ELEG 6913-P17: Deep Learning

Spring 2017

Lecture 3: Deep Feedforward Networks

Dr. Lijun Qian and Dr. Xishuang Dong

Outline

- **Deep Learning**
- **Deep Feedforward Networks**
- **Regularization for Deep Learning**
- **Optimization for Training Deep Models**
- **Summary**

(Acknowledgment: some parts of the slides are from Sargur N. Srihari, and various other sources.

Outline

- **Deep Learning**
- **Deep Feedforward Networks**
- **Regularization for Deep Learning**
- **Optimization for Training Deep Models**
- **Summary**

Artificial Intelligence (AI)

- **Creating machines that think like human beings**

History of AI

- **Advent of Programmable computers led to whether AI was possible**
  - **– Lovelace 1842, Turing 1940s**

- **Today, AI is a thriving field**
  - **– Understand natural language**
  - **– Understand speech or images**
  - **– Make diagnosis in medicine**
  - **– Support basic science research**

Challenge of AI

- **Early days of AI:**

– Solved problems intellectually difficult for humans problems described by small set of rules

- **Success of Deep Blue, (little world knowledge needed)**

Challenge of AI

https://qz.com/877721/the-ai-master-bested-the-worlds-top-go-players-and-then-revealed-itself-as-googles-alphago-in-disguise/

Solution to Everyday Problems

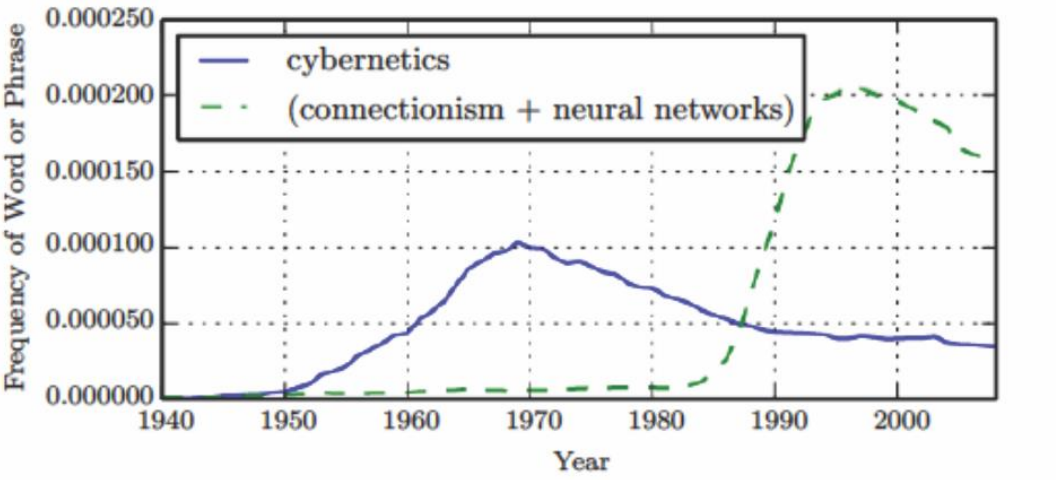
- **AI systems need their own ability to acquire needed knowledge.**

- **Approach avoids need for human operators to formally specify knowledge.**

- **Machine Learning is the capability to acquire knowledge: extracting patterns from raw data.**
  - **– Allows acquiring knowledge of the real world and make decisions that seem subjective**
    - **– Logistic regression: decide whether email is spam**

History of Deep Learning

- **Dates back to 1940s**
- **Three historical waves:**
  - – Cybernetics, peaked in 1970
  - – Connectionism/neural networks, peaked in 1995
  - – Deep learning, 2006+ (layerwise training, big data)



Breakthrough of Machine Learning (ML)

Deep Learning: machine learning algorithms based on learning multiple levels of representation / abstraction.

Amazing improvements in error rate in object recognition, object detection, speech recognition, and more recently, in natural language processing / understanding

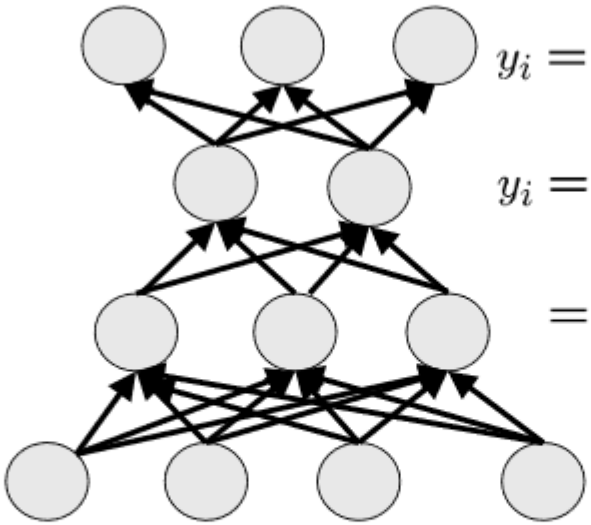
Deep architectures

Defintion: Deep architectures are composed of *multiple levels* of non-linear operations, such as neural nets with many hidden layers.

Output layer

Hidden layers

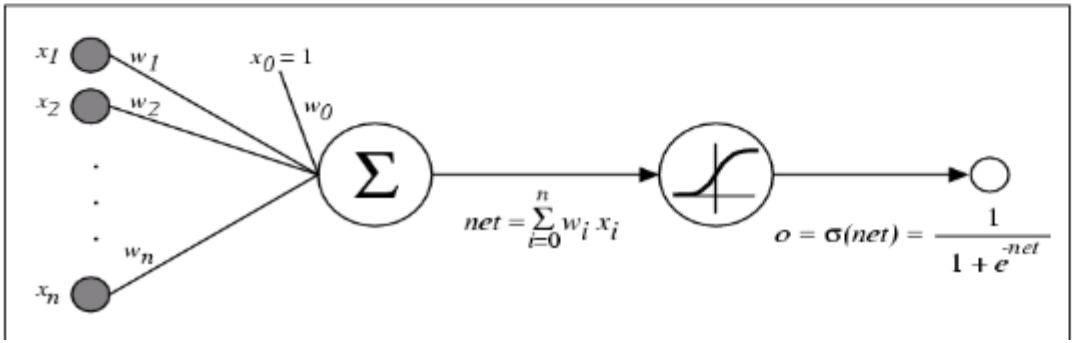
Input layer



$$y_i = \tanh(\sum_j W_{ij} x_j + W_{i0})$$

$$y_i = \sigma(\sum_j W_{ij} x_j + W_{i0})$$

$$= \frac{1}{1 + \exp[-(\sum_j W_{ij} x_j + W_{i0})]}$$



Goal of Deep architectures

Goal: Deep learning methods aim at

- learning *feature hierarchies*
- where features from higher levels of the hierarchy are formed by lower level features.

