retrain bigger part of the network

Slide credit: Andrej Karpathy

Recap: Visualizing CNNs

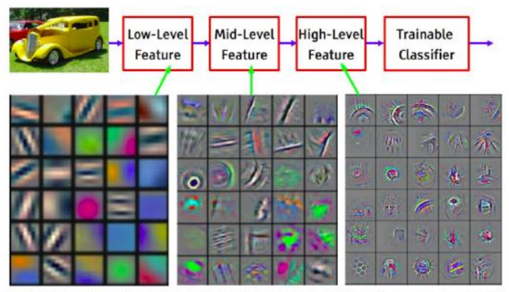
DeconvNet



ConvNet

Image source: M. Zeiler, R. Fergus

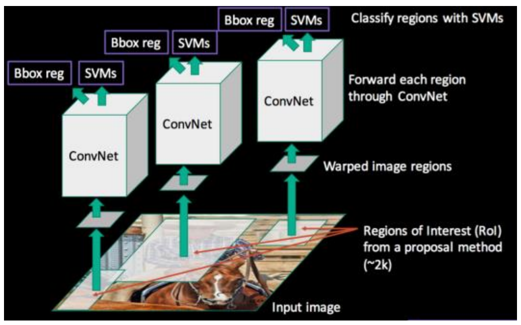
Recap: Visualizing CNNs



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

Slide credit: Yann LeCun

Recap: R-CNN for Object Deteection

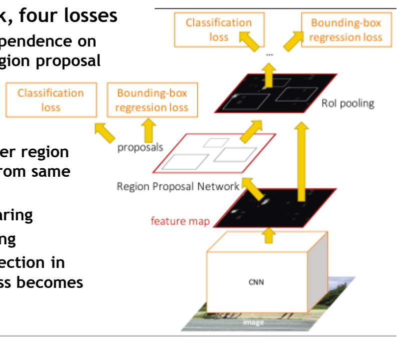

Bbox reg
SVMs
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Slide credit: Ross Girshick

Recap: Faster R-CNN

- One network, four losses
  - Remove dependence on external region proposal algorithm.
  - Instead, infer region proposals from same CNN.
  - Feature sharing
  - Joint training

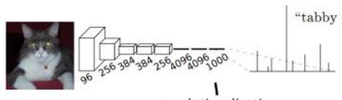
  ⇒ Object detection in a single pass becomes possible.



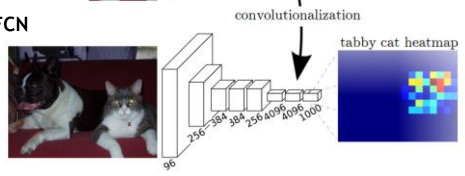
Slide credit: Ross Girshick

Recap: Fully Convolutional Networks

- CNN



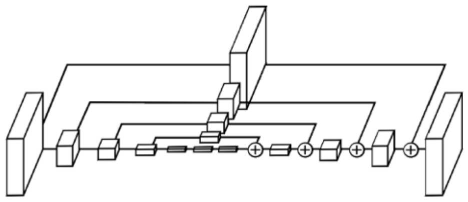
- FCN



- 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: Image Segmentation Networks



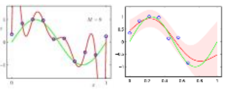
- Encoder-Decoder Architecture
  - Problem: FCN output has low resolution
  - Solution: perform upsampling to get back to desired resolution
  - Use skip connections to preserve higher-resolution information

Image source: Newell et al.

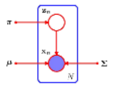
This Lecture: *Advanced Machine Learning*

$f : \mathcal{X} \to \mathbb{R}$

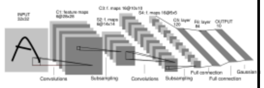
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes



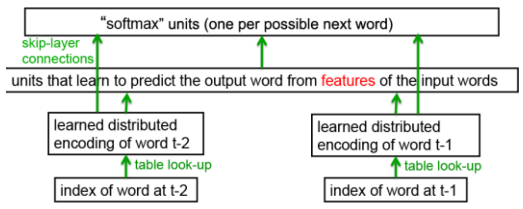
- Approximate Inference
  - Sampling Approaches
  - MCMC



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



Recap: Neural Probabilistic Language Model


"softmax" units (one per possible next word)
skip-layer connections
units that learn to predict the output word from features of the input words
learned distributed encoding of word t-2
learned distributed encoding of word t-1
table look-up
table look-up
index of word at t-2
index of word at t-1

- 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.

