Day 10: Review

Introduction to Machine Learning Summer School
June 18, 2018 - June 29, 2018, Chicago

Instructor: Suriya Gunasekar, TTI Chicago

29 June 2018

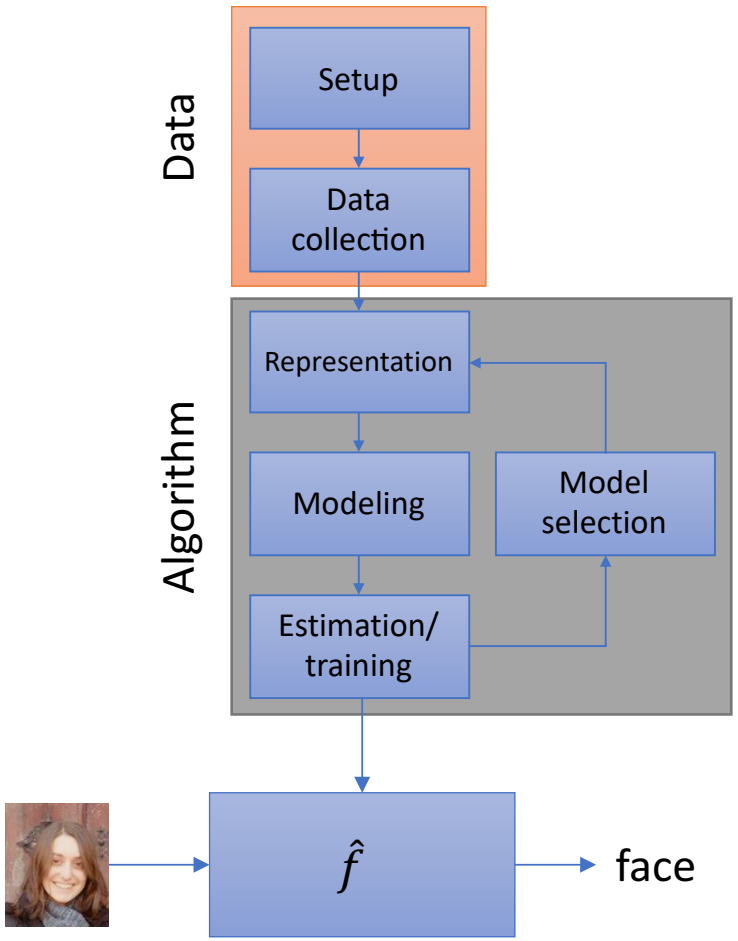
TOYOTA TECHNOLOGICAL INSTITUTE AT CHICAGO

THE UNIVERSITY OF CHICAGO

NSF

Review

Supervised learning – key questions



- Data: what kind of data can we get? how much data can we get?
- Model: what is the correct model for my data? – want to minimize the effort put into this question!
- Training: what resources - computation/memory - does the algorithm need to estimate the model $\hat{f}$?
- Testing: how well will $\hat{f}$ perform when deployed? what is the computational/memory requirement during deployment?

Linear regression

- Input $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, output $y \in \mathbb{R}$, want to learn $f: \mathcal{X} \to \mathbb{R}$
- Training data $S = \{(\boldsymbol{x}^{(i)}, y^{(i)}): i = 1,2, \ldots, N\}$
- Parameterize candidate $f: \mathcal{X} \to \mathbb{R}$ by linear functions,

$$\mathcal{H} = \{\boldsymbol{x} \to \boldsymbol{w}.\boldsymbol{x}: w \in \mathbb{R}^d\}$$

- Estimate $\boldsymbol{w}$ by minimizing loss on training data

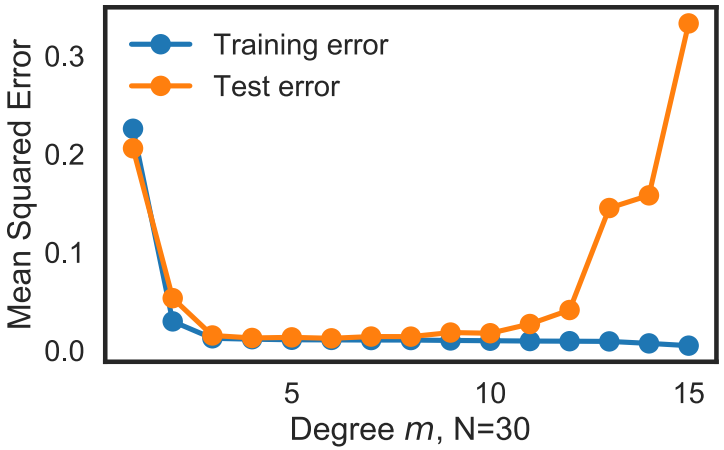
$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} J_S^{LS}(\boldsymbol{w}) := \sum_{i=1}^{N} \left(\boldsymbol{w}.\boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

  - $J_S^{LS}(\boldsymbol{w})$ is convex in $\boldsymbol{w}$ → minimize $J_S^{LS}(\boldsymbol{w})$ by setting gradient to 0
  - $\nabla_{\boldsymbol{w}} J_S^{LS}(\boldsymbol{w}) = \sum_{i=1}^{N} \left(\boldsymbol{w}.\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$
  - Closed form solution $\widehat{\boldsymbol{w}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{y}$

- Can get non-linear functions by mapping $\boldsymbol{x} \to \phi(\boldsymbol{x})$ and doing linear regression on $\phi(\boldsymbol{x})$

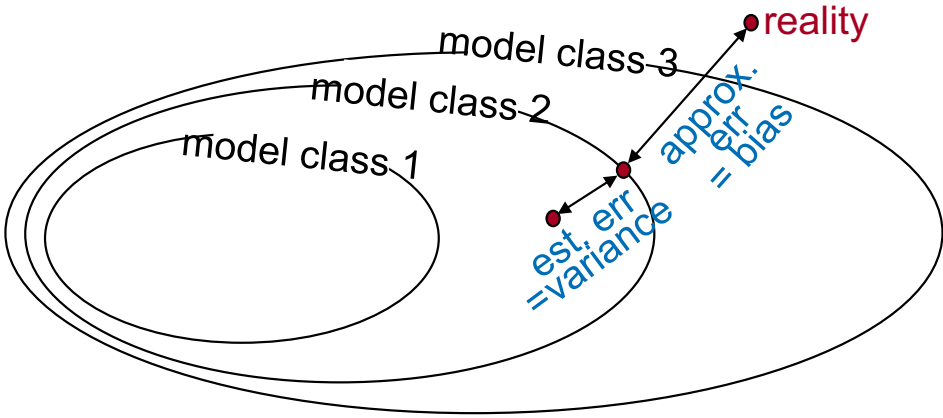
Overfitting

- For same amount of data, more complex models (e.g., higher degree polynomials) overfit more
- or need more data to fit more complex models
- complexity ≈ number of parameters



Model selection

- m model classes $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m\}$
- $S = S_{train} \cup S_{val} \cup S_{test}$
- Train on S_{train} to pick best $\hat{f}_r \in \mathcal{H}_r$
- Pick $\hat{f}^*$ based on validation loss on S_{val}
- Evaluate test loss $L_{S_{test}}(\hat{f}^*)$



Regularization

- Complexity of model class can also be controlled by norm of parameters – smaller range of values allowed
- Regularization for linear regression

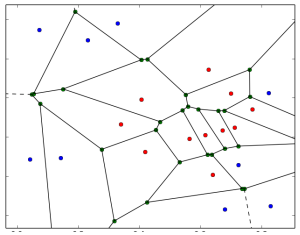
$$\underset{\boldsymbol{w}}{\operatorname{argmin}}\, J_S^{LS}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2$$

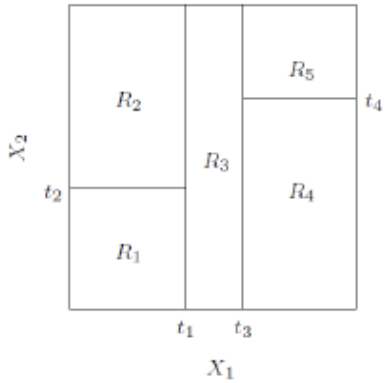
$$\underset{\boldsymbol{w}}{\operatorname{argmin}}\, J_S^{LS}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1$$

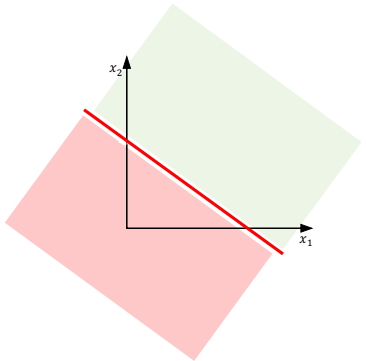
- Again do model selection to pick λ– using S_{val} or cross-validation

Classification

- Output $y \in \mathcal{Y}$ takes discrete set of values, e.g., $\mathcal{Y} = \{0,1\}$ or $\mathcal{Y} = \{-1,1\}$ or $\mathcal{Y} = \{spam, nospam\}$
  - Unlike regression, label-values do not have meaning
- Classifiers divide the space of input $\mathcal{X}$ (often $\mathbb{R}^d$) to "regions" where each region is assigned a label
- Non-parametric models
  - k-nearest neighbors – regions defined based on nearest neighbors
  - decision trees – structured rectangular regions
- Linear models – classifier regions are halfspaces



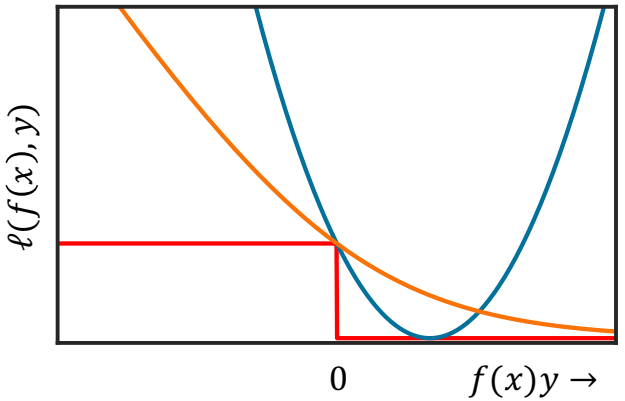




Classification – logistic regression

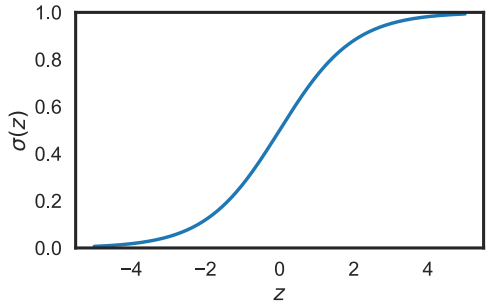
Logistic loss

$$\ell(f(\boldsymbol{x}), y) = \log\left(1 + \exp(-f(\boldsymbol{x})y)\right)$$



- $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1,1\}, S = \{(\boldsymbol{x}^{(i)}, y^{(i)}): i = 1,2,\ldots,N\}$
- Linear model $f(\boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}.\boldsymbol{x}$
- Output classifier $\hat{y}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}.\boldsymbol{x})$
- Empirical risk minimization:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\text{argmin}} \sum_i \log\left(1 + \exp\left(-\boldsymbol{w}.\boldsymbol{x}^{(i)} y^{(i)}\right)\right)$$