edges, local shapes, object parts

Low level representation

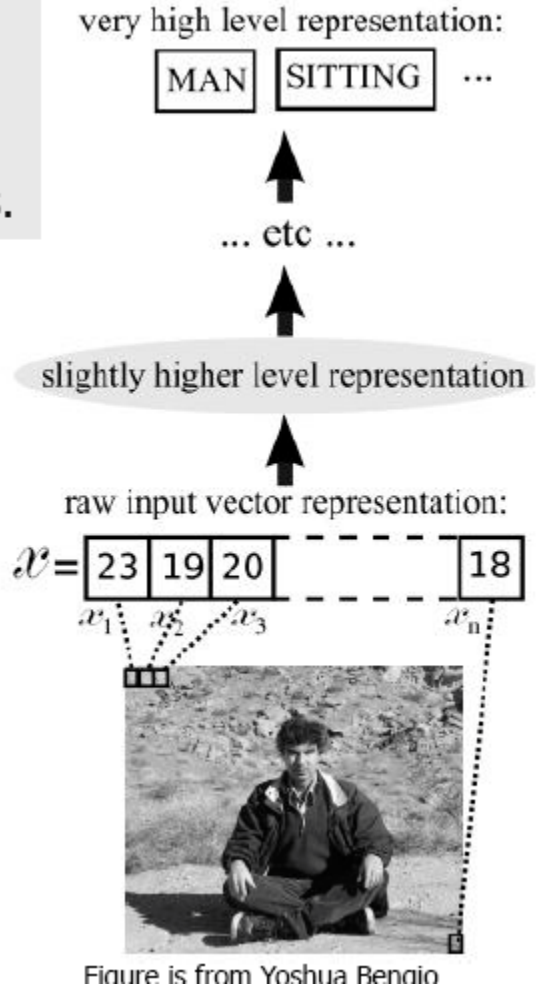


Figure is from Yoshua Bengio

Representation for Machine Learning

- **Simple ML algorithms depend heavily on representation of given data**
  - **– Feature Extraction and Selection**
  - **– Feature Relations: Linear vs Non-linear**

- **Logistic regression to recommend delivery does not examine patient directly**

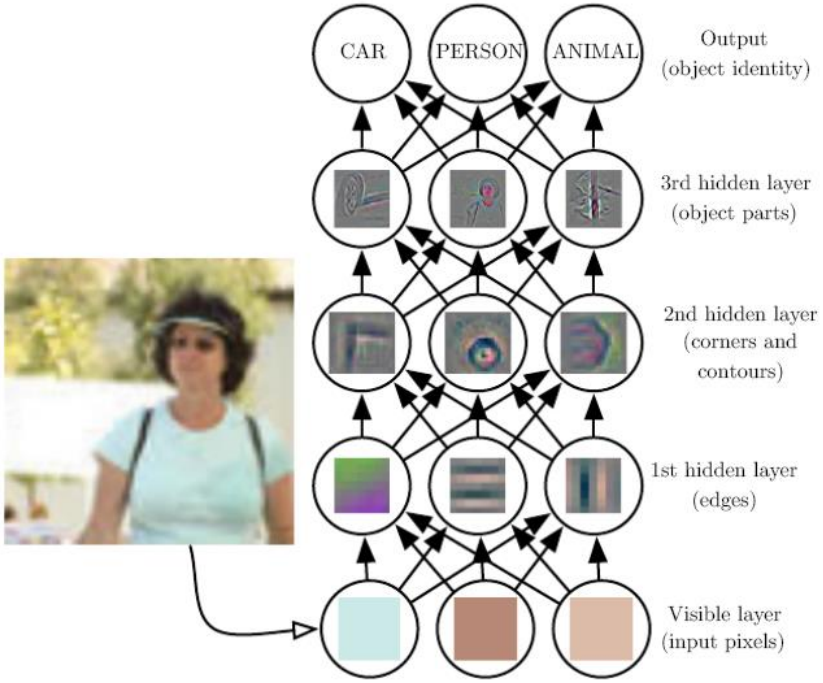
Representation Learning

- **Learn not only the mapping from representation to output but the representation itself**
- **Learned representations often provide much better results than hand-coded representations**
- **Allow AI systems to rapidly adapt to new tasks**
  - Designing features can take great deal of human effort

Illustration of Deep Learning

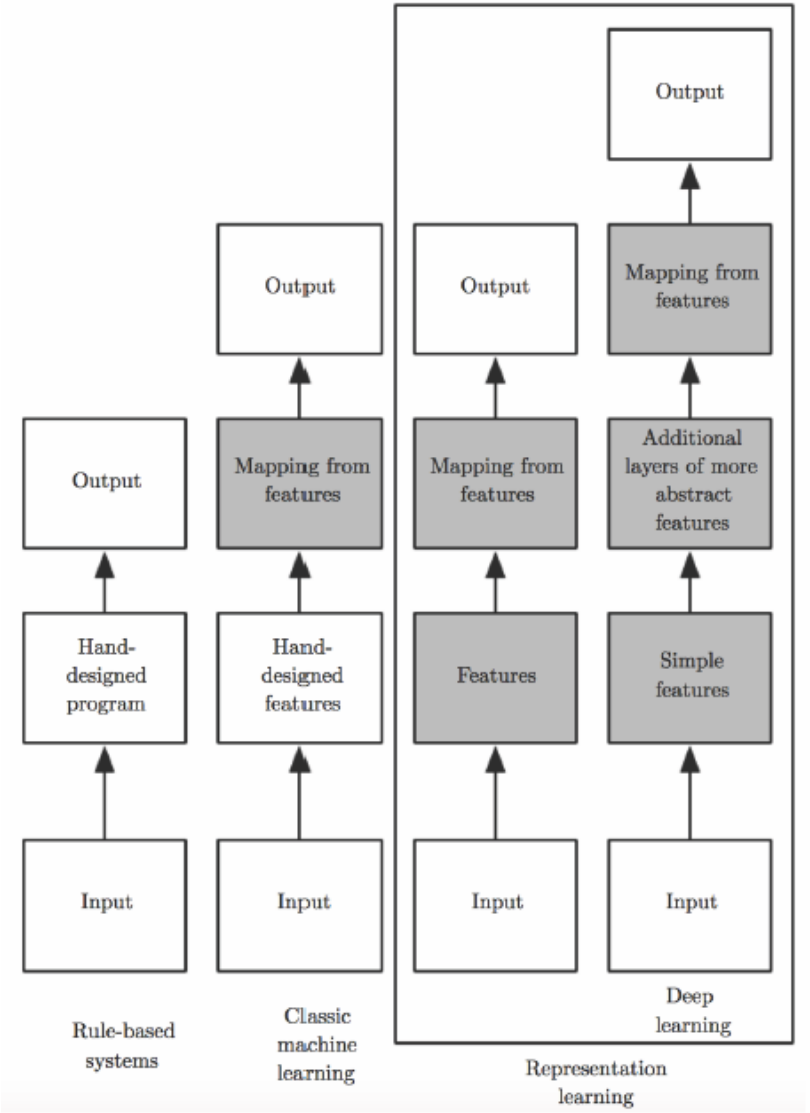
- **Function mapping from pixels to object identity is complicated**