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.

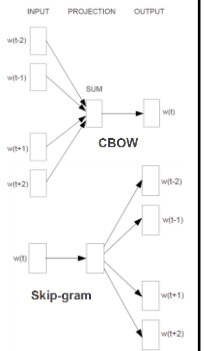
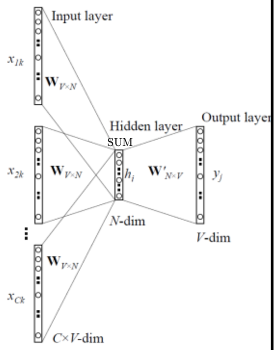


Image source: Mikolov et al., 2015

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)

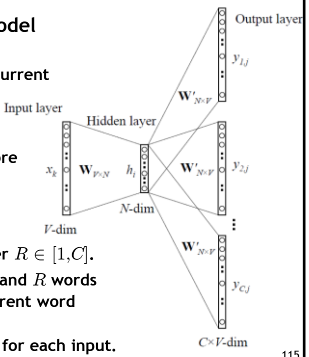
$\Rightarrow$ Bag-of-Words model (order of the words does not matter anymore)



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
- Implementation
  - Randomly choose a number $R \in [1,C]$.
  - Use R words from history and R words from the future of the current word as correct labels.

  $\Rightarrow$ $R+R$ word classifications for each input.

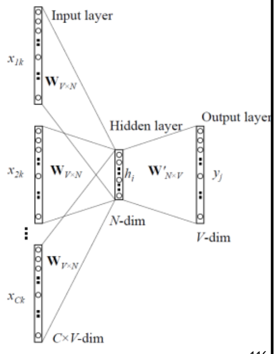


Problems with 100k-1M outputs

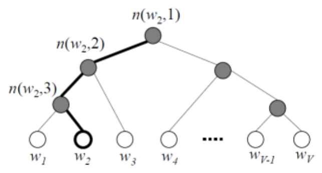
- Weight matrix gets huge!
  - Example: CBOW model
  - One-hot encoding for inputs

  $\Rightarrow$ Input-hidden connections are just vector lookups.
  - This is not the case for the hidden-output connections!
  - State h is not one-hot, and vocabulary size is 1M.

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



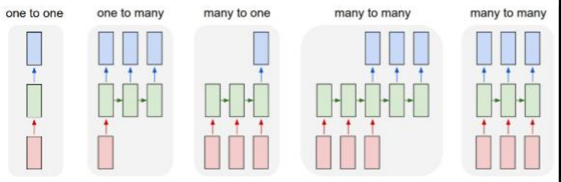
Recap: Hierarchical Softmax



- 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.

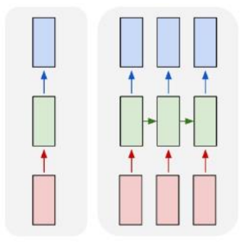
Recap: Recurrent Neural Networks



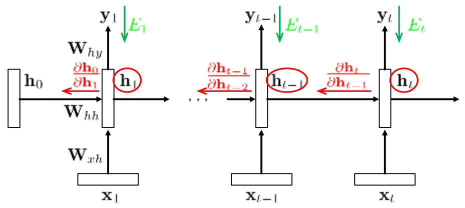
- Up to now
  - Simple neural network structure: 1-to-1 mapping of inputs to outputs
- Recurrent Neural Networks
  - Generalize this to arbitrary mappings

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.



Recap: Backpropagation Through Time (BPTT)



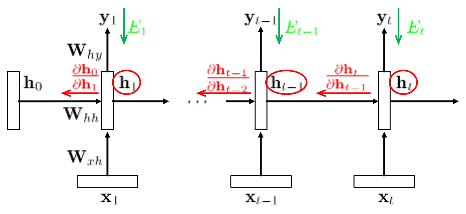
- Configuration

$$\mathbf{h}_t = \sigma\left(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + b\right)$$
$$\hat{\mathbf{y}}_t = \mathrm{softmax}\left(\mathbf{W}_{hy}\mathbf{h}_t\right)$$

- Backpropagated gradient
  - For weight w_{ij}:

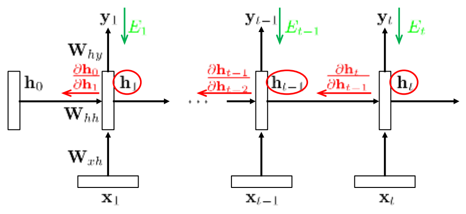
$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1\le k\le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$

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)

- 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 $\mathbf{h}_0$?

