

Deep Learning I

Supervised Learning

Russ Salakhutdinov

Machine Learning Department
Carnegie Mellon University
Canadian Institute for Advanced Research

**Carnegie
Mellon
University**



Mining for Structure

Massive increase in both computational power and the amount of data available from web, video cameras, laboratory measurements.

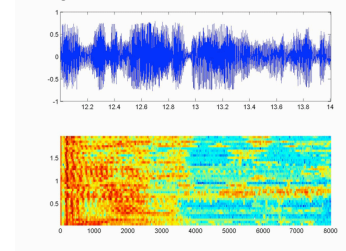
Images & Video



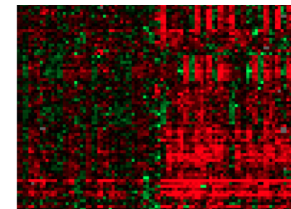
Text & Language



Speech & Audio



Gene Expression



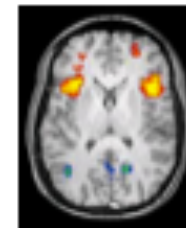
Product Recommendation



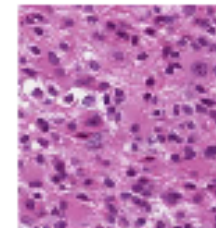
Relational Data/
Social Network



fMRI



Tumor region



- Develop statistical models that can discover underlying structure, cause, or statistical correlation from data in **unsupervised** or **semi-supervised** way.
- Multiple application domains.

Mining for Structure

Massive increase in both computational power and the amount of data available from web, video cameras, laboratory measurements.

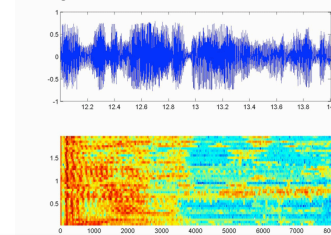
Images & Video



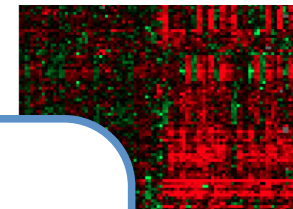
Text & Language



Speech & Audio



Gene Expression

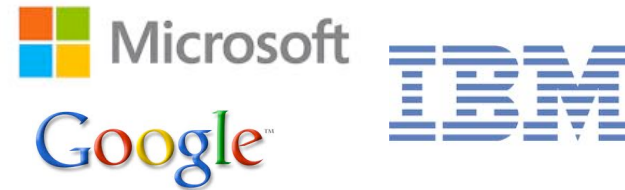


Deep Learning Models that support inferences and discover structure at multiple levels.

- Develop statistical models that can discover underlying structure, cause, or statistical correlation from data in **unsupervised** or **semi-supervised** way.
- Multiple application domains.

Impact of Deep Learning

- Speech Recognition



- Computer Vision



- Recommender Systems



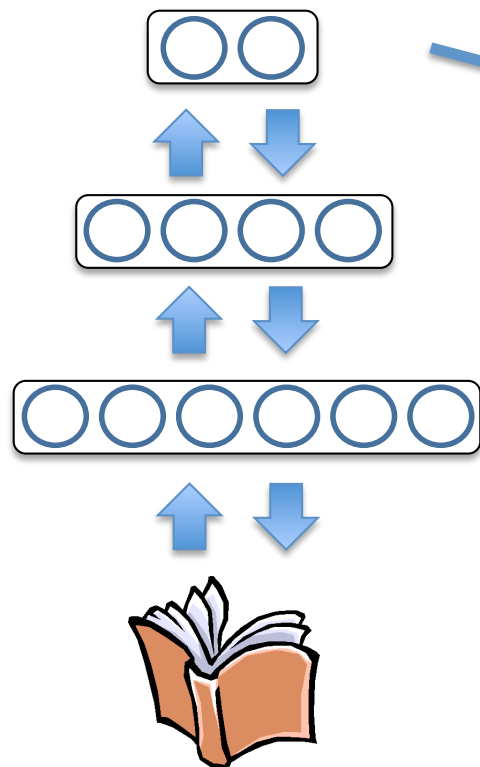
- Language Understanding

- Drug Discovery and Medical
Image Analysis



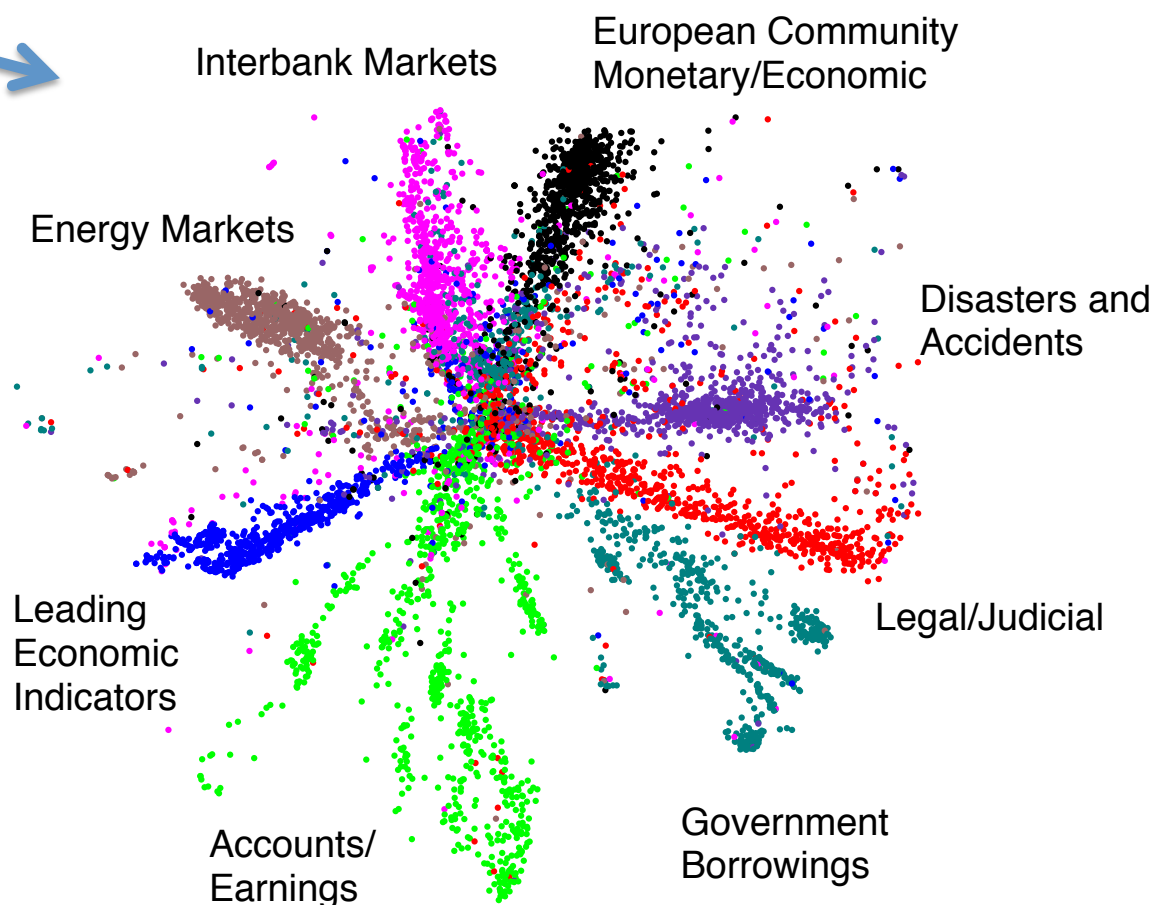
Deep Generative Model

Model $P(\text{document})$



Bag of words

Reuters dataset: 804,414
newswire stories: **unsupervised**



(Hinton & Salakhutdinov, Science 2006)

Example: Understanding Images



TAGS:

strangers, coworkers, conventioners,
attendants, patrons

Nearest Neighbor Sentence:

people taking pictures of a crazy person

Model Samples

- a group of people in a crowded area .
- a group of people are walking and talking .
- a group of people, standing around and talking .

Caption Generation



a car is parked in the middle of nowhere .



a wooden table and chairs arranged in a room .



there is a cat sitting on a shelf .



a ferry boat on a marina with a group of people .



a little boy with a bunch of friends on the street .

Talk Roadmap

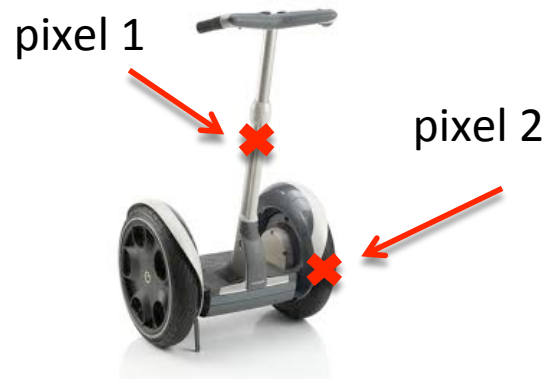
Part 1: Supervised Learning: Deep Networks

- Definition of Neural Networks
- Training Neural Networks
- Recent Optimization / Regularization Techniques

Part 2: Unsupervised Learning: Learning Deep Generative Models

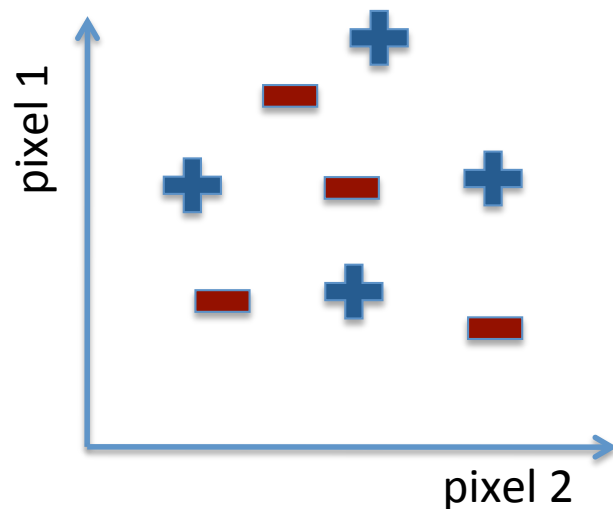
Part 3: Open Research Questions

Learning Feature Representations



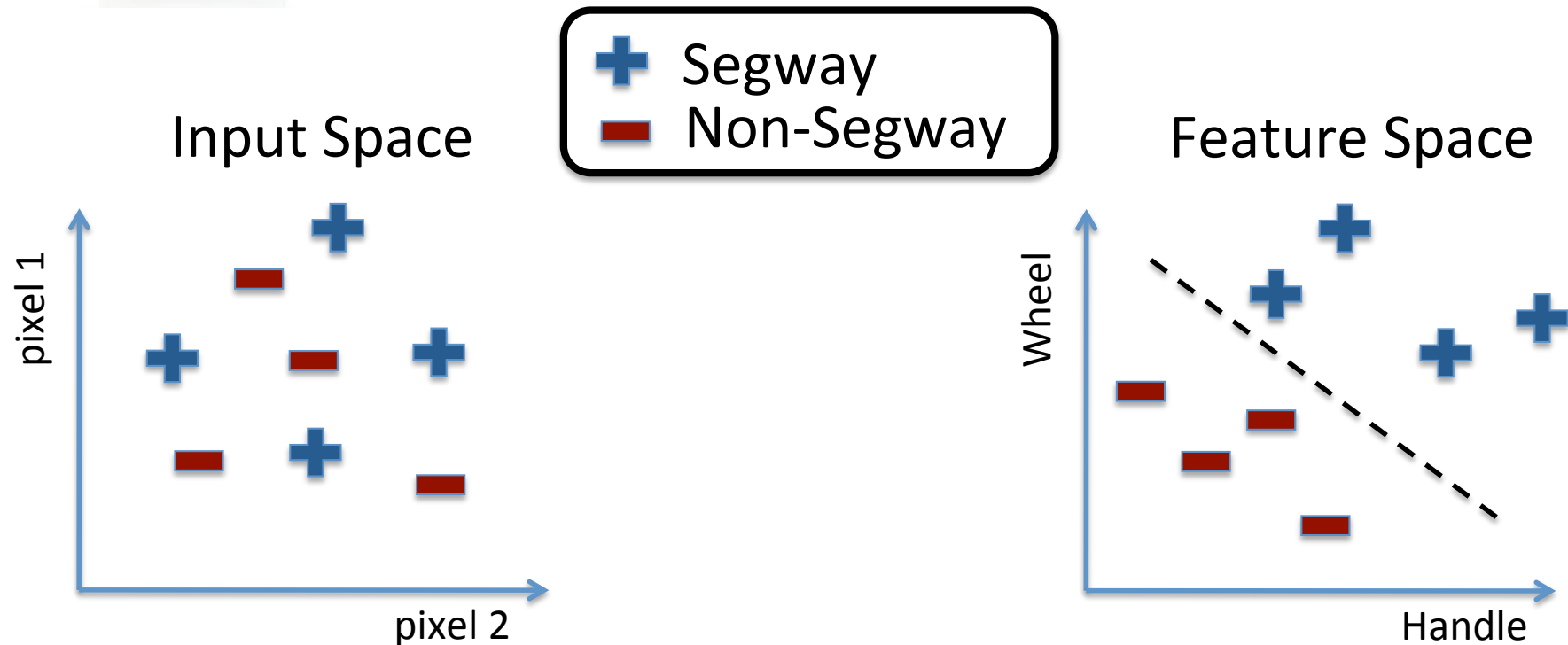
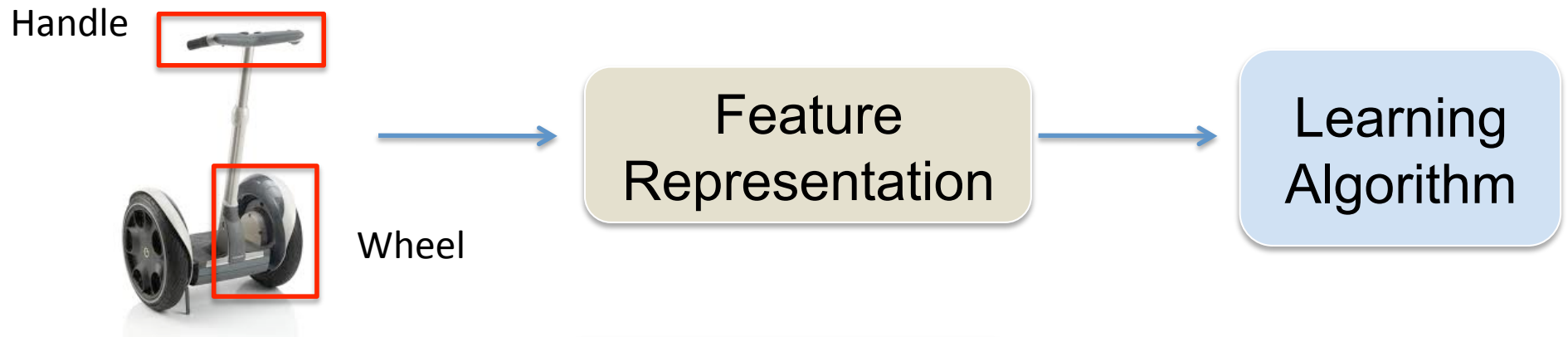
Learning
Algorithm

Input Space



+ Segway
- Non-Segway

Learning Feature Representations



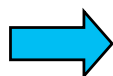
Traditional Approaches



Object
detection



Image



vision features

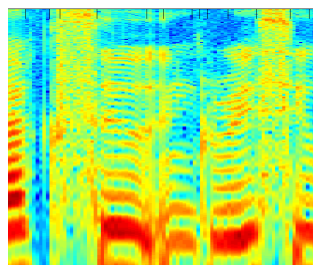
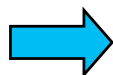


Recognition

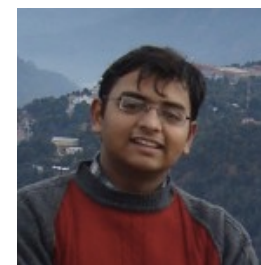
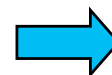
Audio
classification