- **Series of hidden layers extract increasingly abstract features**



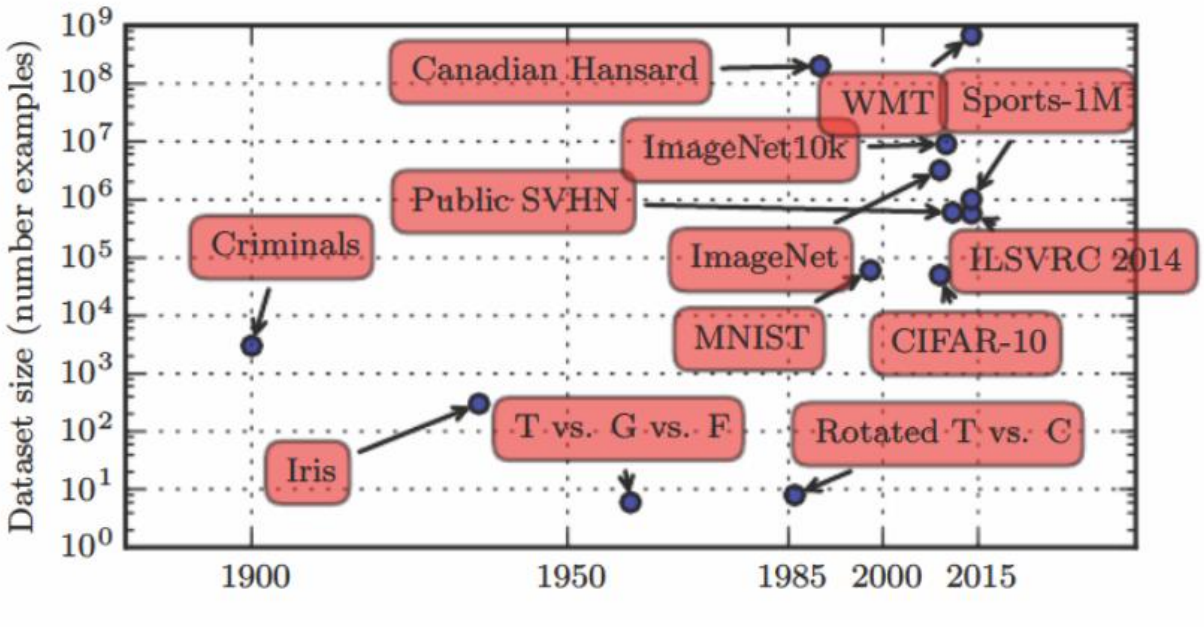
Different AI Models

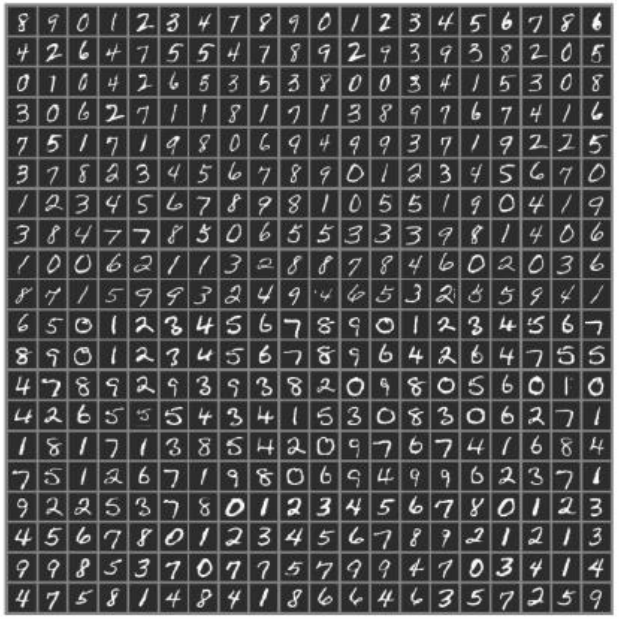
- **Relationship of different parts of system in different AI disciplines**
- **Shaded boxes involve ML**



Large Data Sets

- **Algorithms can be provided with data sets needed to succeed**





Increased Connectivity

- **Intelligence achieved when many neurons work together**

1.Adaptive Linear element (Adaline), 1960

2. Neocognitron

3. GPU CNN

4. Deep Boltzmann

5. Unsup. CNN

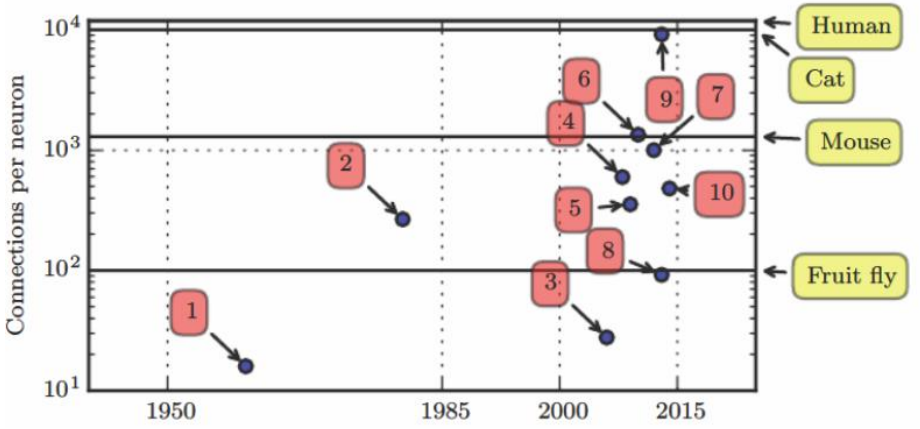
6. GPU MLP

7. Distributed Autoencoder

8. Multi-GPU CNN

9. COTS Unsupervised CNN

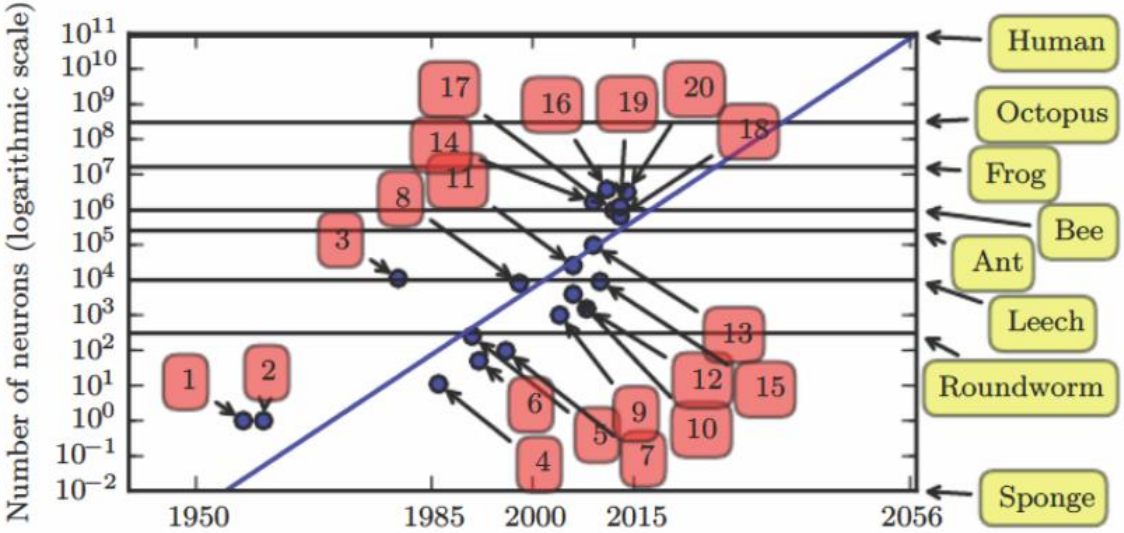
10. GoogLeNet



Increased no. of neurons

- **Since introduction of hidden units, doubling of neurons every 2.4 years**

1. Perceptron
2. Adaline
3. Neocognitron
4. Early backprop
5. RNN speech
6. MLP speech
7. Mean-field Belief Net
8. LeNet5
9. Echo-state network
10. Deep Belief Net
11. GPU CNN

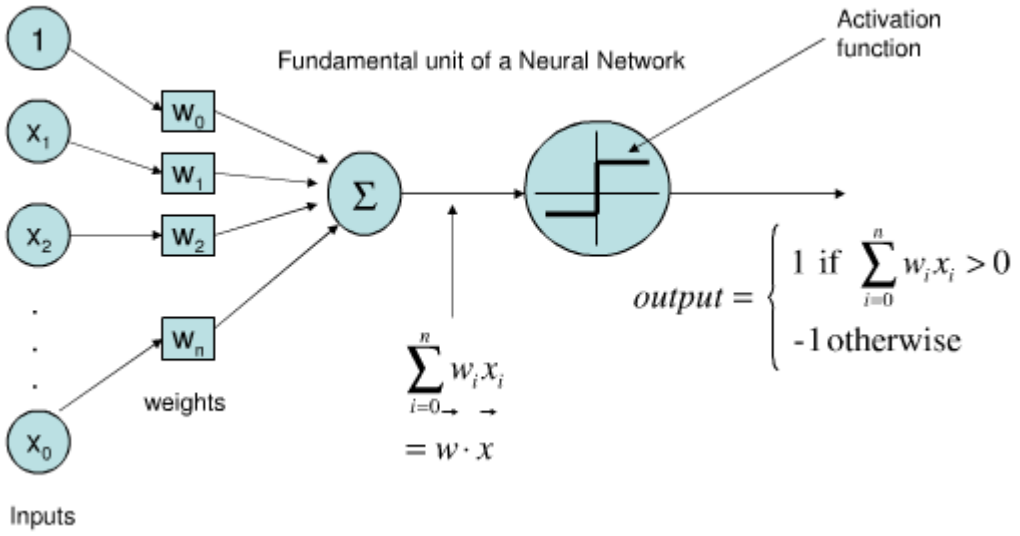


Outline

- Deep Learning
- **Deep Feedforward Networks**
- Regularization for Deep Learning
- Optimization for Training Deep Models
- Summary