  ⇒ Learn this together with all the other parameters.

Recap: Exploding / Vanishing Gradient Problem

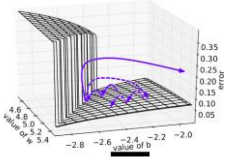
- BPTT equations:
$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial w_{ij}} \right)$$
$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}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

⇒ 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 ⇒ Gradients *may* explode.
    - Largest eigenvalue < 1 ⇒ Gradients *will* vanish.
    - This is very bad...

Recap: Gradient Clipping

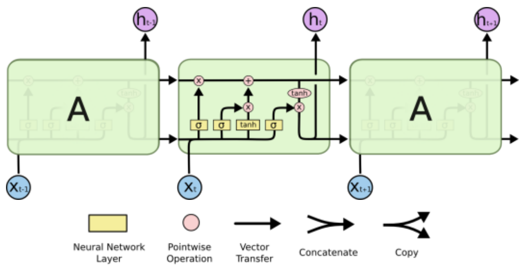
- Trick to handle exploding gradients
  - If the gradient is larger than a threshold, clip it to that threshold.

```
Algorithm 1 Pseudo-code
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \ge threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if
```



  - This makes a big difference in RNNs

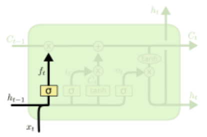
Recap: Long Short-Term Memory (LSTM)



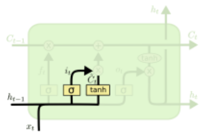
- LSTMs
  - Inspired by the design of memory cells
  - Each module has 4 layers, interacting in a special way.

Recap: Elements of LSTMs

- Forget gate layer
  - Look at $\mathbf{h}_{t-1}$ 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.
- 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.



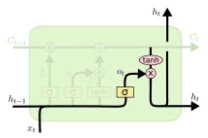
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



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

Recap: Elements of LSTMs

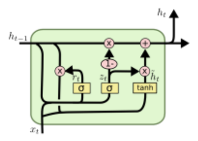
- Output gate layer
  - Output is a filtered version of our gate state.
  - First, apply sigmoid layer to decide what parts of the cell state to output.
  - Then, pass the cell state through a tanh (to push the values to be between -1 and 1) and multiply it with the output of the sigmoid gate.



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

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.

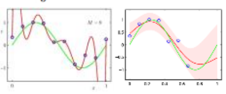


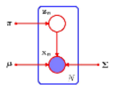
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

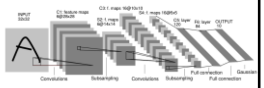
This Lecture: *Advanced Machine Learning*

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

$$f : \mathcal{X} \to \mathbb{R}$$







Recap: Reinforcement Learning

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

$$\begin{tikzcd}\text{Agent} \arrow[r, bend left, "\text{action}"] & \text{Environment} \arrow[l, bend left, "\text{observation, reward}"]\end{tikzcd}$$

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

Slide adapted from: David Silver, Sergey Levine

Recap: Reward vs. Return

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

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

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

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

$\Rightarrow$ Temporal credit assignment problem

Recap: Policy

- Definition
  - A policy determines the agent's behavior
  - Map from state to action $\pi: \mathcal{S} \rightarrow \mathcal{A}$
- Two types of policies
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \Pr\{A_t = a | S_t = s\}$
- Note
  - $\pi(a|s)$ denotes the probability of taking action a when in state s.

Recap: Value Function

- Idea
  - Value function is a prediction of future reward
  - Used to evaluate the goodness/badness of states
  - And thus to select between actions
- Definition
  - The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s]$$

  - The value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$, is the expected return starting from s, taking action a, and following π thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$$

Recap: Optimal Value Functions

- Bellman optimality equations
  - For the optimal state-value function v_*:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a)$$
$$= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

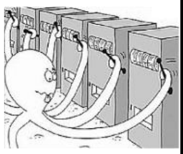
  - v_* is the unique solution to this system of nonlinear equations.
  - For the optimal action-value function q_*:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a')\right]$$

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

$\Rightarrow$ If the dynamics of the environment $p(s',r|s,a)$ are known, then in principle one can solve those equation systems.