- Alternative, probabilistic formulation:

$$\Pr(y = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}.\boldsymbol{x})}$$

- Multi-class generalization: $\mathcal{Y} = \{1,2,\ldots,m\}$

$$\Pr(y|\boldsymbol{x}) = \frac{\exp(-\boldsymbol{w}_y.\boldsymbol{x})}{\sum_{y'} \exp(-\boldsymbol{w}_{y'}.\boldsymbol{x})}$$

- Can again get non-linear decision boundaries by mapping $\boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$

Classification – maximum margin classifier

Separable data

- Original formulation

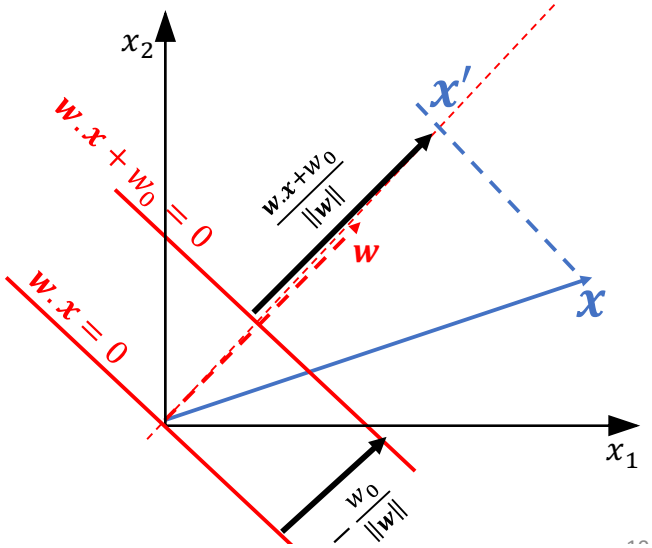
$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmax}} \quad \min_i \frac{y^{(i)}\boldsymbol{w}.\boldsymbol{x}^{(i)}}{\|\boldsymbol{w}\|}$$

- Fixing $\|\boldsymbol{w}\| = 1$

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \quad \min_i y^{(i)}\left(\boldsymbol{w}.\boldsymbol{x}^{(i)}\right) \quad \text{s.t.} \quad \|\boldsymbol{w}\| = 1$$

- Fixing $\min_i \; y^{(i)}\boldsymbol{w}.\boldsymbol{x}^{(i)} = 1$

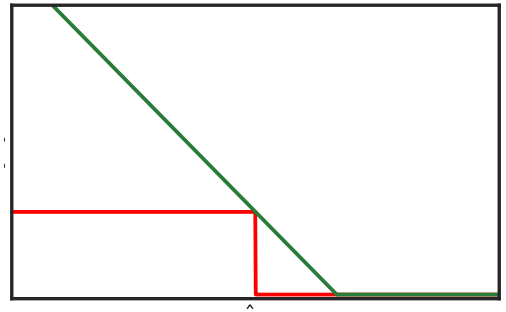
$$\widetilde{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 \quad \text{s.t.} \quad \forall i, y^{(i)}(\boldsymbol{w}.\boldsymbol{x}^{(i)}) \geq 1$$



Slack variables for non-separable data

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w},\{\xi_i\geq 0\}}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 + \lambda \sum_i \xi_i \quad \text{s.t.} \quad \forall i, y^{(i)}\left(\boldsymbol{w}.\boldsymbol{x}^{(i)}\right) \geq 1 - \xi_i$$

$$= \underset{\boldsymbol{w},\{\xi_i\geq 0\}}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 + \lambda \sum_i \max\left(0, 1 - y^{(i)}\left(\boldsymbol{w}.\boldsymbol{x}^{(i)}\right)\right)$$



Kernel trick

- Using representor theorem $\boldsymbol{w} = \sum_{i=1}^{N} \beta_i \boldsymbol{x}^{(i)}$

$$\min_{\boldsymbol{w}} \|\boldsymbol{w}\|^2 + \lambda \sum_i \max\left(0, 1 - y^{(i)}\, \boldsymbol{w}.\boldsymbol{x}^{(i)}\right)$$

$$\equiv \min_{\boldsymbol{\beta} \in \mathbb{R}^N} \boldsymbol{\beta}^\top \boldsymbol{G} \boldsymbol{\beta} + \lambda \sum_i \max\left(0, 1 - y^{(i)} (\boldsymbol{G}\boldsymbol{\beta})_i\right)$$

$\boldsymbol{G} \in \mathbb{R}^{N \times N}$ with $G_{ij} = \boldsymbol{x}^{(i)}.\boldsymbol{x}^{(j)}$ is called the gram matrix

- Optimization depends on $\boldsymbol{x}^{(i)}$ only through $G_{ij} = \boldsymbol{x}^{(i)}.\boldsymbol{x}^{(j)}$
- For prediction $\widehat{\boldsymbol{w}}.\boldsymbol{x} = \sum_i \beta_i\, \boldsymbol{x}^{(i)}.\boldsymbol{x}$, we again only need $\boldsymbol{x}^{(i)}.\boldsymbol{x}$
- Function $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}.\boldsymbol{x}'$ is called the Kernel
- When learning non-linear classifiers using feature transformations $\boldsymbol{x} \to \phi(\boldsymbol{x})$ and $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}.\phi(\boldsymbol{x})$
  - Classifier fully specified in terms of $K_\phi(\boldsymbol{x}, \boldsymbol{x}') = K(\phi(\boldsymbol{x}), \phi(\boldsymbol{x}'))$
  - $\phi(\boldsymbol{x})$ itself can be very very high dimensional (maybe even infinite dimensional)
    → e.g., polynomial kernels, RBF kernel

Optimization

- ERM+regularization optimization problem

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} J_S^{\lambda}(\boldsymbol{w}) := \sum_{i=1}^{N} \ell(\boldsymbol{w}.\phi(\boldsymbol{x}^{(i)}), y^{(i)}) + \lambda \|\boldsymbol{w}\|$$

- If $J_S^{\lambda}(\boldsymbol{w})$ is convex in $\boldsymbol{w}$, then $\widehat{\boldsymbol{w}}$ is optimum if and only if gradient at $\widehat{\boldsymbol{w}}$ is 0, i.e., $\nabla J_S^{\lambda}(\widehat{\boldsymbol{w}}) = 0$
- Gradient descent: start with initialization $\boldsymbol{w^0}$ and iteratively update
  - $\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \eta^t \nabla J_S^{\lambda}(\boldsymbol{w^t})$
  - where $\nabla J_S^{\lambda}(\boldsymbol{w^t}) = \sum_{\boldsymbol{i}} \nabla \ell\left(\boldsymbol{w^t}.\phi\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right) + \lambda \nabla \|\boldsymbol{w^t}\|$
- Stochastic gradient descent:
  - use gradients from only one example
  - $\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \eta^t \, \hat{\nabla}^{(i)} J_S^{\lambda}(\boldsymbol{w^t})$
  - where $\hat{\nabla}^{(i)} J_S^{\lambda}(\boldsymbol{w^t}) = \nabla \ell\left(\boldsymbol{w^t}.\phi\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right) + \frac{\lambda}{N} \nabla \|\boldsymbol{w^t}\|$ for a random sample $(\boldsymbol{x}^{(i)}, y^{(i)})$

Other classification models

- Optimal unrestricted predictor
  - Regression + squared loss→ $f^{**}(\boldsymbol{x}) = \mathbf{E}[y|\boldsymbol{x}]$
  - Classification + 0-1 loss → $\hat{y}^{**}(\boldsymbol{x}) = \text{argmax}_c \Pr(y = c|\boldsymbol{x})$
- Discriminative models: directly model $\Pr(y|\boldsymbol{x})$, e.g., logistic regression
- Generative models: model full joint distribution $\Pr(y, \boldsymbol{x}) = \Pr(\boldsymbol{x}|y) \Pr(y)$
- Why generative models?
  - One conditional might be simpler to model with prior knowledge, e.g., compare specifying $\Pr(\text{image}|\text{digit} = 1)$ vs $\Pr(\text{digit} = 1|\text{image})$
  - Naturally handles missing data
- Two examples of generative models
  - Naïve Bayes classifier – digit recognition, document classification
  - Hidden Markov model – POS tagging

Other classifiers

- Naïve Bayes classifier: with d features $x = [x_1, x_2, \dots, x_d]$ where each $x_1, x_2, \dots, x_d$ can take one of K values → $C\ K^d$ parameters
  - NB assumption: features are independent given class y → $C\ K\ d$ params.

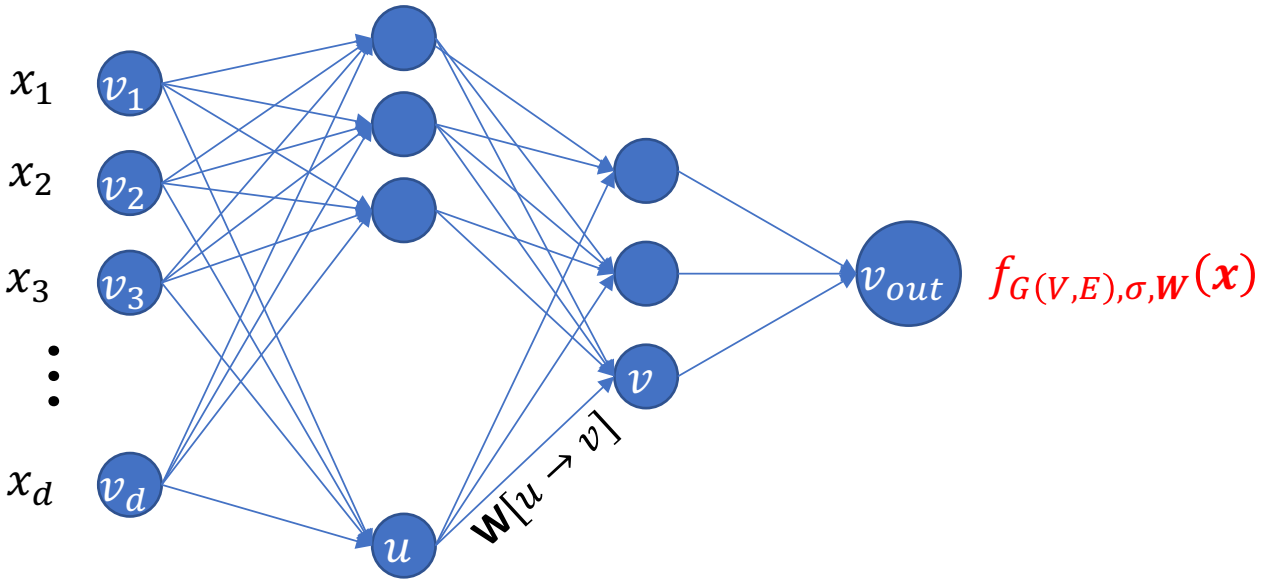
$$\Pr(x_1, x_2, \dots, x_d|y) = \Pr(x_1|y)\Pr(x_2|y)\dots\Pr(x_d|y) = \prod_{k=1}^{d} \Pr(x_k|y)$$