Goal of Feedforward Networks

- **Deep Feedforward Networks are also called as**
  - **– Feedforward neural networks or**
  - **– Multilayer Perceptrons (Why?)**

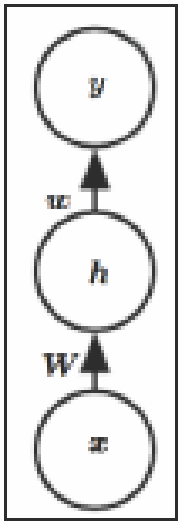


Goal of Feedforward Networks

- **Their Goal: approximate some function *f'***
  - **E.g., a classifier $y = f'(x)$ maps an input x to a category y**
  - **Feedforward Network defines a mapping**
    - **$y = f'(x ; \theta)$**
    - **and learns the values of the parameters θ that result in the best function approximation**

Feedforward Network

- **Models are called Feedforward because:**
  - **Information flows through function being evaluated from *x* through intermediate computations used to define f and finally to output *y***
- **There are no feedback connections**
  - **No outputs of the model are fed back**

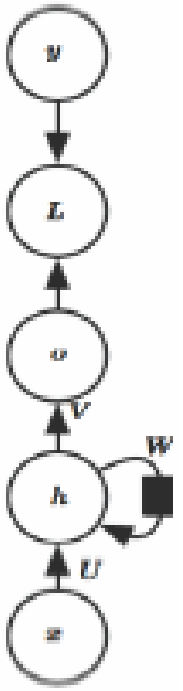


Feedforward Network

- **US Presidential Election**
  - **– Inputs are raw votes cast**
  - **– Hidden layer is electoral college**
  - **– Output are candidates**

Feedforward vs. Recurrent

- **When extended to include feedback connections they are called *Recurrent Neural Networks (RNN)***



Importance of Feedforward Networks

- **They are extremely important to ML practice**
  - **– Many commercial applications**
  - **– E.g., convolutional networks used for recognizing objects from photos**

- **They are a conceptual stepping stone on path to recurrent networks**
  - **– Which power many NLP applications**

Feedforward Neural Network Structures

- **Called networks because they are composed of many different functions**
- **Model is associated with a directed acyclic graph describing how functions composed**
- **These chain structures are the most commonly used structures of neural networks**

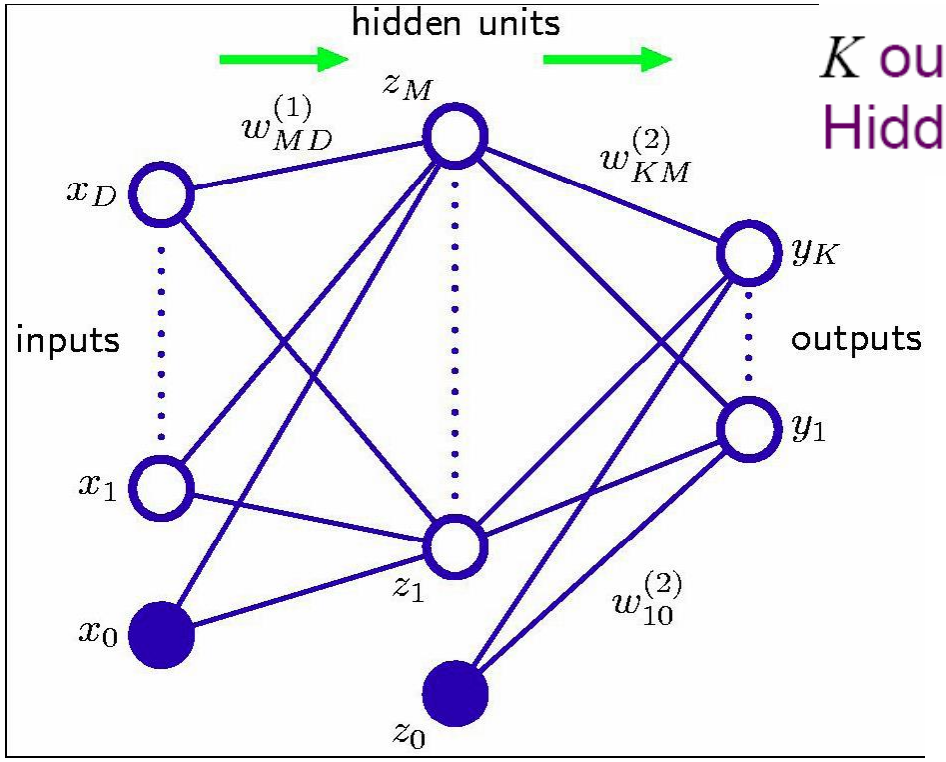
Feedforward Neural Network Structures

- **Model is associated with a directed acyclic graph describing how functions composed**
  - **E.g., functions** $f^{(1)}, f^{(2)}, f^{(3)}$ **connected in a chain to form** $f(\boldsymbol{x}) = f^{(3)}[f^{(2)}[f^{(1)}(\boldsymbol{x})]]$
    - $f^{(1)}$ **is called the first layer of the network**
    - $f^{(2)}$ **is called the second layer, etc**

Definition of Depth

- **Overall length of the chain is the depth of the model**
- **The name deep learning arises from this terminology**
- **Final layer of a feedforward network is called the output layer**

A Feed-forward Neural Network



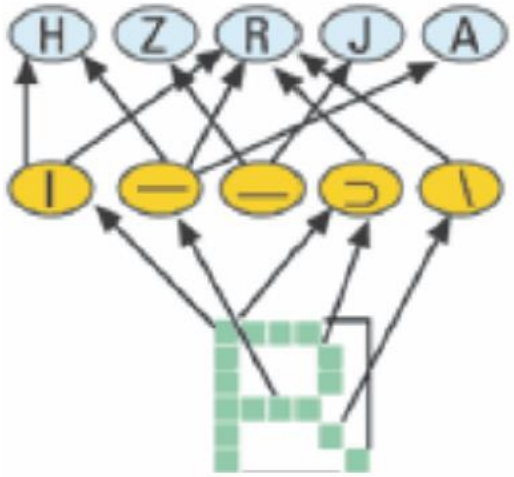
K outputs $y_1,..y_K$ for a given input x
Hidden layer consists of M units

$$y_k(\mathrm{x},\mathrm{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

$$f_m{}^{(1)} = z_m = h(\mathrm{x},\mathrm{w}^{(1)}),\ m=1,..M$$
$$f_k{}^{(2)} = \sigma\,(\mathrm{z},\mathrm{w}^{(2)}),\ k=1,..K$$

Example of Feedforward Network

- **Optical Character Recognition (OCR)**



Hidden layer compares raw pixel inputs to component patterns

Training the Network

- **In network training we drive *f(x)* to match *f'(x)***

- **Training data provides us with noisy, approximate examples of *f'(x)* evaluated at different training points**

Training the Network

- **Each example accompanied by a label $y \approx f'(x)$**

- **Training examples specify directly what the output layer must do at each point x**
  - **It must produce a value y' that is close to y**

What are Hidden Layers?

