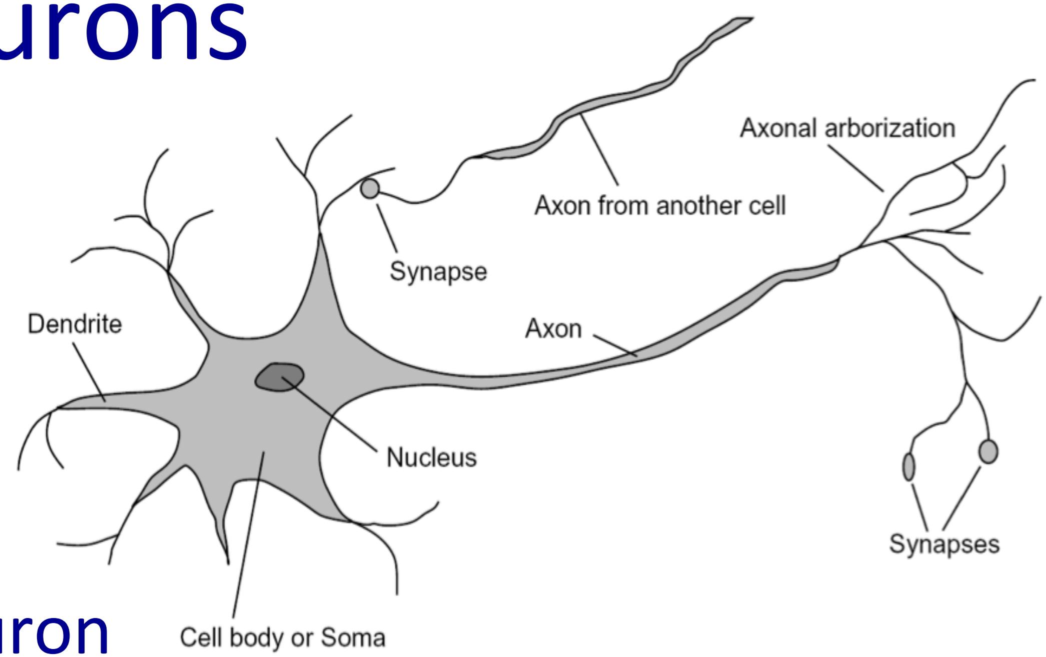
Neural Networks

Instructor: Alan Ritter

Many Slides from Carlos Guestrin and Richard Socher

Human Neurons



- Switching time
  - ~ 0.001 second
- Number of neurons
  - 10^{10}
- Connections per neuron
  - $10^{4\text{-}5}$
- Scene recognition time
  - 0.1 seconds
- Number of cycles per scene recognition?
  - 100 → much parallel computation!

How to Extend Linear Models to Learn Nonlinear Functions?

» Transform the input (e.g. kernel methods)

$$x \to \phi(x)$$

» Q: how do we decide on the right representation?

-Very high-dimensional representation?

How to Extend Linear Models to Learn Nonlinear Functions?

» Transform the input (e.g. kernel methods)

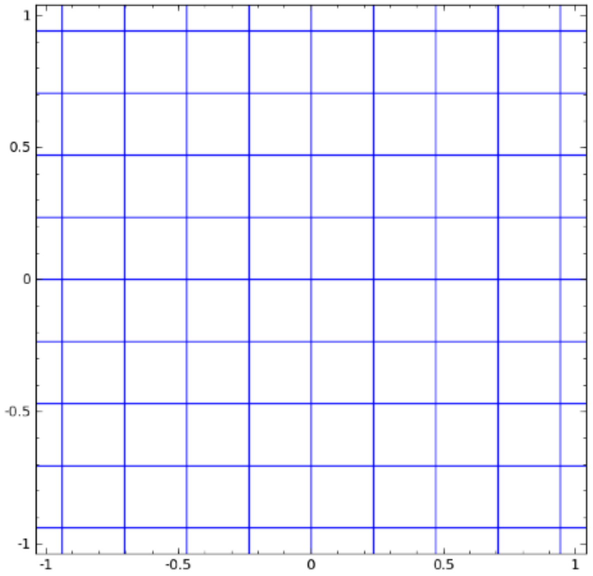
$$x \rightarrow \phi(x)$$

» Q: how do we decide on the right representation?