  - Training amounts to averaging samples across classes
- Hidden Markov model: variable length input/observations $\{x_1, x_2, \dots, x_m\}$ (e.g., words) and variable length output/state $\{y_1, y_2, \dots, y_m\}$ (e.g., tags)
  - HMM assumption: a) current state conditioned on immediate previous state is conditionally independent of all other variables, and (b) current observation conditioned on current state is conditionally independent of all other variables.

$$\Pr(x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_m) = \Pr(y_1)\Pr(x_1|y_1)\prod_{k=2}^{m}\Pr(y_k|y_{k-1})\Pr(y_k|x_k)$$

  - Parameters estimated using MLE dynamic programming

Feed-Forward Neural Networks

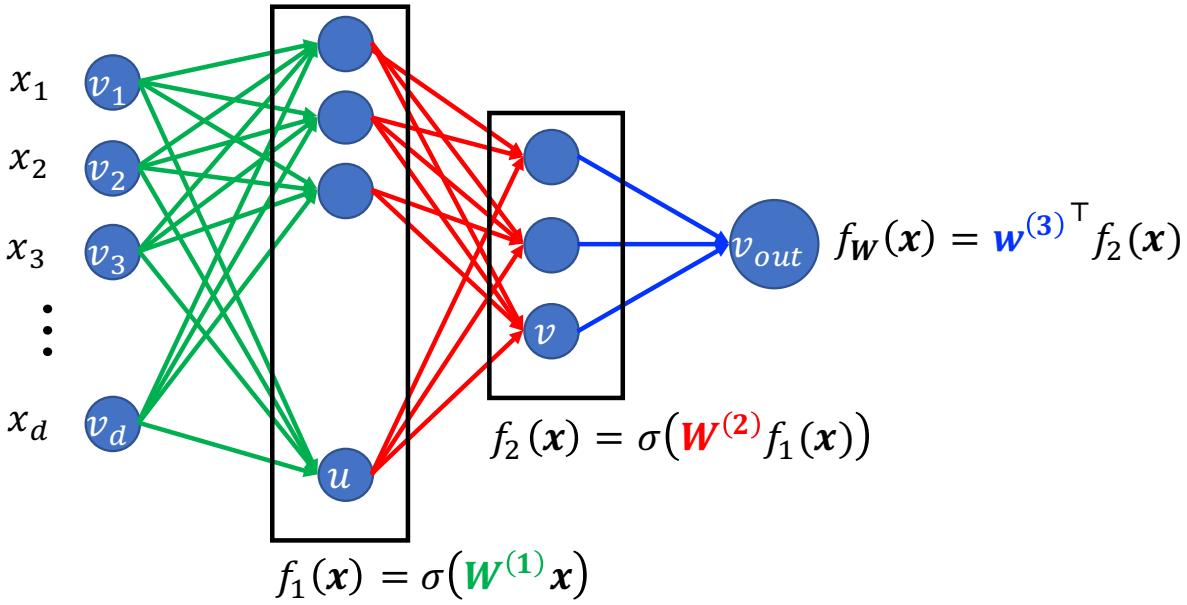


Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \ldots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \to v$ has weight $\boldsymbol{W}[u \to v]$
    - Pre-activation $a[v] = \sum_{u \to v \in E} \boldsymbol{W}[u \to v]\, o[u]$
    - Output value $o[v] = \sigma(\, a[v]\,)$
  - "Output Unit" $v_{out} \in V$, $f_W(\boldsymbol{x}) = a[v_{out}]$

Figure credit: Nati Srebro

Feed forward fully connected network

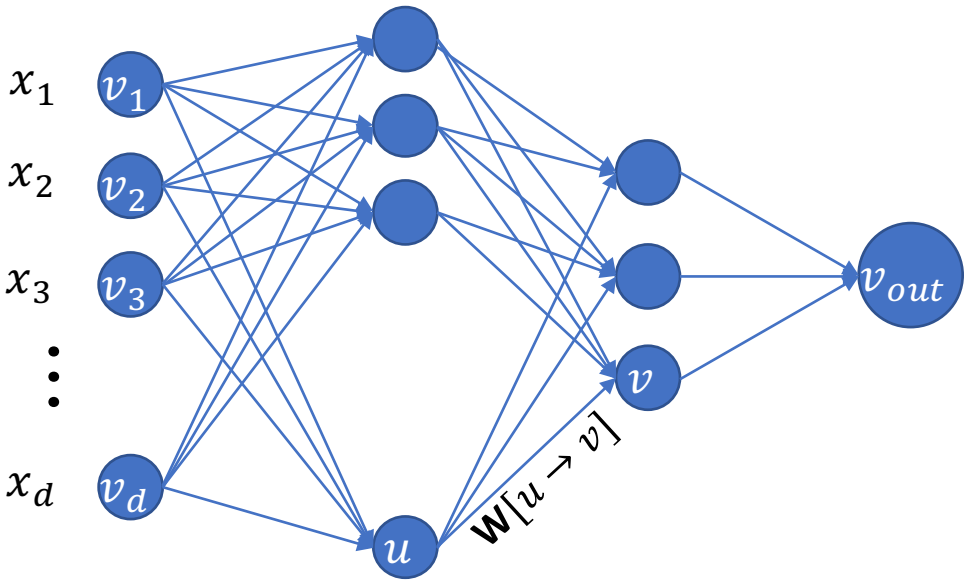


- L hidden layers with layer l havinb d_l hidden units
- Parameters:
  - for each intermediate layer $\boldsymbol{W}^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ where $d_0 = d$
  - final layer weights $\boldsymbol{w}^{(L+1)} \in \mathbb{R}^{d_L}$
- For 2-hidden layer $f_W(\boldsymbol{x}) = \boldsymbol{w}^{(3)\top}\sigma\left(\boldsymbol{W}^{(2)}\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)\right)$. More generally,

$$f_W(\boldsymbol{x}) = \boldsymbol{w}^{(L+1)\top}\sigma\left(\boldsymbol{W}^{(L-1)} \ldots \sigma\left(\boldsymbol{W}^{(2)}\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)\right)\right)$$

Back-Propagation

- Efficient calculation of $\nabla_{\boldsymbol{W}} \ell(f_{\boldsymbol{W}}(\boldsymbol{x}), y)$ using chain rule



$$a[v] = \sum_{u \to v \in E} \boldsymbol{W}^{(t)}[u \to v]\, o[u]$$

$$o[v] = \sigma(\, a[v]\,)$$

$$z[v_{out}] = \ell'(a[v_{out}], y)$$

$$z[u] = \sigma'(a[u]) \sum_{u \to v} \boldsymbol{W}^{(t)}[u \to v] z[v]$$

- Forward propagation: calculate activations $a[v]$ and outputs $o[v]$
- Backward propagation: calculate $z[v] \stackrel{\text{def}}{=} \frac{\partial \ell(f_{\boldsymbol{W}}(\boldsymbol{x}), y)}{\partial \boldsymbol{a}[v]}$
- Gradient descent update: using $\frac{\partial \ell(f_{\boldsymbol{W}}(\boldsymbol{x}), y)}{\partial W^{(t)}[u \to v]} = z[v] o[u]$

$$\boldsymbol{W}^{(t+1)}[u \to v] = \boldsymbol{W}^{(t+1)}[u \to v] - \eta^{(t)} \frac{\partial \ell(f_{\boldsymbol{W}}(\boldsymbol{x}), y)}{\partial W^{(t)}[u \to v]}$$