Audio

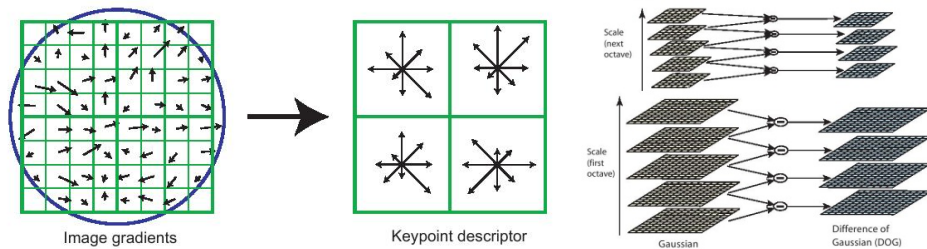


audio features

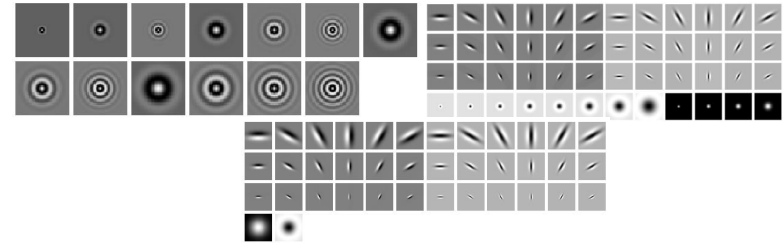


Speaker
identification

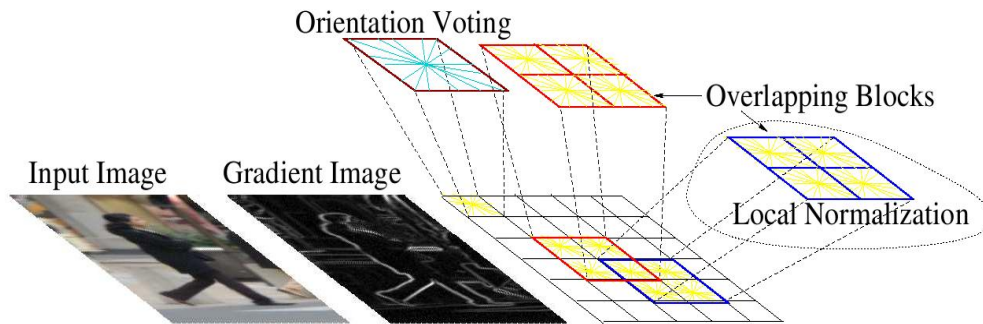
Computer Vision Features



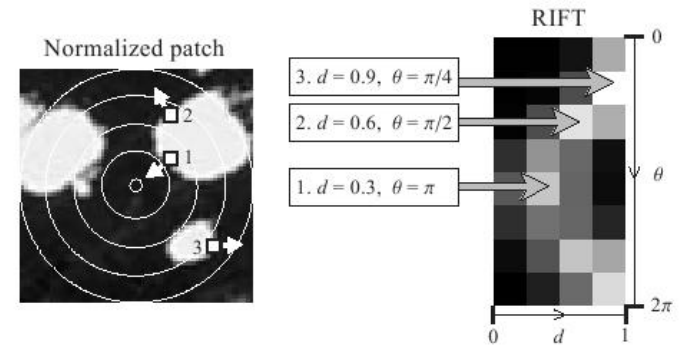
SIFT



Textons

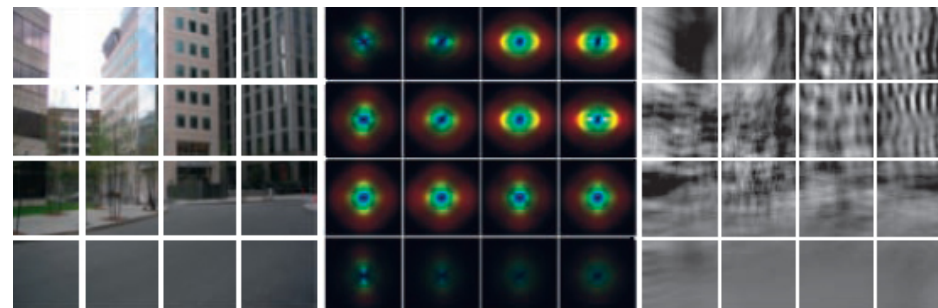


HoG

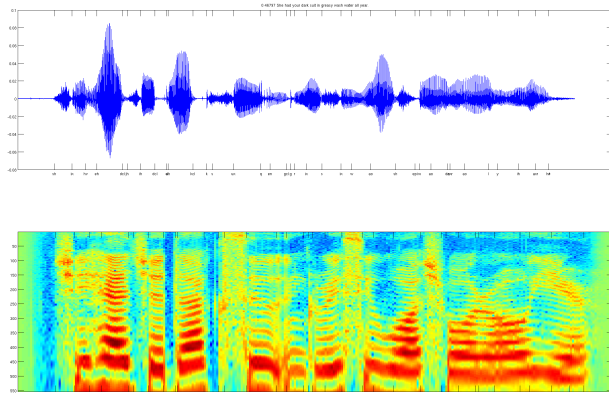


RIFT

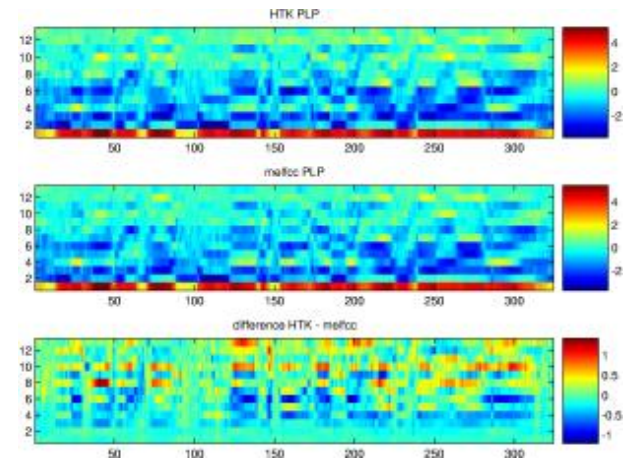
GIST



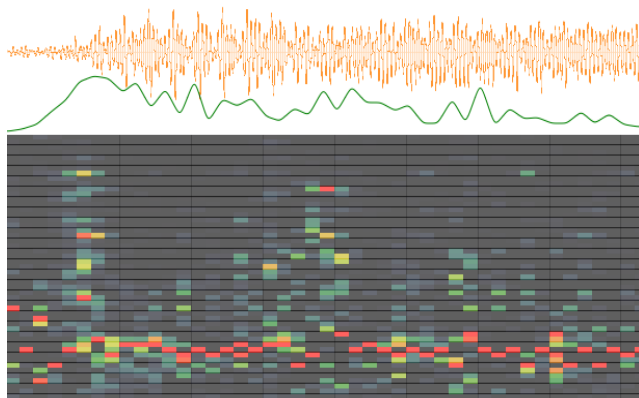
Audio Features



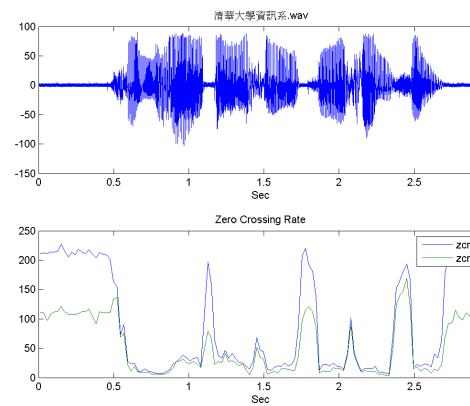
Spectrogram



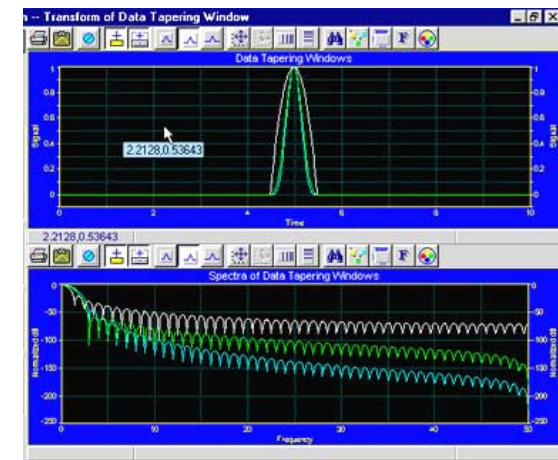
MFCC



Flux

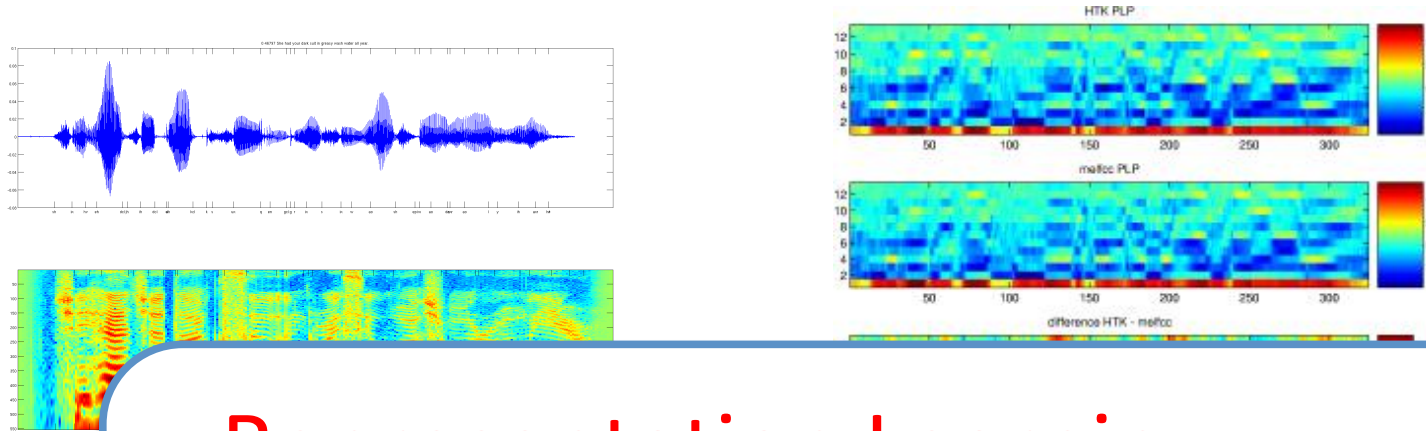


ZCR



Rolloff

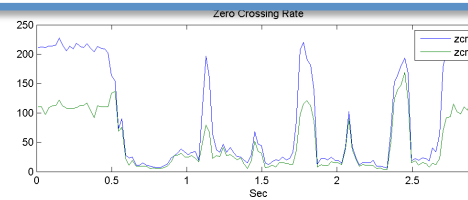
Audio Features



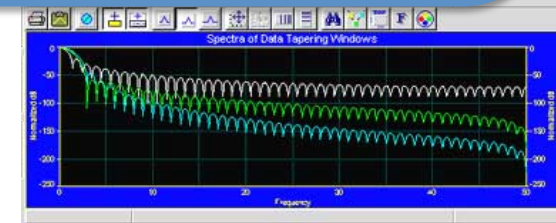
Representation Learning:
Can we automatically learn
these representations?



Flux



ZCR



Rolloff

Neural Networks Online Course

- **Disclaimer:** Some of the material and slides for this lecture were borrowed from Hugo Larochelle's class on Neural Networks:
<https://sites.google.com/site/deeplearningsummerschool2016/>

http://info.usherbrooke.ca/hlarochelle/neural_networks

- Hugo's class covers many other topics: convolutional networks, neural language model, Boltzmann machines, autoencoders, sparse coding, etc.

- We will use his material for some of the other lectures.

RESTRICTED BOLTZMANN MACHINE

Click with the mouse or tablet to draw with pen 2

Topics: RBM, visible layer, hidden layer, energy function

The diagram illustrates the architecture of a Restricted Boltzmann Machine (RBM). It consists of two layers of binary units: a hidden layer (h) and a visible layer (x). The hidden layer units are represented by circles with bias values b_j , and the visible layer units are represented by circles with bias values c_k . The units are connected by weights W , labeled as "connections". A line labeled "bias" points to the bias values b_j and c_k .

Energy function: $E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^T \mathbf{W} \mathbf{x} - \mathbf{c}^T \mathbf{x} - \mathbf{b}^T \mathbf{h}$

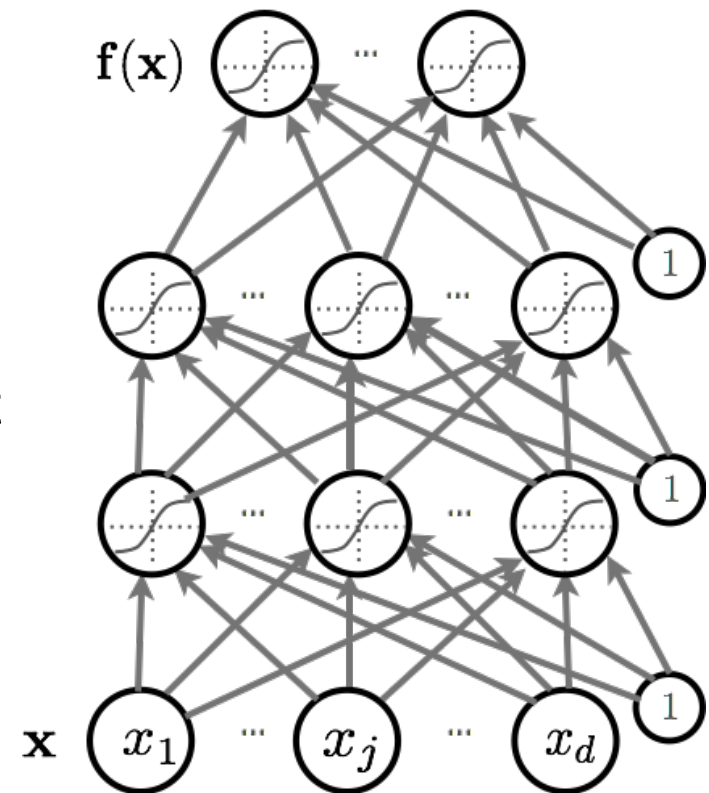
$$= -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j$$

Distribution: $p(\mathbf{x}, \mathbf{h}) = \exp(-E(\mathbf{x}, \mathbf{h})) / Z$ ← partition function (intractable)

A small video inset in the bottom right corner shows a person with glasses and a beard, likely the lecturer, speaking.

Feedforward Neural Networks

- ▶ Definition of Neural Networks
 - Forward propagation
 - Types of units
 - Capacity of neural networks
- ▶ How to train neural nets:
 - Loss function
 - Backpropagation with gradient descent
- ▶ More recent techniques:
 - Dropout
 - Batch normalization
 - Unsupervised Pre-training



Artificial Neuron

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron output activation:

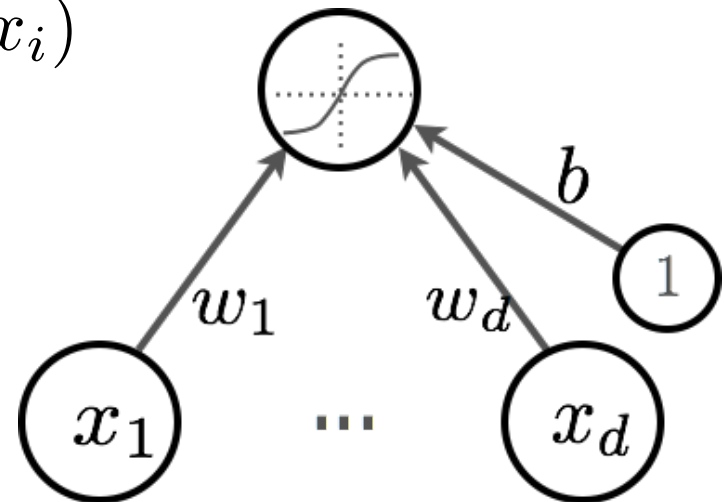
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

where

\mathbf{W} are the weights (parameters)

b is the bias term

$g(\cdot)$ is called the activation function

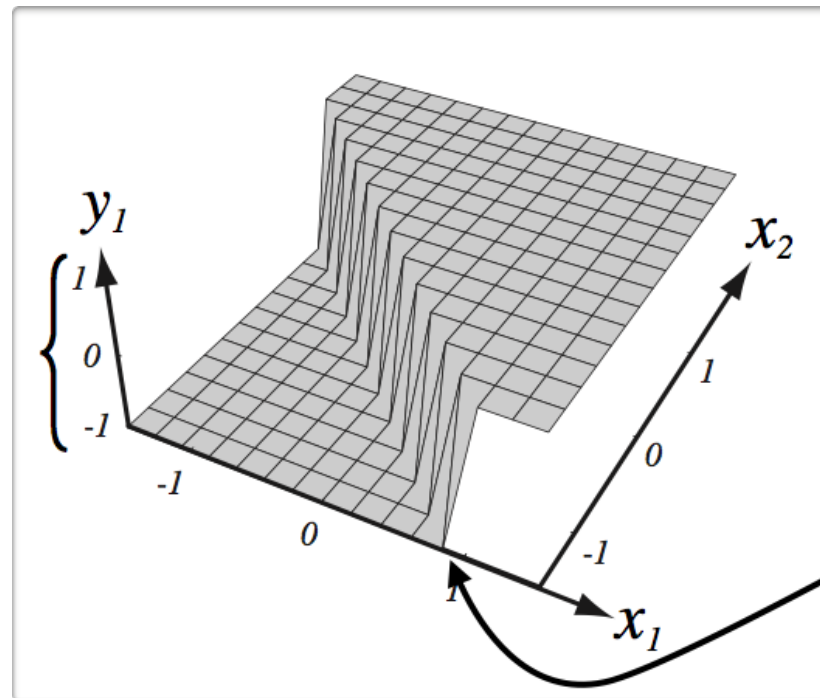


Artificial Neuron

- Output activation of the neuron:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

Range is
determined
by $g(\cdot)$



(from Pascal Vincent's slides)

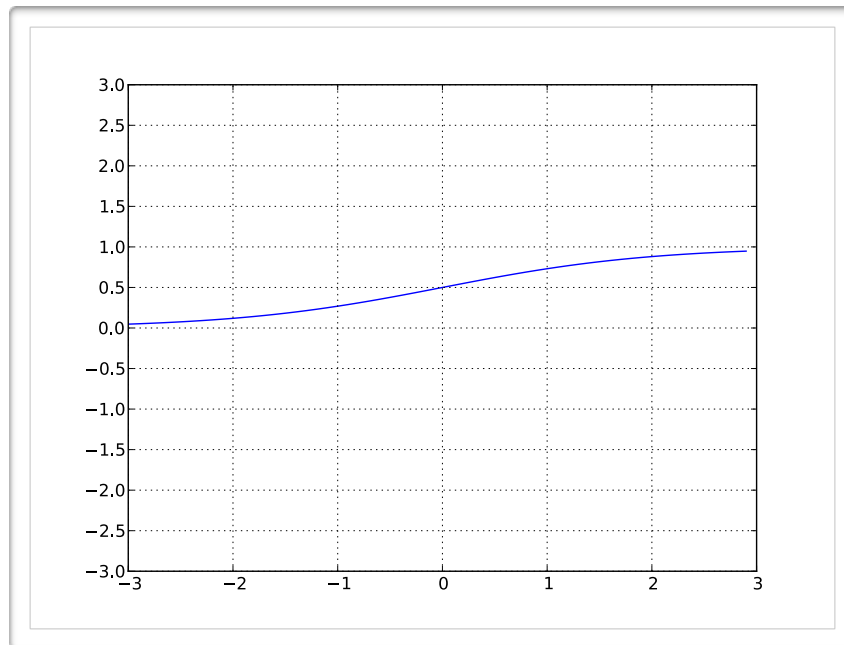
Bias only changes
the position of the
riff

Activation Function

- Sigmoid activation function:

- Squashes the neuron's output between 0 and 1
- Always positive
- Bounded
- Strictly Increasing

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

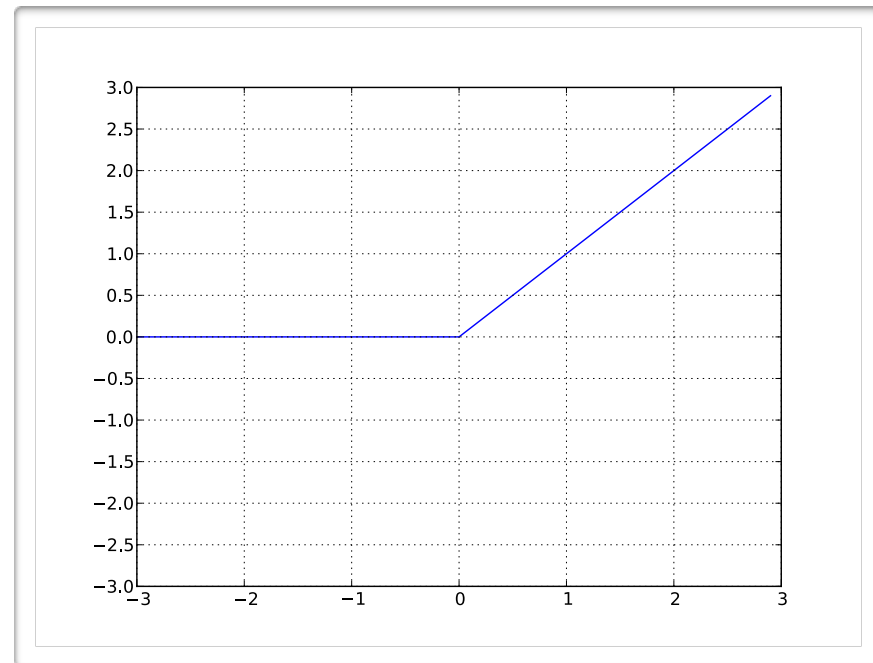


Activation Function

- Rectified linear (ReLU) activation function:

- Bounded below by 0 (always non-negative)
- Tends to produce units with sparse activities
- Not upper bounded
- Strictly increasing

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



Single Hidden Layer Neural Net

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

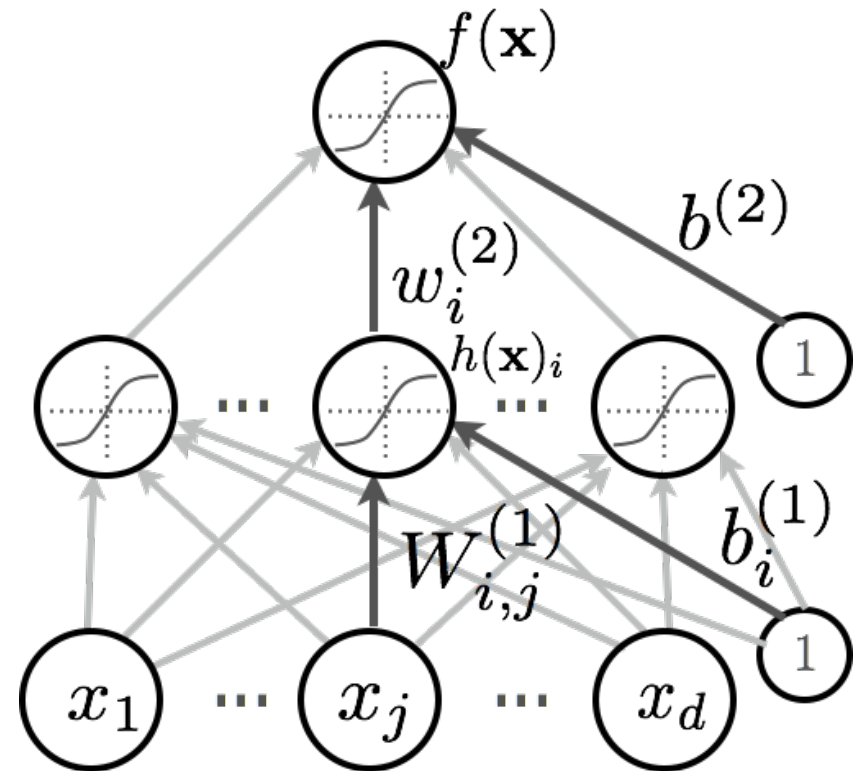
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o \left(b^{(2)} + \mathbf{w}^{(2)\top} \mathbf{h}^{(1)} \mathbf{x} \right)$$

Output activation
function



Multilayer Neural Net

- Consider a network with L hidden layers.