- **Behavior of other layers is not directly specified by the data**

- **Learning algorithm must decide how to use those layers to produce value that is close to *y***

What are Hidden Layers?

- **Training data does not say what individual layers should do**

- **Since the desired output for these layers is not shown, they are called *hidden layers***

Why are they called neural?

- **These networks are loosely inspired by neuroscience**
- **Each hidden layer is typically vector-valued**
- **Each unite receives inputs from many other units and computes its own activation value**

Why are they called neural?

- **Each hidden layer is typically vector-valued**
  - **– Dimensionality of hidden layer is width of the model**
  - **– Each element of vector viewed as a neuron**
  - **– Instead of thinking of it as a vector-vector function, they are regarded as units in parallel**

Neural networks and Brains

- **Choice of functions $f^{(i)}(\boldsymbol{x})$:**
  - **Loosely guided by neuroscientific observations about biological neurons**
    - **Modern neural networks are guided by many mathematical and engineering disciplines**
    - **Not perfectly model the brain**

Neural networks and Brains

- **Think of feedforward networks as function approximation machines**
  - **– Designed to achieve statistical generalization**
  - **– Occasionally draw insights from what we know about the brain**
  - **– Rather than as models of brain function**

View as Extension of Linear Models

- **Begin with linear models and see limitations**

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

  - **Linear regression:**

$$E_D(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N} \left\{ t_n - \mathrm{w}^T \phi(\mathrm{x}_n) \right\}^2$$

    - **Simple closed form solutions:**

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

$$\Phi = \begin{pmatrix} \phi_0(\mathrm{x}_1) & \phi_1(\mathrm{x}_1) & \cdots & \phi_{M-1}(\mathrm{x}_1) \\ \phi_0(\mathrm{x}_2) & & & \\ & & & \\ \phi_0(\mathrm{x}_N) & & & \phi_{M-1}(\mathrm{x}_N) \end{pmatrix}$$

    - **Or solved with convex optimization:**

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} - \eta \nabla E_n$$

$$\nabla E_n = -\sum_{n=1}^{N} \left\{ t_n - \mathrm{w}^T \phi(\mathrm{x}_n) \right\} \phi(\mathrm{x}_n)^T$$

View as Extension of Linear Models

- **Begin with linear models and see limitations**
  - **Logistic regression:** $y(\boldsymbol{x}, \mathrm{w}) = \sigma(\mathrm{w}^T \phi(\boldsymbol{x}))$
    - **No closed-form solution**
    - **Convex Optimization:**

$$\boxed{\mathrm{w}^{\tau+1} = \mathrm{w}^{\tau} - \eta \nabla E_n} \qquad \boxed{\nabla E_n = (y_n - t_n)\phi(\boldsymbol{x}_n)}$$

- **If $\phi(x) = x$ model capacity is limited to linear functions and model has no understanding of interaction between any two input variables**

Extending Linear Models

- **To represent non-linear functions of x apply linear model not to x but to a transformed input $\phi(x)$ where ϕ is a nonlinear transformation**
  - **Equivalently apply the kernel trick to obtain a nonlinear algorithm**
- **We can think of ϕ as providing a set of features describing x, or as providing a new representation for x**

How to choose mapping ϕ?

- **Replace x by generic feature function $\phi(x)$**
  - ***1. RBF: $N(x; x^{(i)}, \sigma 2I)$ centered at $x^{(i)}$***
  - ***2. SVM: $f(x)=w^T x+b$ written as $b + \Sigma_i \alpha_i \phi(x)\phi(x^{(i)})$***
    - ***Choose $k(x,x^{(i)})=\phi(x)\phi(x^{(i)})$***
    - ***Use linear regression on Lagrangian for weights α^i***
    - ***Evaluate f over samples for non-zero (support vectors)***

How to choose mapping ϕ?

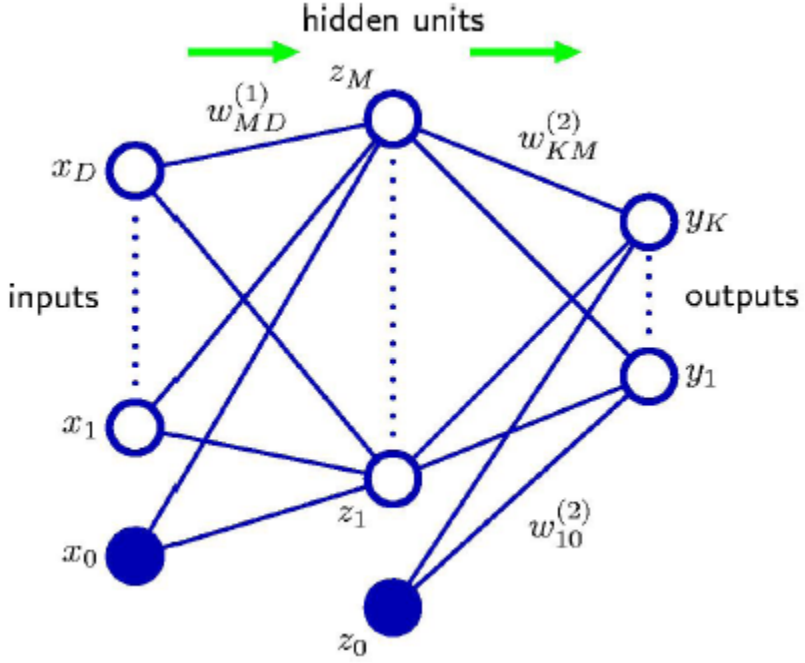
2. Manually engineer ϕ

- **decades of effort: e.g., speech reco, computer vision**
- **laborious, non-transferable between domains**

3. Learn ϕ

- **Approach used in deep learning**

Neural Nets extend Linear Methods



K outputs $y_1,..y_K$ for a given input x
Hidden layer consists of M units

$$y_k(\mathbf{x},\mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- **Can be viewed as a generalization of linear models**
  - **A nonlinear function σ operating on a linear model with basis functions, *h***
  - **Basis functions are learnt**

How to Learn Features?

- **Model is $y=f(x;\theta,w) = \phi(x;\theta)^T w$**
  - **Parameters θ are used to learn ϕ from a broad class of functions**
  - **Parameters w map from $\phi(x)$ to output**
  - **Defines feed-forward network with ϕ defining a hidden layer**

How to Learn Features?

- **Unlike other two, this approach gives-up on convexity of the training problem**
  - **– Use optimization to find θ that corresponds to a good representation**
  - **– Use families of $\phi(x;\theta)$ using domain knowledge**

Ex: XOR problem

- **Two binary variables x_1 and x_2**
- **When exactly one value equals 1 it returns 1, otherwise it returns 0**
- **Target function is $y=f^*(x)$ that we want to learn**
- **Our model is $y=f(x;\theta)$ which we want to learn, i.e., adapt parameters θ to make it as similar as possible to f^***
- **We are given four training points which we must fit: $X=\{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$**

ML for XOR: linear model doesn't fit

- **The MSE loss function is**

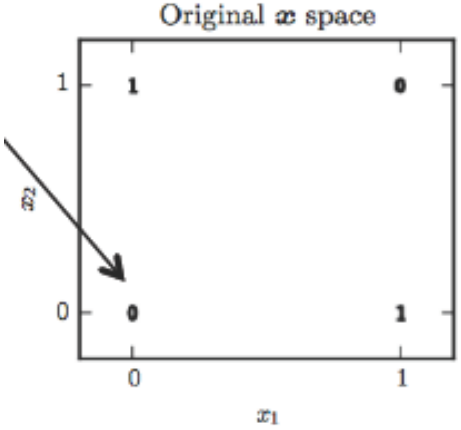
$$J(\theta) = \frac{1}{4}\sum_{x \in X}\left(f^*(x) - f(x;\theta)\right)^2$$