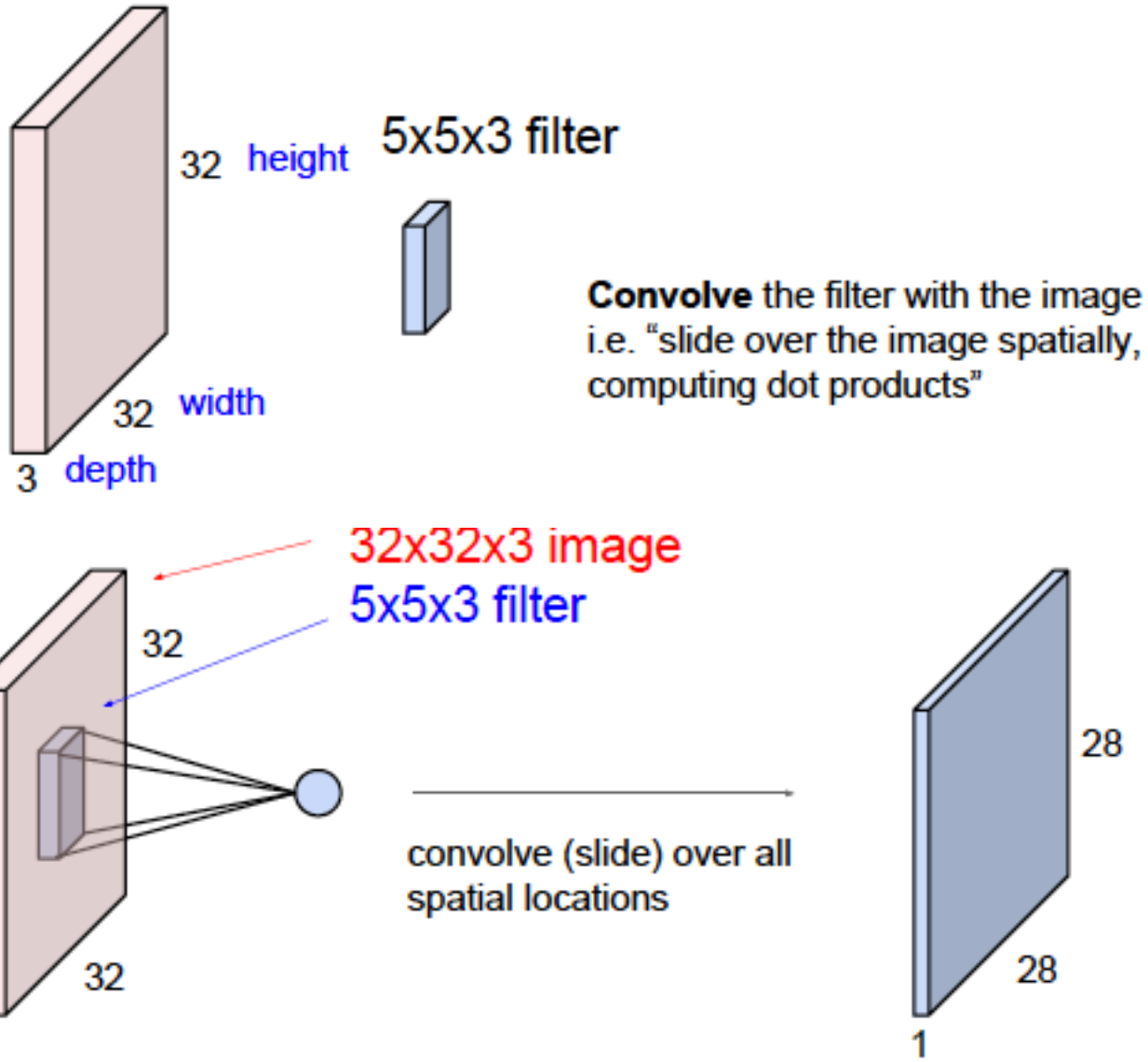
Optimization for NN training

- Check
  - Add gradCheck()
  - Randomly permute data for SGD sequence
- Choose activations to avoid
  - Gradient clipping
  - Gradient explosion
- SGD "knobs" in NN training
  - Initialization → Kaiming/Xavier, or warm start initialization.
  - Step size/learning rate → very important to tune based on training/ validation loss
  - SGD variants
    - Momentum for SGD → usually added with SGD (default parameter momentum=0.9 often works well)
    - Adaptive variants of SGD → common alternative to SGD+momentum is Adam with $\beta_2 \gg \beta_1, e.g., \beta_2 = 0.999, \beta_1 = 0.9$
  - Mini-batch SGD → ~ 128 common
  - Batch normalization → use batch normalization

Regularization in NN

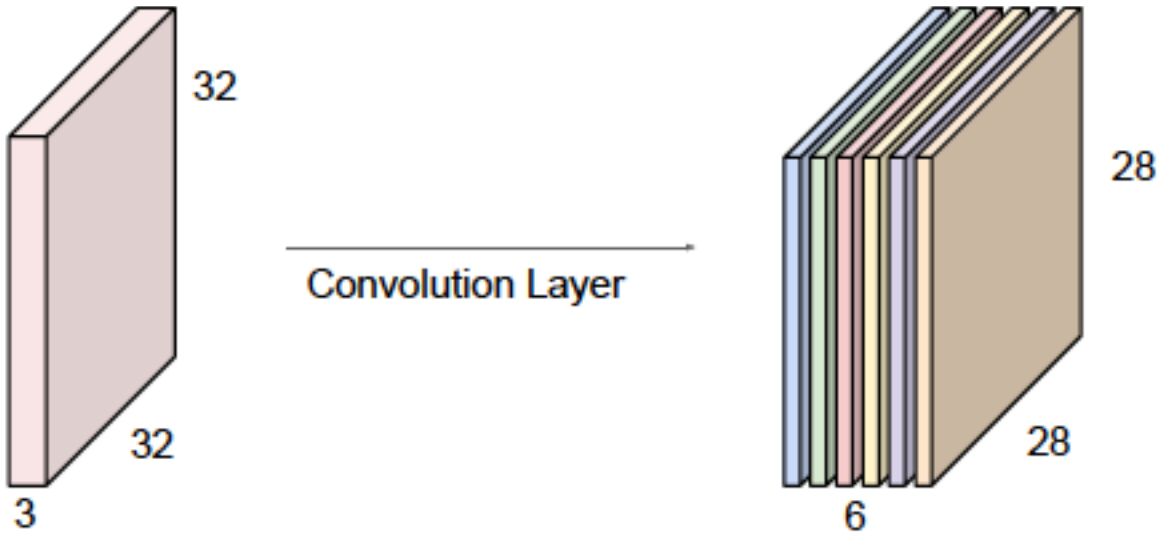
- Explicit regularization
  - Data augmentation → Augment training data with known invariances/noise models → very effective
    - think of what is the right data augmentation for your problem
  - Weight decay → $\arg\min_{\boldsymbol{W}} L_S(f_{\boldsymbol{W}}) + \frac{\lambda}{2}\|\boldsymbol{W}\|^2$
    - tune step sizes/ λ parameter
  - Dropout → Randomly (temporarily) remove p fraction of the units in each step of SGD → usually very useful
  - Early stopping
- Choice of architecture affects validation performance/generalization!
- Many optimization choices also affect validation performance—unlike convex optimization problems with a unique global minimum, where optimization algorithm only changes the speed/computation of training → Not well understood phenomenon
  - Keep in mind while making choices in previous slides

NN architectures – CNNs



Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/

NN architectures – CNNs

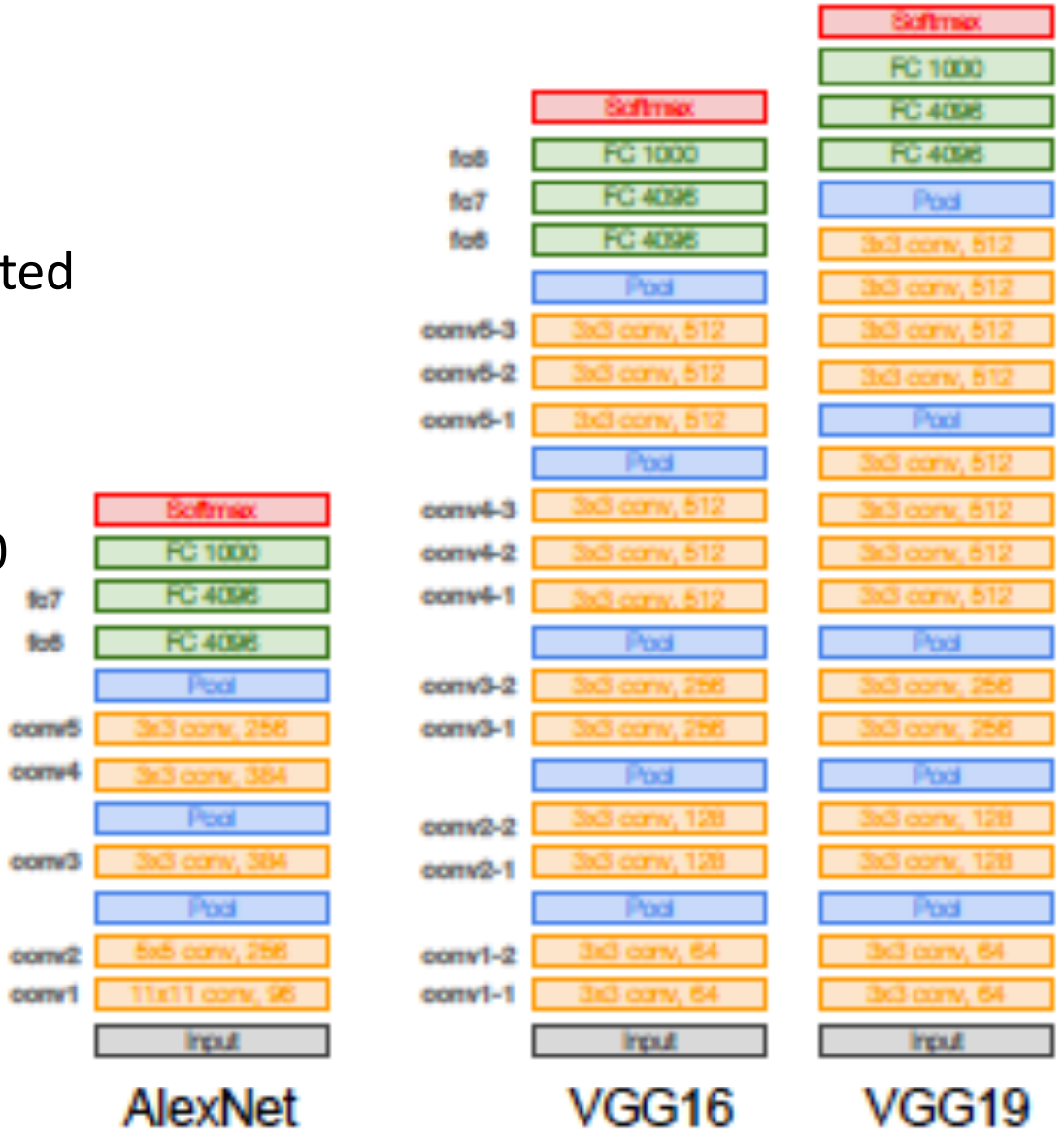


- Each convolution layer has input of size $W_{in} \times H_{in} \times D_{in}$
- Hyperparameters: Number of filters D_{out}; Size of filters $K_1 \times K_2$; Stride S; Zero padding P
- Parameters: $K_1 \times K_2 \times D_{in} \times D_{out}$
- Output: $W_{out} \times H_{out} \times D_{out}$ where
  - $W_{out} = (W\_in - K\_1 - 2P)/S + 1$
  - $H_{out} = (H\_in - K\_2 - 2P)/S + 1$

Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/

CNNs

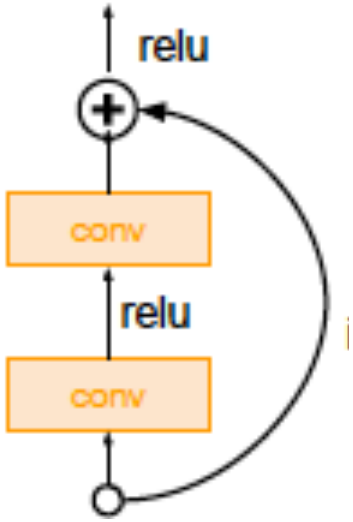
- Typical layers
  - Convolution+ReLU
  - Max-pooling
  - Final few fully connected layers
- Common datasets
  - MNIST (small)
  - CIFAR-10 & CIFAR-100
  - ImageNet
  - MS COCO
- Tip: Try warm-start initialization from models pre-trained on imageNet