- layer pre-activation for $k > 0$

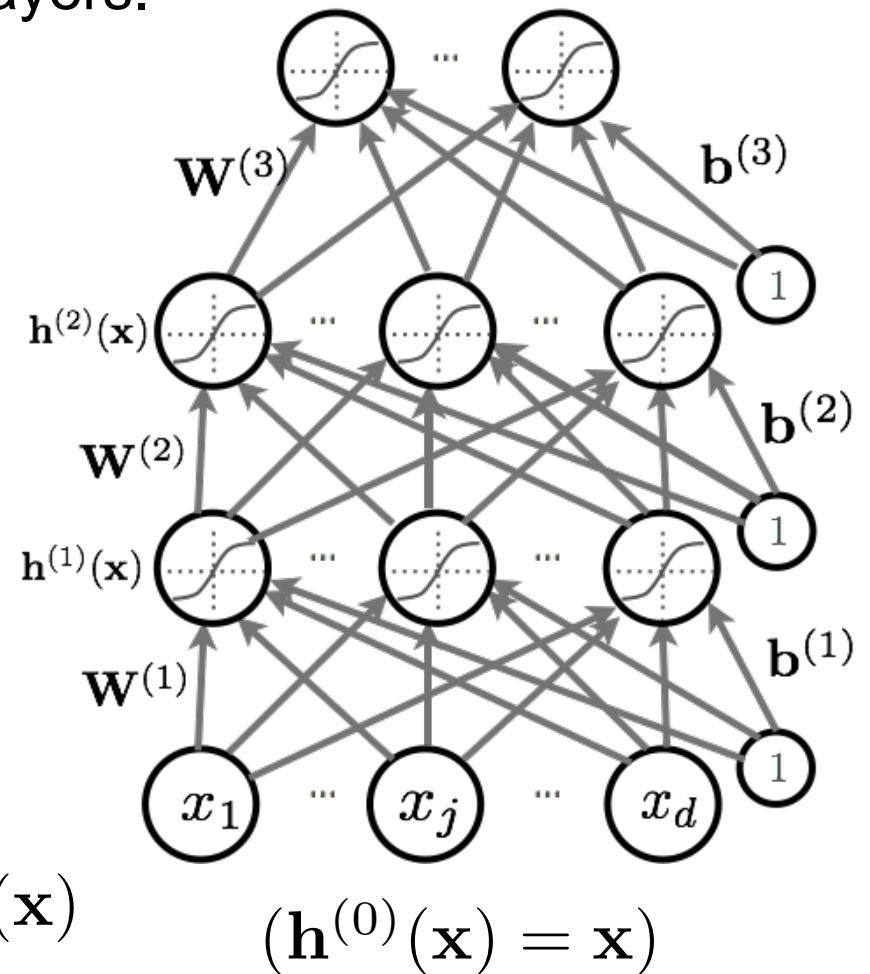
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)} \mathbf{h}^{(k-1)}(\mathbf{x})$$

- hidden layer activation from 1 to L :

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

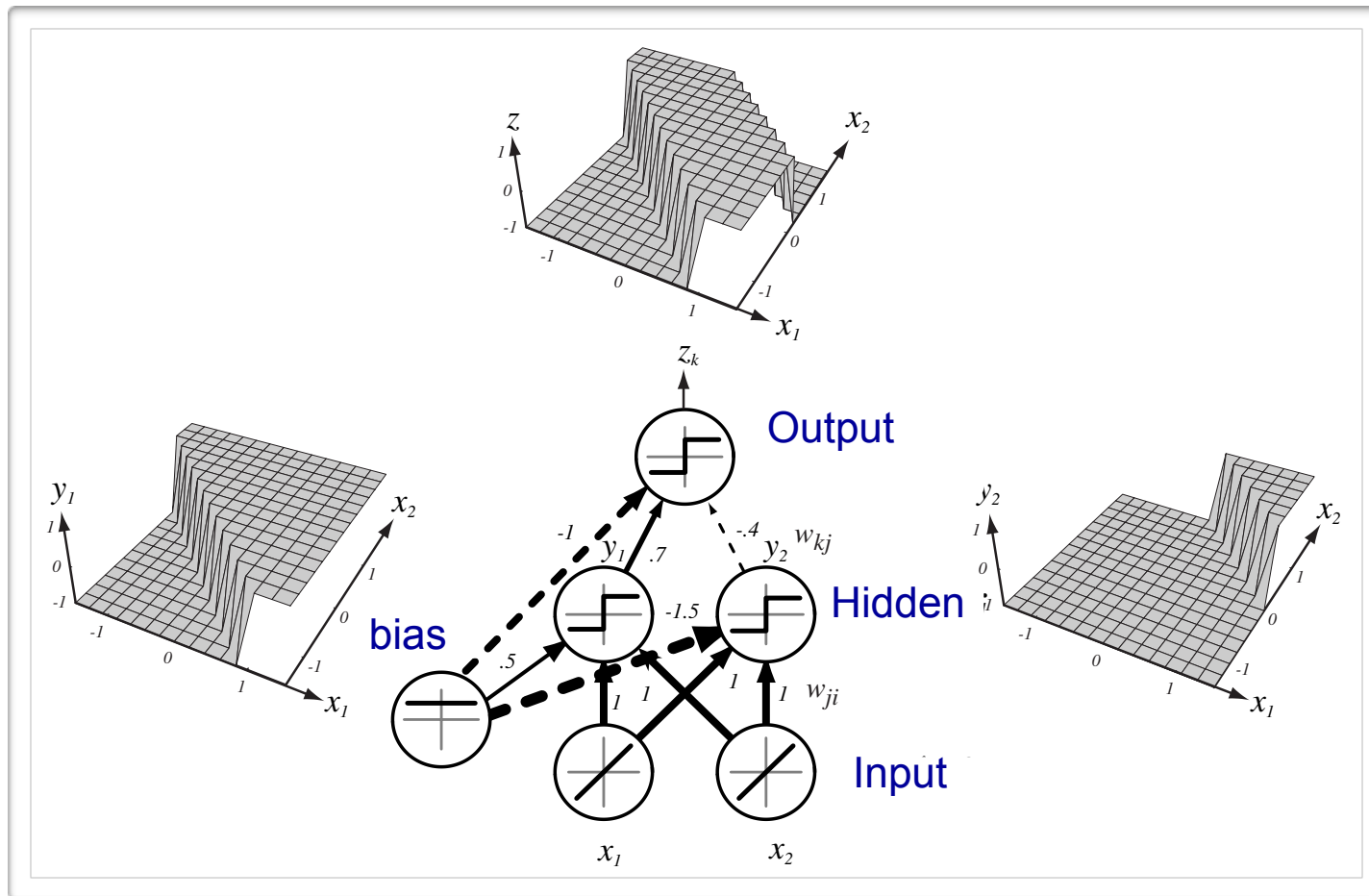
- output layer activation ($k=L+1$):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Capacity of Neural Nets

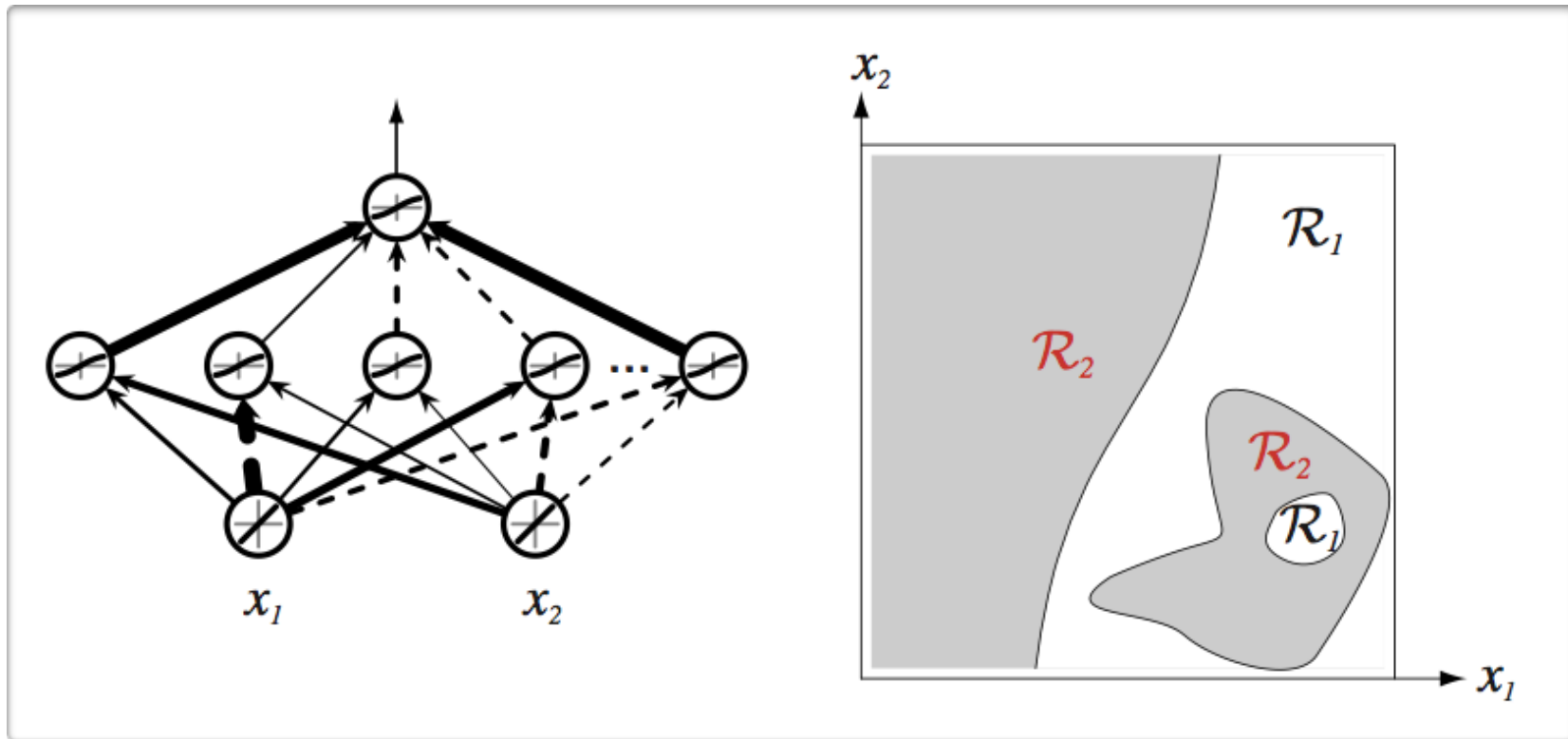
- Consider a single layer neural network



(from Pascal Vincent's slides)

Capacity of Neural Nets

- Consider a single layer neural network



(from Pascal Vincent's slides)

Universal Approximation

- Universal Approximation Theorem (Hornik, 1991):
 - “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”
- This applies for sigmoid, tanh and many other activation functions.
- However, this does not mean that there is learning algorithm that can find the necessary parameter values.

Feedforward Neural Networks

- ▶ How neural networks predict $f(\mathbf{x})$ given an input \mathbf{x} :

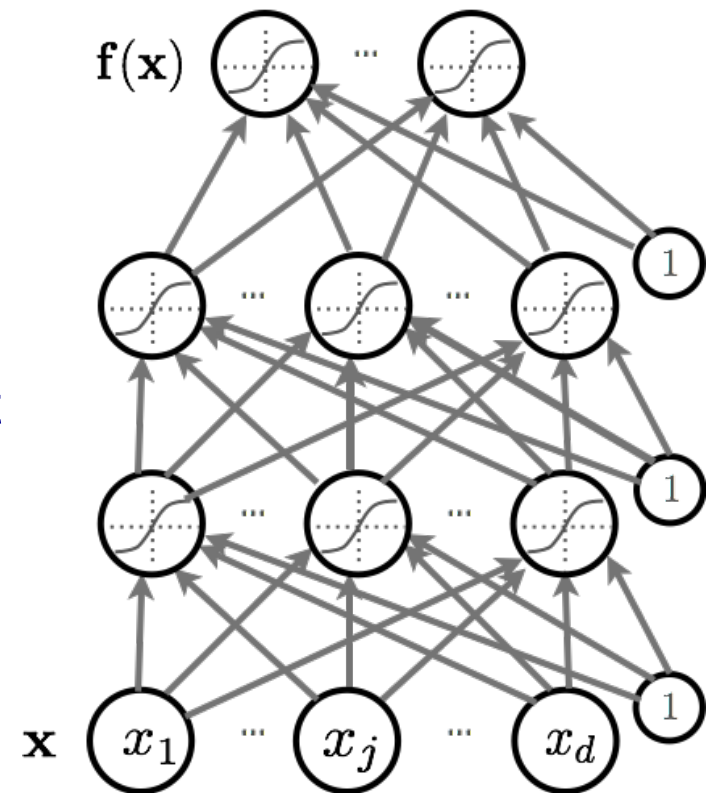
- Forward propagation
- Types of units
- Capacity of neural networks

- ▶ How to train neural nets:

- Loss function
- Backpropagation with gradient descent

- ▶ More recent techniques:

- Dropout
- Batch normalization
- Unsupervised Pre-training



Training

- Empirical Risk Minimization:

$$\arg \min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \underbrace{\lambda \Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

- Learning is cast as optimization.
 - For classification problems, we would like to minimize classification error.
 - Loss function can sometimes be viewed as **a surrogate for what we want to optimize** (e.g. upper bound)

Stochastic Gradient Descend

- Perform updates after seeing each example:

- Initialize: $\theta \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$

- For $t=1:T$

- for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)$$

$$\theta \leftarrow \theta + \alpha \Delta$$

} Training epoch
=
Iteration of all examples

- To train a neural net, we need:

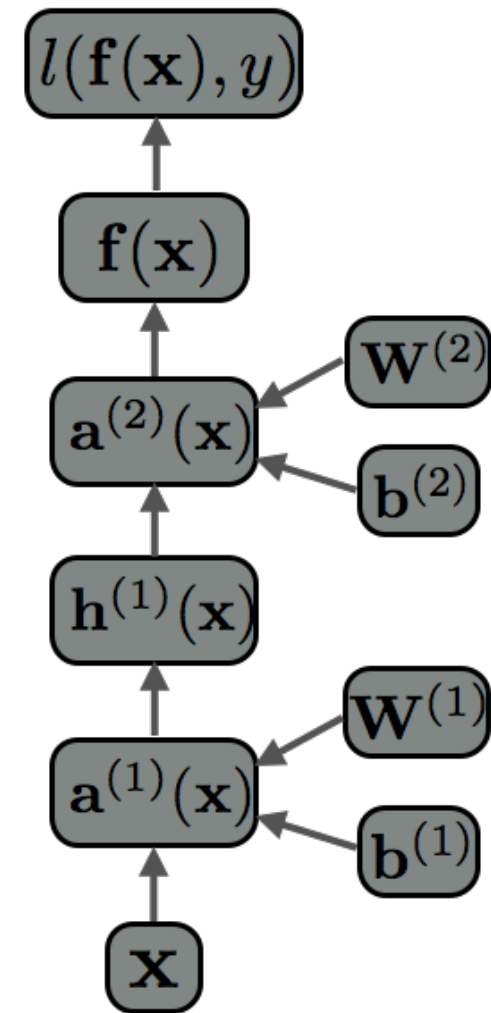
- **Loss function:** $l(\mathbf{f}(\mathbf{x}^{(t)}; \theta), y^{(t)})$

- A procedure to **compute gradients:** $\nabla_{\theta} l(\mathbf{f}(\mathbf{x}^{(t)}; \theta), y^{(t)})$

- **Regularizer** and its gradient: $\Omega(\theta), \nabla_{\theta} \Omega(\theta)$

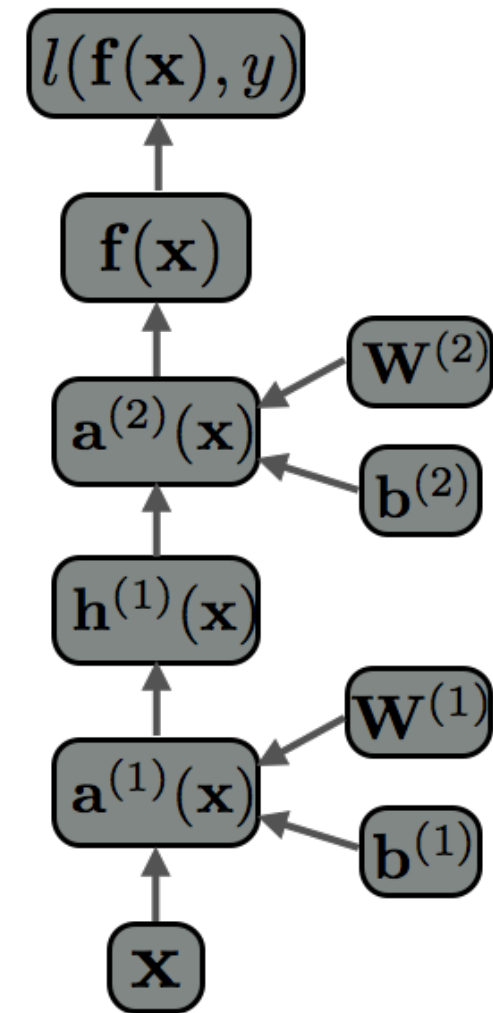
Computational Flow Graph

- Forward propagation can be represented as an acyclic flow graph
- Forward propagation can be implemented in a modular way:
 - Each box can be an object with an **fprop method**, that computes the value of the box given its children
 - Calling the fprop method of each box in the right order yields forward propagation