-Very high-dimensional representation?

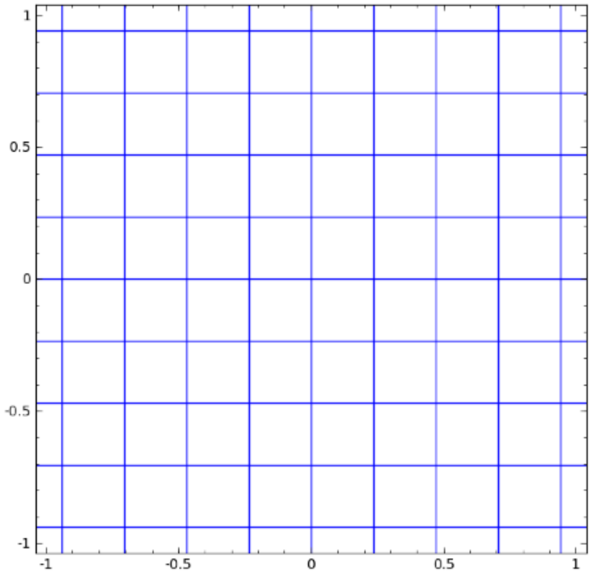
-Neural Networks: Let's learn the representation!

Neural Networks



Neural Networks

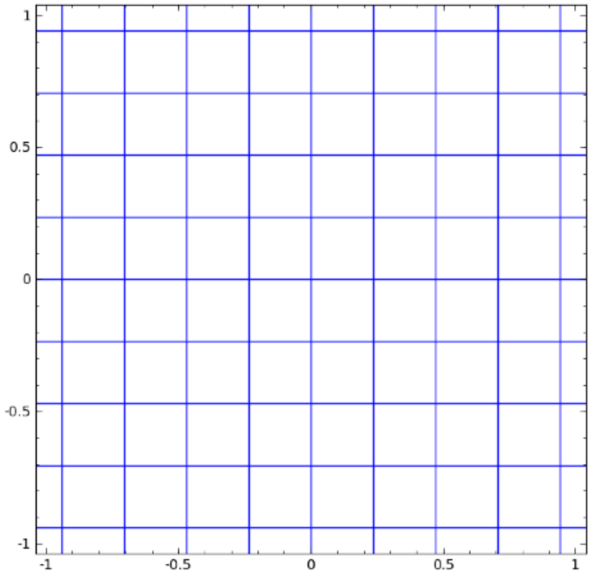
Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$



Neural Networks

Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

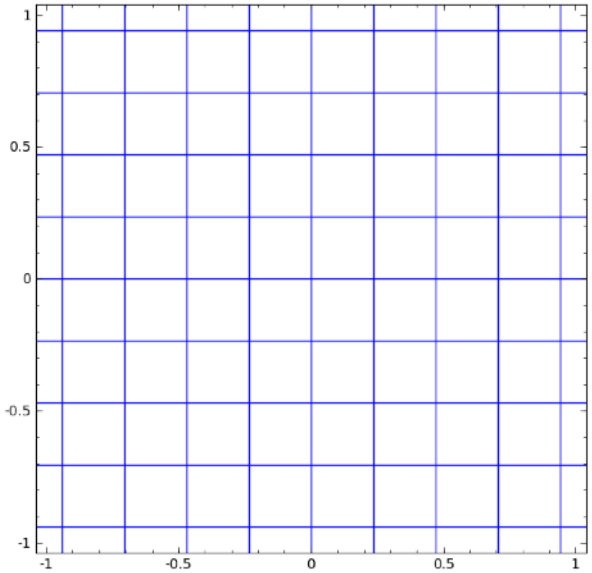


Neural Networks

Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$
$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Nonlinear transformation