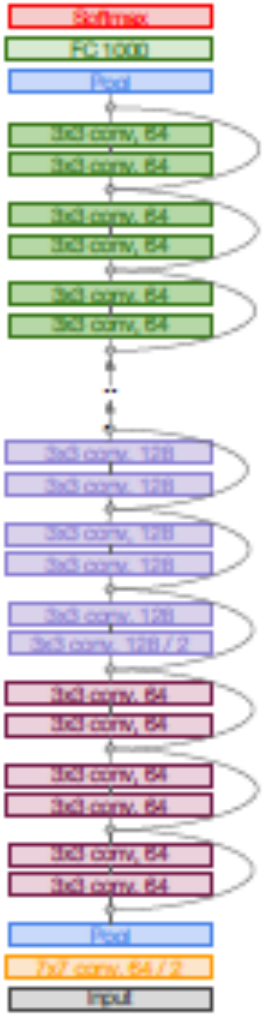


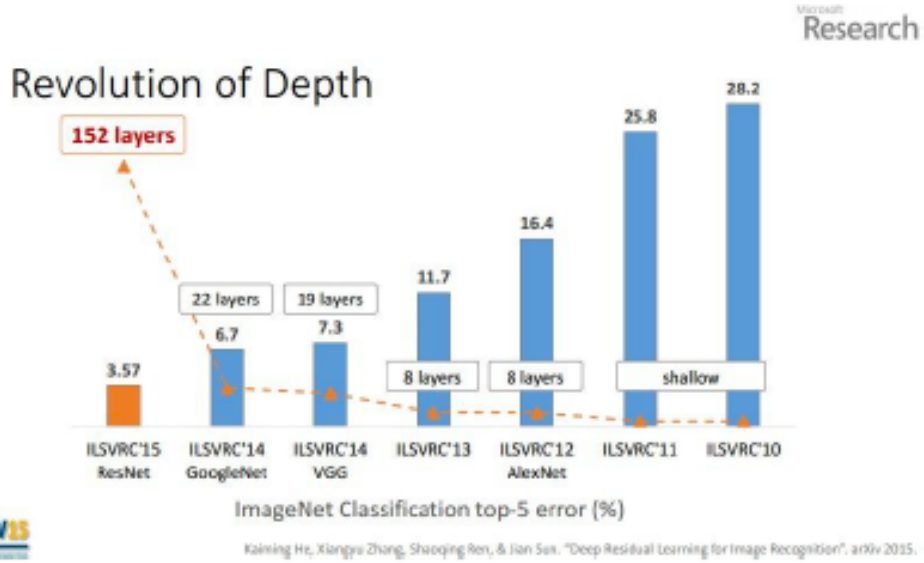
Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/

Residual Networks

- $h_l = h_{l-1} + ReLU(\text{Conv}(h_{l-1}))$
- Avoids gradient saturation
- Enabled training of really deep networks
  - Typical choice is 152 layers
  - 1000+ layers have been trained with ResNets
- Can also extend for other architectures like FCNs/RNNs





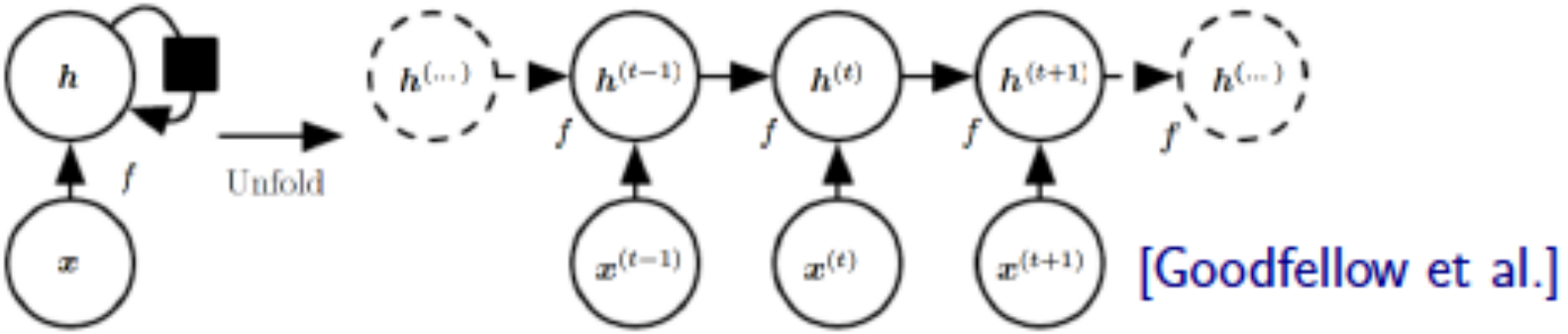


Total depths of 34, 50, 101, or 152 layers for ImageNet

Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/

NN architectures – RNNs

- Input: each example is a sequence
$$[\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n} \in \mathbb{R}^d]$$
- Labels: can be single label $\boldsymbol{y}$ or another sequence
- Output of RNNs: $[h_1, h_2, \dots, h_n \in \mathbb{R}^{d'}]$
- Note: this is just one example, the training dataset will contain many such examples
- RNN model: For $i = 1,2, \dots, n$
$$\boldsymbol{h_i} = \tanh(\boldsymbol{W x_i} + \boldsymbol{V h_{i-1}})$$



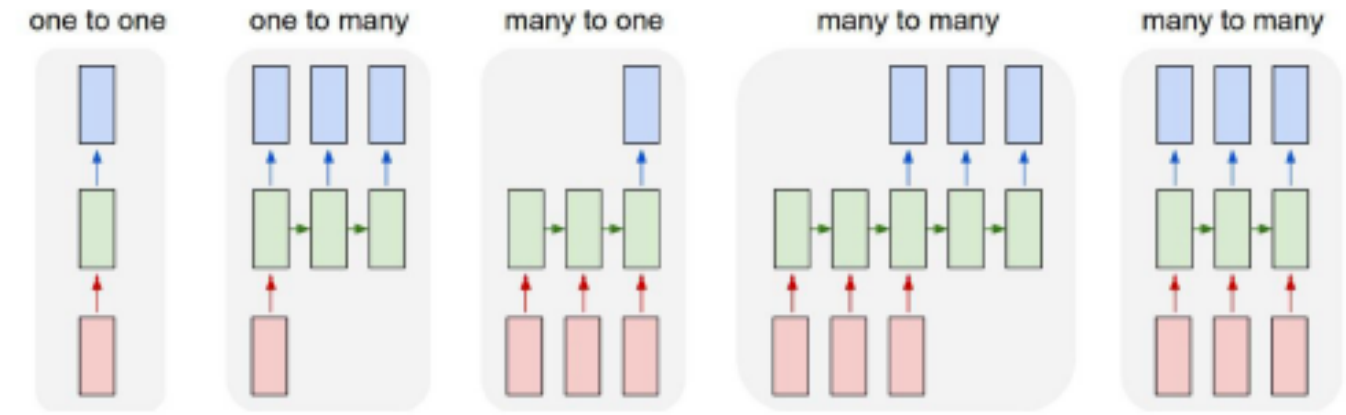
[Goodfellow et al.]

NN architectures – RNNs

- RNN model: For $i = 1,2,\dots,n$

$$\boldsymbol{h_i} = \tanh(\boldsymbol{W x_i} + \boldsymbol{V h_{i-1}})$$

- $\boldsymbol{h_n} = \tanh\left(\boldsymbol{W x_n} + \boldsymbol{V}\tanh\left(\boldsymbol{W x_{n-1}} + V(\dots + \tanh(\boldsymbol{W x_1} + \boldsymbol{V h_0}))\right)\right)$
  - Like fully connected networks, but parameters are reused
- loss $\ell([\boldsymbol{h_1}, \boldsymbol{h_2}, \dots, \boldsymbol{h_n}], \boldsymbol{y})$



[A. Karpathy]

- Can create deeper networks by using $[h_1, h_2, \dots, h_n \in \mathbb{R}^{d'}]$ as sequential input to next layer

NN Architectures LSTMs

- RNN produces a sequence of output vectors

$$x_1 \dots x_N \longrightarrow h_1 \dots h_N$$

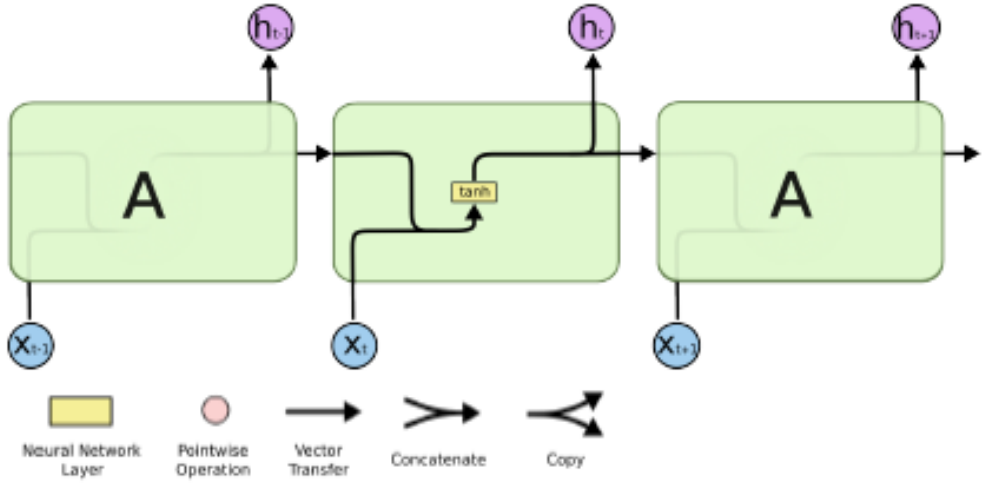
- LSTM produces "memory cell vectors" along with output

$$x_1 \dots x_N \longrightarrow c_1 \dots c_N, h_1 \dots h_N$$

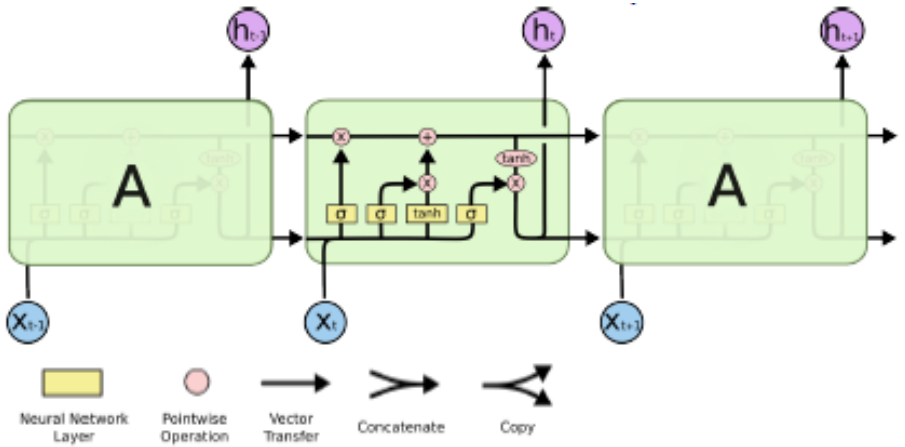
- These $c_1 \dots c_N$ enable the network to keep or drop information from previous states.