Computational Flow Graph

- Each object also has a **bprop method**
 - it computes the gradient of the loss with respect to each child box.
- By calling bprop in the **reverse order**, we obtain backpropagation

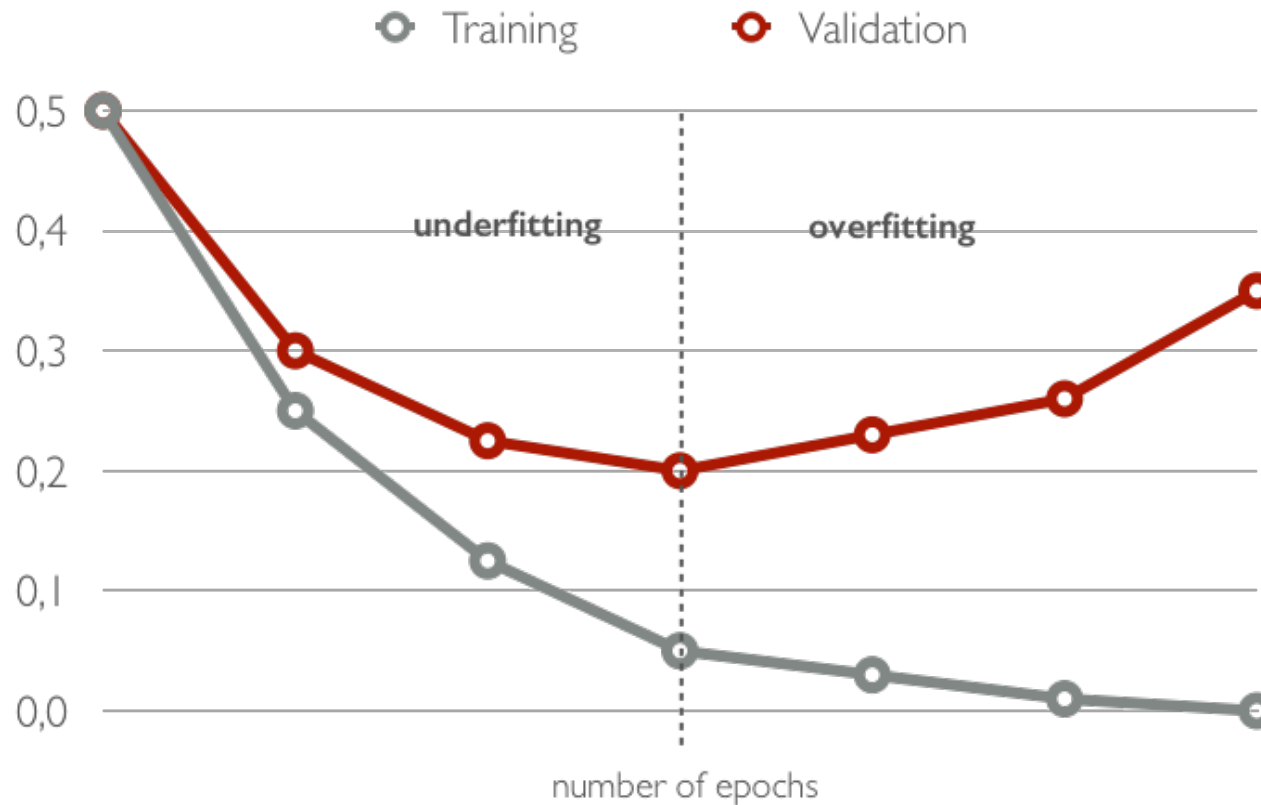


Model Selection

- Training Protocol:
 - Train your model on the **Training Set** $\mathcal{D}^{\text{train}}$
 - For model selection, use **Validation Set** $\mathcal{D}^{\text{valid}}$
 - Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
 - Estimate generalization performance using the **Test Set** $\mathcal{D}^{\text{test}}$
- Generalization is the behavior of the model on **unseen examples**.

Early Stopping

- To select the number of epochs, stop training when validation set error increases (with some look ahead).



Mini-batch, Momentum

- Make updates based on a mini-batch of examples (instead of a single example):

- the gradient is the average regularized loss for that mini-batch
- can give a more accurate estimate of the gradient
- can leverage matrix/matrix operations, which are more efficient

- **Momentum**: Can use an exponential average of previous gradients:

$$\overline{\nabla}_{\theta}^{(t)} = \nabla_{\theta} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\theta}^{(t-1)}$$

- can get pass plateaus more quickly, by “gaining momentum”

Feedforward Neural Networks

- ▶ How neural networks predict $f(\mathbf{x})$ given an input \mathbf{x} :

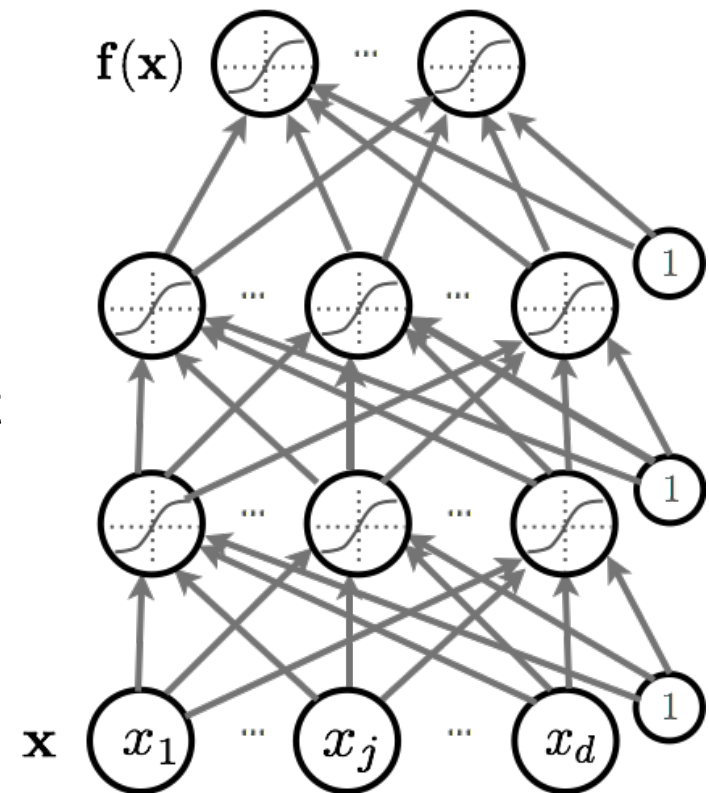
- Forward propagation
- Types of units
- Capacity of neural networks

- ▶ How to train neural nets:

- Loss function
- Backpropagation with gradient descent

- ▶ More recent techniques:

- Dropout
- Batch normalization
- Unsupervised Pre-training



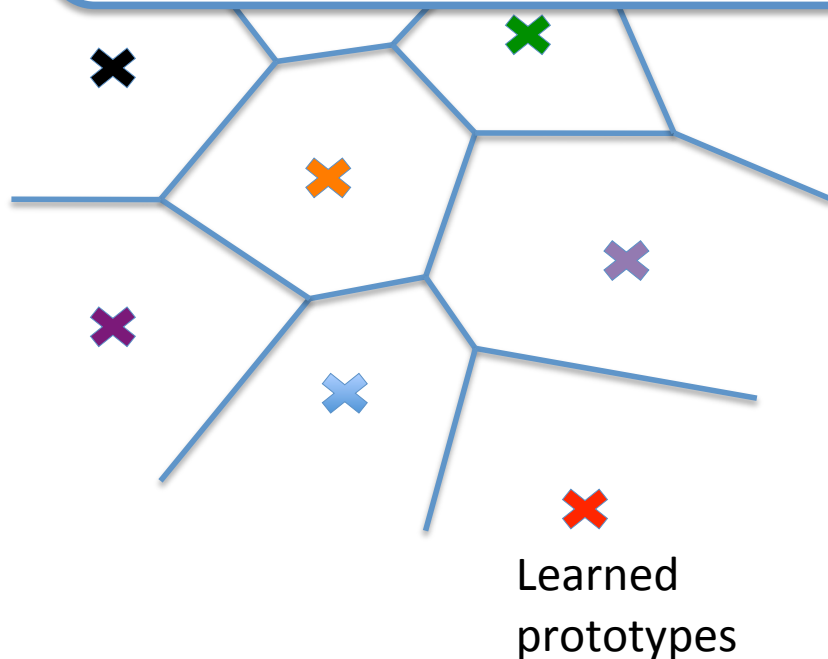
Learning Distributed Representations

- Deep learning is research on learning models with **multilayer representations**
 - multilayer (feed-forward) neural networks
 - multilayer graphical model (deep belief network, deep Boltzmann machine)
- Each layer learns “**distributed representation**”
 - Units in a layer are not mutually exclusive
 - each unit is a separate feature of the input
 - two units can be “active” at the same time
 - Units do not correspond to a partitioning (clustering) of the inputs
 - in clustering, an input can only belong to a single cluster

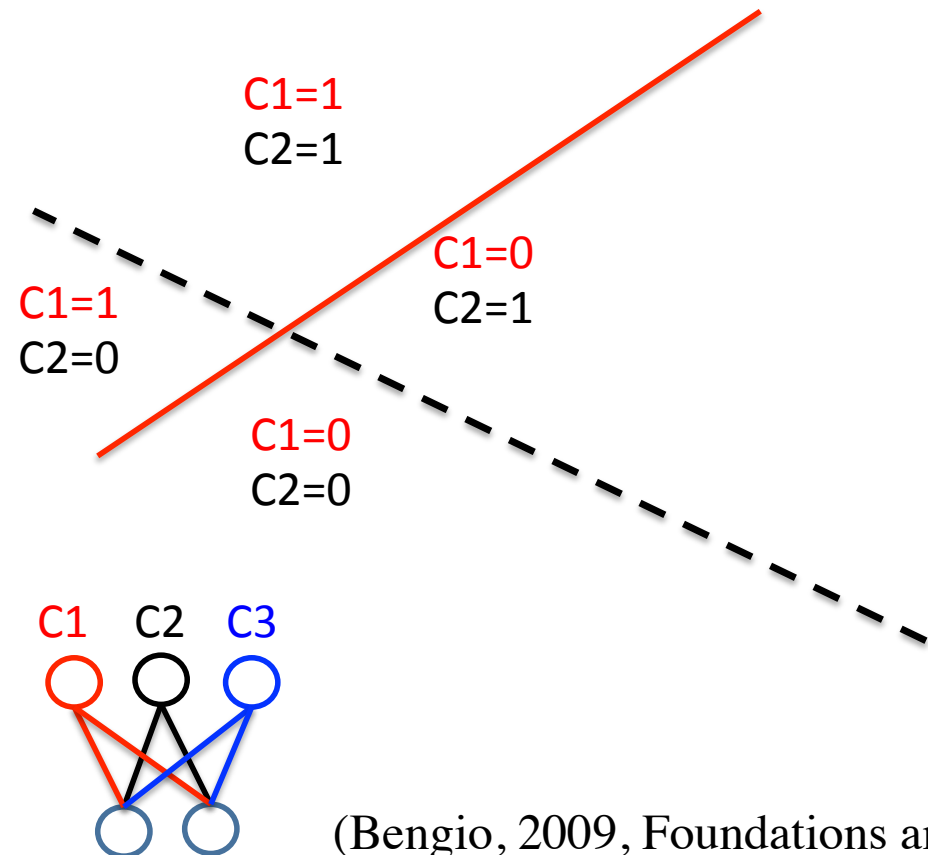
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators

- Parameters for each region.
- # of regions is linear with # of parameters.



- RBMs, Factor models, PCA, Sparse Coding, Deep models

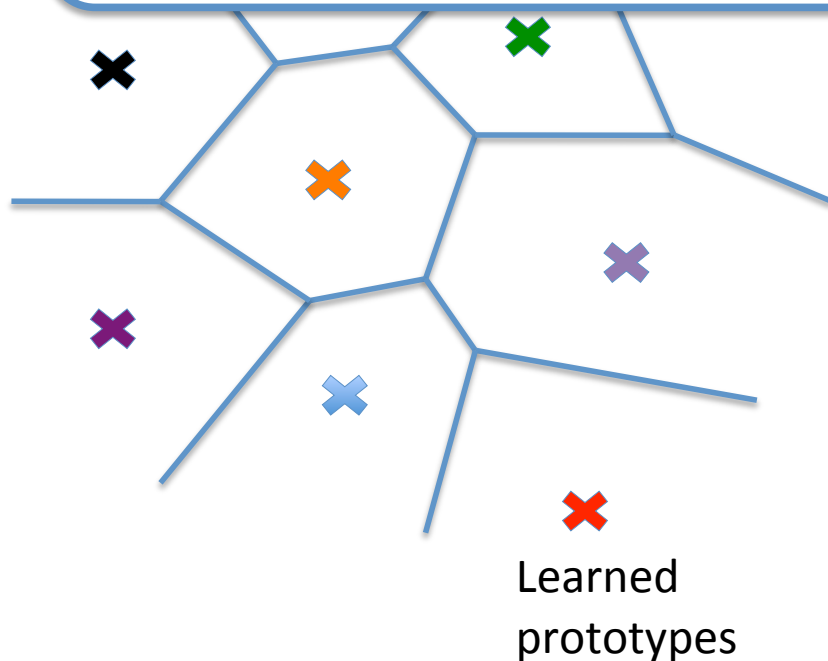


(Bengio, 2009, Foundations and Trends in Machine Learning)

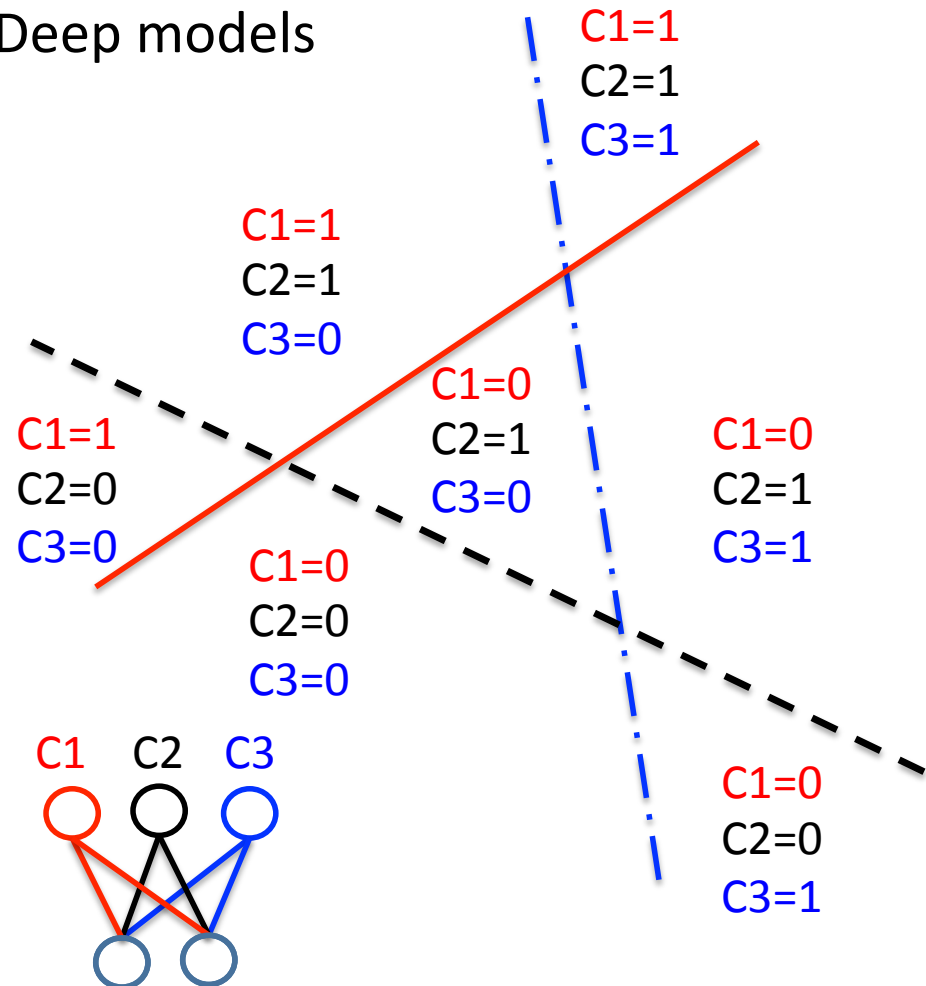
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators

- Parameters for each region.
- # of regions is linear with # of parameters.



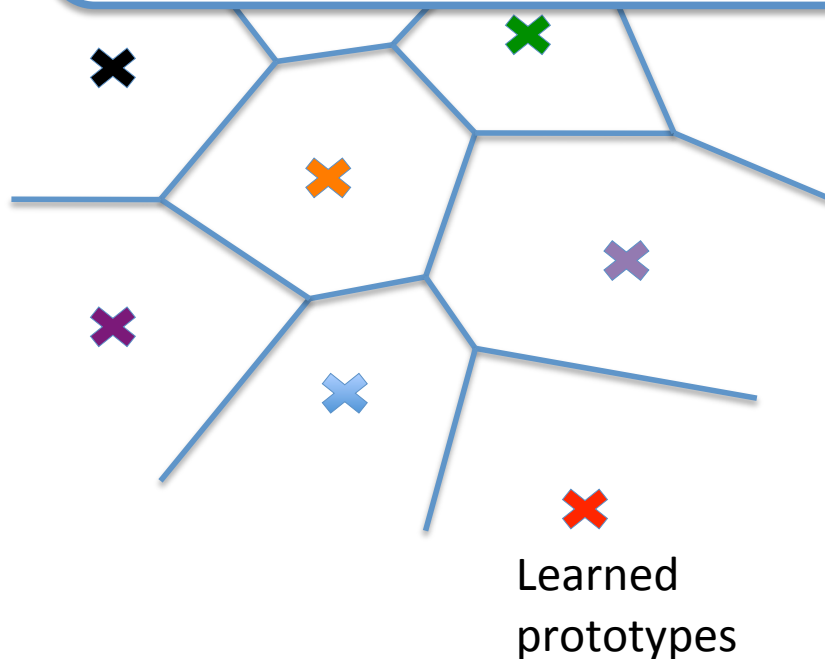
- RBMs, Factor models, PCA, Sparse Coding, Deep models



Local vs. Distributed Representations

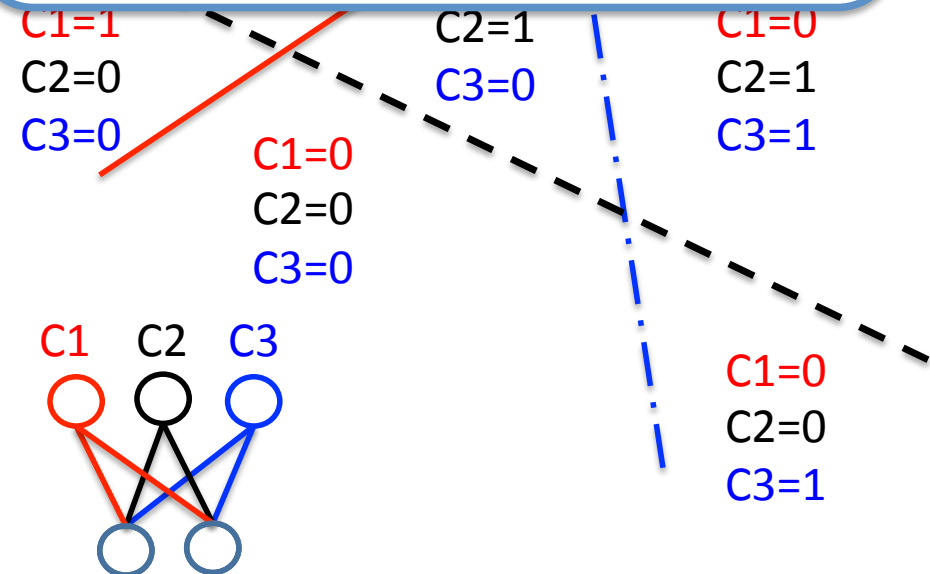
- Clustering, Nearest Neighbors, RBF SVM, local density estimators

- Parameters for each region.
- # of regions is linear with # of parameters.