Recap: Exploration-Exploitation Trade-off



- Example: N-armed bandit problem
  - Suppose we have the choice between N actions $a_1, \ldots, a_N$.
  - If we knew their value functions $q_*(s, a_i)$, it would be trivial to choose the best.
  - However, we only have estimates based on our previous actions and their returns.
- We can now
  - Exploit our current knowledge
    - And choose the greedy action that has the highest value based on our current estimate.
  - Explore to gain additional knowledge
    - And choose a non-greedy action to improve our estimate of that action's value.

Recap: TD-Learning

- Policy evaluation (the prediction problem)
  - For a given policy π, compute the state-value function v_π.
- One option: Monte-Carlo methods
  - Play through a sequence of actions until a reward is reached, then backpropagate it to the states on the path.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Target: the actual return after time t

- Temporal Difference Learning – TD(λ)
  - Directly perform an update using the estimate $V(S_{t+\lambda+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)]$$

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

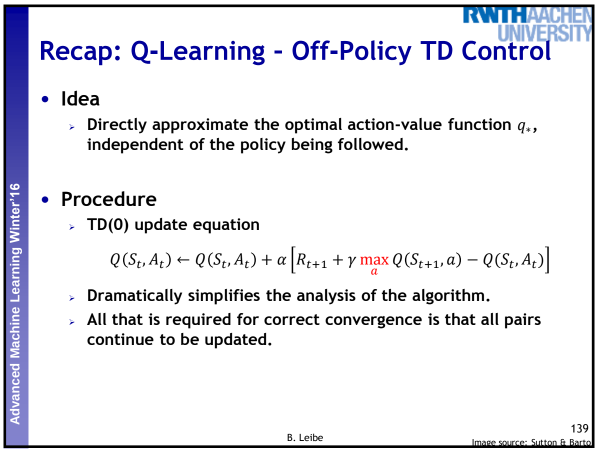
Recap: SARSA – On-Policy TD Control

- Idea
  - Turn the TD idea into a control method by always updating the policy to be greedy w.r.t. the current estimate
- Procedure
  - Estimate $q_\pi(s,a)$ for the current policy π and for all states s and actions a.
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

  - This rule is applied after every transition from a nonterminal state S_t.
  - It uses every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

$\Rightarrow$ *Hence, the name SARSA.*

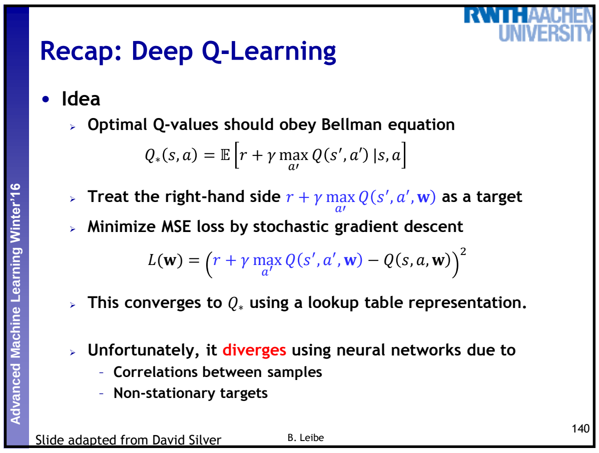


Recap: Q-Learning - Off-Policy TD Control

- Idea
  - Directly approximate the optimal action-value function q_*, independent of the policy being followed.
- Procedure
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

  - Dramatically simplifies the analysis of the algorithm.
  - All that is required for correct convergence is that all pairs continue to be updated.



Recap: Deep Q-Learning

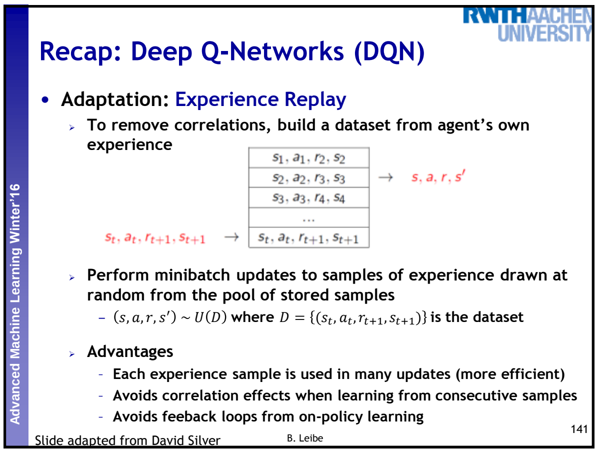
- Idea
  - Optimal Q-values should obey Bellman equation

$$Q_*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s', a') \,|s,a\right]$$