NN Architectures LSTMs

- Simple RNNs

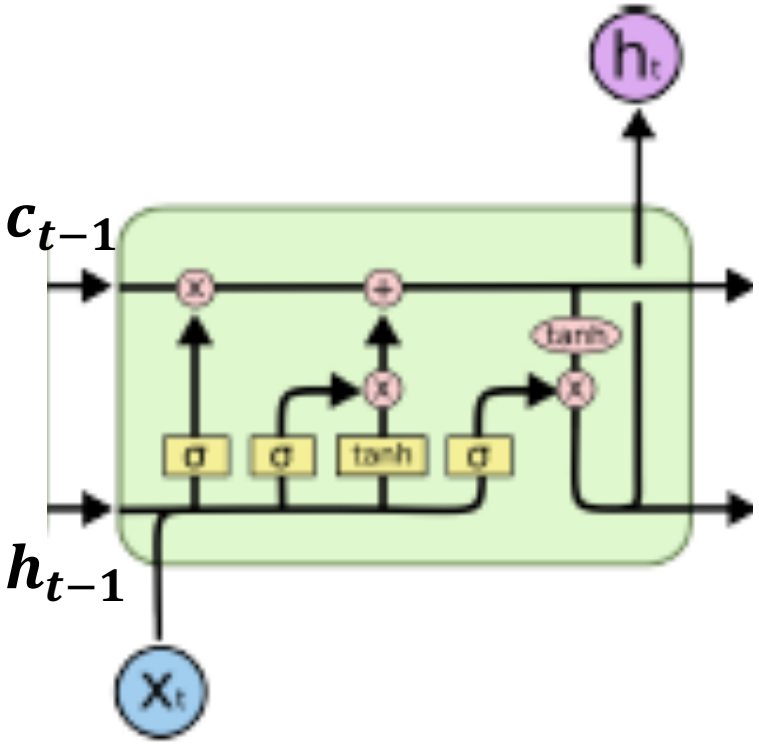


- In LSTMs, each time frame associated with a complex cell

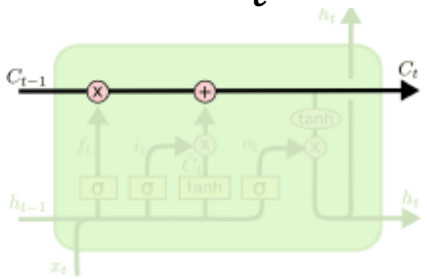


Figures taken from blog post on LSTMs by C. Olah

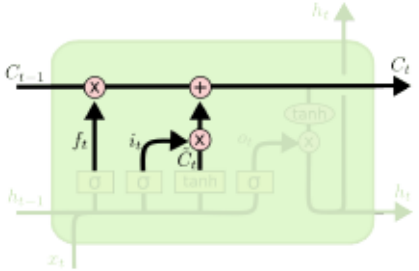
NN Architecture LSTMs



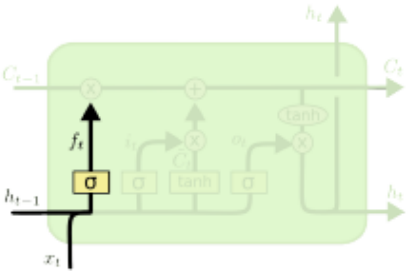
- Cell state c_t



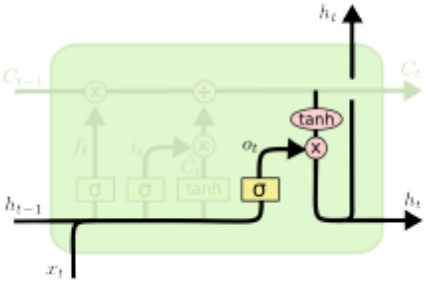
- Cell state update



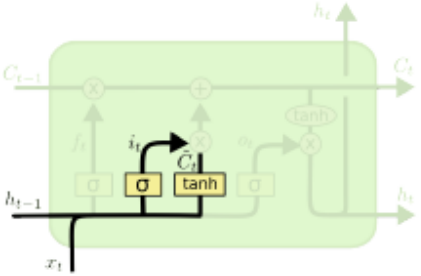
- Forget gate q_t



- Output gate



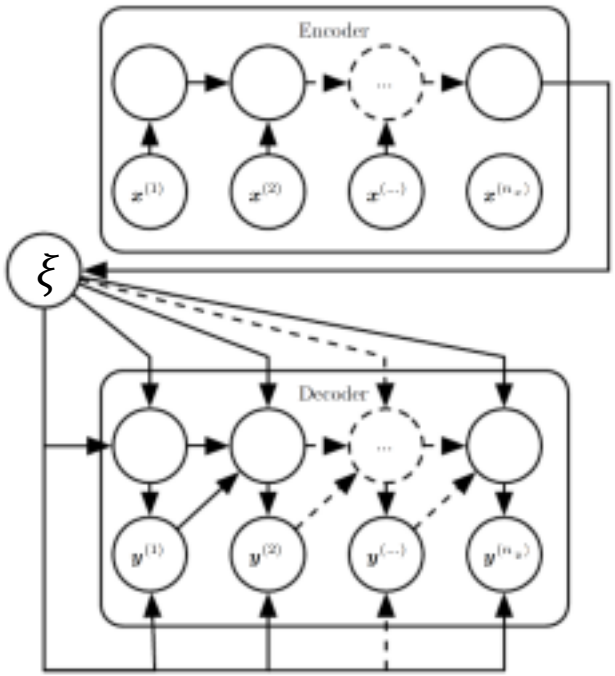
- Input gate



See lecture slides
for exact equations

Figures taken from blog post on LSTMs by C. Olah

NN architectures – encoder-decoder



See lecture slides for exact equations

- Encoder RNN: First encodes in the input and captures the context in ξ
- Decoder RNN: decodes the output from ξ
- Decoder with attention: instead of relying just on final context ξ, use a linear combination of all the hidden states in the encoder (not depicted in figure)

Slide credit: Greg Shaknarovich

Ensembles

- Reduce bias:
  - build ensemble of low-variance, high-bias predictors sequentially to reduce bias
  - AdaBoost: binary classication, exponential surrogate loss
- Reduce variance:
  - build ensemble of high-variance, low-bias predictors in parallel and use randomness and averaging to reduce variance
  - random forests, bagging
- Problems
  - Computationally expensive (train and test time)
  - Often loose interpretability

Bagging: Bootstrap aggregation

Averaging independent models reduces variance without increasing bias.

- But we don't have independent datasets!
  - Instead take repeated bootstrap samples from training set S
- Bootstrap sampling: Given dataset $S = \{(x^{(i)}, y^{(i)}): i = 1,2, \dots, N\}$, create S' by drawing N examples at random **with replacement** from S
- Bagging:
  - Create M bootstrap datasets $S_1, S_2, \dots, S_M$
  - Train distinct models $f_m: \mathcal{X} \to \mathcal{Y}$ by training only on S_m
  - Output final predictor $F(x) = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$ (for regression) or $F(x) = \text{majority}(f_m(x))$ (for classification)

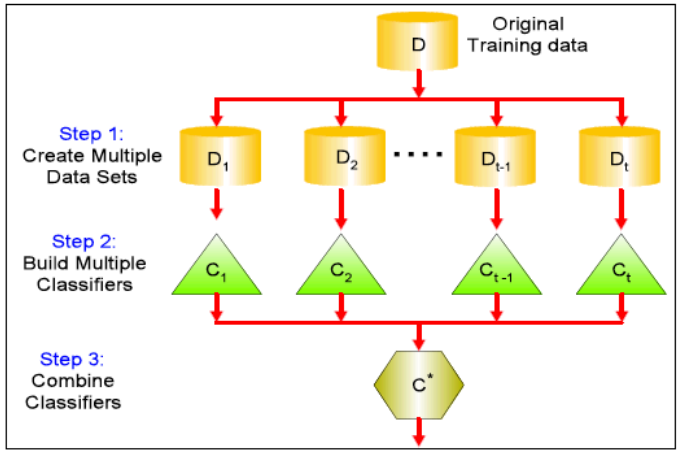
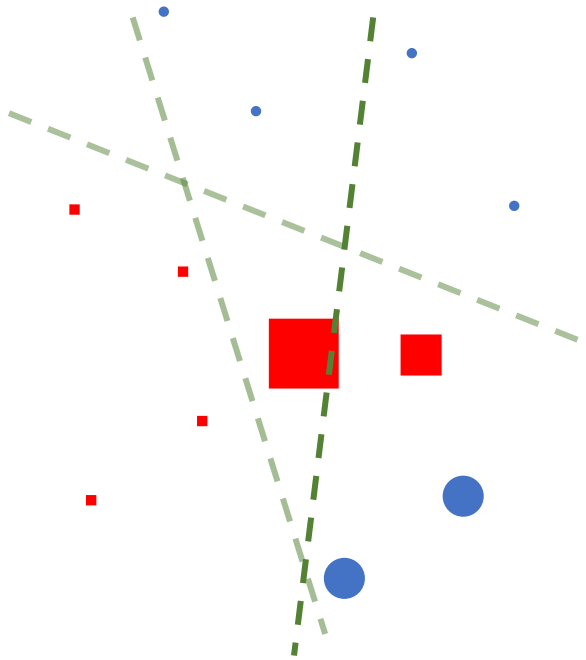


Figure credit: David Sontag

Adaboost

Training data $S = \{(x^{(i)}, y^{(i)}): i = 1,2, \dots, N\}$

- Maintain weights $W_i^{(t)}$ for each example $(x^{(i)}, y^{(i)})$, initially all $W_i^{(1)} = \frac{1}{N}$
- For $t = 1,2, \dots, \mathrm{T}$
  - Normalize weights $D_i^{(t)} = \frac{W_i^{(t)}}{\sum_i W_i^{(t)}}$
  - Pick a classifier f_t has better than 0.5 weighted loss
    $\epsilon_t = \sum_{i=1}^{N} D_i^{(t)} \ell^{01}(f_t(x^{(i)}), y^{(i)})$
  - Set $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  - Update weights
    $W_i^{(t+1)} = W_i^{(t)} \exp\left(-\alpha_t y^{(i)} f_t(x^{(i)})\right)$

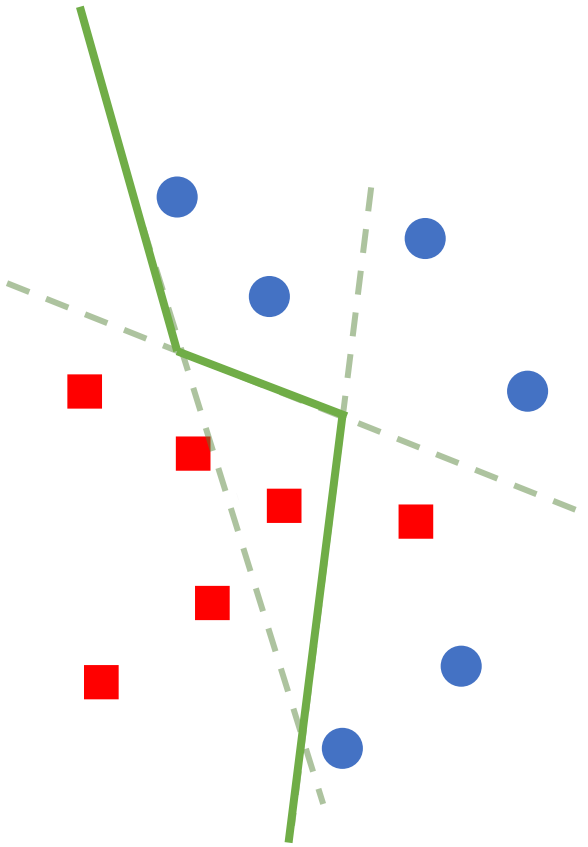


Example credit: Greg Shaknarovich

Adaboost

Training data $S = \{(x^{(i)}, y^{(i)}): i = 1,2, \dots, N\}$

- Maintain weights $W_i^{(t)}$ for each example $(x^{(i)}, y^{(i)})$, initially all $W_i^{(1)} = \frac{1}{N}$
- For $t = 1,2, \dots, \mathrm{T}$
  - Normalize weights $D_i^{(t)} = \frac{W_i^{(t)}}{\sum_i W_i^{(t)}}$
  - Pick a classifier f_t has better than 0.5 weighted loss
    $\epsilon_t = \sum_{i=1}^{N} D_i^{(t)} \ell^{01}(f_t(x^{(i)}), y^{(i)})$
  - Set $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  - Update weights
    $W_i^{(t+1)} = W_i^{(t)} \exp\left(-\alpha_t y^{(i)} f_t(x^{(i)})\right)$
- Output strong classifier $F_T(x) = \mathrm{sign}(\sum_t \alpha_t f_t(x))$