- **We must choose the form of the model**
- **Suppose we choose a linear model with $\theta = \{w,b\}$:**

$$f(x;w,b) = x^T w + b$$

- **We can minimize $J(\theta)$ in closed-form wrt w and b and obtain $w=0$ and $b=1/2$**
- **The linear model simply outputs 0.5 everywhere**

Linear model cannot solve XOR



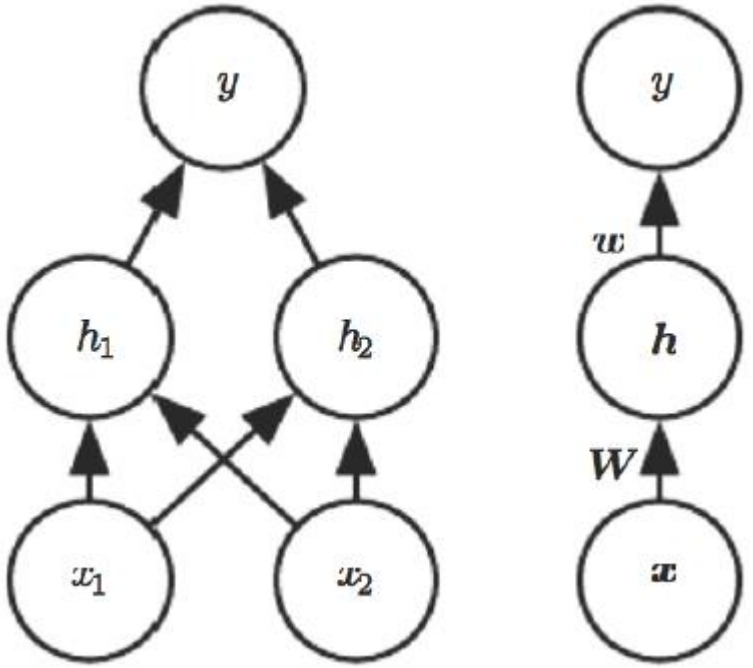
- **Bold nos: values system must output**
  - **Linear model incapable**
    - **When x_1=0, output has to increase with x_2**
    - **When x_1=1, output has to decrease with x_2**
    - **Linear model has to assign a single weight to x_2, so it cannot solve this problem**

Linear model cannot solve XOR

- **One way to solve problem: use a model to learn a different representation in which a linear model is able to represent the solution**
  - **We use a simple feedforward network**
    - **one hidden layer containing two hidden units**

Feedforward Network for XOR

- **Same network drawn in two different styles**
  - **Matrix W describes mapping from x to h**
  - **Vector w describes mapping from h to y**
  - **Intercept parameters b are omitted**



Functions computed by Network

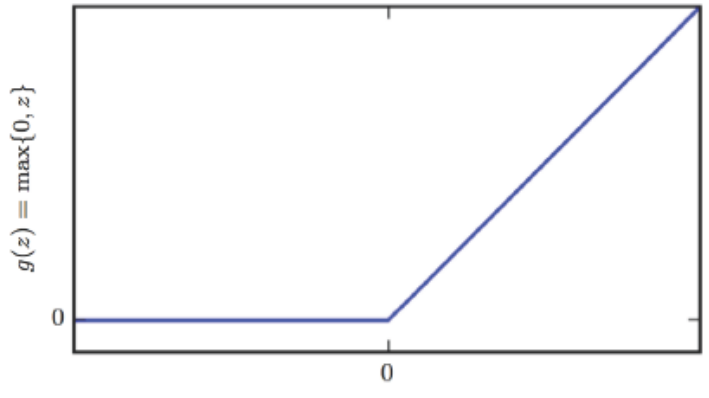
- **Layer 1 (hidden layer): vector of hidden units *h* computed by function $f^{(1)}(x; W,c)$**
  - **c are bias variables**
- **Layer 2 (output layer) computes $f^{(2)}(h; w,b)$**
  - ***w* are linear regression weights**
- **Output is linear regression applied to *h* rather than to *x***
- **Complete model is $f(x;W,c,w,b)=f^{(2)}(f^{(1)}(x))$**

Choice of functions

- **If we choose both $f^{(1)}$ and $f^{(2)}$ to be linear, the total function will still be linear**
  - **Suppose $f^{(1)}(x) = W^T x$ and $f^{(2)}(h) = h^T w$**
  - **Then we could represent this function as $f(x) = x^T w'$ where $w' = Ww$**
- **So we must use a nonlinear function to describe the features**
- **We use the strategy of neural networks by using a nonlinear activation function $h = g(W^T x + c)$**

Rectified Linear Activation Function

- **Activation: g(z)=max{0,z}**
  - **Default for feed-forward networks**
  - **Piecewise linear**
  - **Preserve properties that make linear models easy to optimize**



- **The complete network specifies**

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x})) = \boldsymbol{w}^T \max\{0, W^T\boldsymbol{x} + \boldsymbol{c}\} + b$$

Complete Network and Solution

- **For solution to XOR, let**

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b = 0$$

- **Walk through how model processes batch**
  - **Design matrix of all four points:** $X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$
  - **First step is XW:** $XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$
  - **Adding c:** $XW + c = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$
  - **Compute h**

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

Complete Network and Solution

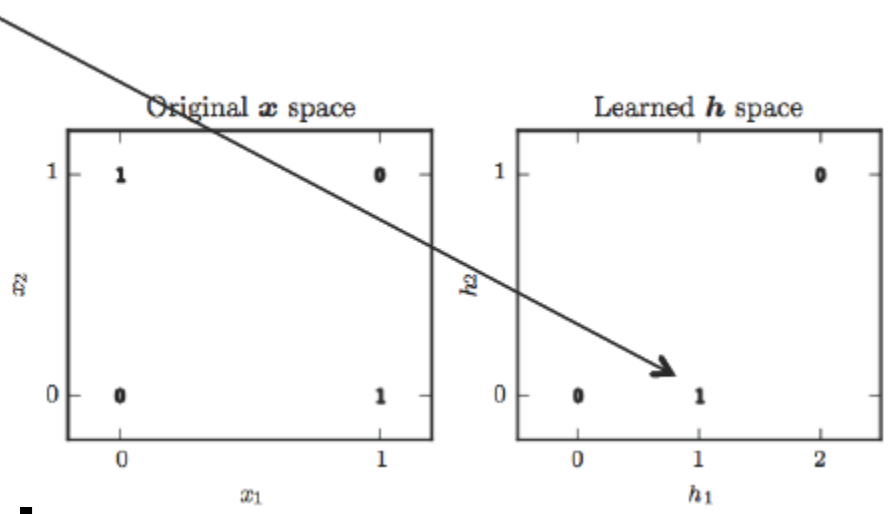
– They now lie in space where linear model suffices

• Finish multiplying by w: $\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

– Network has obtained correct answer for every example

Learned representation for XOR

- **Two points that must have output *1* have been collapsed into one**
- **Points $x=[0,1]^T$ and $x=[1,0]^T$ have been**
- **mapped into $h=[0,1]^T$**
- **Described in linear model**
  - **Output increases in h_1 and decreases in h_2**



Outline

- Deep Learning
- Deep Feedforward Networks
- **Regularization for Deep Learning**
- Optimization for Training Deep Models
- Summary

What is Regularization?

