Deep Learning

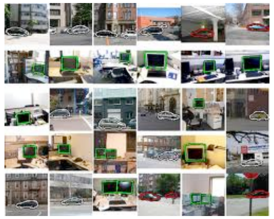
Russ Salakhutdinov

Machine Learning Department
Carnegie Mellon University
Canadian Institute for Advanced Research
MLSS 2017: Lecture 1

Mining for Structure

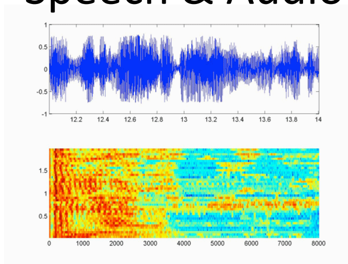
Massive increase in both computational power and the amount of data available from web, video cameras, laboratory measurements.

Images & Video

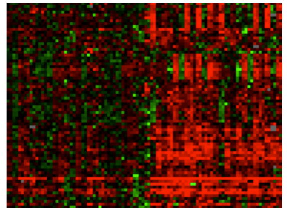


Text & Language

Speech & Audio

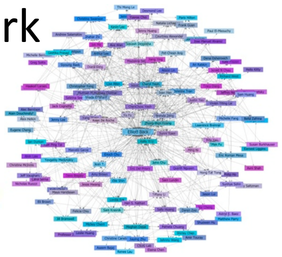


Gene Expression

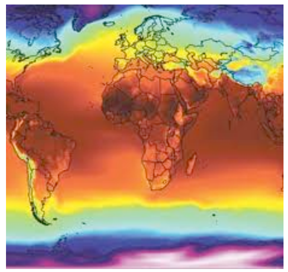


Product Recommendation

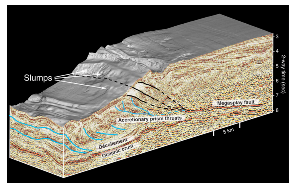
Relational Data/ Social Network



Climate Change



Geological Data



- Develop statistical models that can discover underlying structure, semantic relations, constraints, or invariances from data.
- Robust, adaptive models models that can deal with missing measurements, nonstationary distributions, multimodal data.

Impact of Deep Learning

- Speech Recognition
- Computer Vision
- Recommender Systems
- Language Understanding
- Drug Discovery and Medical Image Analysis

Example: Understanding Images

TAGS:

strangers, coworkers, conventioneers, 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 .

Tutorial Roadmap

Part 1: Supervised (Discriminative) Learning: Deep Networks

Part 2: Unsupervised Learning: Deep Generative Models

Part 3: Open Research Questions

Supervised Learning

- Given a set of labeled training examples: $\{\mathbf{x}^{(t)}, y^{(t)}\}$, we perform Empirical Risk Minimization:

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

where

- $f(\mathbf{x}^{(t)}; \theta)$ (non-linear) function mapping inputs to outputs, parameterized by θ -> Non-convex optimization

- $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is the loss function.

Supervised Learning

- Given a set of labeled training examples: $\{\mathbf{x}^{(t)}, y^{(t)}\}$, we perform Empirical Risk Minimization:

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

where

- $f(\mathbf{x}^{(t)}; \theta)$ (non-linear) function mapping inputs to outputs, parameterized by θ -> Non-convex optimization
- $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is the loss function.
- $\Omega(\boldsymbol{\theta})$ is a regularization term.

Supervised Learning

- Given a set of labeled training examples: $\{\mathbf{x}^{(t)}, y^{(t)}\}$, we perform 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.

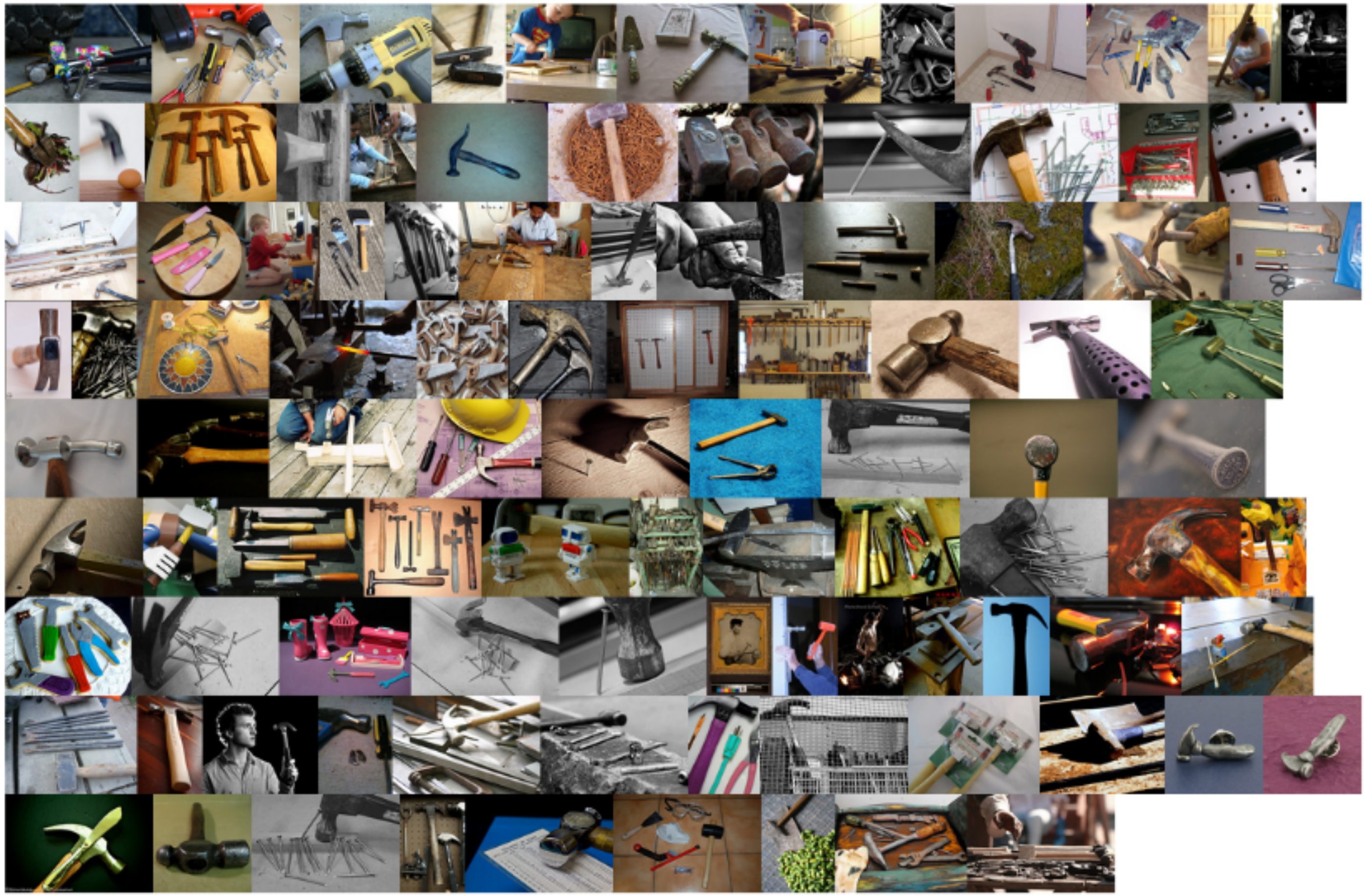
  - For classification problems, we would like to minimize classification error.

  - Loss function can sometimes be viewed as a surrogate for what we want to optimize (e.g. upper bound)

Example: ImageNet Dataset

- 1.2 million (225 x 225) images, 1000 classes

Examples of Hammer



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

AlexNet

- Input: 225 x 225 image
- Output: Softmax over 1000 classes

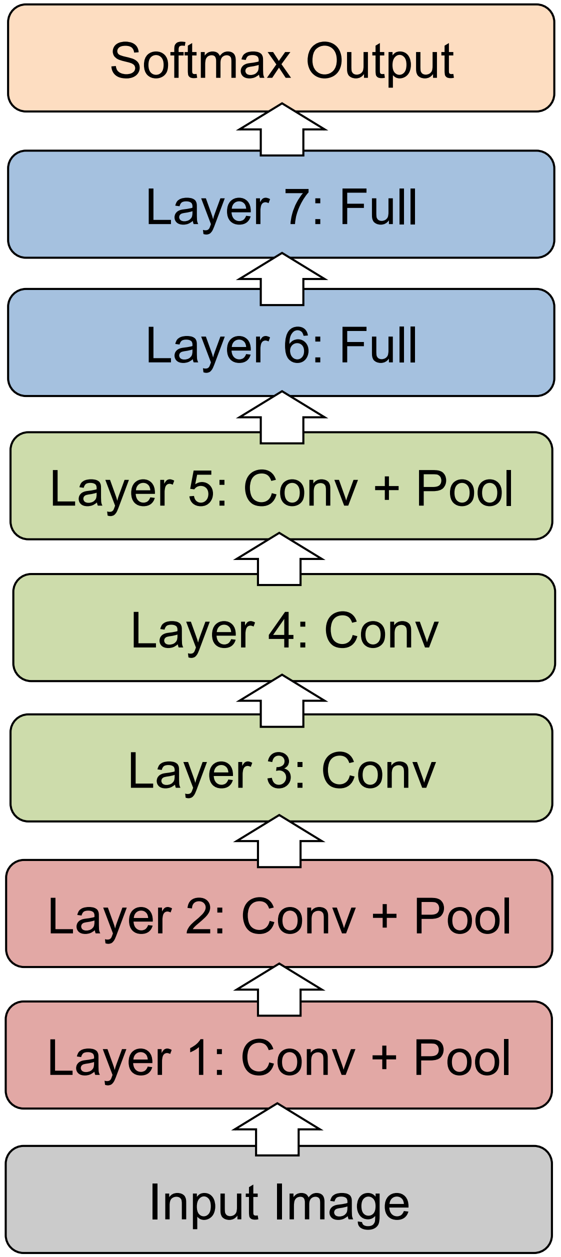
$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda\Omega(\boldsymbol{\theta})$$

➢ $f(\mathbf{x}^{(t)}; \theta)$ differentiable, non-nonlinear function parameterized by θ: 8 layers, 60M parameters -> Non-convex optimization

➢ $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is the cross entropy loss.