Example credit: Greg Shaknarovich

Supervised learning summary

- Linear regression
- Classification
  - Logistic regression
  - Maximum margin classifiers, kernel trick
  - Generative models: Naïve Bayes, HMMs
  - Neural networks
- Ensemble methods
- Main concepts:
  - Detecting and avoiding overfitting and the tradeoff between bias and complexity
  - Learning parameters using empirical risk minimization (ERM) plus regularization
  - Optimization techniques: specially (stochastic) gradient descent → for both convex and non-convex problems

Unsupervised learning

- Unsupervised learning: Requires data $x \in \mathcal{X}$, but no labels
- Goal?: Compact representation of the data by detecting patterns
  - e.g. Group emails by topic
- Useful when we don't know what we are looking for
  - makes evaluation tricky
- Applications in visualization, exploratory data analysis, semi-supervised learning

Figure credit: David Sontag

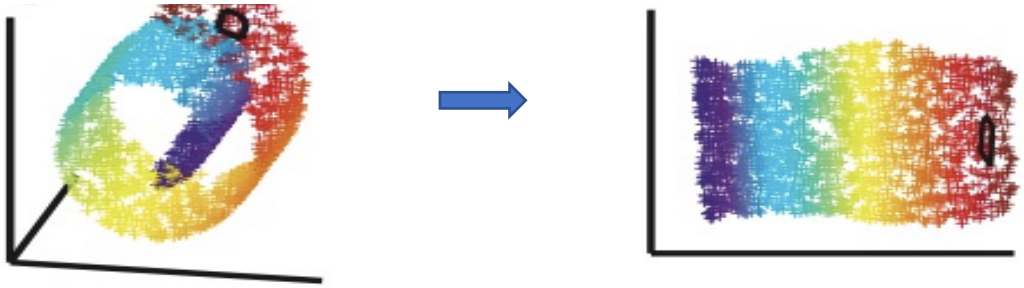
Linear dimensionality reduction

- Problem: Given high dimensional feature $\boldsymbol{x} = [x_1, x_2, \dots, x_d] \in \mathbb{R}^d$ find transformations $z(\boldsymbol{x}) = [z_1(\boldsymbol{x}), z_2(\boldsymbol{x}), \dots, z_k(\boldsymbol{x})] \in \mathbb{R}^k$ so that "almost all useful information" about $\boldsymbol{x}$ is retained in $z(\boldsymbol{x})$
  - Learn $z(\boldsymbol{x})$ from dataset of examples $S = \{\boldsymbol{x^{(i)}} \in \mathbb{R}^d : i = 1,2, \dots, N\}$
- Linear dimensionality reduction: $z(\boldsymbol{x})$ restricted to be a linear function
- PCA: given data $x \in \mathbb{R}^d$, find $\mathrm{U} \in \mathbb{R}^{k \times d}$ to minimize

$$\min_{\mathrm{U}} \sum_i \left\| \boldsymbol{U^\top U x^{(i)}} - \boldsymbol{x^{(i)}} \right\|_2^2 \; s.t. \; \boldsymbol{U U^\top} = \boldsymbol{I}$$

  - solution given by eigenvalue decomposition of $\hat{\Sigma}_{xx} = \frac{1}{N} \sum_{i=1}^N \boldsymbol{x^{(i)}} \boldsymbol{x^{(i)}}^\top$
  - finds directions of maximum variation in data
  - check: make sure to center the data so that each feature has zero mean
- Can get non-linear embedding by doing PCA on $\phi(\boldsymbol{x})$ → Kernel PCA

Non linear dimensionality reduction

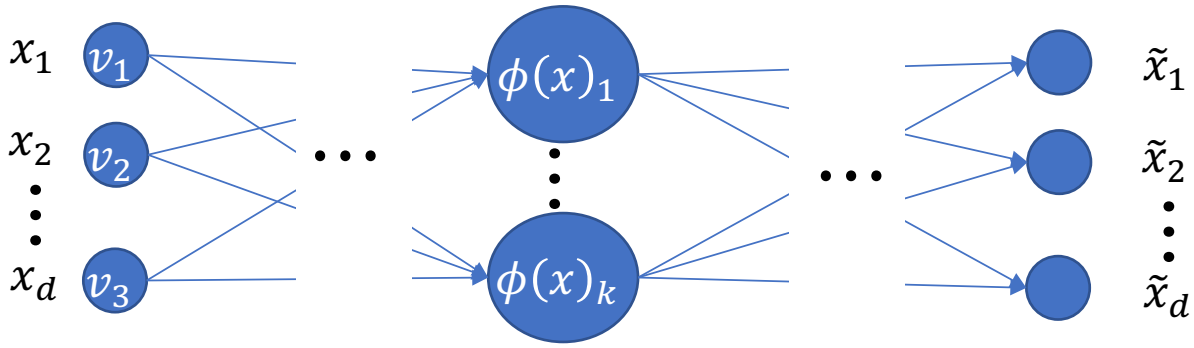


- **Isomap:** Neighborhood of points represented using the kNN-graph with weights proportional to distance between the points
  - geodesic distance $d(x, x') =$ length of shortest path in the graph
  - Use any shortest path algorithm can be used to construct a matrix $M \in \mathbb{R}^{N \times N}$ with $M_{ij} = d\left(x^{(i)}, x^{(j)}\right)$ for all $x^{(i)}, x^{(j)} \in S$
  - MDS: Find a (low dimensional) embedding $z(x)$ of x so that geodesic distance match the Euclidean distance in the transformed space

    $\min_{z} \sum_{i,j \in [N]} \left(\left\| z\left(x^{(i)}\right) - z\left(x^{(j)}\right) \right\| - M_{ij} \right)^2$
- Works well for small scale problems

Non linear dimensionality reduction

- **Autoencoders:**



- $\phi(x) = f_{\boldsymbol{W_1}}(\boldsymbol{x})$
- $\tilde{\boldsymbol{x}} = f_{\boldsymbol{W_2}}(\phi(\boldsymbol{x}))$
- some loss $\ell(\tilde{x}, x)$

$$\widehat{W}_1, \widehat{W}_2 = \min_{W_1, W_2} \sum_{i=1}^{N} \ell\left(f_{\boldsymbol{W_2}}\left(f_{\boldsymbol{W_1}}\left(\boldsymbol{x}^{(i)}\right)\right), \boldsymbol{x}^{(i)}\right)$$

- learn using SGD with backpropagation

MLE of latent variable models

- Generative model:
  - Observed variables $x \in \mathcal{X}$
  - Latent variables $z \in \mathcal{Z}$
  - Probabilistic generative model parameterized by parameters Φ is

$$P_\Phi(x, z) = P_\Phi(z)P_\Phi(x|z)$$

    - For each example x, first sample $z \sim P_\Phi(z)$, then sample $\mathrm{x} \sim P_\Phi(x|z)$
    - Note: we never see z, appears only in generative assumption
    - Latent variables allows for easier specification of $\Pr(x)$

- MLE estimation: given dataset $S = \{x^{(i)} : i = 1,2, \dots, N\}$

$$\Phi^* = \underset{\Phi}{\operatorname{argmax}} \sum_{i=1}^{N} \log \Pr\left(x^{(i)}\right)$$

$$\Phi^* = \underset{\Phi}{\operatorname{argmax}} \sum_{i=1}^{N} \left(\log \sum_{z \in \mathcal{Z}} P_\Phi\left(x^{(i)}, z\right) \right)$$

Expectation Maximization high-level algo

$$\Phi^* = \underset{\Phi}{\operatorname{argmax}} \sum_{i=1}^{N} \left(\log \sum_{z^{(i)} \in \mathcal{Z}} P_\Phi\left(x^{(i)}, z^{(i)}\right) \right)$$

- Main idea: Say we are looking at problems where the above optimization is "easy" if we "know" $z^{(i)}$! but we don't know $z^{(i)}$.
  - Fix-alternate between estimating $z^{(i)}$ and Φ
- Start with some estimate $\Phi^{(0)}$ of parameters we want to estimate:
  - **Expectation step (E-step):** Compute an expectation to "fill in" missing variables $z^{(i)}$ assuming our current estimate of parameter $\Phi^{(t)}$ is correct.
  - **Maximization step (M-step):** Assuming our estimates $z^{(i)}$ from above E-step is correct, solve maximization to estimate $\Phi^{(t+1)}$
    - Recall that if we pretend to know $z^{(i)}$, the optimization is "easy"
- No magic! still optimizing hard non-convex function with lots of local optima
  - not guaranteed to converge to global optima and
  - but often also give good enough solutions even if they are local optima

EM algorithm

$$\Phi^* = \underset{\Phi}{\operatorname{argmax}} \sum_{i=1}^{N} \left(\log \sum_{z^{(i)} \in \mathcal{Z}} P_\Phi\left(x^{(i)}, z^{(i)}\right) \right)$$

- **Expectation step (E-step):** "fill in" missing variables $z^{(i)}$ assuming our current estimate of $\Phi^{(t)}$ is correct.
- How to do this?
  - Specify an auxiliary model $P_\Psi(z|x)$
  - Instead of filling in one value of z this gives a distribution over $z|x$
  - Idea: find a way to estimate Ψ under this model! If the model is correct, we in turn get a good estimate of z

$$ELBO_x(\Phi, \Psi) = \mathbb{E}_{\boldsymbol{z} \sim P_\Psi(.|x)} \log P_\Phi(x|z) + D_{KL}\left(P_\Psi(z|x) || P_\Phi(z)\right)$$

- For any Ψ, $ELBO_x(\Phi, \Psi) \leq \log(\sum_{z \in \mathcal{Z}} P_\Phi(x, z))$ and maximized when $P_\Psi(z|x) = P_\Phi(z) = \sum_{x \in \mathcal{X}} P_\Phi(z, x)$