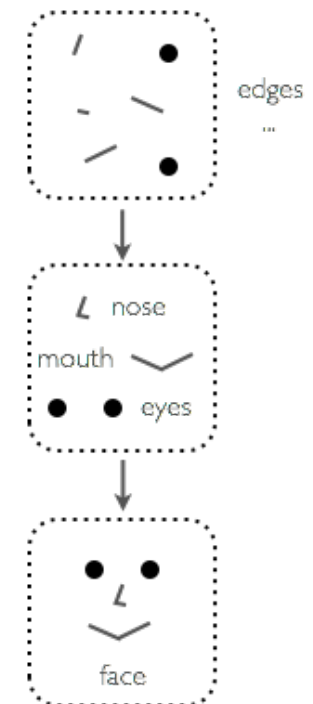
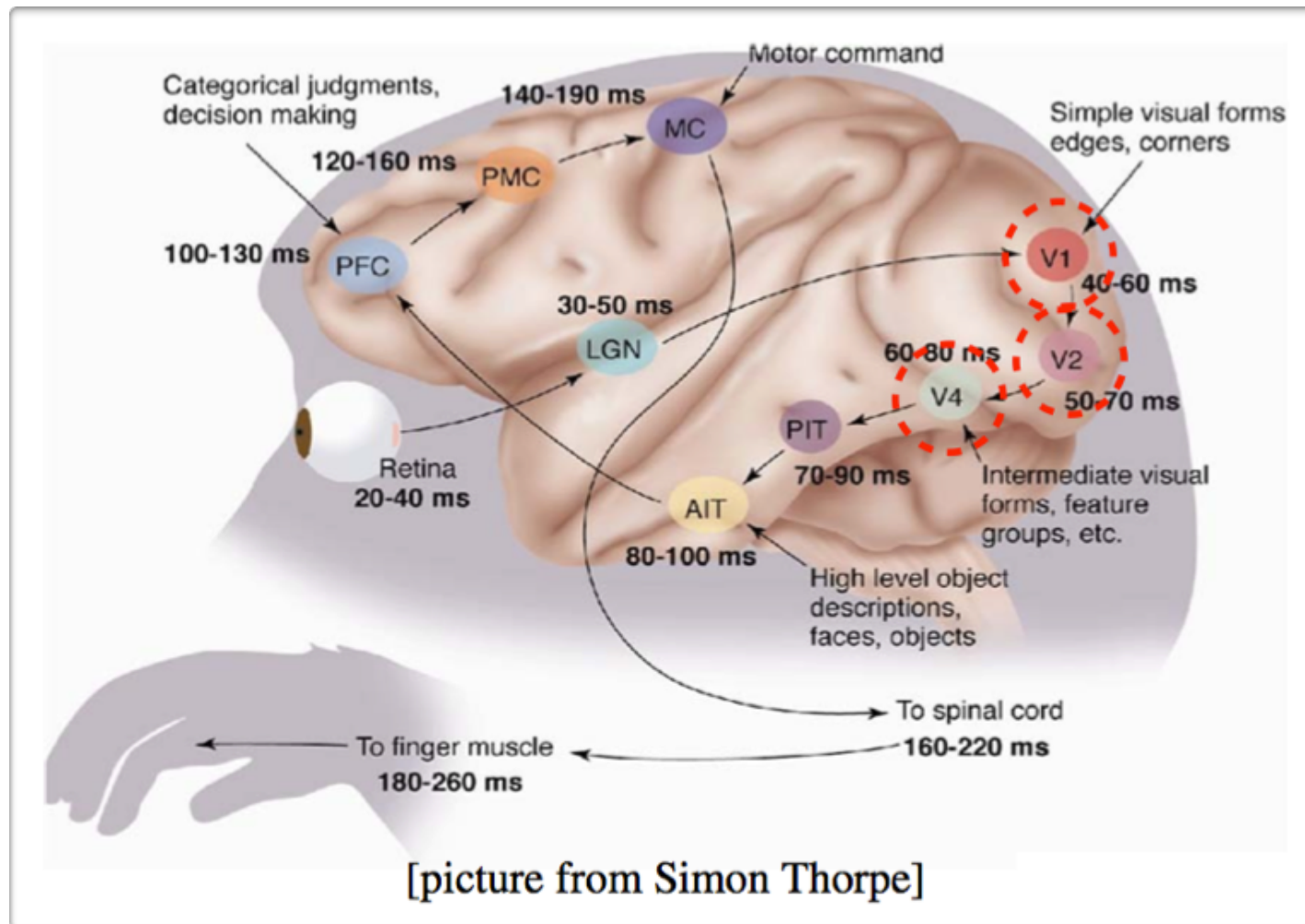


- RBMs, Factor models, PCA, Sparse Coding, Deep models

- Each parameter affects many regions, not just local.
- # of regions grows (roughly) exponentially in # of parameters.

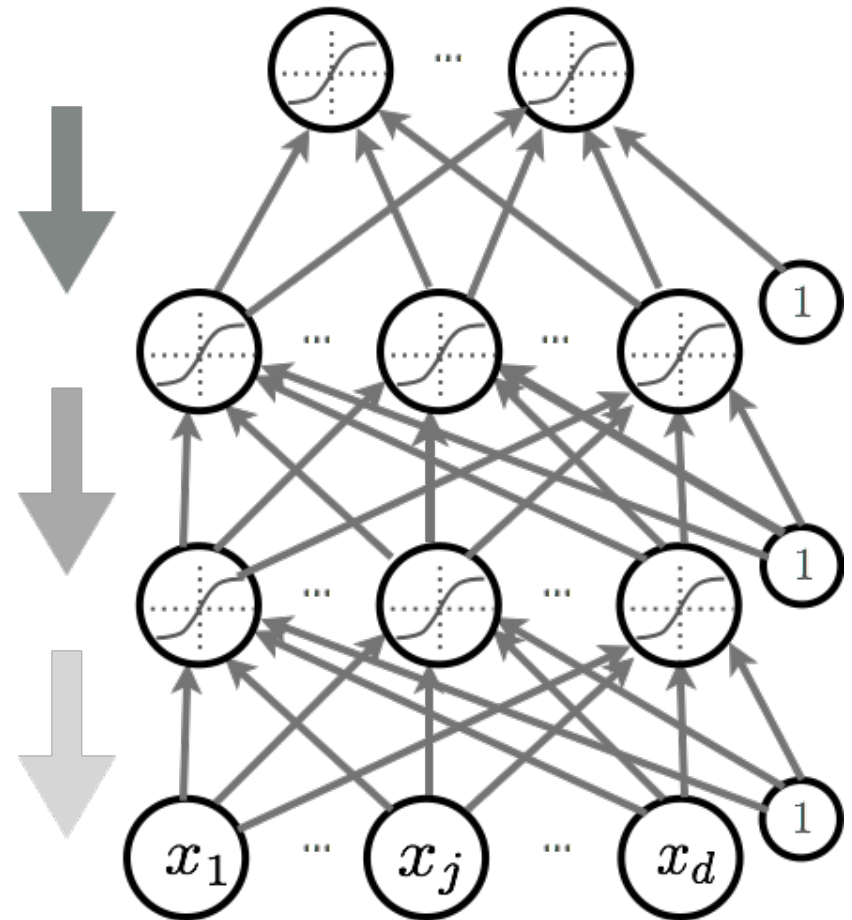


Inspiration from Visual Cortex



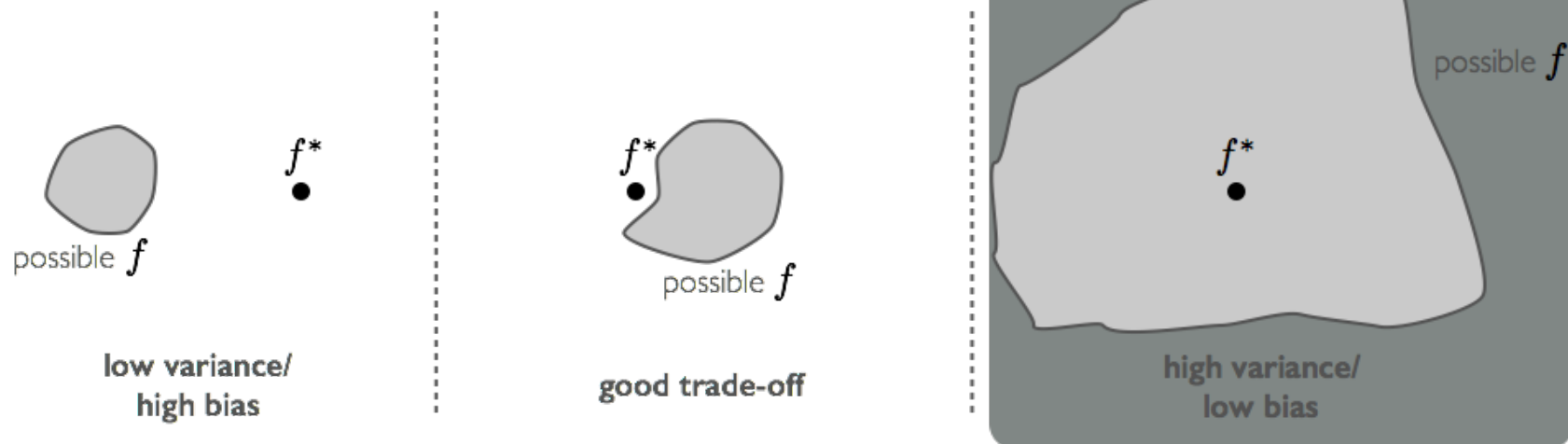
Why Training is Hard

- First hypothesis: **Hard optimization problem** (underfitting)
 - vanishing gradient problem
 - saturated units block gradient propagation
- This is a well known problem in recurrent neural networks



Why Training is Hard

- Second hypothesis: Overfitting
 - we are exploring a space of complex functions
 - deep nets usually have lots of parameters
- Might be in a high variance / low bias situation

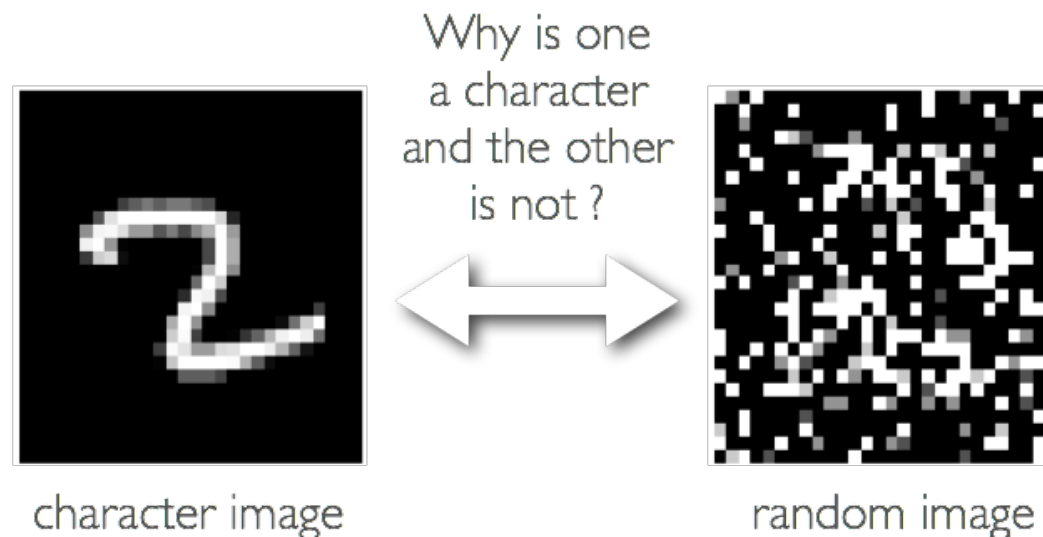


Why Training is Hard

- First hypothesis (**underfitting**): better optimize
 - Use better optimization tools (e.g. batch-normalization, second order methods, such as KFAC)
 - Use GPUs, distributed computing.
- Second hypothesis (**overfitting**): use better regularization
 - Unsupervised pre-training
 - Stochastic drop-out training
- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

Unsupervised Pre-training

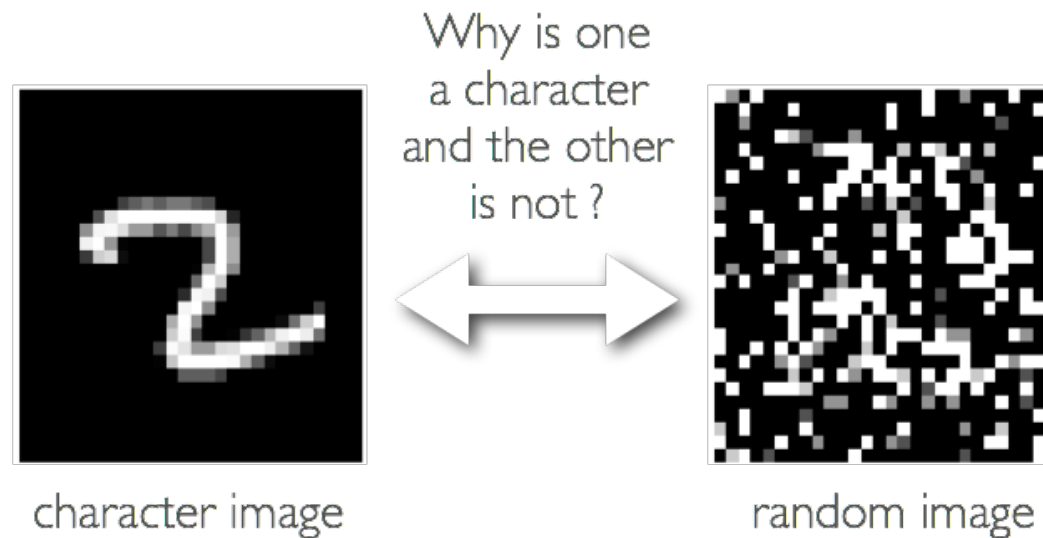
- Initialize hidden layers using **unsupervised learning**
 - Force network to represent latent structure of input distribution



- Encourage hidden layers to encode that structure

Unsupervised Pre-training

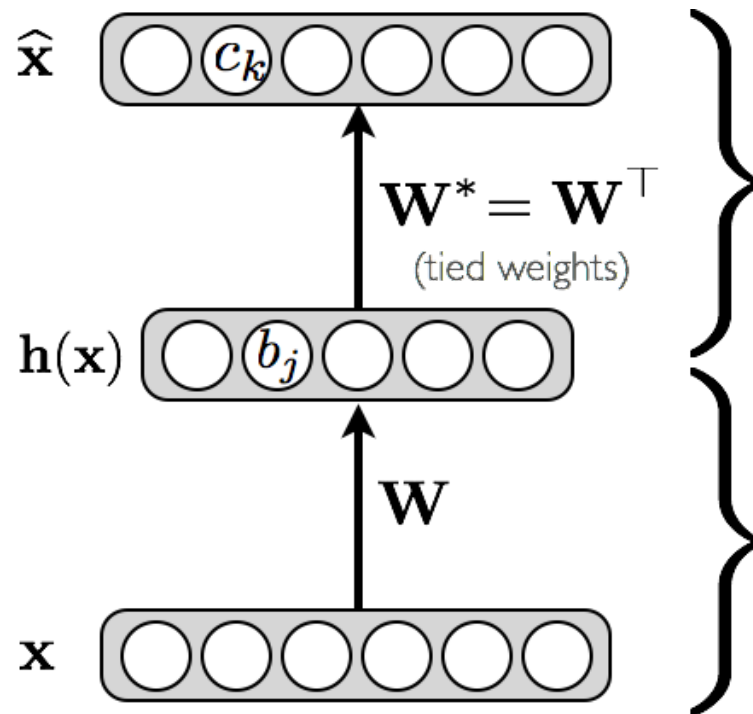
- Initialize hidden layers using **unsupervised learning**
 - This is a harder task than supervised learning (classification)



- Hence we expect less overfitting

Autoencoders: Preview

- Feed-forward neural network trained to reproduce its input at the output layer



Decoder

$$\begin{aligned}\hat{\mathbf{x}} &= o(\hat{\mathbf{a}}(\mathbf{x})) \\ &= \text{sigm}(\underbrace{\mathbf{c} + \mathbf{W}^* \mathbf{h}(\mathbf{x})}_{\text{For binary units}})\end{aligned}$$

Encoder

$$\begin{aligned}\mathbf{h}(\mathbf{x}) &= g(\mathbf{a}(\mathbf{x})) \\ &= \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})\end{aligned}$$

Autoencoders: Preview

- Loss function for **binary inputs**

$$l(f(\mathbf{x})) = - \sum_k (x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k))$$