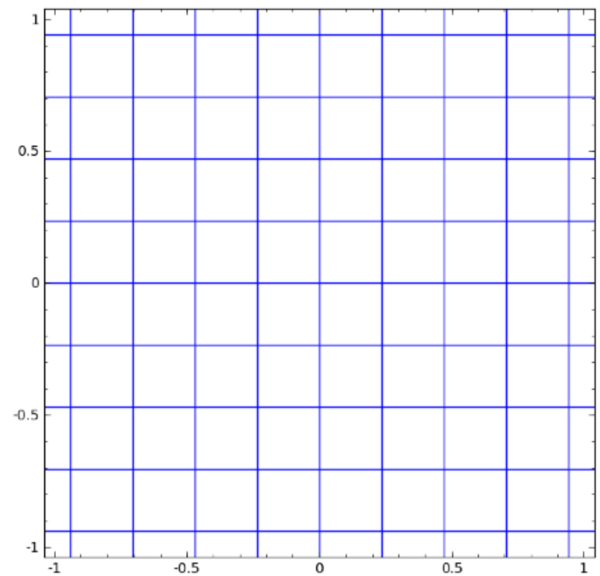
Neural Networks

Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$
$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Nonlinear transformation

Warp space



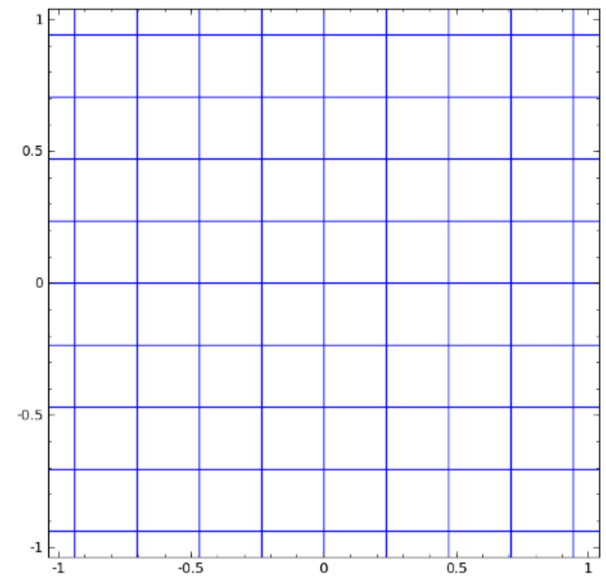
Neural Networks

Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Nonlinear transformation　Warp space　Shift



Neural Networks

Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

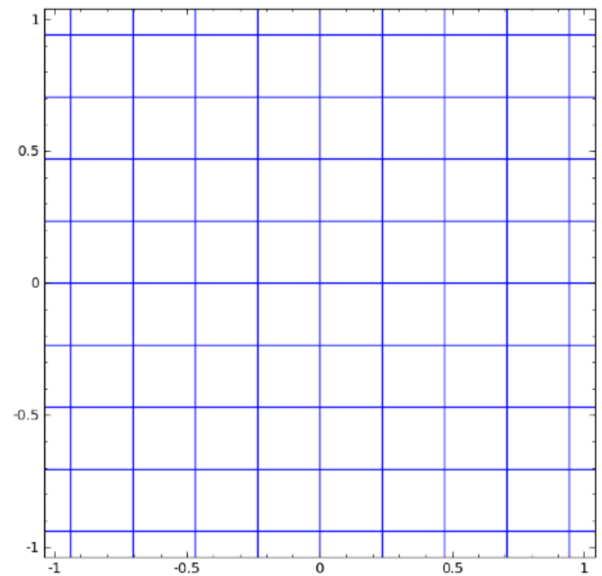
$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Nonlinear transformation

Warp space

Shift



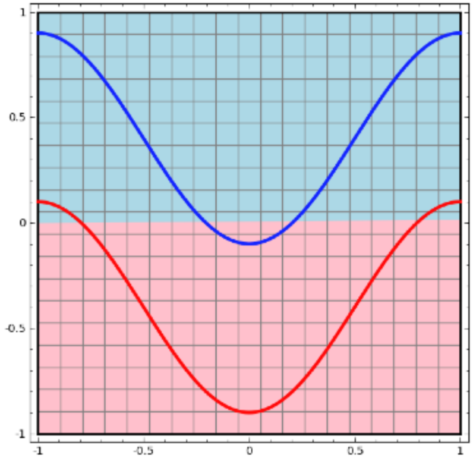
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