➢ $\Omega(\boldsymbol{\theta})$: L_2, early stopping, drop-out

- Achieves: 18.2% top-5 error

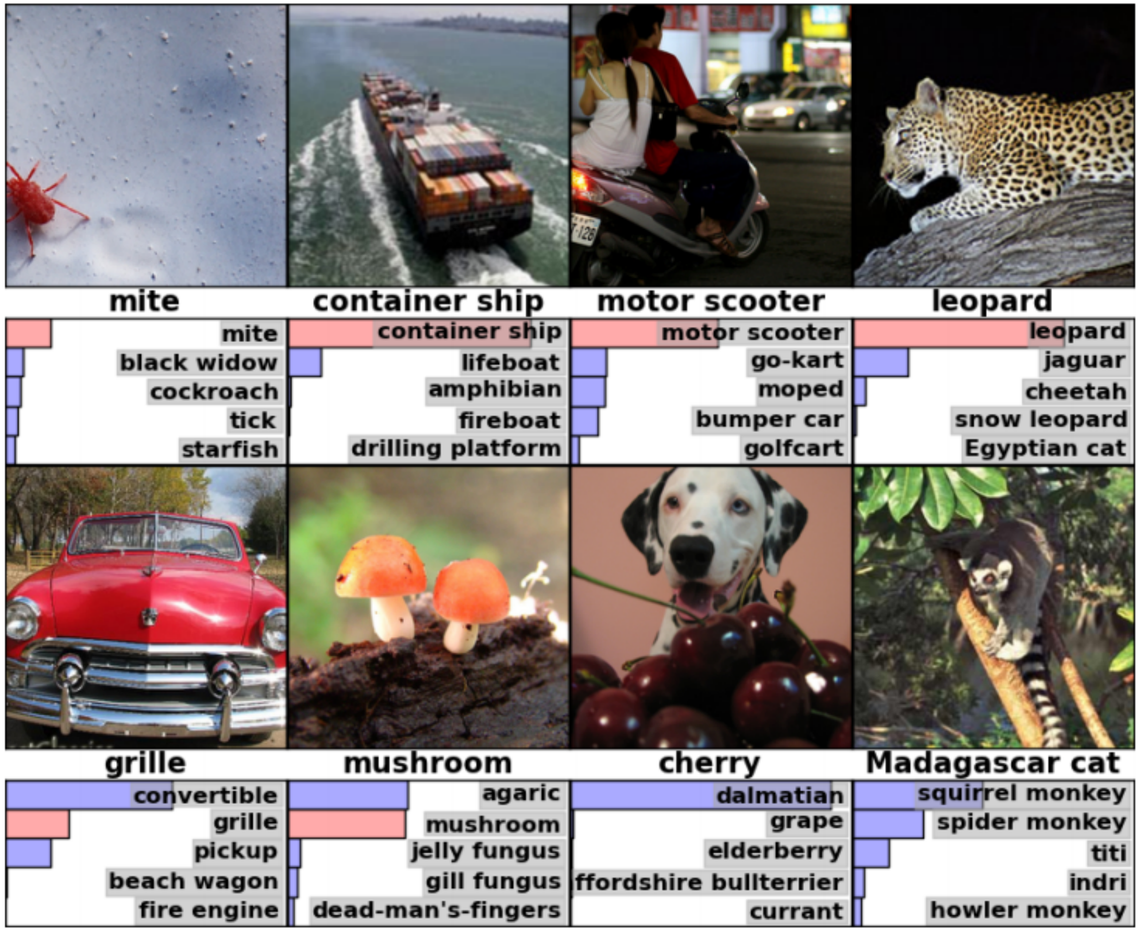


(Krizhevsky, Sutskever, Hinton, NIPS, 2012)

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.



Unsupervised Learning

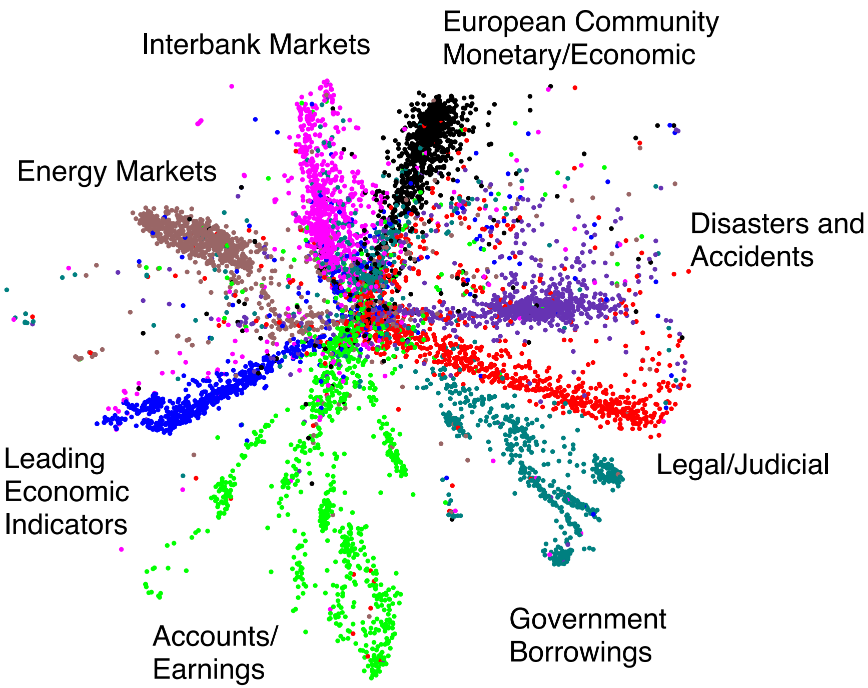
- Given a set of unlabeled training examples $\{\mathbf{x}^{(t)}\}$:

$$P(\mathbf{x}) = \frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \exp\left[\mathbf{x}^\top \mathbf{W} \mathbf{h}\right]$$

Vector of word counts
on a webpage

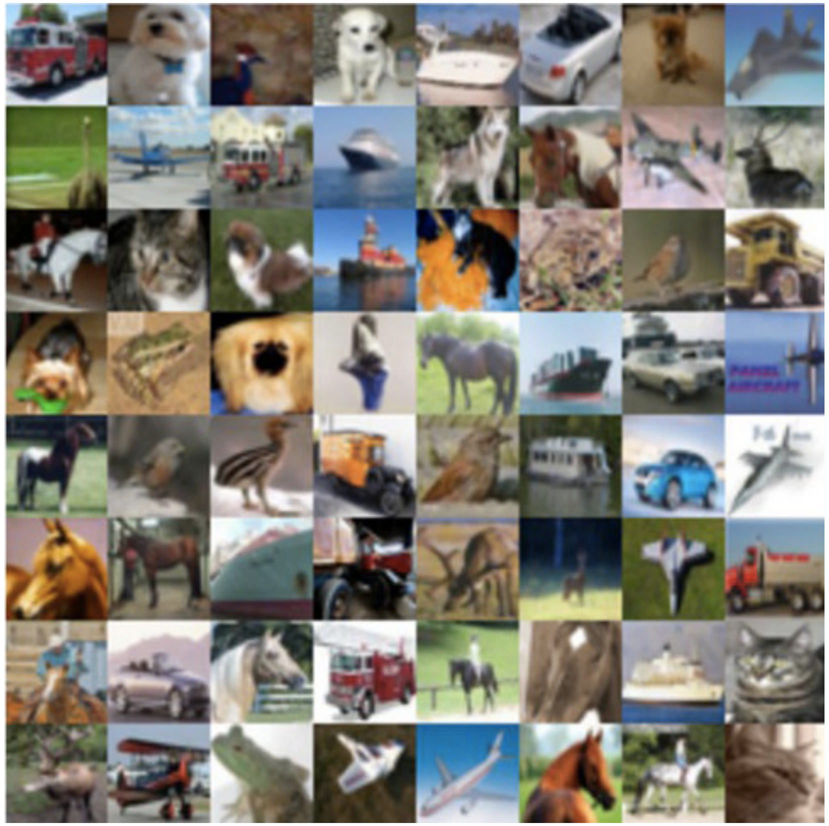
Latent variables:
hidden topics

804,414 unlabelled
newswire stories

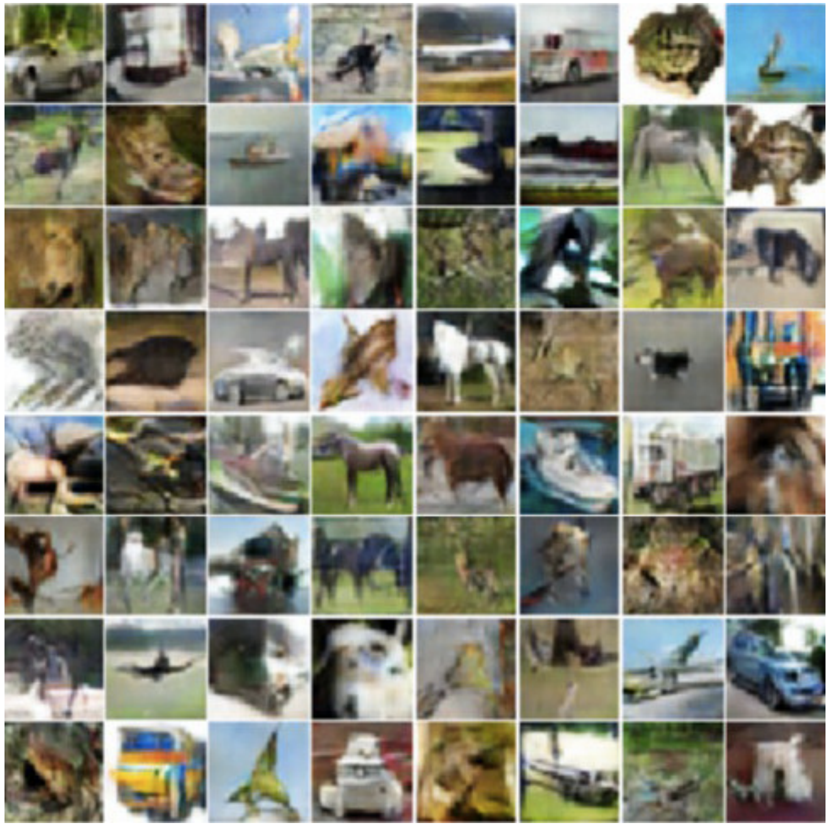


(Hinton & Salakhutdinov, Science, 2006)

Generative Adversarial Net Trained on ImageNet



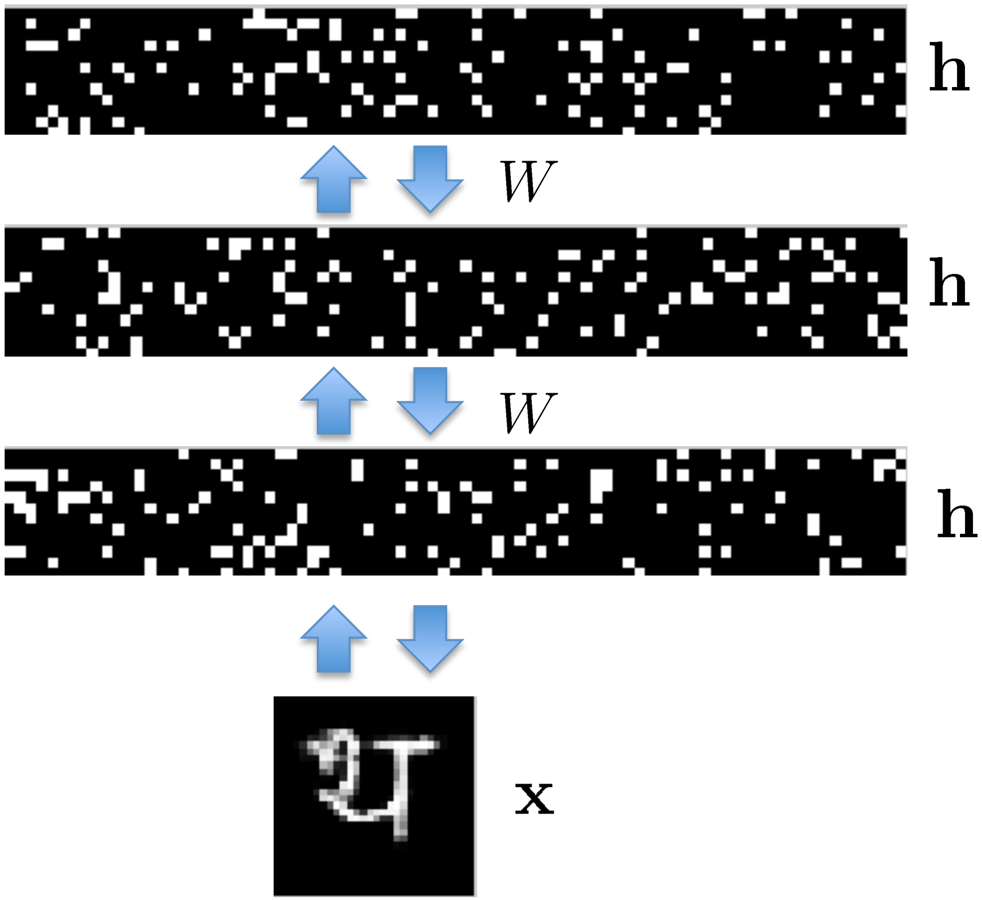
Training



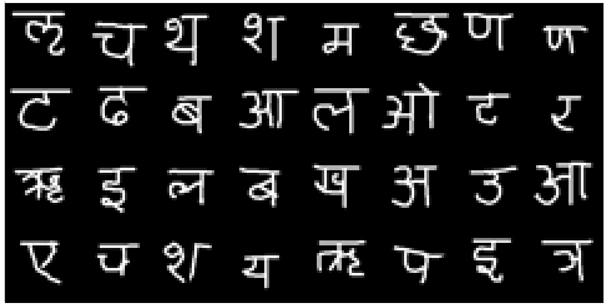
Samples

(Salimans et. al., 2016)

Boltzmann Machine



Observed Data



25,000 characters from 50 alphabets around the world.