- Cross-entropy error function $f(\mathbf{x}) \equiv \hat{\mathbf{x}}$

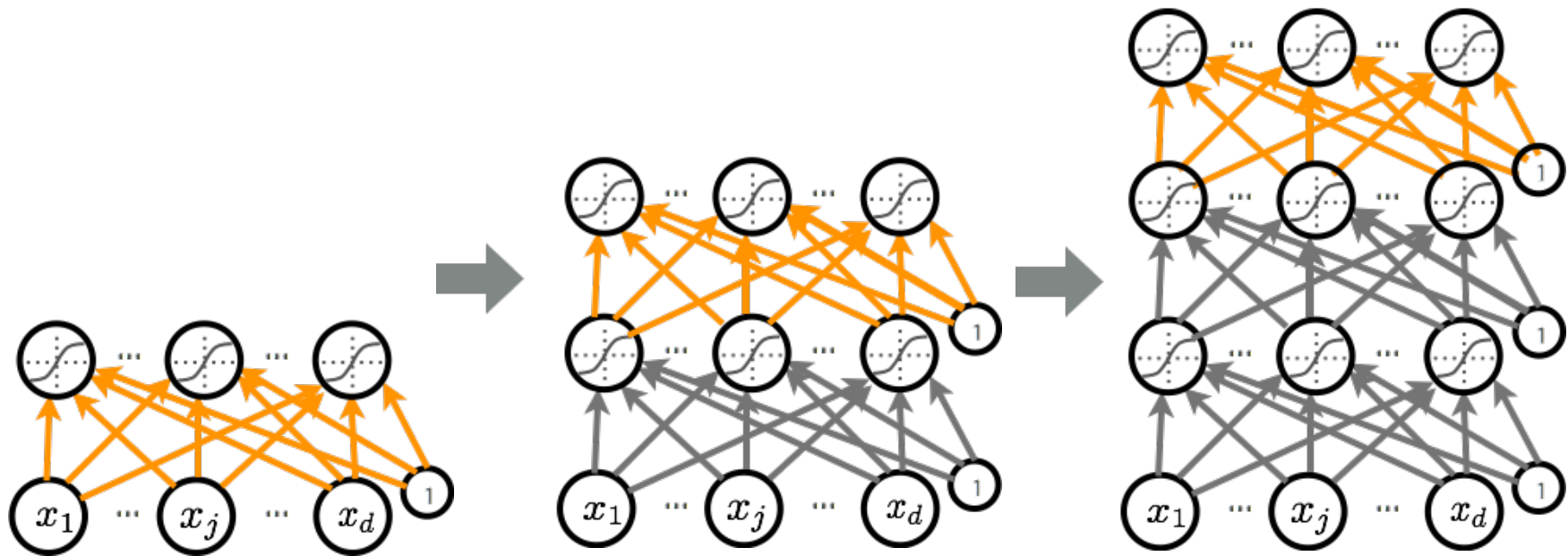
- Loss function for **real-valued inputs**

$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2$$

- sum of squared differences
- we use a linear activation function at the output

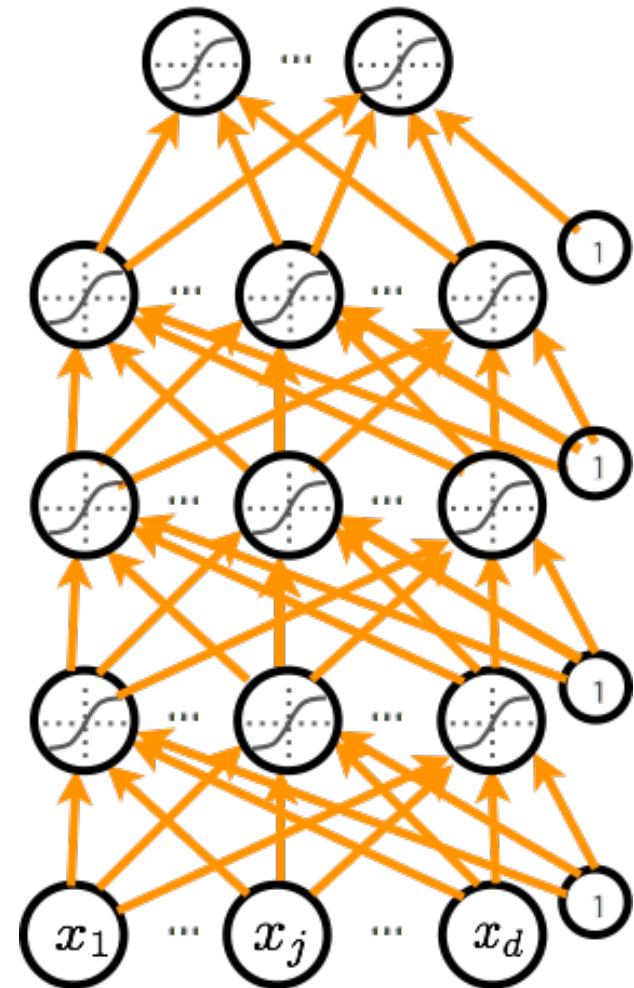
Pre-training

- We will use a greedy, layer-wise procedure
 - Train one layer at a time with unsupervised criterion
 - Fix the parameters of previous hidden layers
 - Previous layers can be viewed as feature extraction



Fine-tuning

- Once all layers are pre-trained
 - add output layer
 - train the whole network using supervised learning
- We call this last phase **fine-tuning**
 - all parameters are “tuned” for the supervised task at hand
 - representation is adjusted to be more discriminative



Why Training is Hard

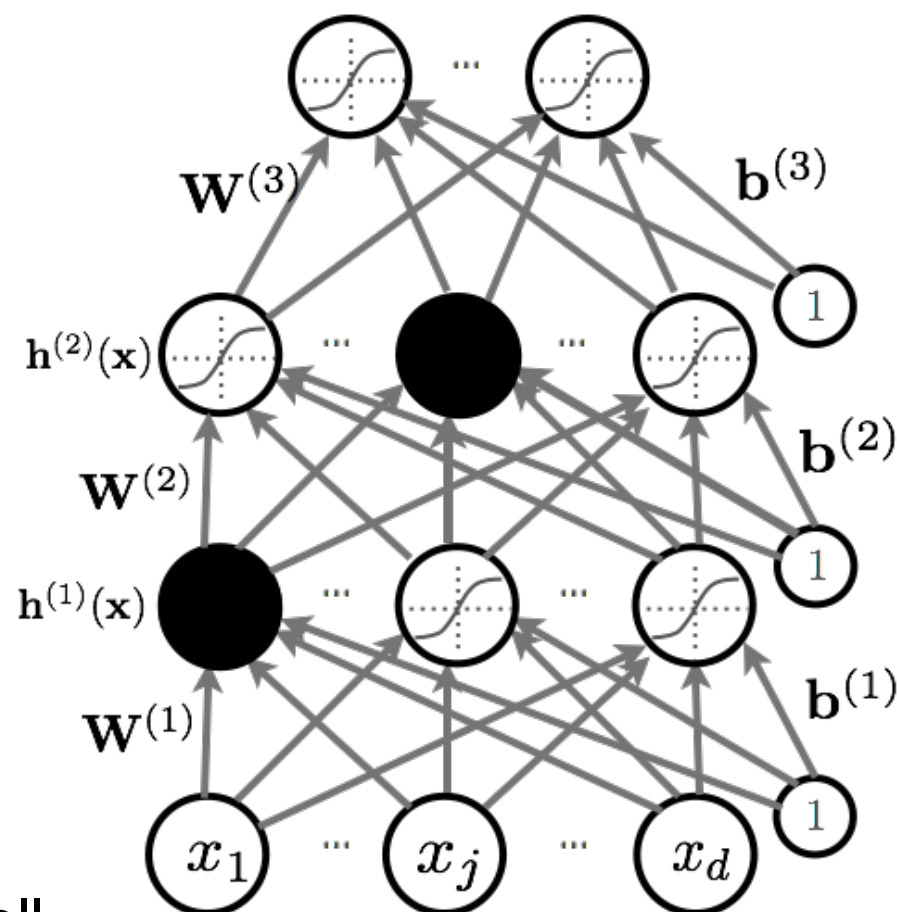
- First hypothesis (underfitting): better optimize
 - Use better optimization tools (e.g. batch-normalization, second order methods, such as KFAC)
 - Use GPUs, distributed computing.
- Second hypothesis (overfitting): use better regularization
 - Unsupervised pre-training
 - Stochastic drop-out training
- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

Dropout

- **Key idea:** Cripple neural network by removing hidden units stochastically

- each hidden unit is set to 0 with probability 0.5
- hidden units cannot co-adapt to other units
- hidden units must be more generally useful

- Could use a different dropout probability, but 0.5 usually works well



Dropout

- Use random binary masks $\mathbf{m}^{(k)}$

- layer pre-activation for $k > 0$

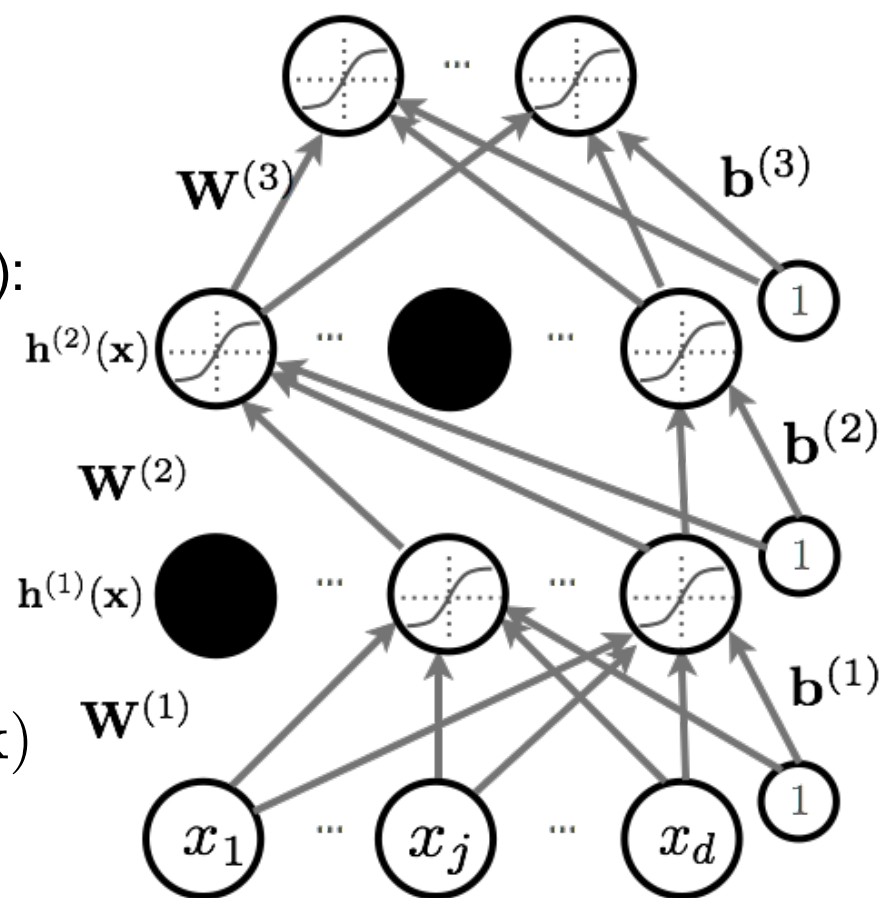
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)} \mathbf{h}^{(k-1)}(\mathbf{x})$$

- hidden layer activation ($k=1$ to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x})) \odot \mathbf{m}^{(k)}$$

- Output activation ($k=L+1$)

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Dropout at Test Time

- At test time, we replace the masks by their **expectation**
 - This is simply the constant vector 0.5 if dropout probability is 0.5
 - For single hidden layer: equivalent to taking the geometric average of all neural networks, with all possible binary masks
- Can be combined with unsupervised pre-training
- Beats regular backpropagation on many datasets
- **Ensemble**: Can be viewed as a geometric average of exponential number of networks.

Why Training is Hard

- First hypothesis (**underfitting**): better optimize
 - Use better optimization tools (e.g. batch-normalization, second order methods, such as KFAC)
 - Use GPUs, distributed computing.
- Second hypothesis (**overfitting**): use better regularization
 - Unsupervised pre-training
 - Stochastic drop-out training
- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

Batch Normalization

- Normalizing the inputs will speed up training (Lecun et al. 1998)
 - could normalization be useful at the level of the hidden layers?
- **Batch normalization** is an attempt to do that (Ioffe and Szegedy, 2014)
 - each unit's pre-activation is normalized (mean subtraction, stddev division)
 - during training, mean and stddev is computed for each minibatch
 - backpropagation takes into account the normalization
 - at test time, the global mean / stddev is used

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β


Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$



Learned linear transformation to adapt to non-linear activation function (γ and β are trained)

Batch Normalization

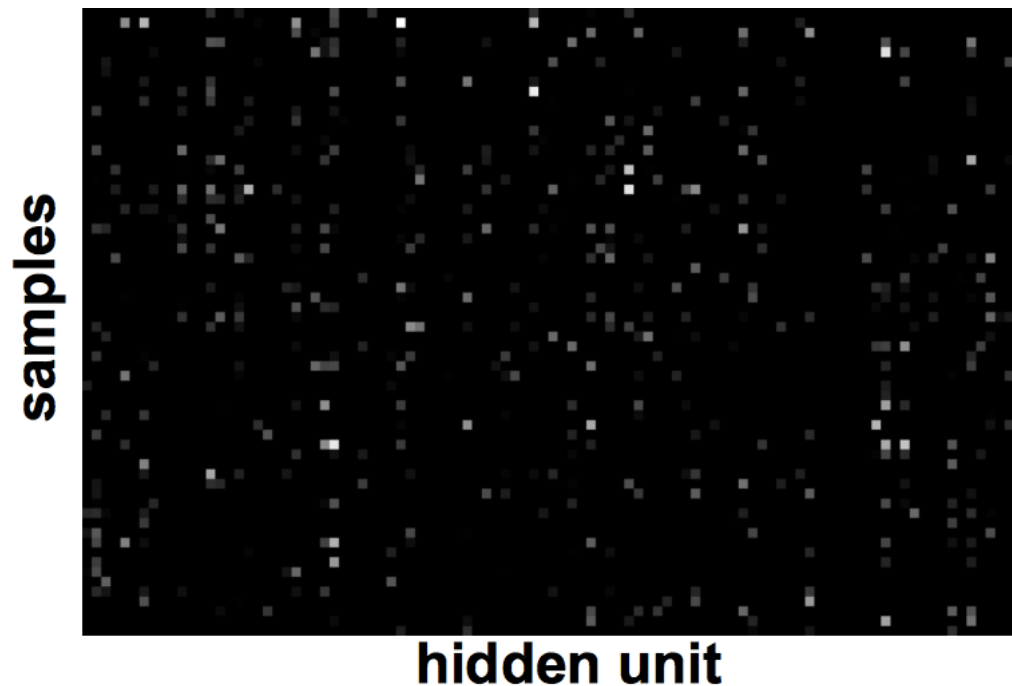
- Why normalize the pre-activation?
 - can help keep the pre-activation in a non-saturating regime (though the linear transform $y_i \leftarrow \gamma \hat{x}_i + \beta$ could cancel this effect)
- Use the **global mean and stddev** at test time.
 - removes the stochasticity of the mean and stddev
 - requires a final phase where, from the first to the last hidden layer
 - propagate all training data to that layer
 - compute and store the global mean and stddev of each unit
 - for early stopping, could use a running average

Optimization Tricks

- SGD with momentum, batch-normalization, and dropout usually works very well
- Pick learning rate by running on a subset of the data
 - Start with large learning rate & divide by 2 until loss does not diverge
 - Decay learning rate by a factor of ~ 100 or more by the end of training
- Use ReLU nonlinearity
- Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.

Visualization

- Check gradients numerically by finite differences
- Visualize features (features need to be uncorrelated) and have high variance

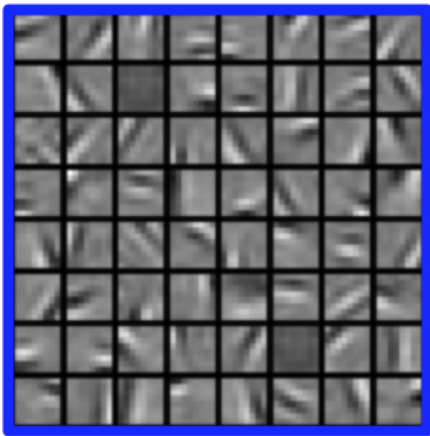


- **Good training:** hidden units are sparse across samples

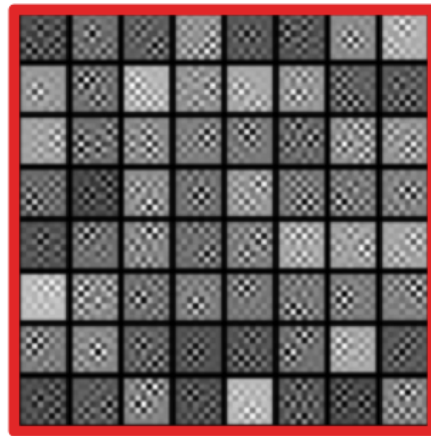
Visualization

- Check gradients numerically by finite differences
- Visualize features (features need to be uncorrelated) and have high variance
- Visualize parameters: learned features should exhibit structure and should be uncorrelated and are uncorrelated

GOOD

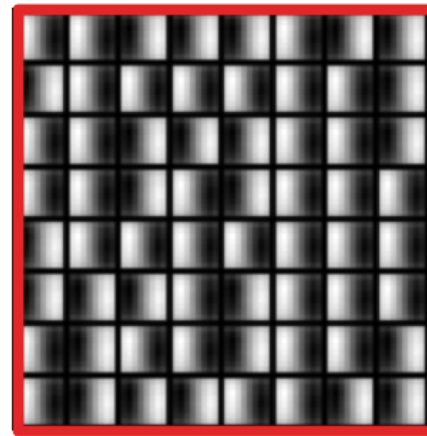


BAD



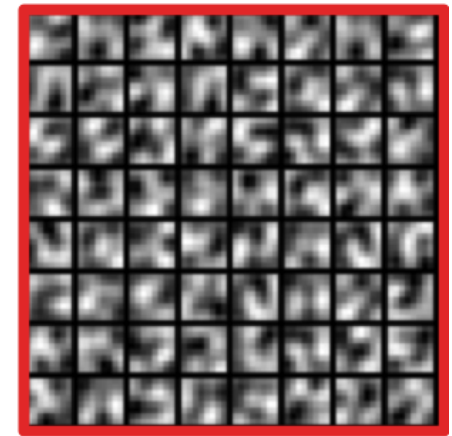
too noisy

BAD



too correlated

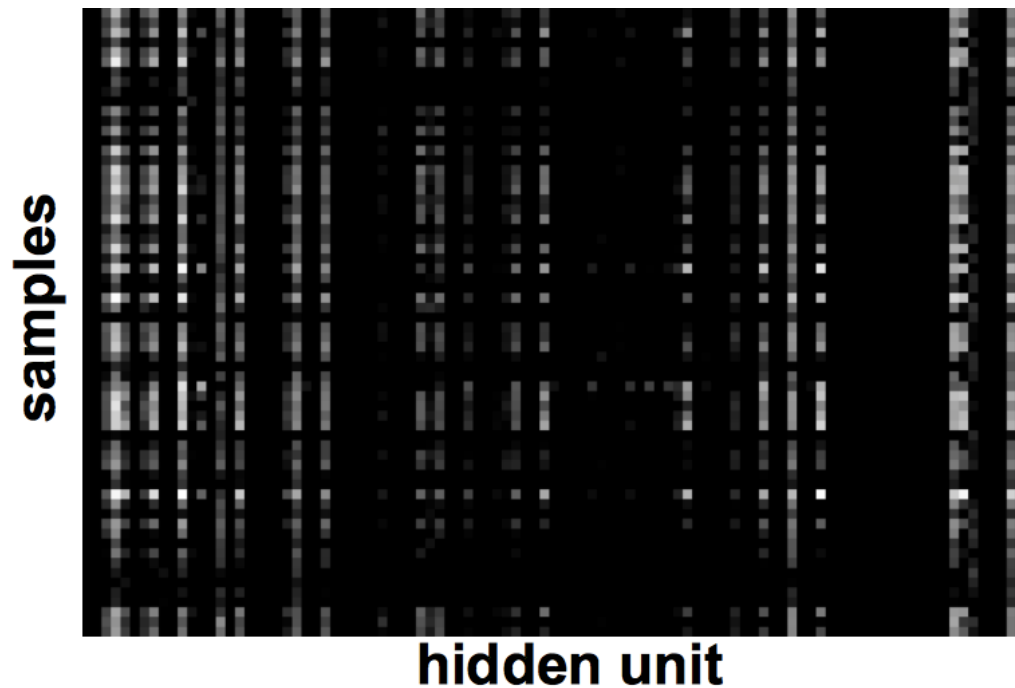
BAD



lack structure

Visualization

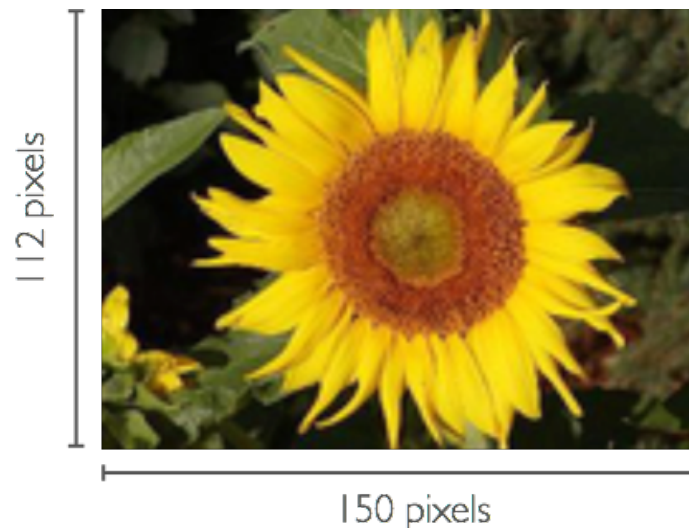
- Check gradients numerically by finite differences
- Visualize features (features need to be uncorrelated) and have high variance