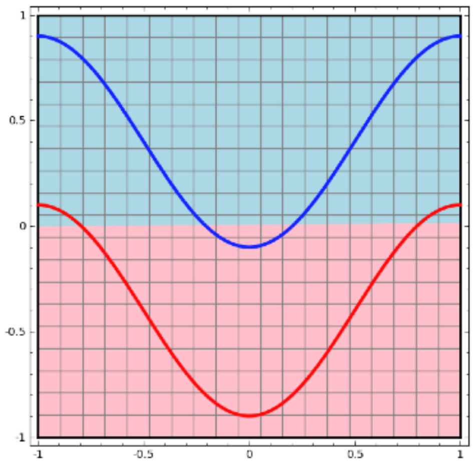
Linear classifier



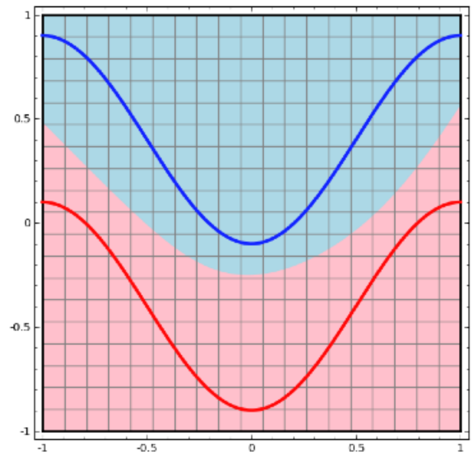
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

Linear classifier



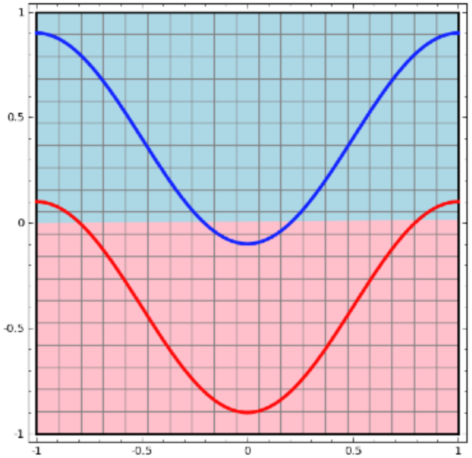
Neural network



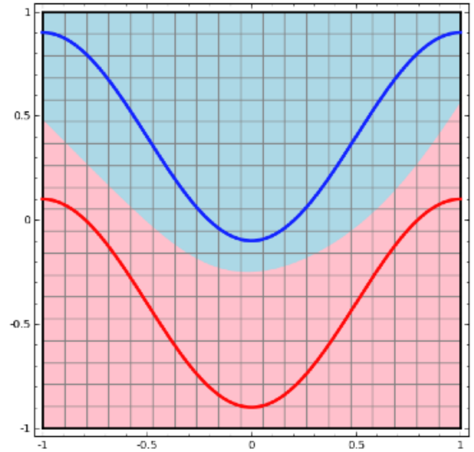
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