  - Treat the right-hand side $r + \gamma \max_{a'} Q(s', a', \mathbf{w})$ as a target
  - Minimize MSE loss by stochastic gradient descent

$$L(\mathbf{w}) = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

  - This converges to Q_* using a lookup table representation.
  - Unfortunately, it diverges using neural networks due to
    - Correlations between samples
    - Non-stationary targets

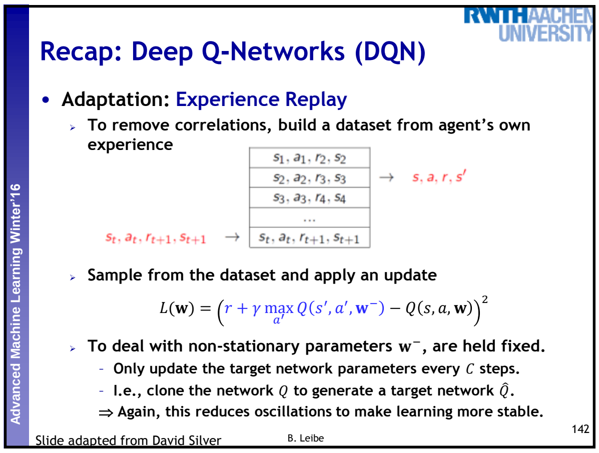


Recap: Deep Q-Networks (DQN)

- Adaptation: Experience Replay
  - To remove correlations, build a dataset from agent's own experience

$$\begin{tikzcd} & s_1, a_1, r_2, s_2 & \\ & s_2, a_2, r_3, s_3 \arrow[r] & s, a, r, s' \\ & s_3, a_3, r_4, s_4 & \\ & \dots & \\ s_t, a_t, r_{t+1}, s_{t+1} \arrow[r] & s_t, a_t, r_{t+1}, s_{t+1} & \end{tikzcd}$$

  - Perform minibatch updates to samples of experience drawn at random from the pool of stored samples
    - $(s,a,r,s') \sim U(D)$ where $D = \{(s_t, a_t, r_{t+1}, s_{t+1})\}$ is the dataset
  - Advantages
    - Each experience sample is used in many updates (more efficient)
    - Avoids correlation effects when learning from consecutive samples
    - Avoids feeback loops from on-policy learning



Recap: Deep Q-Networks (DQN)

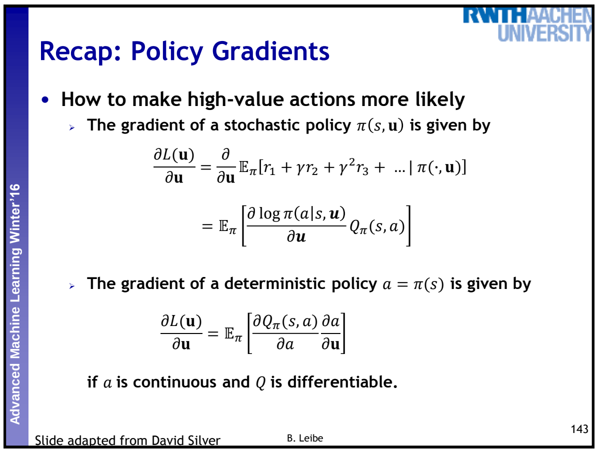
- Adaptation: Experience Replay
  - To remove correlations, build a dataset from agent's own experience

$$\begin{tikzcd} & s_1, a_1, r_2, s_2 & \\ & s_2, a_2, r_3, s_3 \arrow[r] & s, a, r, s' \\ & s_3, a_3, r_4, s_4 & \\ & \dots & \\ s_t, a_t, r_{t+1}, s_{t+1} \arrow[r] & s_t, a_t, r_{t+1}, s_{t+1} & \end{tikzcd}$$

  - Sample from the dataset and apply an update

$$L(\mathbf{w}) = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s,a,\mathbf{w})\right)^2$$

  - To deal with non-stationary parameters $\mathbf{w}^-$, are held fixed.
    - Only update the target network parameters every C steps.
    - I.e., clone the network Q to generate a target network $\hat{Q}$.
    - ⇒ Again, this reduces oscillations to make learning more stable.