- **Bad training**: many hidden units ignore the input and/or exhibit strong correlations

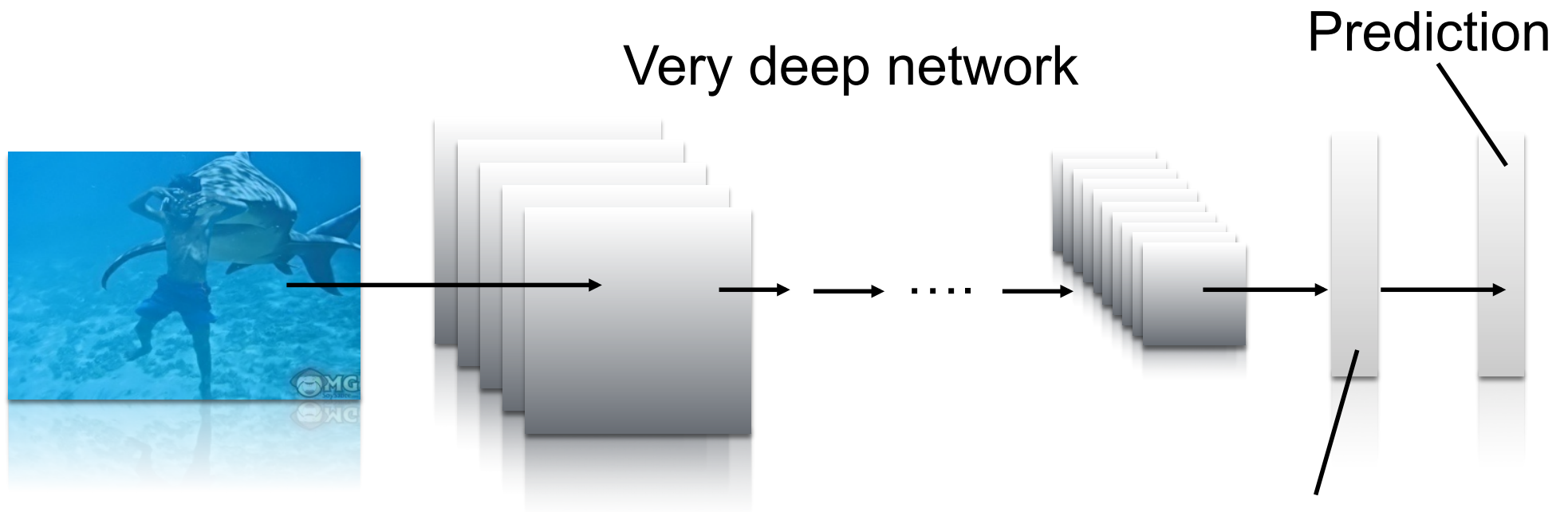
Computer Vision

- Design algorithms that can process visual data to accomplish a given task:
 - For example, **object recognition**: Given an input image, identify which object it contains



“sun flower”

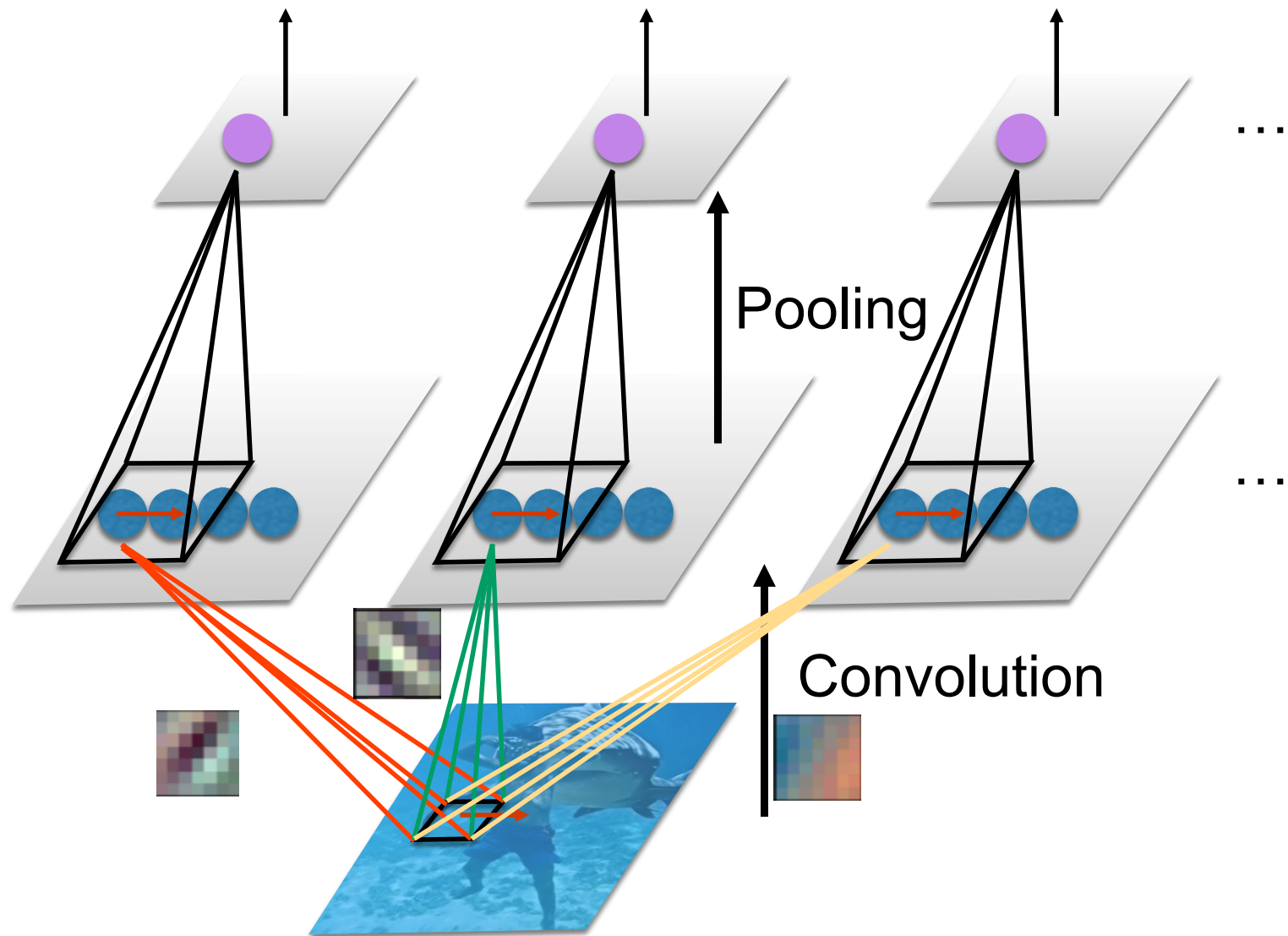
Deep Convolutional Nets



- Convolution
- Pooling
- Normalization
- Densely connected

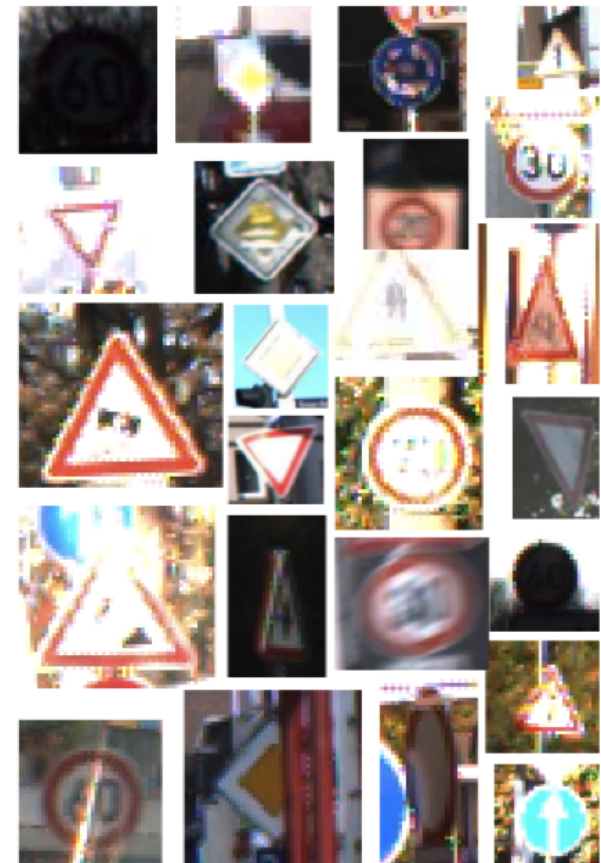
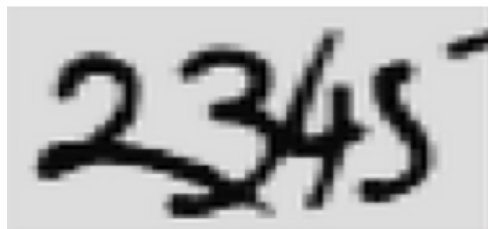
High-level feature
space

Deep Convolutional Nets



ConvNets: Examples

- Optical Character Recognition, House Number and Traffic Sign classification



Ciresan et al. "MCDNN for image classification" CVPR 2012

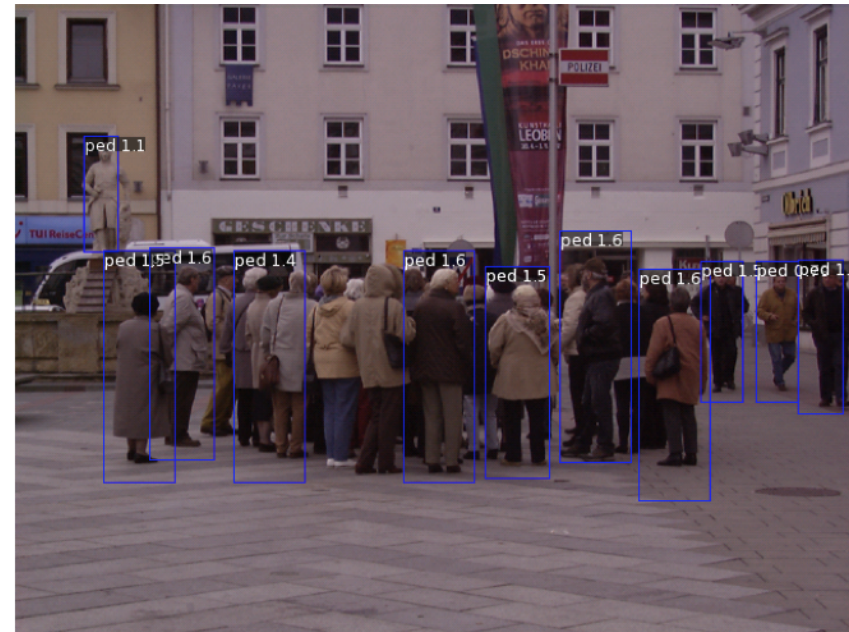
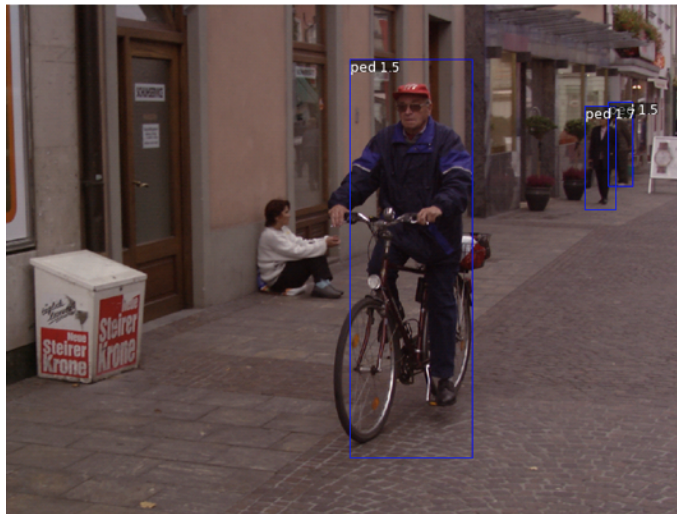
Wan et al. "Regularization of neural networks using dropconnect" ICML 2013

Goodfellow et al. "Multi-digit number recognition from StreetView..." ICLR 2014

Jaderberg et al. "Synthetic data and ANN for natural scene text recognition" arXiv 2014

ConvNets: Examples

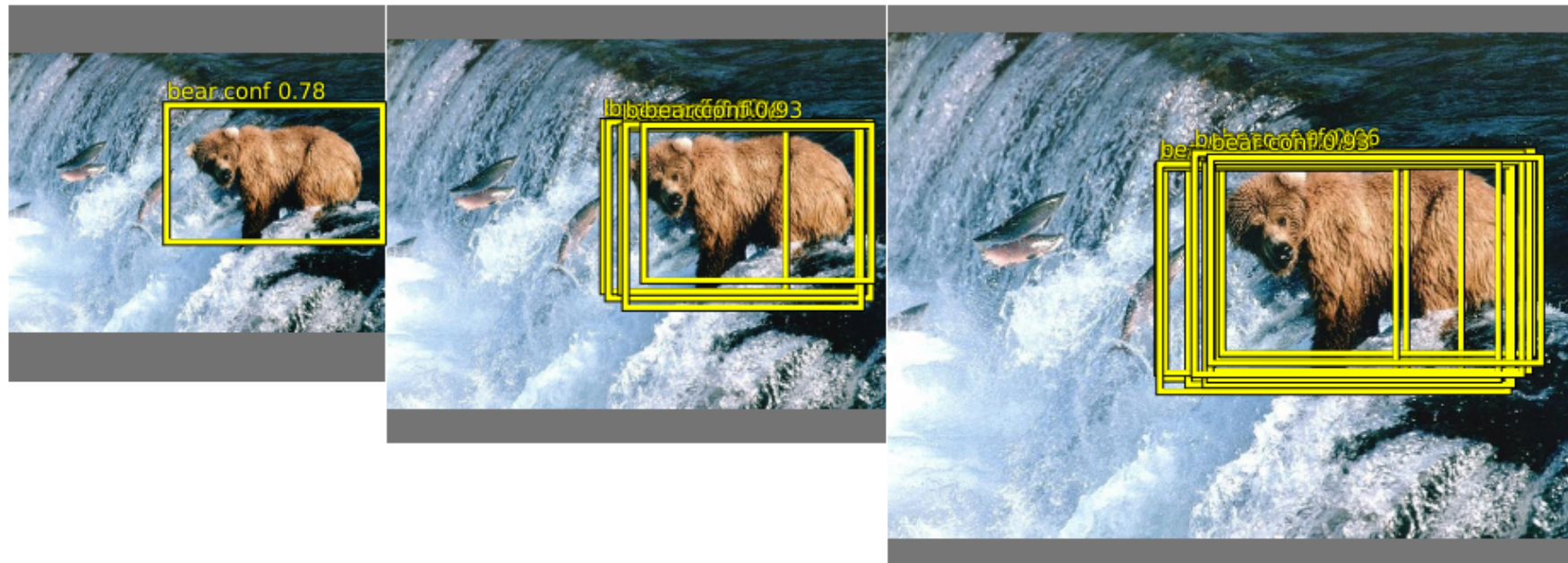
- Pedestrian detection



(Sermanet et al., Pedestrian detection with unsupervised multi-stage, CVPR 2013)

ConvNets: Examples

- Object Detection



Sermanet et al., OverFeat: Integrated recognition, localization, 2013

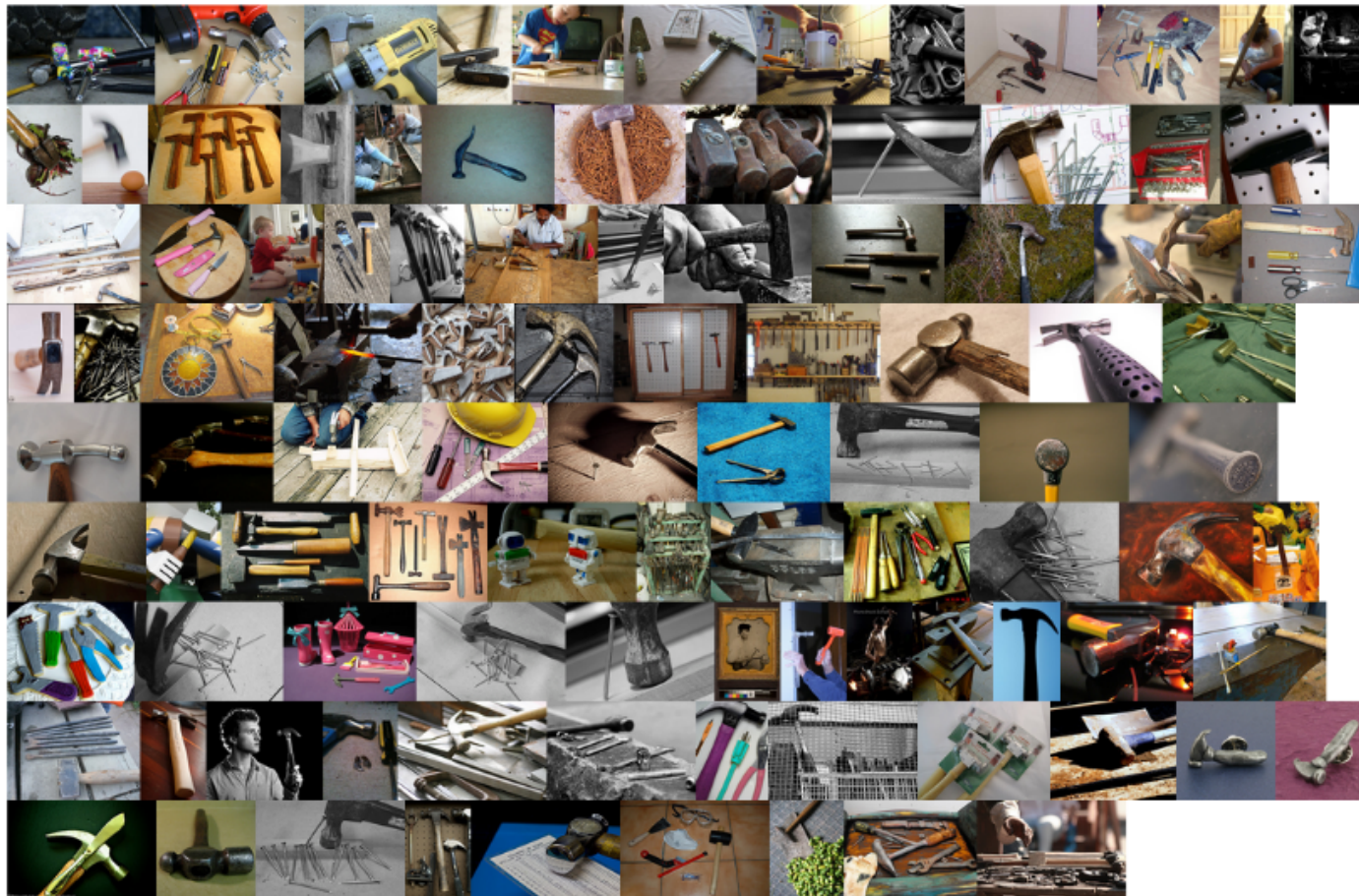
Girshick et al., Rich feature hierarchies for accurate object detection, 2013