Simulate a Markov chain whose stationary distribution is $P(\mathbf{x}|\mathbf{y} = \text{Sanskrit})$.

Talk Roadmap

Part 1: Supervised Learning: Deep Networks

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

Part 2: Unsupervised Learning: Learning Deep Generative Models

Part 3: Open Research Questions

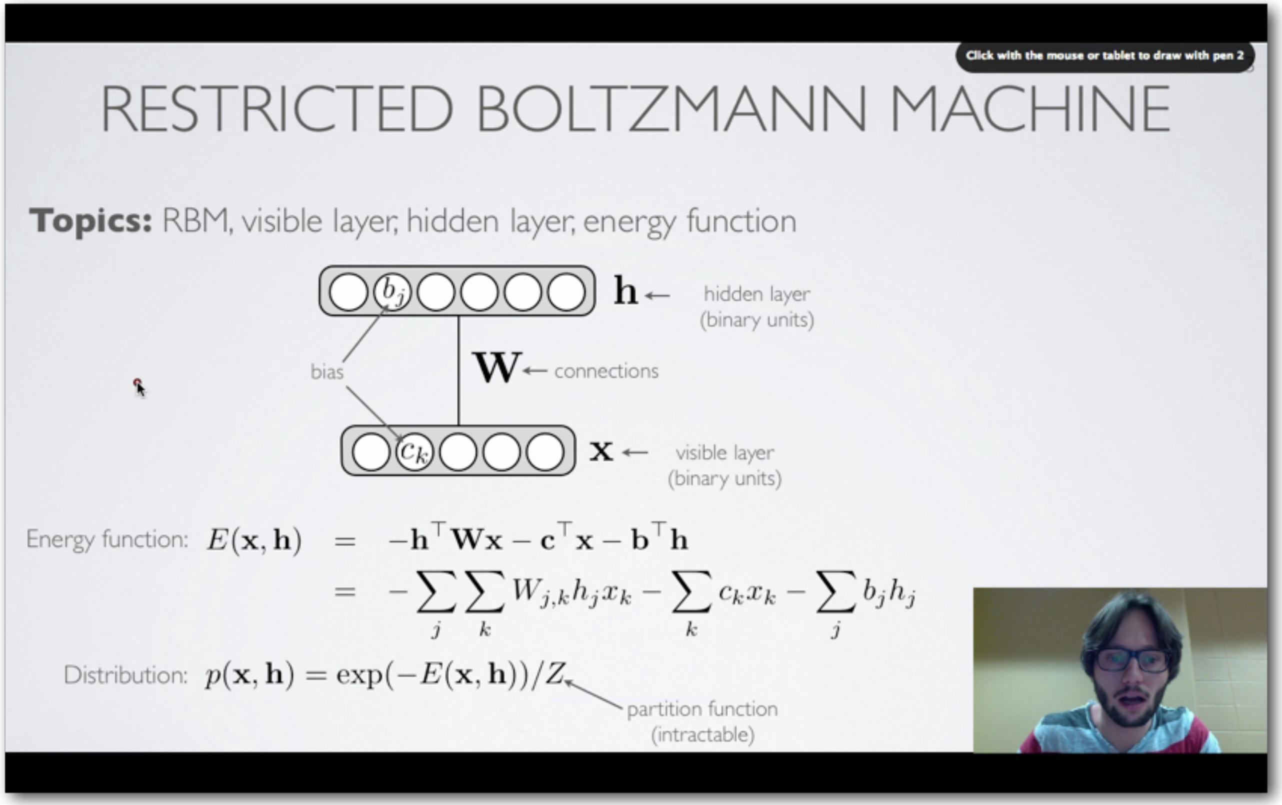
Neural Networks Online Course

- **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.

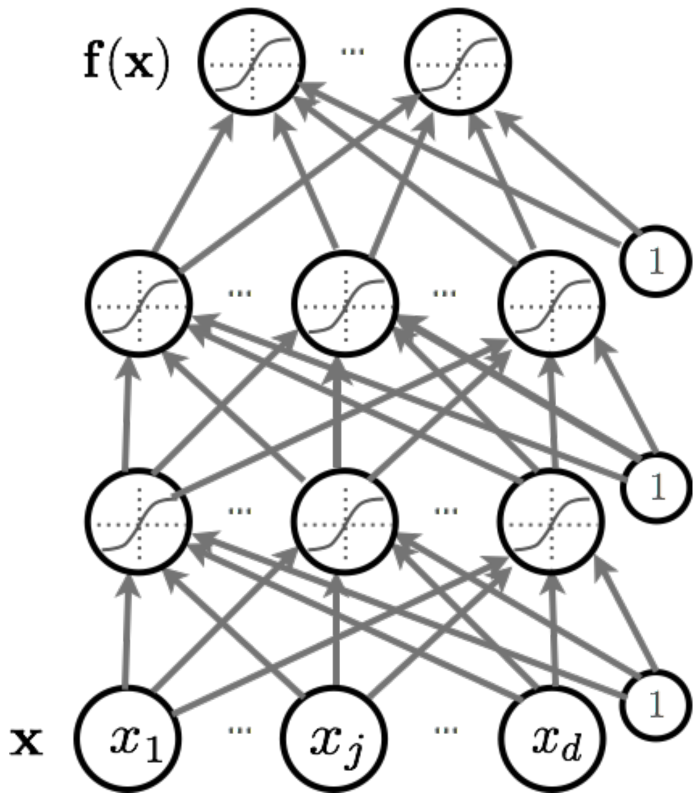


Feedforward Neural Networks

- 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



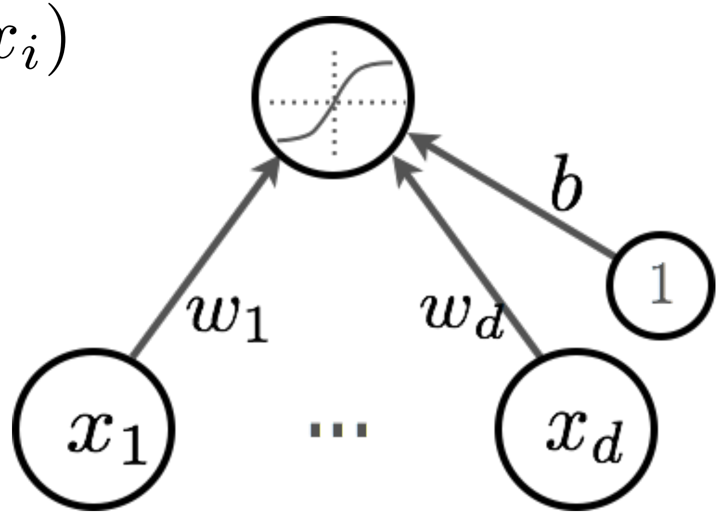
Artificial Neuron

- 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

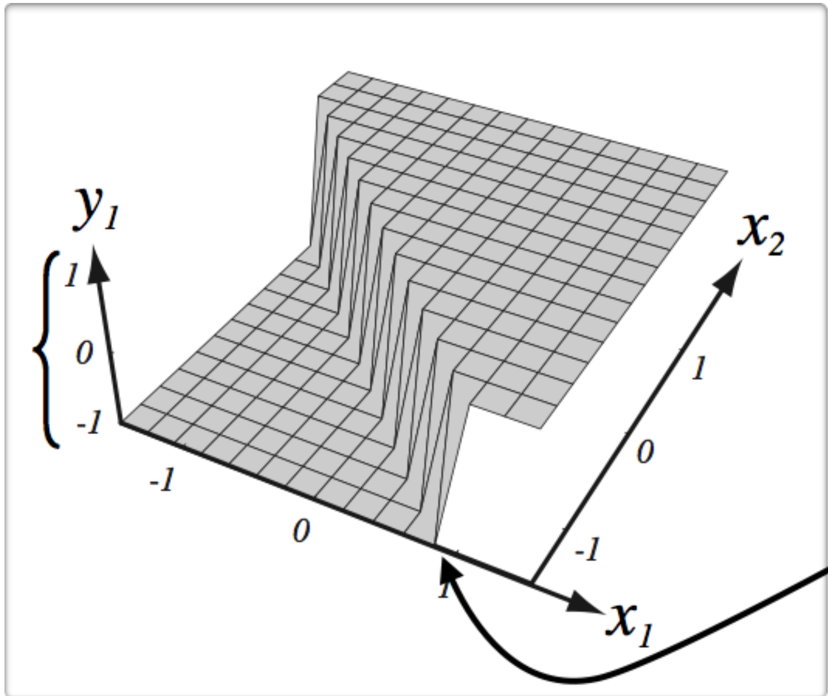
$g(\cdot)$ is called the activation function

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)

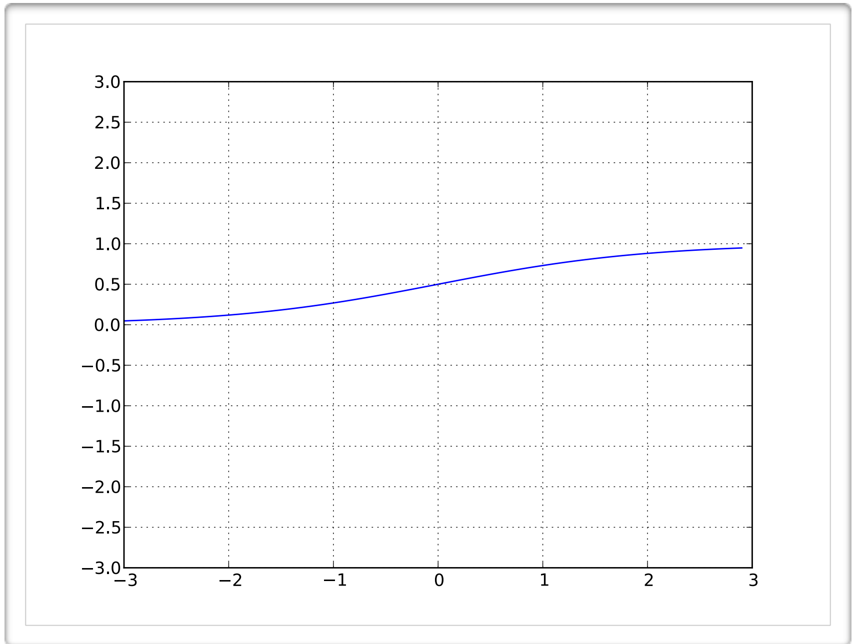
Bias only changes the position of the riff

Activation Function

- Sigmoid activation function:

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

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

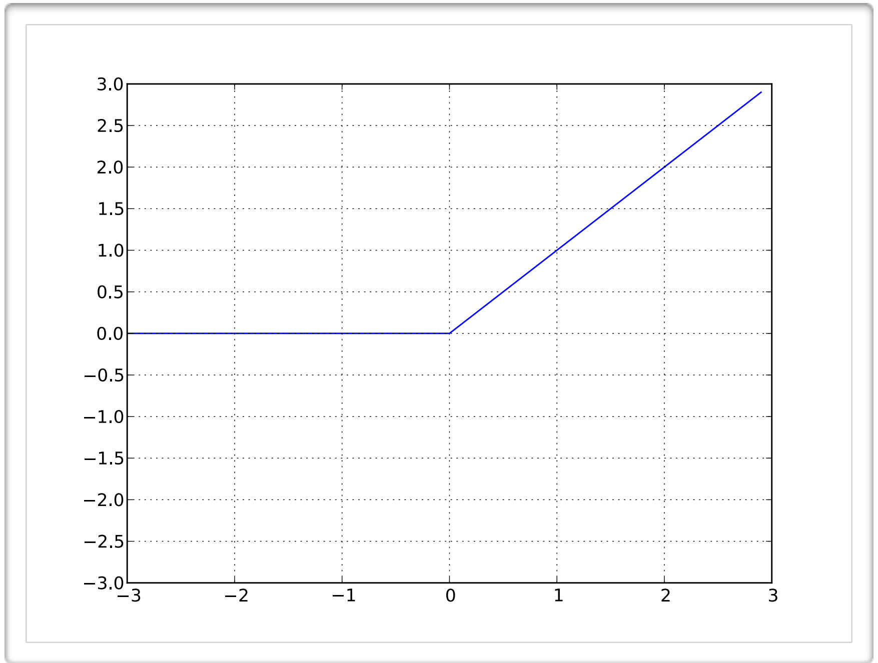


Activation Function

- Rectified linear (ReLU) activation function:

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

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



Single Hidden Layer Neural Net

- Hidden layer pre-activation:

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

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

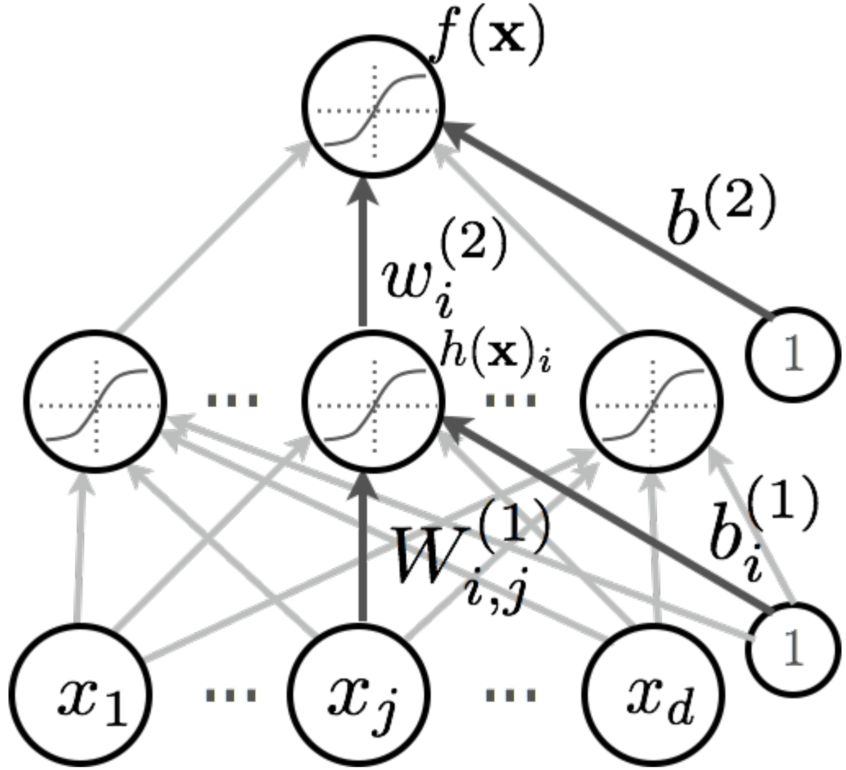
- Hidden layer activation:

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

- 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

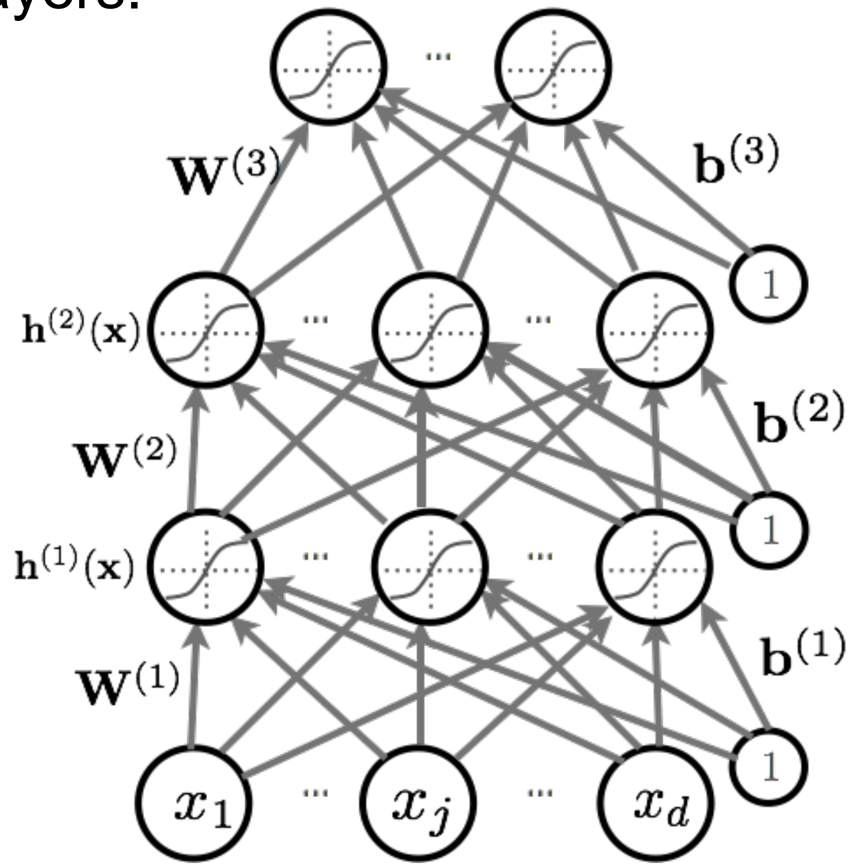
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - 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):

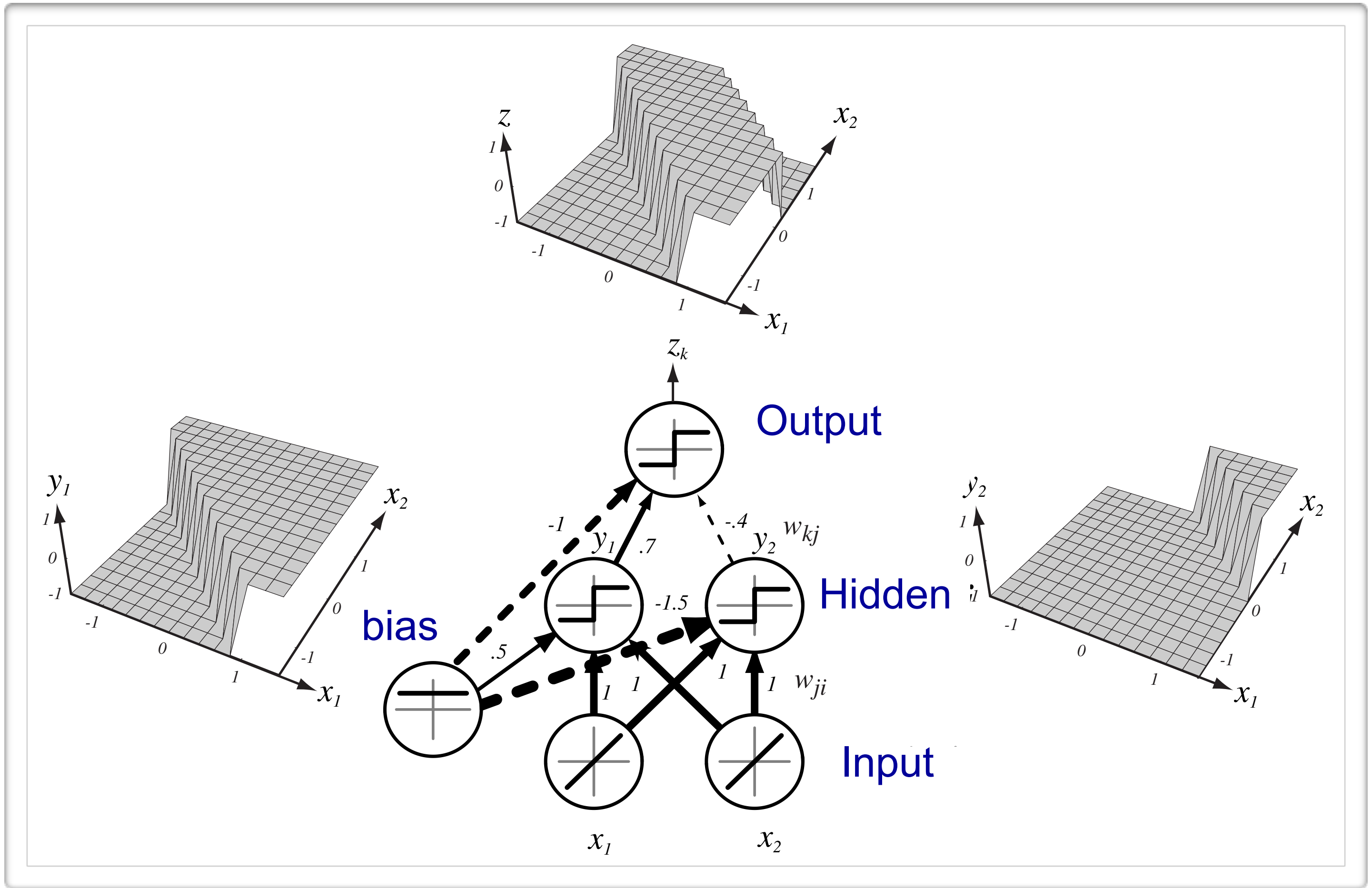
$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