EM algorithm

$$\Phi^* = \underset{\Phi}{\operatorname{argmax}} \sum_{i=1}^{N} \left(\log \sum_{z^{(i)} \in \mathcal{Z}} P_\Phi\left(x^{(i)}, z^{(i)}\right) \right)$$

- Specify joint models $P_\Phi(z, x)$ and auxiliary model $P_\Psi(z|x)$
- Initialize $\Phi^{(0)}, \Psi^{(0)}$
- For $t = 1,2, \ldots,$
  - $\Psi^{(t)} = \max_\Psi ELBO(\Phi^{(t-1)}, \Psi)$
  - $\Phi^{(t)} = \max_\Phi ELBO\left(\Phi, \Psi^{(t)}\right)$

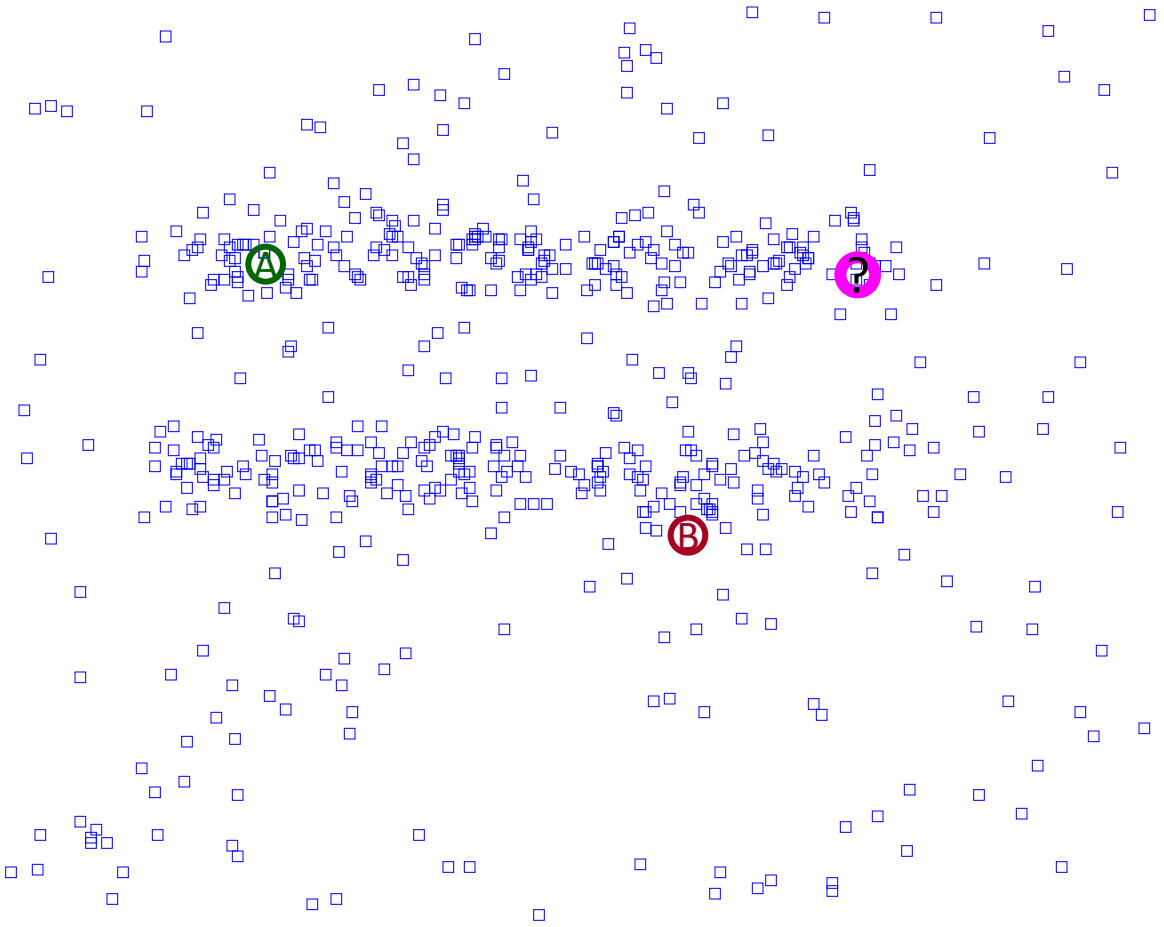
Unsupervised learning – clustering

- k-means clustering
  - hard clustering
  - Initialize cluster centroid
  - Alternatingly
    - Compute cluster memberships (hard membershipts)
    - Update cluster centroids
- Gaussian mixture models
  - soft clustering: cluster membership is a probability vector $\pi \in \Delta^{k-1}$ over k mixture components and mixture components are Gaussians with means $\mu_1, \mu_2, \ldots, \mu_k$
  - EM algorithm alternatingly:
    - Computes soft cluster memberships $\pi^{(t)}$
    - Updates mixture component means $\mu_1^{(t)}, \mu_2^{(t)}, \ldots, \mu_k^{(t)}$
- Main modeling in specifying distance or learning representation

Topics not covered

Semi-Supervised Learning

Using unlabeled data to help predictions



Slide credit: Nati Srebro

Active Learning

- Training data is randomly drawn/fixed
- What if we could explicitly ask for specific training data?
  - E.g. we could query an expert (a teacher, a user, someone on mechanical turk) about a specific point
- Setting
  - We have a large collection of unlabeled points
  - Can query labels for specific unlabeled examples
  - Each query has a cost associated, so we want to minimize the number of queries
  - Goal is to still learn a mapping from input to some label/output
- How to design the querying system so that we learn good models with smallest amount of data?

Limited/partial Feedback

- Instead of getting correct label, we only know if the prediction was correct or not
- Only know loss/payoff of label/action chosen, not of others
- "Bandit" problems: ad placement, recommendation systems, …

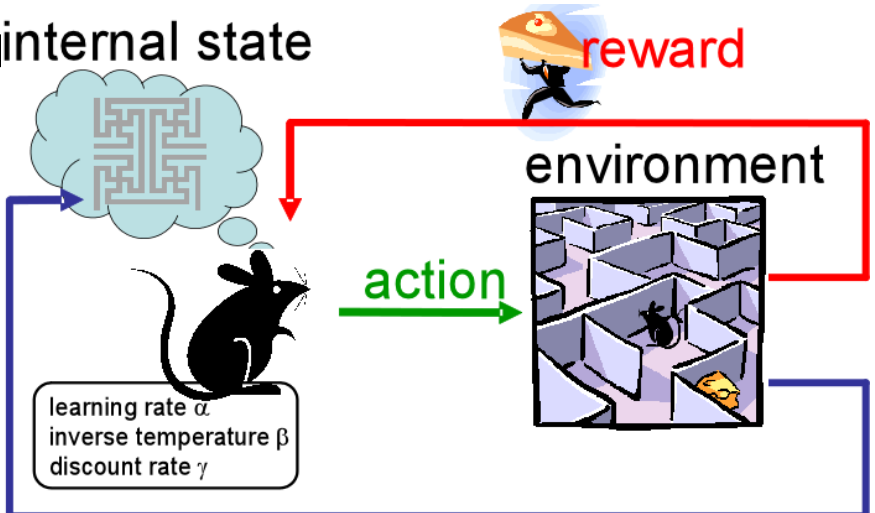
- New challenge: Exploration vs Exploitation

Slide credit: Nati Srebro

Reinforcement Learning

- Control agent (robot) in environment, only see reward when you get it
- Long term planning to finish a task
  - At time t you are in some (unknown to you) state s_t
  - You choose an action a_t, based on which you move to a new state $s_{t+1} = f(s_t, a_t)$ (maybe with some randomness) and receive reward $r(s_{t+1})$.
  - You don't know $f(\cdot,\cdot)$ and $r(\cdot)$ (need to learn them)
  - You **only** know the rewards $r(s_t)$ you get, and possibly other limited feedback about the state $o(s_t)$
  - Goal: maximize rewards

- E.g.: mouse moving in a maze
  - State = location and direction
  - Action = move forward, turn left or turn rig
  - Reward = cheese
  - Observation(State) = (front wall, left wall, right wall, back wall)



Slide credit: Nati Srebro

Probabilistic Models

- Probabilistic models define models for $\Pr(x, y)$ or $\Pr(x|y)$ or $\Pr(y|x)$
- We saw some simple examples of this flavor
- More complex models often use many latent variables
  - typically represented as using graphical models such as Bayesian Networks and Markov Random Fields
- Techniques for
  - Modeling : how to represent $\Pr(x, y)$ or $\Pr(x|y)$ or $\Pr(y|x)$
  - Inference : inferring the values of latent variables
  - Learning : prediction
- Many times the optimization problems are non-convex and sometimes even non-computable
  - approximate inference algorithms are very common

Machine Learning Landscape

Convex (= Linear)

- **Linear/logistic reg.**
- **SVMs**
- **Boosting**
- Many other models

Main optimization tools: LP/SDP solvers and SGD

Combinatorial Classes

- Formulas (DNFs)
- **Decision trees**

Main optimization tools: greedy, combinatorial search (using pruning, genetic programming, simulated annealing, etc)

Non-Parametric

- **Nearest-Neighbor**
- Parzan Window
- Random walk on example graph

Non-Convex

- **Neural Networks**
- Dictionary and representation learning

Main optimization tools: SGD with tricks

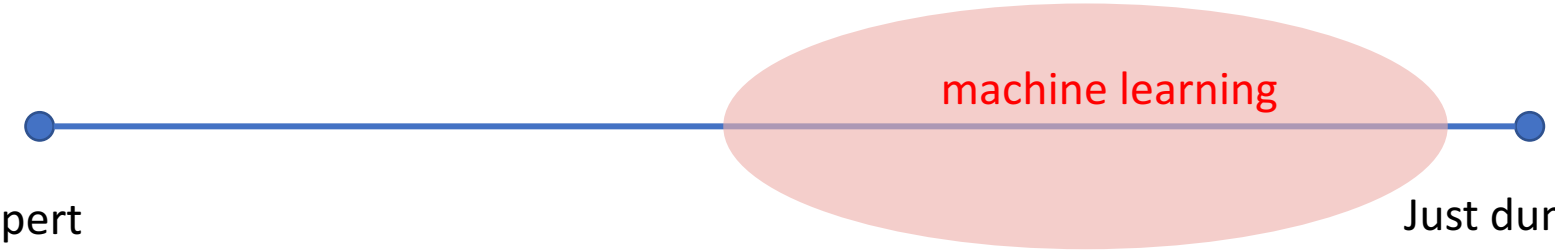
Probabilistic Models

- Fit data to generative model
- Bayes nets, graphical models
- Latent variable models

Typically non-convex, same issues as non-convex models

Slide credit: Nati Srebro

Expert designed → data driven



C. M. Bishop: "…a training set is used to tune the parameters of an adaptive model"