- **Central problem of ML is to design algorithms that will perform well not just on training data but on new inputs as well**
- **Regularization is any strategy to reduce test error even at the expense of increasing training error**
- **Many forms of regularization available**
  - **Major efforts of the field are to develop better regularization**

Some Goals of Regularization

- **Put extra constraints on objective function**
  - **They are equivalent to a soft constraint on parameter values**
    - **Result in improved performance on test set**
- **Some goals of regularization**
  1. **Encode prior knowledge**
  2. **Express preference for simpler model**
  3. **Needed to make underdetermined problem determined**

Regularizing Estimators

- **In Deep Learning, regularization means regularizing estimators**
- **Involves increased bias for reduced variance**
  - **Good regularizes reduces variance significantly while not overly increasing bias**

Model Family and Regularization

- **Three types of model families**
  1. **Excludes the true data generating process**
     - **Implies underfitting and high bias**
  2. **Matches the true data generating process**
  3. **Overfits**
     - **Includes true data generating process but also many other processes**
- **Goal of regularization is to take model from third regime to second**

Importance of Regularization

- **Overly complex family does not necessarily include target function, true data generating process, or even an approximation**
- **Most deep learning applications are where true data generating process is outside family**
  - **Complex domains of images, audio sequences and text true generation process may involve entire universe**
    - **Fitting square hole (data generating process) to round hole (model family)**

What is the Best Model?

- **Best fitting model obtained not by finding the right number of parameters**
- **Instead, best fitting model is a large model that has been regularized appropriately**
- **We look at strategies of how to create such a large, deep regularized model**

Regularization Strategies

1. Parameter Norm Penalties
   – (L2- and L1- regularization)
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness

More Regularization Strategies

6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Outline

- Deep Learning
- Deep Feedforward Networks
- Regularization for Deep Learning
- **Optimization for Training Deep Models**
- Summary

The Optimization Task

- **Find parameters θ of a neural network that significantly reduce a cost function $J(\theta)$**
- **$J(\theta)$ typically include a performance measure evaluated on the entire training set as well as additional regularization terms**

Plan of discussion

- How training optimization differs from pure optimization
- Challenges that make optimization difficult
- Several practical algorithms for
  - Optimization
  - Strategies for initializing parameters
  - Advanced algorithms adapt learning rates
- Optimization Strategies

ML Training vs. Pure Optimization

- **There are several differences**
- **ML acts indirectly**
  - **We care about some performance measure *P* which may be intractable**
  - **We reduce a different cost function *J(θ)* in the hope that doing so will reduce *P***
- **Pure optimization: minimizing *J* is a goal in itself**
- **Optimizing Deep models: specialization on structure of ML objective function**

Cost Function for Training

- **Cost is average over the training set**

$$J(\boldsymbol{\theta}) = E_{(\boldsymbol{x},y)\sim\hat{p}_{data}} \left(L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \right)$$

  - **where L is the per-example loss function**
  - **$f(x ; \theta)$ is the predicted output when the input is x**
  - **$\hat{p}_{data}$ is the empirical distribution; we may prefer over all p_{data}**
  - **In supervised learning, y is target output**

Cost Function for Training

- **We consider the supervised unregularized case**
  - **Where arguments of L are $f(x; \theta)$ and y**
  - **Trivial to extend to case where θ and x are arguments or y is excluded**
    - **To develop various forms of regularization or unsupervised learning**

Empirical Risk Minimization

- **True risk is**

$$J(\boldsymbol{\theta}) = E_{(\boldsymbol{x},y)\sim\hat{p}_{data}}\left(L(f(\boldsymbol{x};\boldsymbol{\theta}),y)\right)$$

- **Empirical risk, with m training examples, is**

$$J(\boldsymbol{\theta}) = E_{(\boldsymbol{x},y)\sim p_{data}}\left(L(f(\boldsymbol{x};\boldsymbol{\theta}),y)\right) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)})$$

Empirical Risk Minimization

- **Empirical risk, with m training examples, is**
  - **– Empirical risk minimization is not very useful since:**
    1. **Prone to overfitting: can simply memorize training set**
    2. **SGD is commonly used, but many useful loss functions have 0-1 loss, with no useful derivatives (0 or undefined everywhere)**
- **We must use a slightly different approach**

Surrogate Loss Functions

- **Exactly minimizing 0-1 loss is typically intractable (exponential in the input dimension) even for a linear classifier**

- **In such situations use a surrogate loss function**
  - **E.g., negative log-likelihood of the correct class is a surrogate for 0-1 loss**

Early Stopping

- **Important difference between optimization in general and learning:**
  - **Training algorithms do not usually halt at a local minimum**
- **ML algorithm usually minimizes a surrogate loss function and halts based on an early stopping criterion**
  - **Typically loss on a validation set designed to stop overfitting**

Batch and Minibatch Algorithms

- **Unlike pure optimization, in learning the objective function decomposes as a sum over training examples**

- **Learning algorithms update the parameters based only a subset of the terms of the cost function**

Need for minibatch

- **Gradient of $J(\theta)$ is also an expectation**

$$\nabla_{\theta} J(\boldsymbol{\theta}) = E_{\boldsymbol{x}, y \sim \hat{p}_{data}} \log p_{\mathrm{model}} \left(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta} \right)$$

- **Computing this expectation is very expensive**
  - **Requires summation over every training sample**
  - **Instead randomly sample small no. of samples**

Minibatch yields better results

- **Standard error for mean from n samples is $\sigma/n^{1/2}$, where σ is std dev of samples**

- **Denominator shows that error decreases less than linearly with no. of samples**
  - **Ex: 100 samples vs 10,000 samples**
    - **Computation increases by a factor of 100**

      but Error decreases by only a factor of 10

- **Optimization algorithms converge much faster**

Another motivation: small sample gradient

- Training set may be redundant
- Worst case: all m examples are same
  - Sampling based estimate could use m times less computation
- In practice, unlikely to find worst case situation

Terminology

- **Batch or deterministic: use all training samples in a batch**
- **Those using a single sample are called stochastic or on-line**
  - **On-line typically means continually created samples, rather than multiple passes over data**
- **More than 1 but less than all are minibatch stochastic or simply stochastic**

Minibatch Size

- **Driven by following factors**
  - **Larger batch -> more accurate gradient but with less than linear returns**
  - **GPU architectures more efficient with power of 2**
    - **Range from 32 to 256, with 16 for large models**

Challenges for optimization

- **Ill-Conditioning**
  - **Ill-conditioning can manifest by causing SGD to get "stuck" in the sense that even very small steps increase the cost function.**
- **Local Minima**
  - **With non-convex functions, such as neural nets, it is possible to have many local minima.**

Challenges for optimization

- **Saddle Points**
  - **In low dimensional spaces, local minima are common.**
  - **In higher dimensional spaces, local minima are rare while saddle points are more common.**

Challenges for optimization

- **Long-Term Dependencies**

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}.$$

  - **Vanishing gradients make it difficult to know which direction the parameters should move to improve the cost function**
  - **Exploding gradients can make learning unstable**
- **Inexact Gradients**
  - **A noisy or even biased estimate**

Several practical algorithms

- **Optimization**

SGD

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Several practical algorithms

- **Optimization**

Polyak, 1964

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Several practical algorithms

- **Optimization**

Sutskever 2013

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
  Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
  Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Parameter Initialization Strategies

- **Modern initialization strategies are simple and heuristic**
  - **Designing improved initialization strategies is a difficult task because neural network optimization is not yet well understood.**
  - **Most initialization strategies are based on achieving some nice properties when the network is initialized.**
  - **Some initial points may be beneficial from the viewpoint of optimization but detrimental from the viewpoint of generalization.**

Parameter Initialization Strategies

- **Always initialize all the weights in the model to values drawn randomly from a Gaussian or uniform distribution.**

- **The scale of the initial distribution, however, does have a large effect**
  - **Outcome of the optimization procedure**
  - **The ability of the network to generalize.**

Parameter Initialization Strategies

- **Normalized initialization**

$$\mathrm{W}_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right).$$

  - **Weights of a fully connected layer with *m* inputs and *n* outputs**
- **Random orthogonal matric**
  - **Guarantees that the total number of training iterations required to reach convergence is independent of depth.**