Capacity of Neural Nets

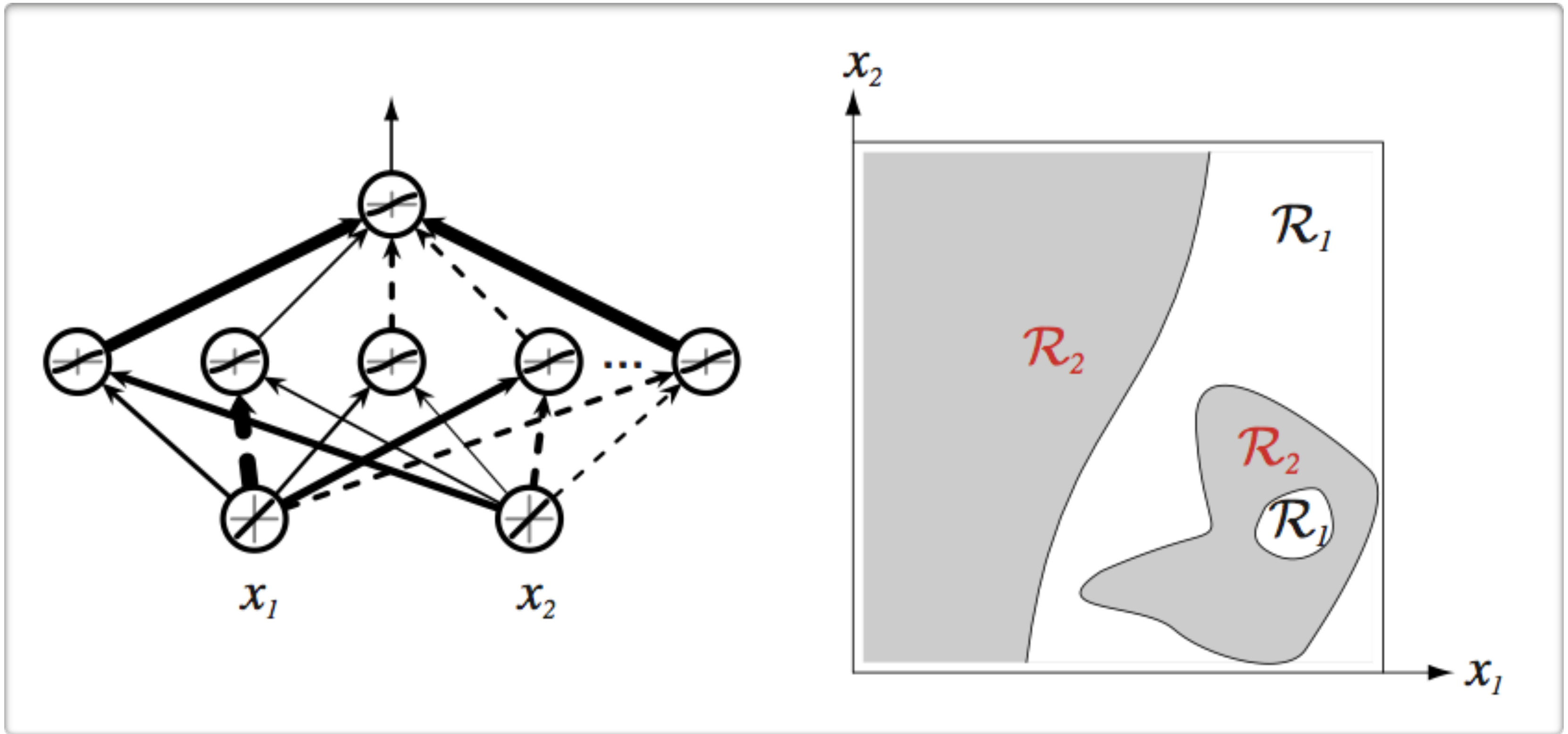
- Consider a single layer neural network



(from Pascal Vincent's slides)

Capacity of Neural Nets

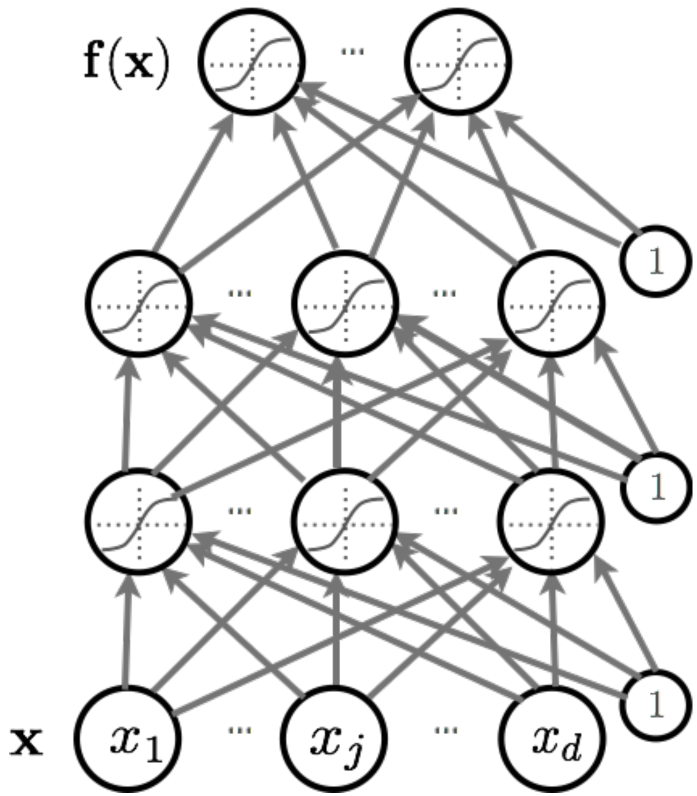
- Consider a single layer neural network



(from Pascal Vincent's slides)

Feedforward Neural Networks

- How neural networks predict f(x) given an input 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:

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

- To train a neural net, we need:
  - Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)};\boldsymbol{\theta}), y^{(t)})$
  - A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)};\boldsymbol{\theta}), y^{(t)})$
  - Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

Stochastic Gradient Descend

- Perform updates after seeing each example:
  - Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  - For t=1:T
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$$

Training epoch
=
Iteration of all examples

- To train a neural net, we need:

  - Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

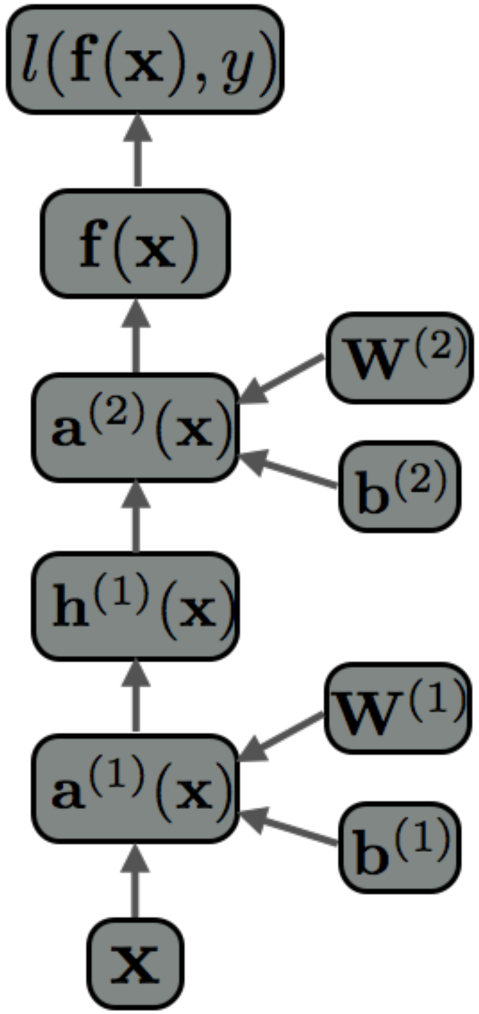
Backpropagation Algorithm: Computational Flow Graph

- 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

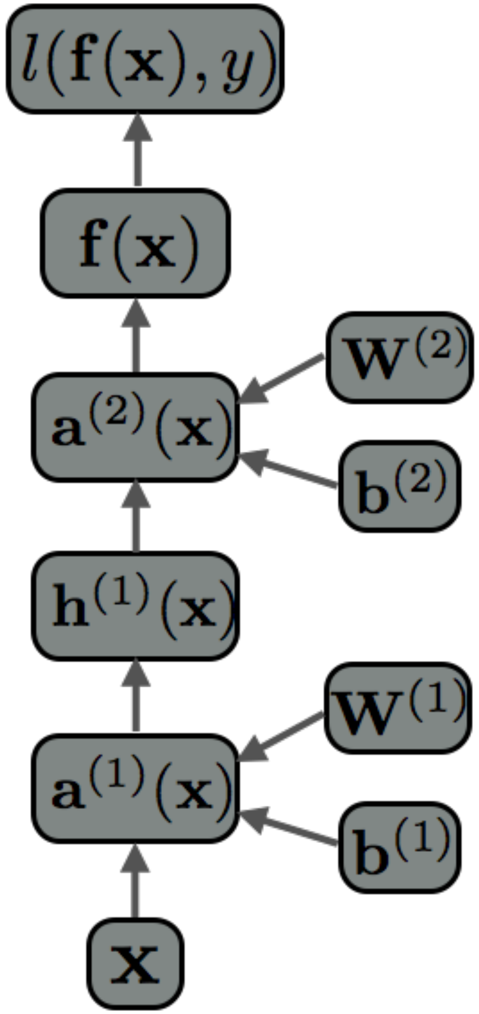
  - Calling the fprop method of each box in the right order yields forward propagation



Backpropagation Algorithm: 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



Weight Decay

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

- L2 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k ||\mathbf{W}^{(k)}||_F^2$$

- L1 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

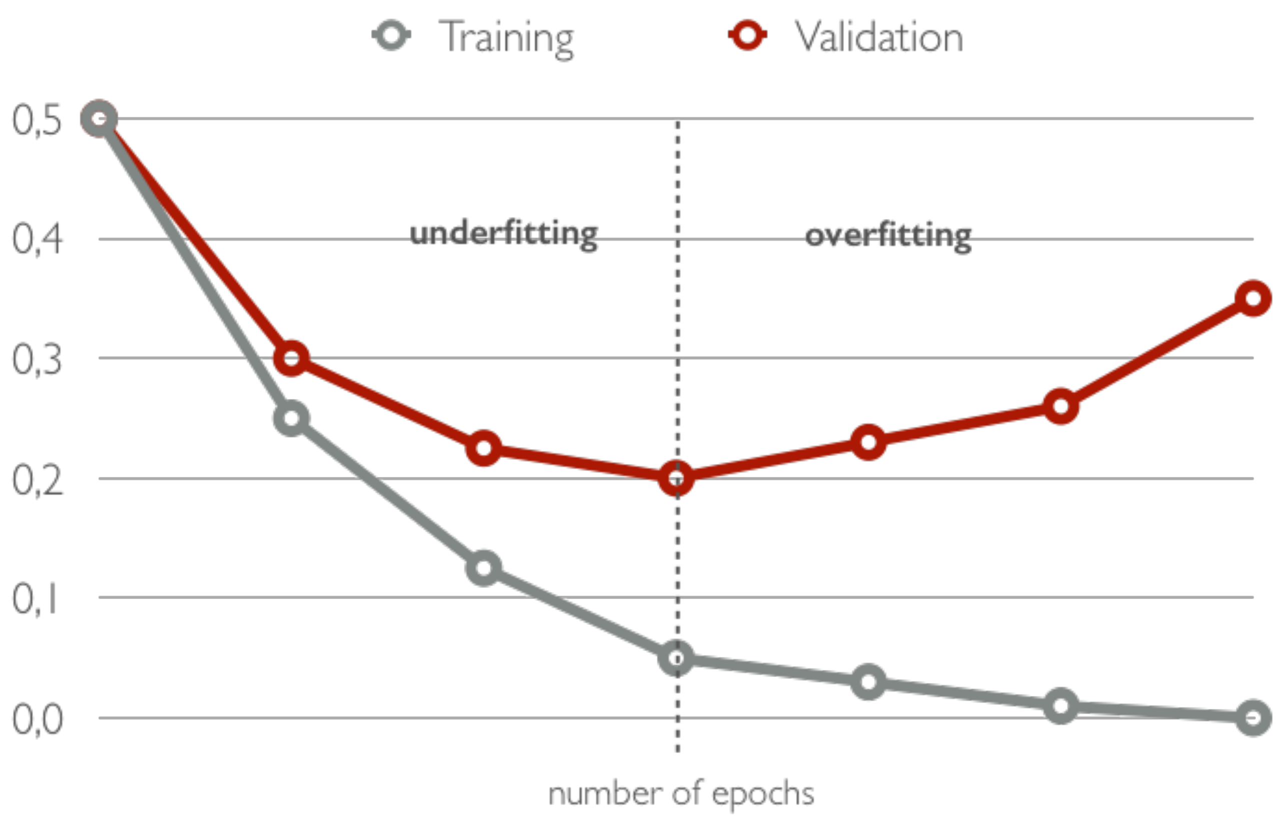
  – Only applies to weights, not biases (weigh decay)

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:

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

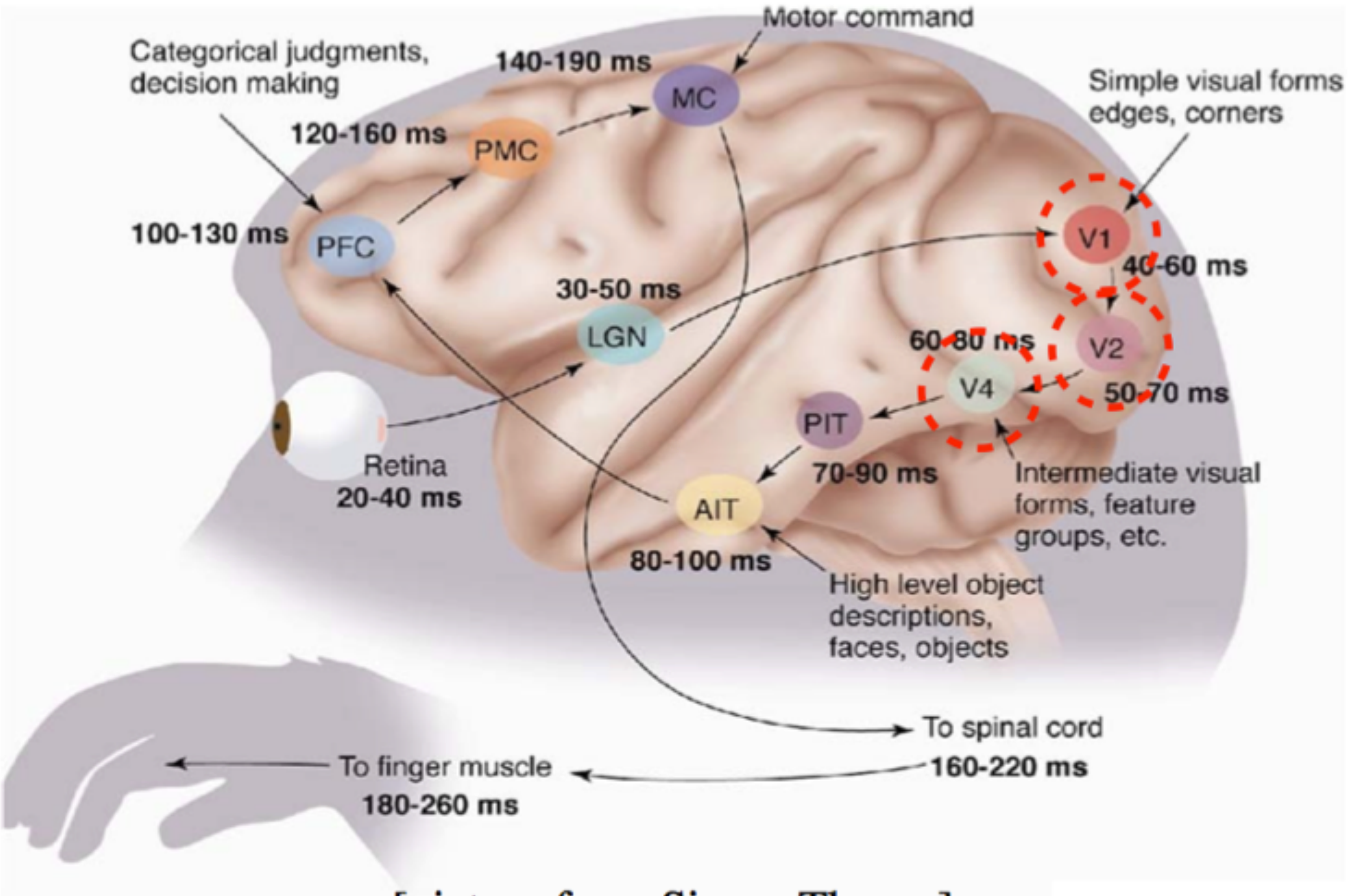
  - can get pass plateaus more quickly, by “gaining momentum”

Learning Distributed Representations

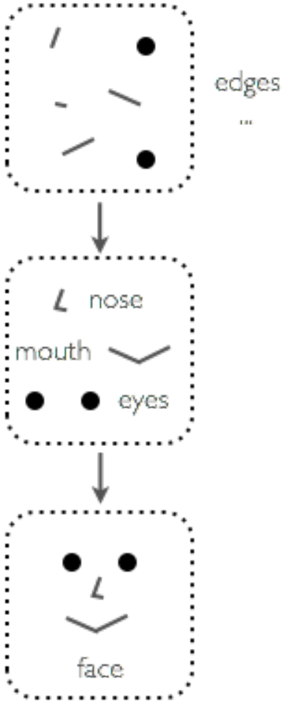
- Deep learning: learning models with multilayer representations
  - multilayer (feed-forward) neural networks
  - multilayer graphical model (deep belief network, deep Boltzmann machine)

- 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

Inspiration from Visual Cortex

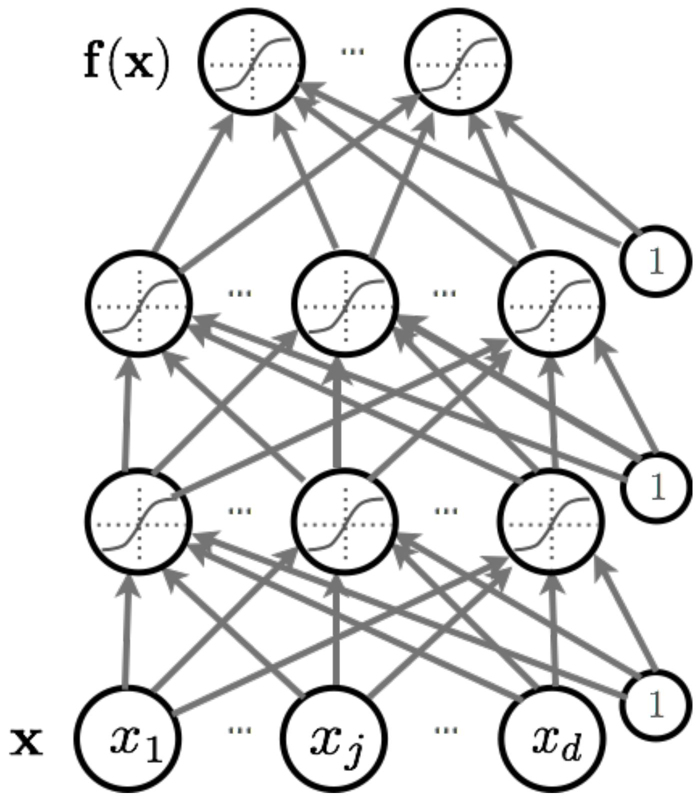


[picture from Simon Thorpe]



Feedforward Neural Networks

- How neural networks predict f(x) given an input 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



Best Practice

- Given a dataset D, pick a model so that:
  - You can achieve 0 training error– Overfit on the training set
- Regularize the model (e.g. using Dropout).
- SGD with momentum, batch-normalization, and dropout usually works very well.

Dropout

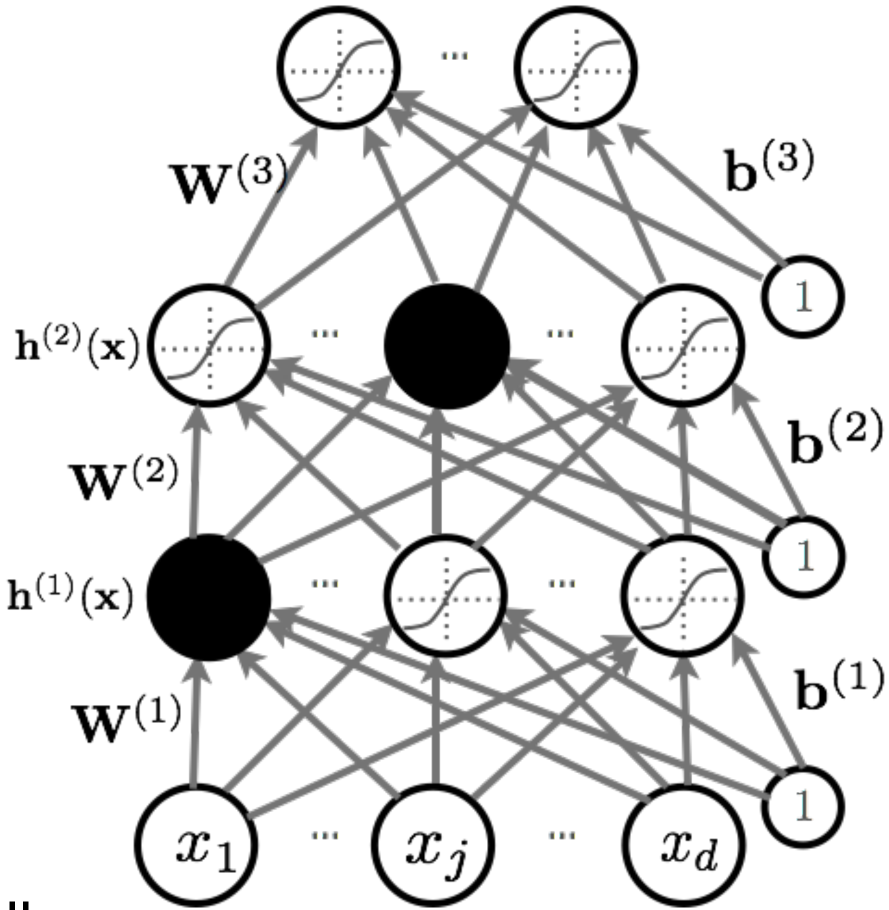
- Key idea: Cripple neural network by removing hidden units stochastically

  - 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



(Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, JMLR 2014)

Dropout

- Use random binary masks $\mathrm{m}^{(k)}$
  - layer pre-activation for k>0

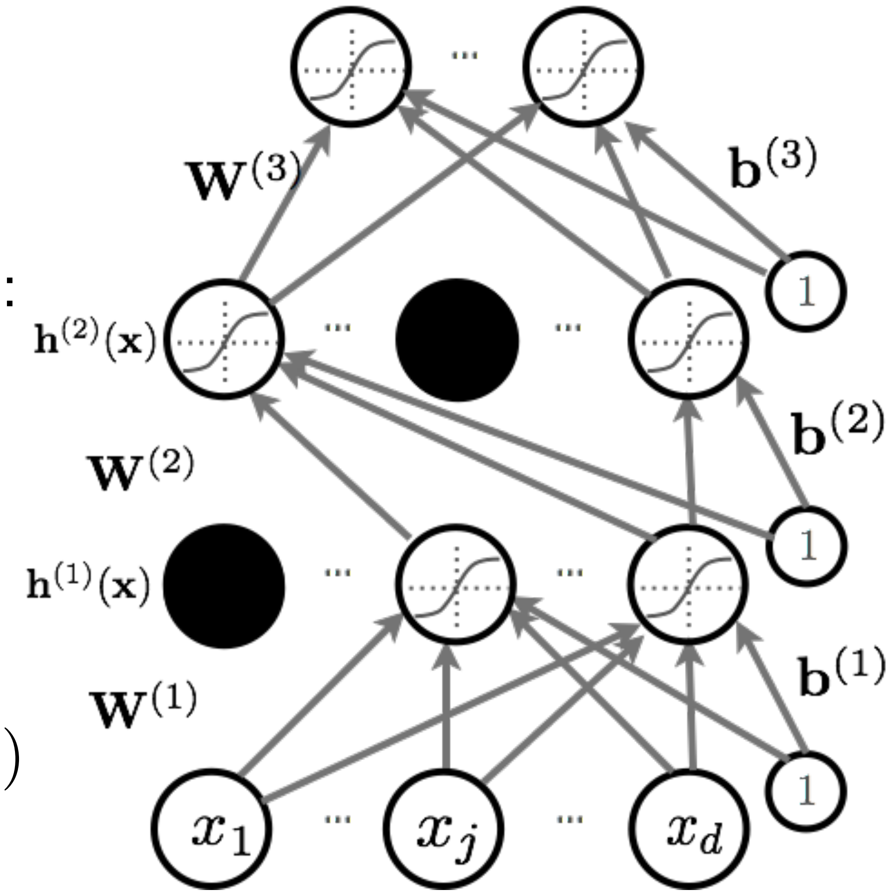
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

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

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

  - Output activation (k=L+1)

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



(Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, JMLR 2014)

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.