Szegedy et al., DNN for object detection, NIPS 2013

ImageNet Dataset

- 1.2 million images, 1000 classes

Examples of Hammer

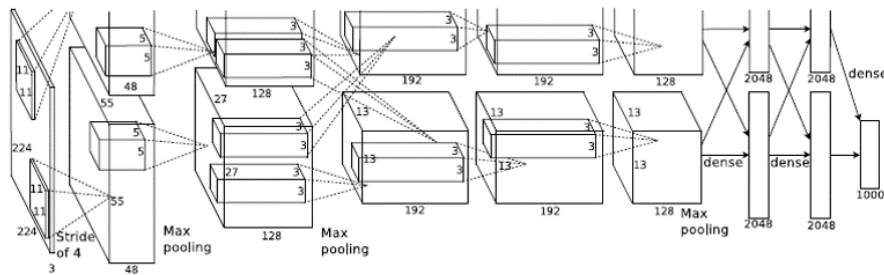


(Deng et al., Imagenet: a large scale hierarchical image database, CVPR 2009)

Important Breakthrough

- Deep Convolutional Nets for Vision (Supervised)

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



IMAGENET

1.2 million training images

1000 classes



Architecture

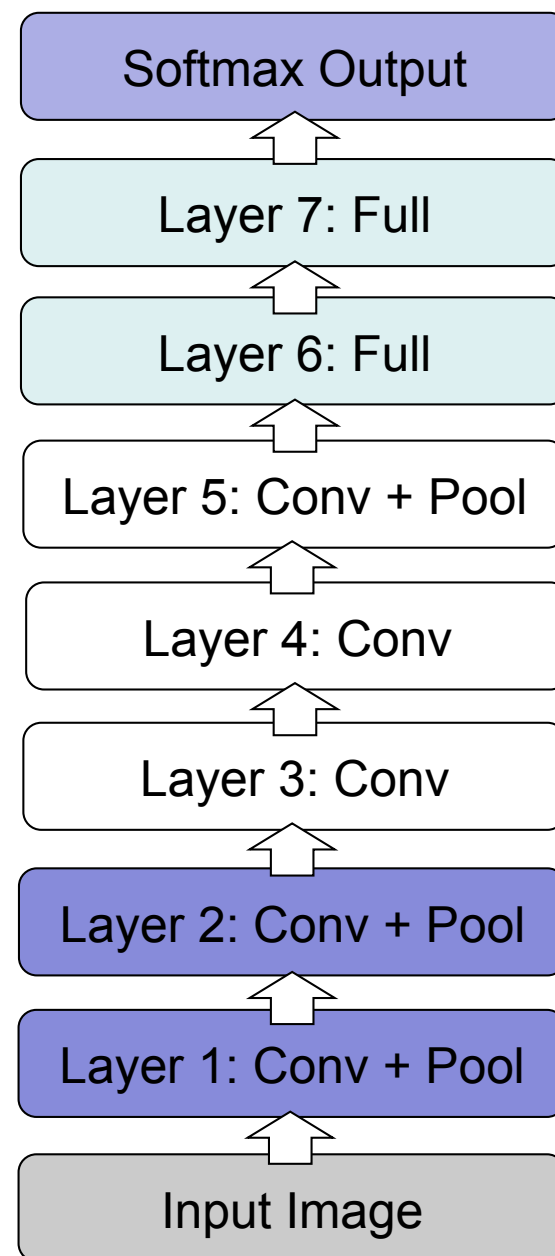
- How can we select the **right architecture**:
 - Manual tuning of features is now replaced with the manual tuning of architectures
- Depth
- Width
- Parameter count

How to Choose Architecture

- Many **hyper-parameters**:
 - Number of layers, number of feature maps
- Cross Validation
- Grid Search (need lots of GPUs)
- Smarter Strategies
 - Random search
 - Bayesian Optimization

AlexNet

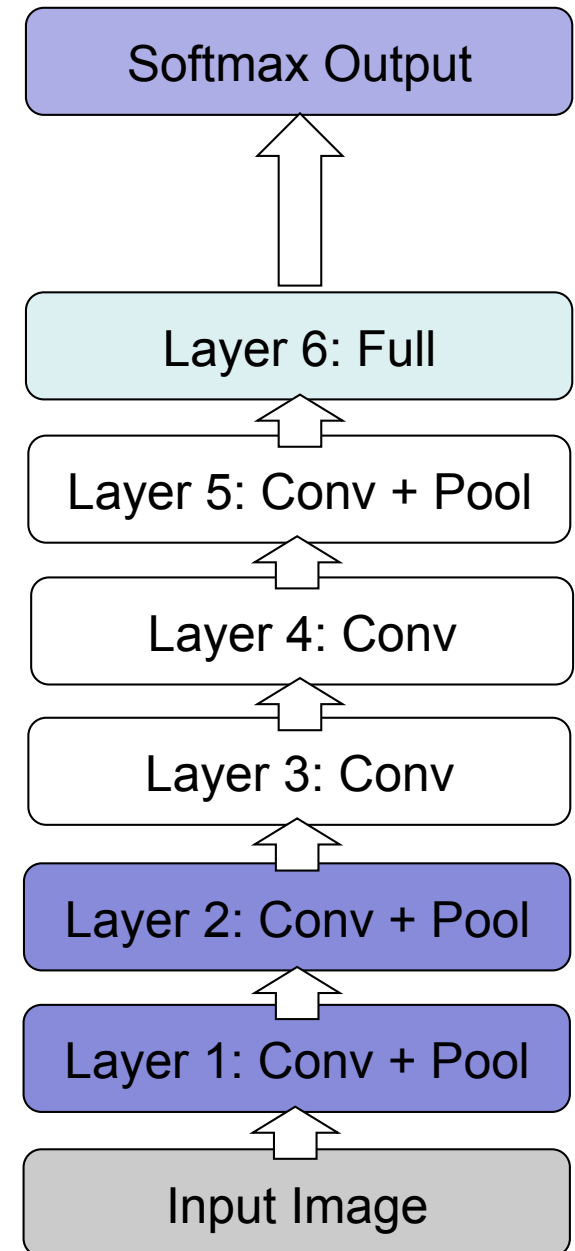
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

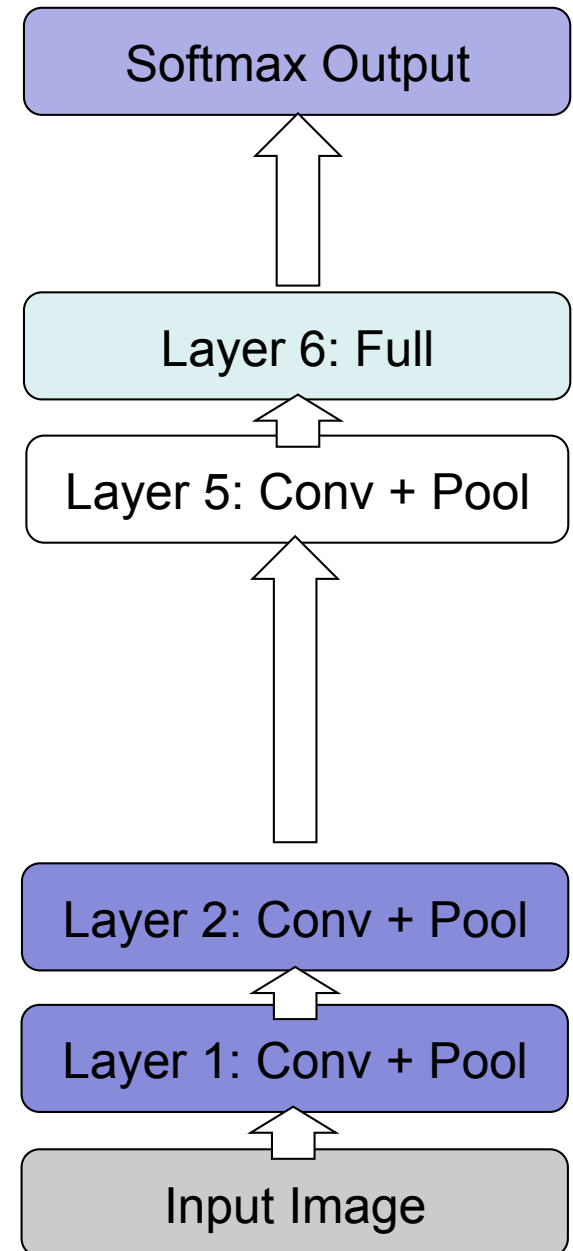
- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in performance!



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

- Let us remove upper feature extractor layers and fully connected:
 - Layers 3,4, 6 and 7
- Drop ~50 million parameters
- **33.5 drop in performance!**
- **Depth of the network is the key.**



[From Rob Fergus' CIFAR 2016 tutorial]

GoogLeNet



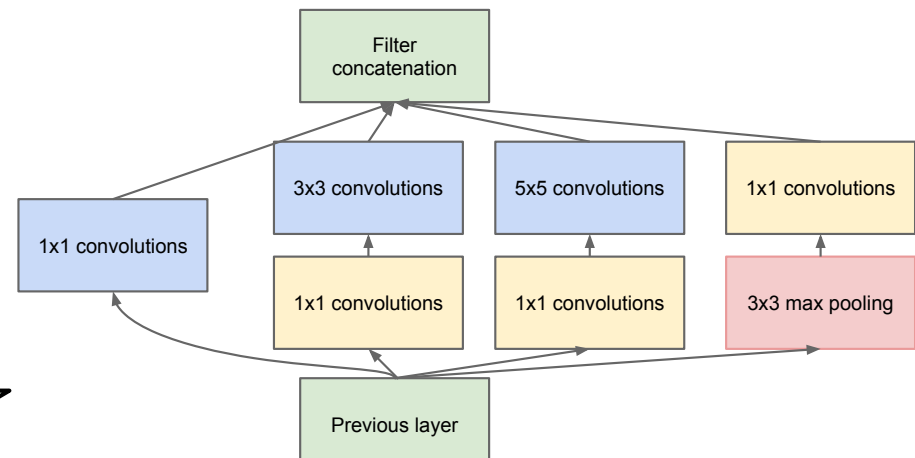
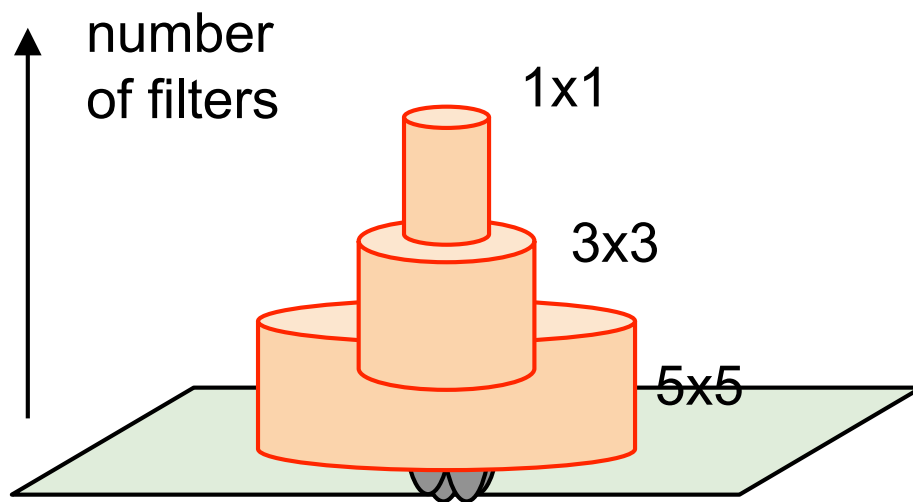
- 24 layer model that uses so-called inception module.

Convolution
Pooling
Softmax
Other

(Szegedy et al., Going Deep with Convolutions, 2014)

GoogLeNet

- GoogLeNet inception module:
 - Multiple filter scales at each layer
 - Dimensionality reduction to keep computational requirements down



(Szegedy et al., Going Deep with Convolutions, 2014)

GoogLeNet

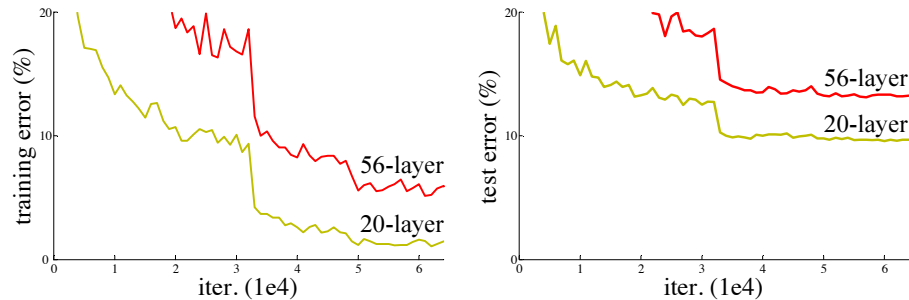


- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

(Szegedy et al., Going Deep with Convolutions, 2014)

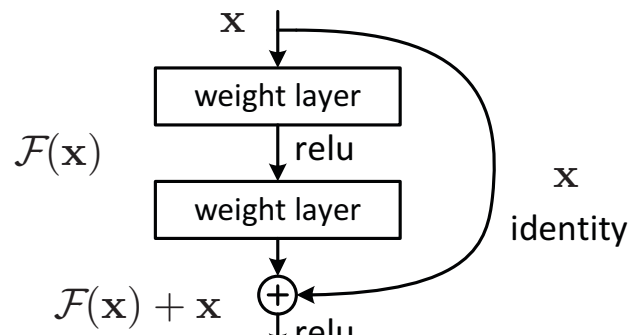
Residual Networks

Really, really deep convnets do not train well,
E.g. CIFAR10:



Key idea: introduce “pass through” into each layer

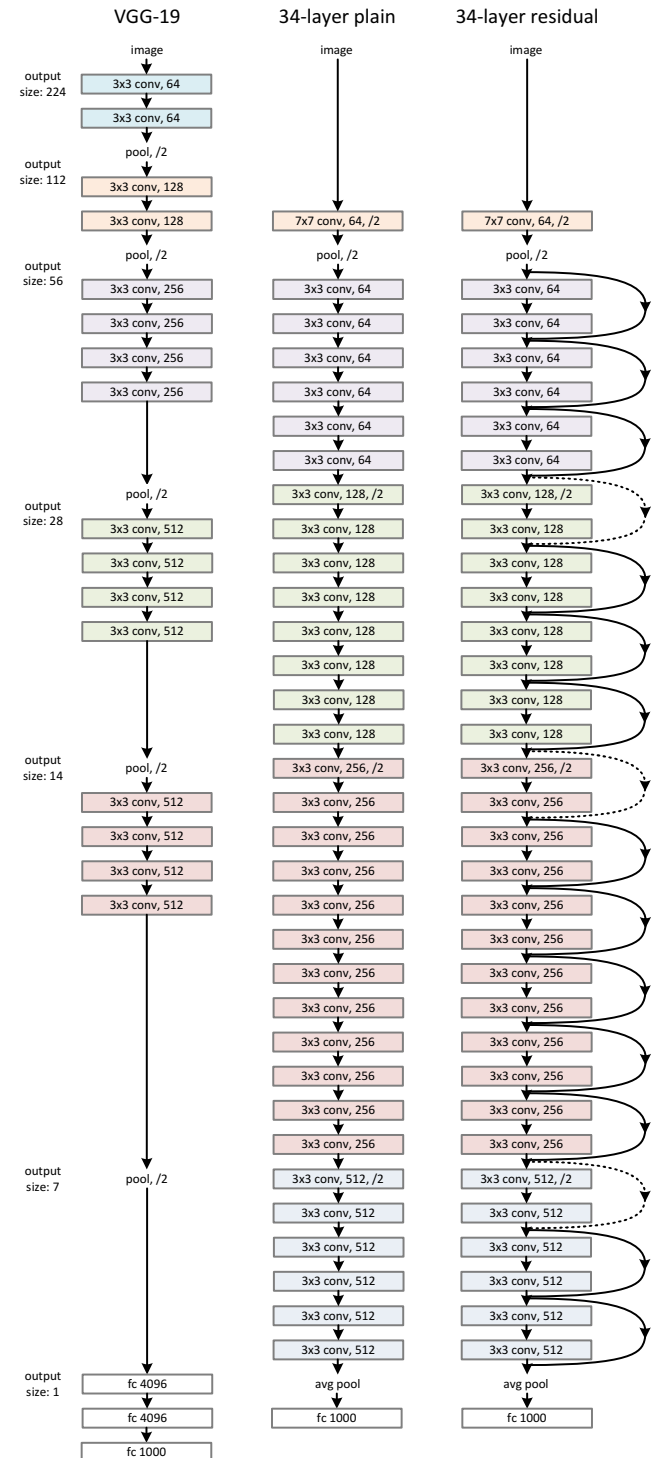
Thus only residual now needs to be learned



method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43 [†]
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	19.38	4.49

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except [†] reported on the test set).

With ensembling, 3.57% top-5 test error on ImageNet



(He, Zhang, Ren, Sun, CVPR 2016)

Choosing the Architecture

- Task dependent
- Cross-validation
- [Convolution \rightarrow pooling]* + fully connected layer
- The more data: the more layers and the more kernels
 - Look at the **number of parameters** at each layer
 - Look at the **number of flops** at each layer
- Computational resources

End of Part 1