Parameter Initialization Strategies

- **Strategies often do not lead to optimal performance.**
  - **Wrong criteria**
  - **Properties imposed at initialization may not persist**
  - **Improving the speed of optimization but inadvertently increase generalization error.**
    - **Increase the testing error**

Algorithms with Adaptive Learning Rates

- **Algorithms**
  - **AdaDelta**
  - **AdaGrad**
  - **RMSProp**
  - **Adam**
- **Schaul et al. (2014)**
  - **Algorithms with adaptive learning rates performed fairly robustly**
  - **No single best algorithm has emerged.**

Adam

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

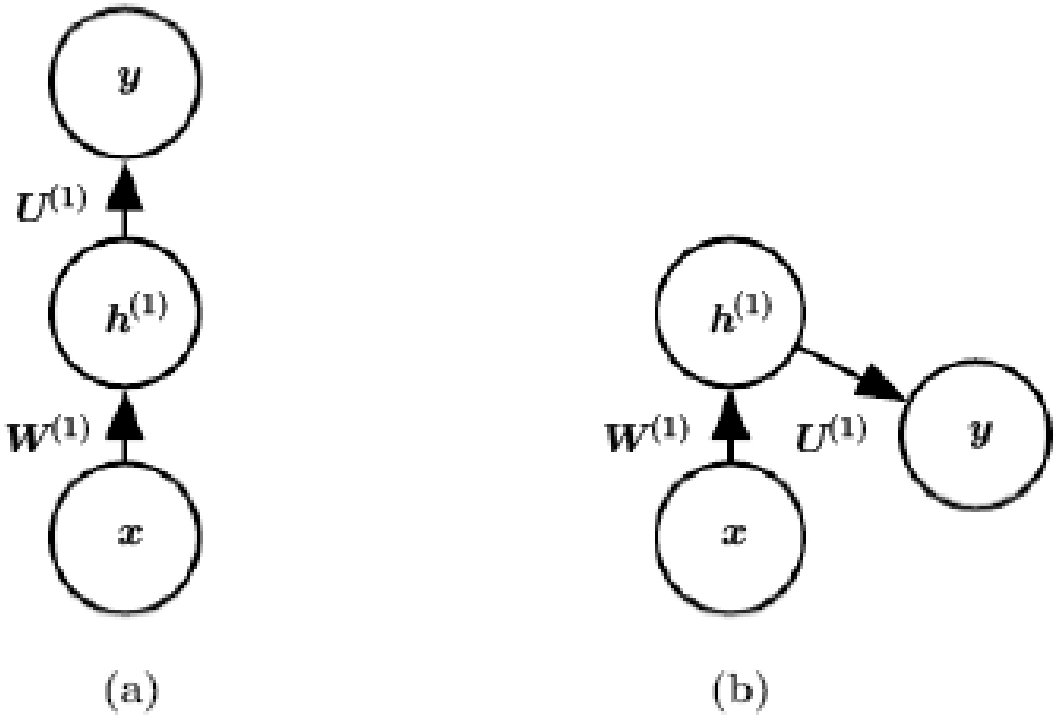
Optimization Strategies

- **Batch Normalization**
- **Coordinate Descent**
- **Supervised Pretraining**
- **Continuation Methods and Curriculum Learning**
- **… …**

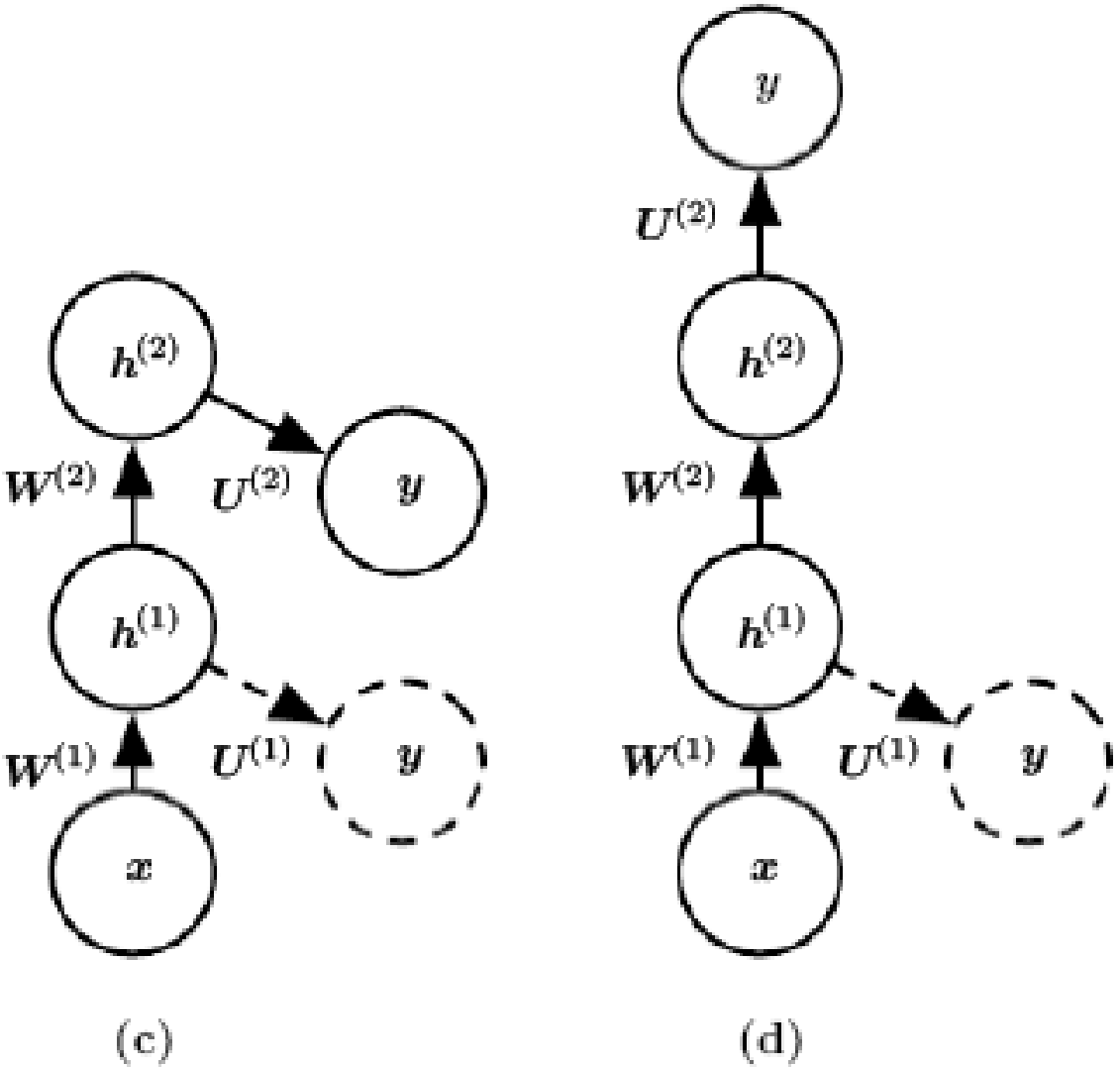
Supervised Pretraining

- **Directly training can be too ambitious if the model is complex and hard to optimize.**
- **Train simple models on simple tasks, then move on to confront the final (complex) task.**

Greedy Supervised Pretraining



Greedy Supervised Pretraining



Outline

- Deep Learning
- Deep Feedforward Networks
- Regularization for Deep Learning
- Optimization for Training Deep Models
- **Summary**

Summary

- **One core goal of Deep Learning is to implement representation Learning of the raw data**

- **The optimal regularization and Optimization are difficult to design**

- **Specific structure of deep neural networks will be helpful for special tasks in certain domains.**

Deep Feedforward Networks

Thank you!

Q&A

Deep Feedforward Networks

Thank you!

Q&A