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, 2015)
  - 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

(Ioffe and Szegedy, ICML 2015)

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

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

$$\boxed{y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)} \qquad \text{// scale and shift}$$

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

(Ioffe and Szegedy, ICML 2015)

Batch Normalization

- Why normalize the pre-activation?
  - can help keep the pre-activation in a non-saturating regime (though the linear transform $y_i \leftarrow \gamma \widehat{x}_i + \beta$ 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

(Ioffe and Szegedy, ICML 2015)

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

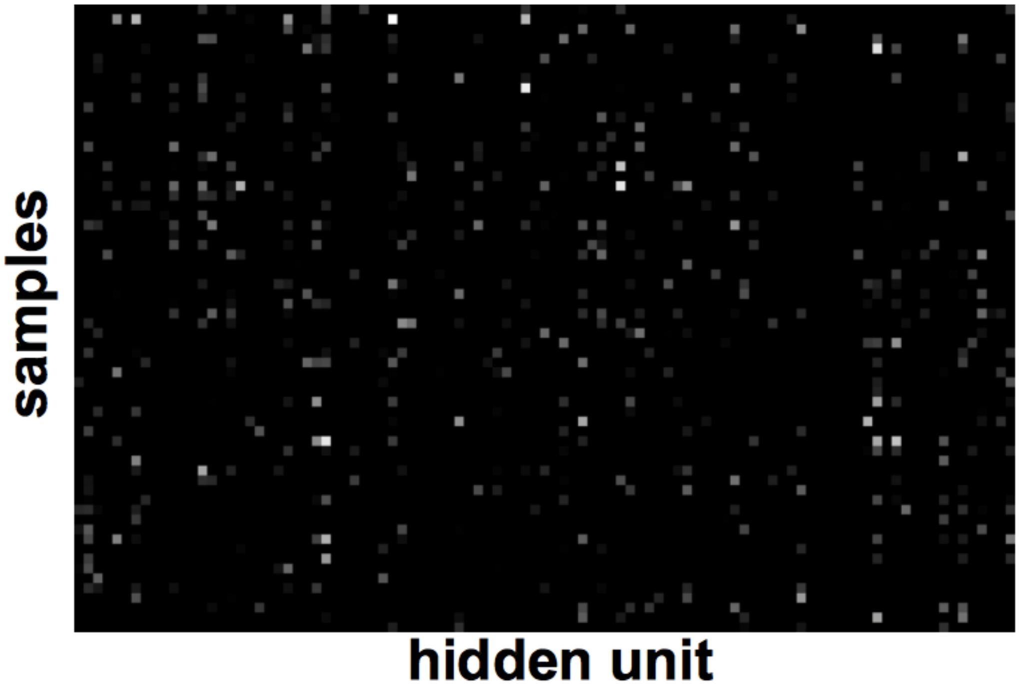
• Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Visualization

• Check gradients numerically by finite differences

• Visualize features (features need to be uncorrelated) and have high variance



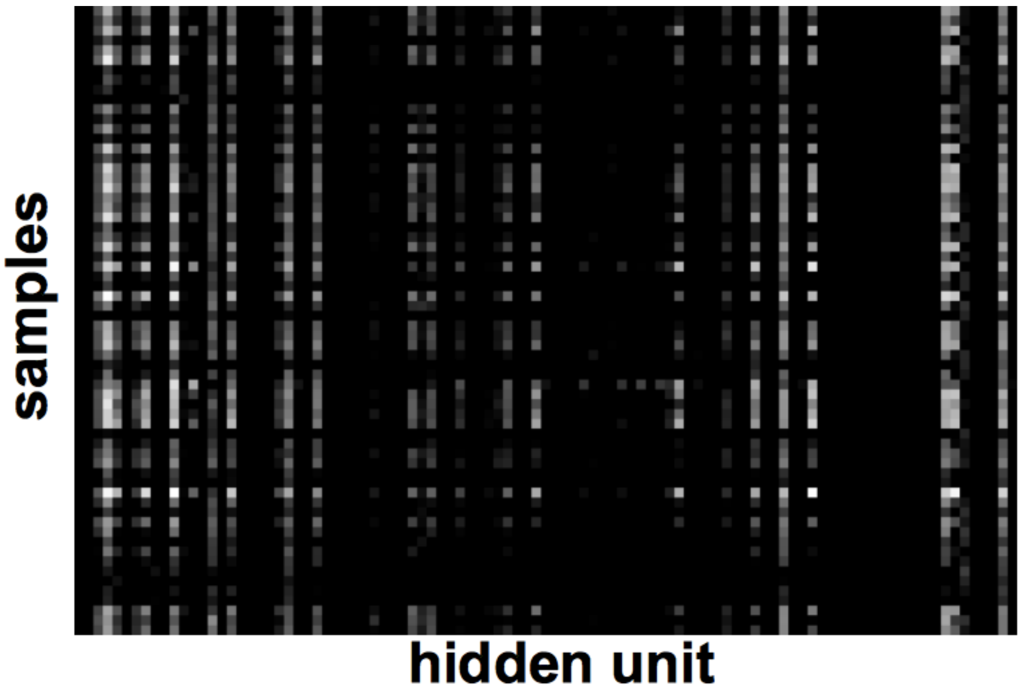
• Good training: hidden units are sparse across samples

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

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

Debugging on Small Dataset

- Next, make sure your model can overfit on a smaller dataset (~ 500-1000 examples)

- If not, investigate the following situations:
  - ➢ Are some of the units saturated, even before the first update?
    - scale down the initialization of your parameters for these units
    - properly normalize the inputs
  - ➢ Is the training error bouncing up and down?
    - decrease the learning rate

- This does not mean that you have computed gradients correctly:
  - ➢ You could still overfit with some of the gradients being wrong

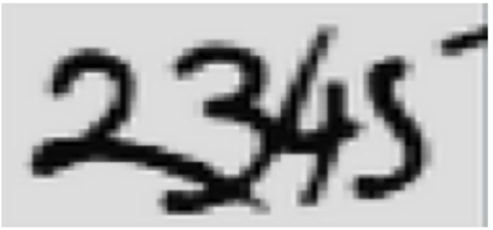
Computer Vision

• 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

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 nuber recognition from StreetView...” ICLR 2014
Jaderberg et al. “Synthetic data and ANN for natural scene text recognition” arXiv 2014

Architecture

- How can we select the right architecture:
  - ➢ Manual tuning of features is now replaced with the manual tuning of architechtures

- Depth
- Width
- Parameter count

How to Choose Architecture

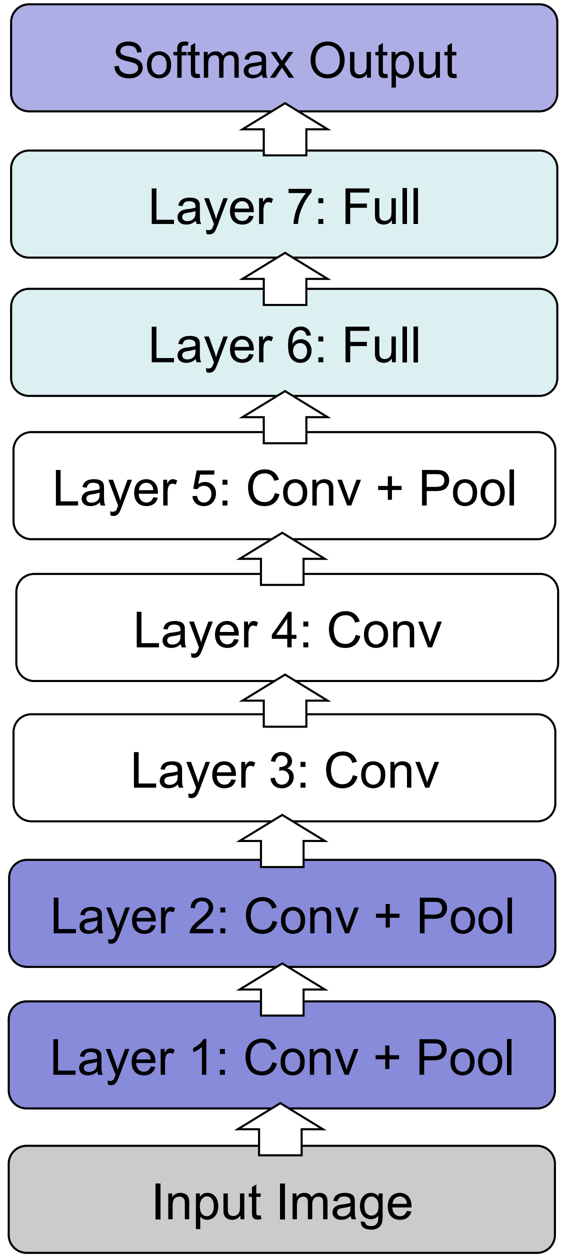
- Many hyper-parameters:
  - Number of layers, number of feature maps
- Cross Validation
- Grid Search (need lots of GPUs)
- Smarter Strategies
  - Random search
  - Bayesian Optimization

AlexNet

- 8 layers total

- Trained on Imagenet dataset [Deng et al. CVPR'09]