Recap: Policy Gradients

- How to make high-value actions more likely
  - The gradient of a stochastic policy $\pi(s, \mathbf{u})$ is given by

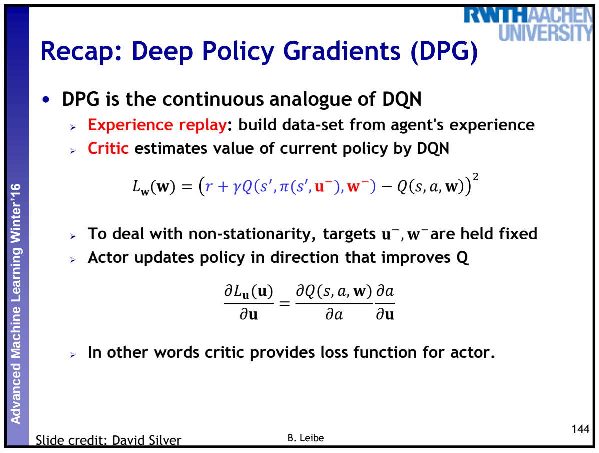
$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \mathbb{E}_\pi[r_1 + \gamma r_2 + \gamma^2 r_3 + \ \dots |\ \pi(\cdot, \mathbf{u})]$$

$$= \mathbb{E}_\pi\left[\frac{\partial \log \pi(a|s, \boldsymbol{u})}{\partial \boldsymbol{u}} Q_\pi(s,a)\right]$$

  - The gradient of a deterministic policy $a = \pi(s)$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}_\pi\left[\frac{\partial Q_\pi(s,a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}}\right]$$

  if a is continuous and Q is differentiable.



Recap: Deep Policy Gradients (DPG)

- DPG is the continuous analogue of DQN
  - Experience replay: build data-set from agent's experience
  - Critic estimates value of current policy by DQN

$$L_{\mathbf{w}}(\mathbf{w}) = \left(r + \gamma Q(s', \pi(s', \mathbf{u}^-), \mathbf{w}^-) - Q(s,a,\mathbf{w})\right)^2$$

  - To deal with non-stationarity, targets $\mathbf{u}^-, \mathbf{w}^-$ are held fixed
  - Actor updates policy in direction that improves Q

$$\frac{\partial L_{\mathbf{u}}(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial Q(s,a,\mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

  - In other words critic provides loss function for actor.

Any Questions?

So what can you do with all of this?

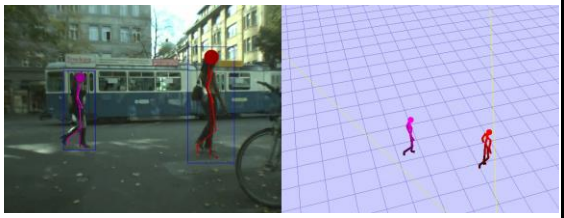
Robust Object Detection & Tracking

Applications for Driver Assistance Systems

Mobile Tracking in Densely Populated Settings

[D. Mitzel, B. Leibe, ECCV'12]

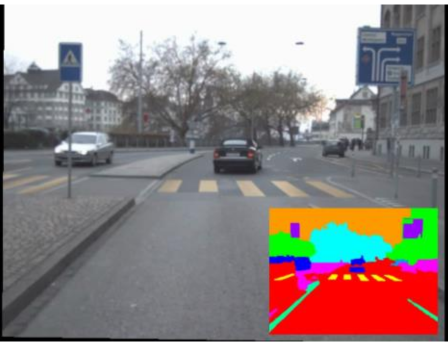
Articulated Multi-Person Tracking



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

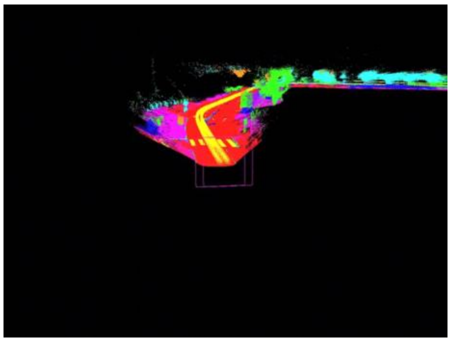
[Gammeter, Ess, Jaeggli, Schindler, Leibe, Van Gool, ECCV'08]

Semantic 2D-3D Scene Segmentation



[G. Floros, B. Leibe, CVPR'12]

Integrated 3D Point Cloud Labels



[G. Floros, B. Leibe, CVPR'12]

Any More Questions?

Good luck for the exam!