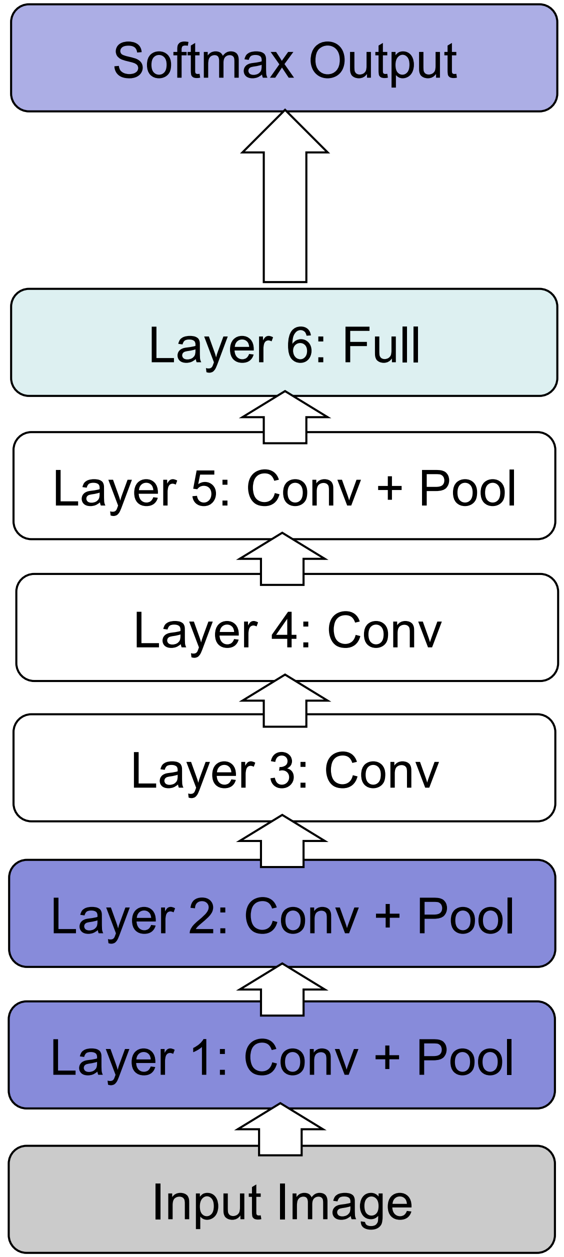
- 18.2% top-5 error



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

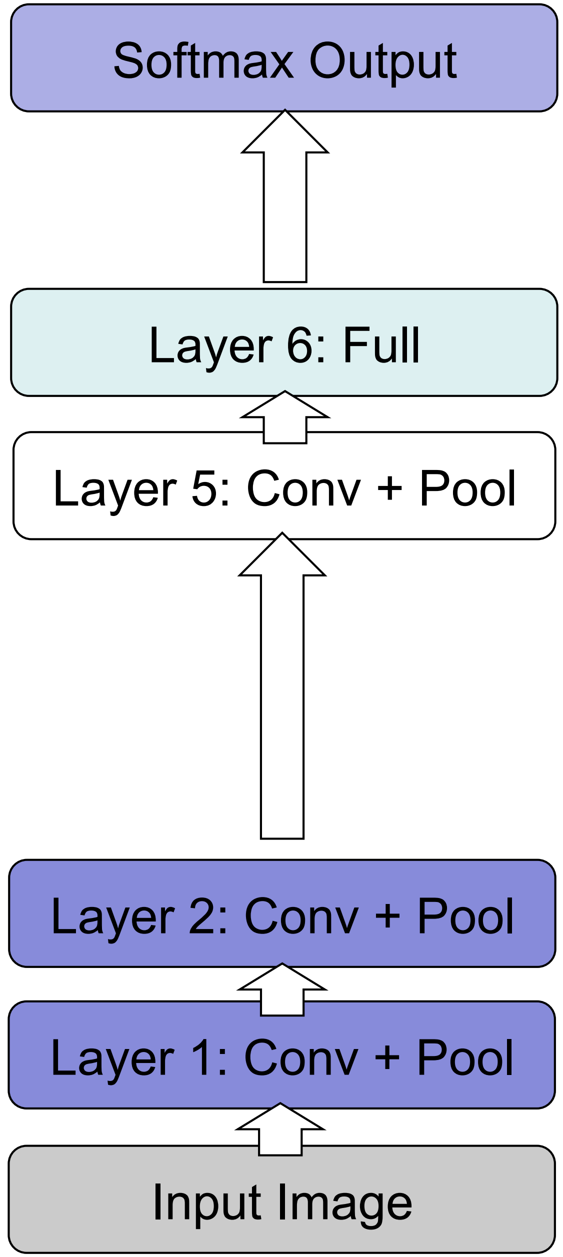
- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in performance!



[From Rob Fergus' CIFAR 2016 tutorial]

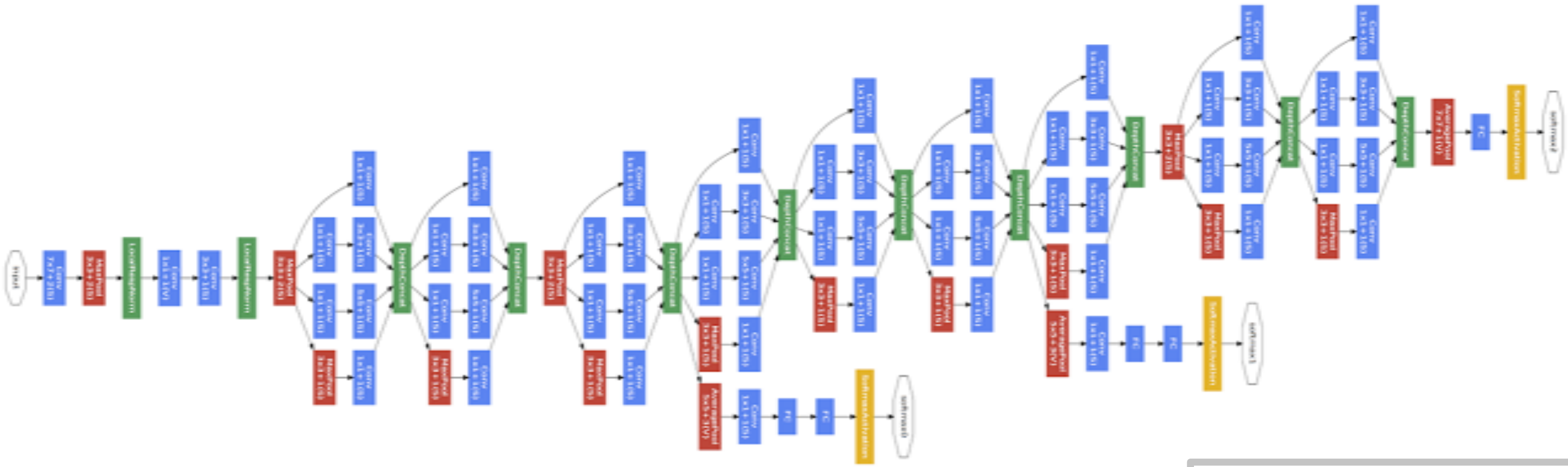
AlexNet

- 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

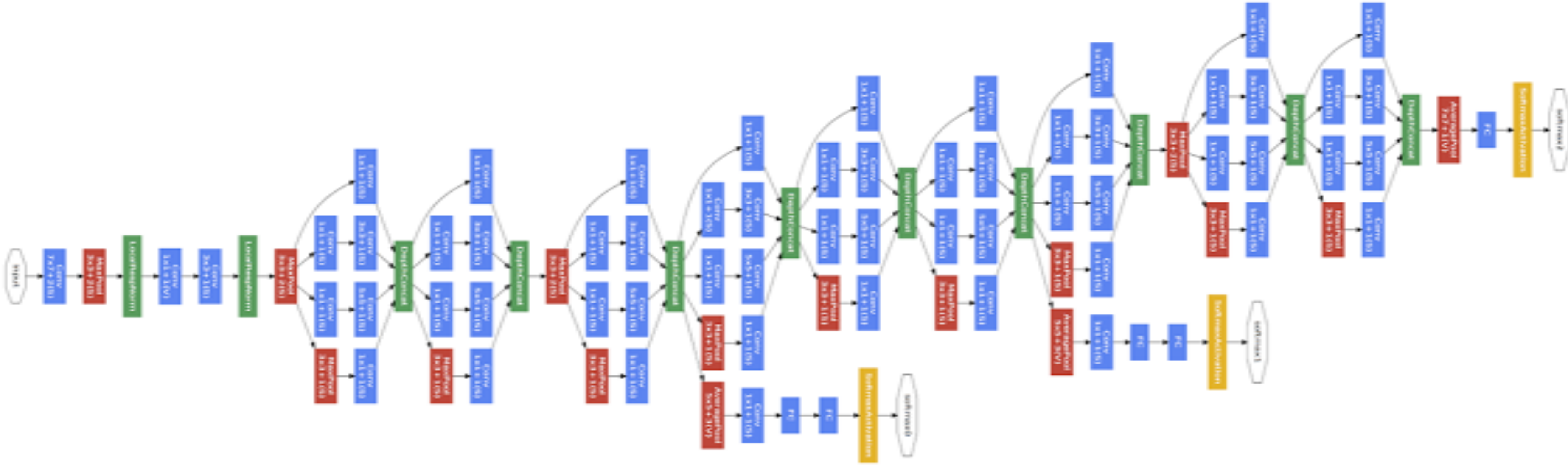


- 24 layer model that uses so-called inception module.

Convolution
Pooling
Softmax
Other

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

GoogLeNet

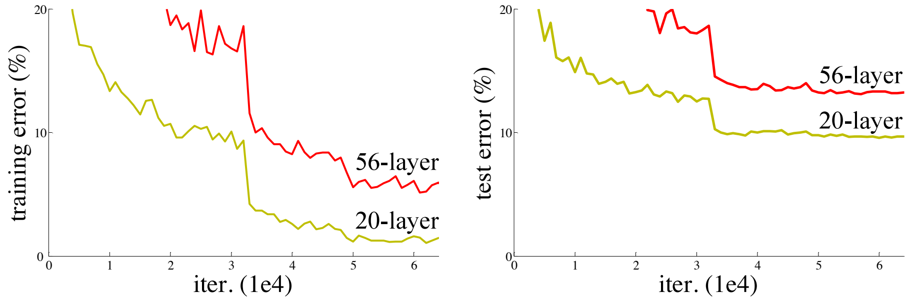


- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

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

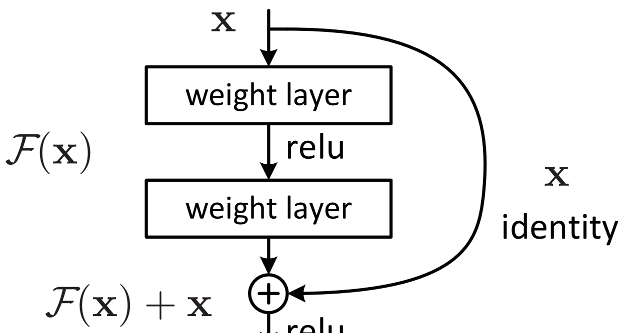
Residual Networks

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



Key idea: introduce “pass through” into each layer

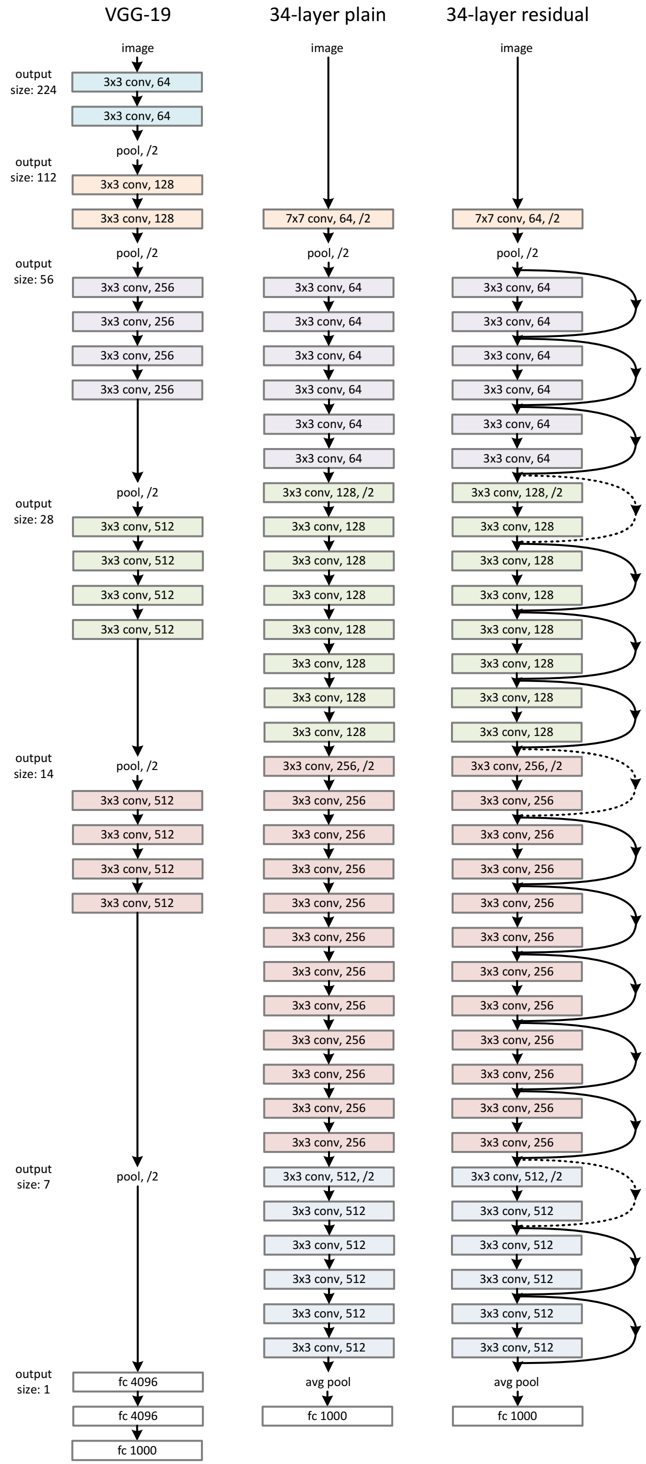
Thus only residual now needs to be learned



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)

End of Part 1