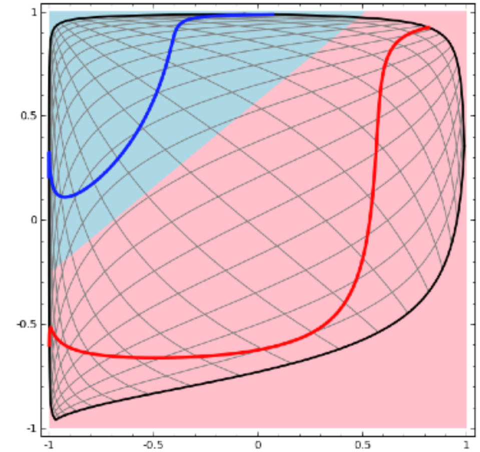
Linear classifier



Neural network

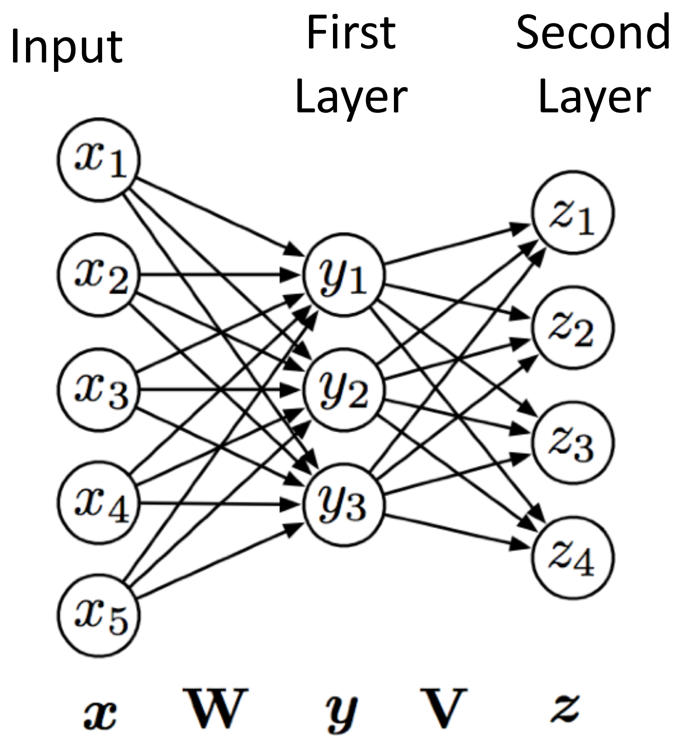


…possible because we transformed the space!



Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

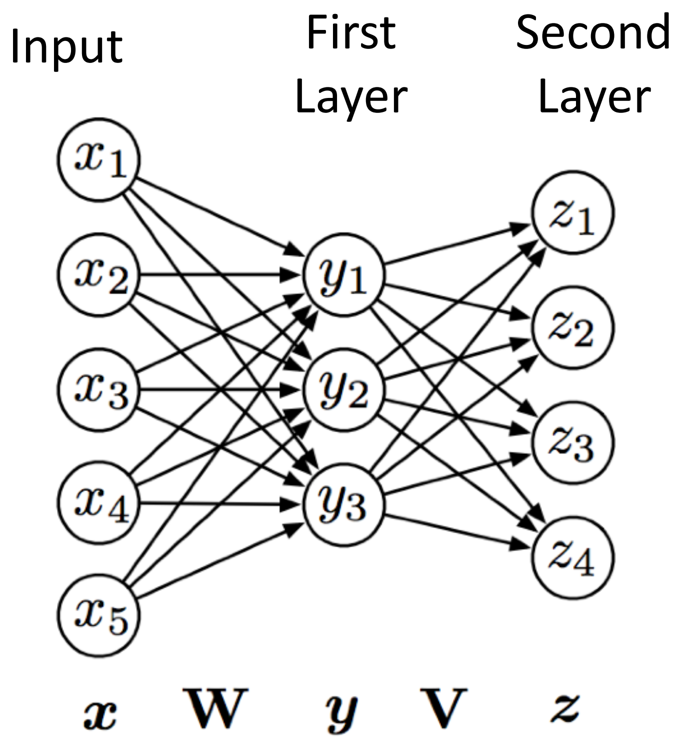
Deep Neural Networks



$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

Adopted from Chris Dyer

Deep Neural Networks

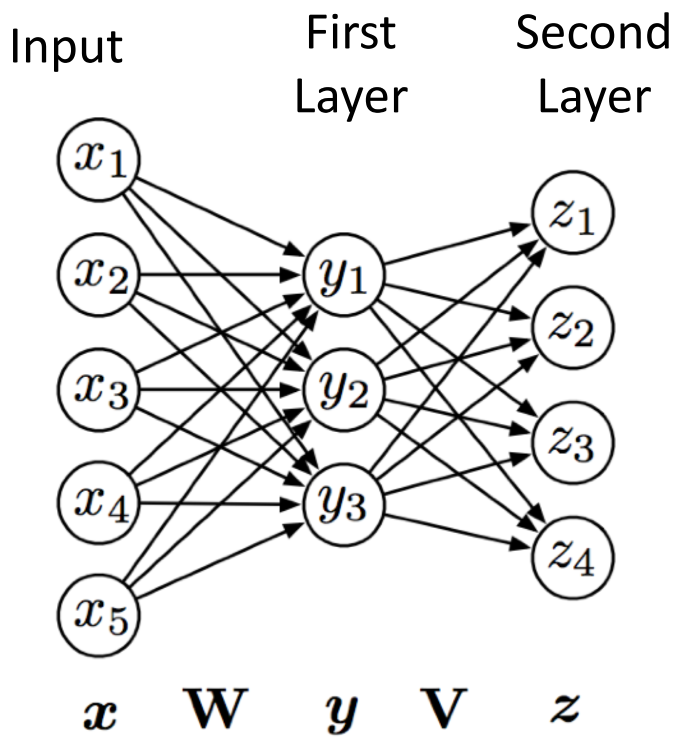


$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$$

Adopted from Chris Dyer

Deep Neural Networks



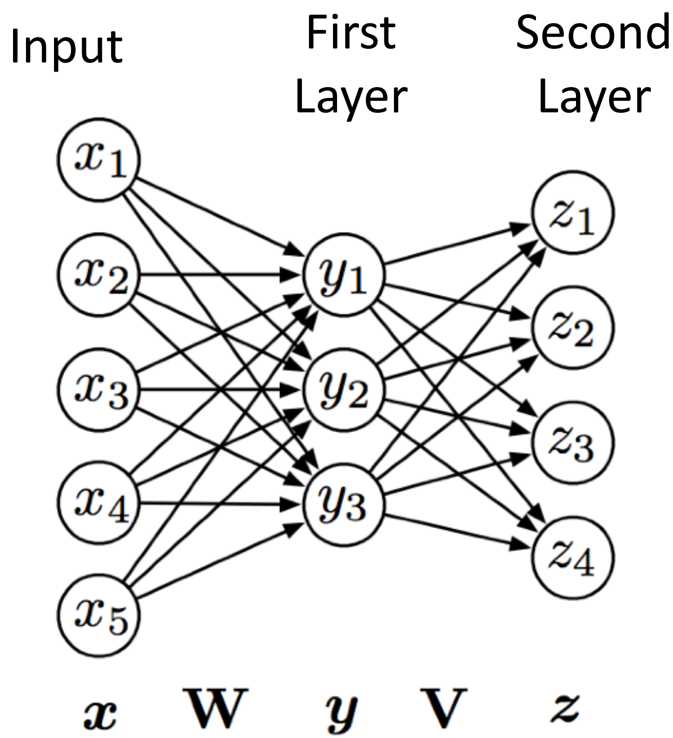
$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$$

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

Adopted from Chris Dyer

Deep Neural Networks



$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

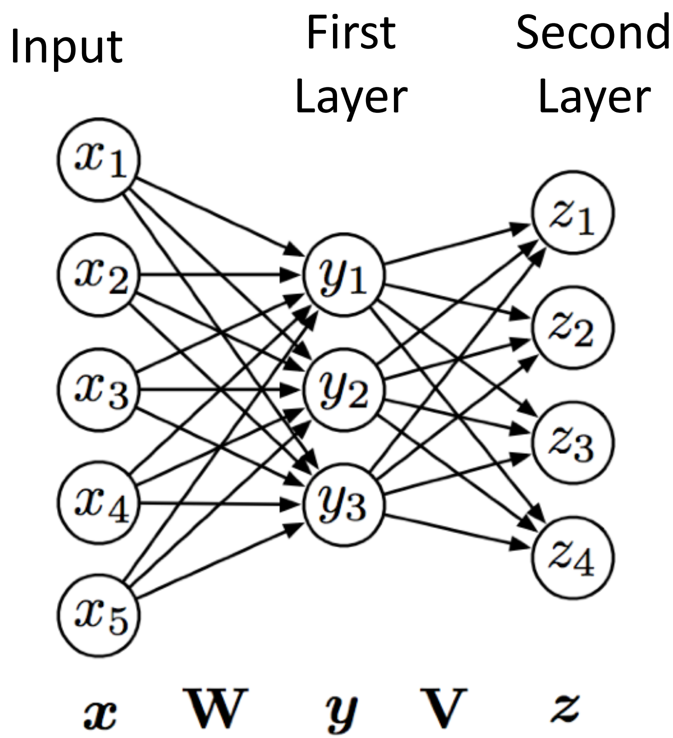
$$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$$

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

"Feedforward" computation (not recurrent)

Adopted from Chris Dyer

Deep Neural Networks



$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$$

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

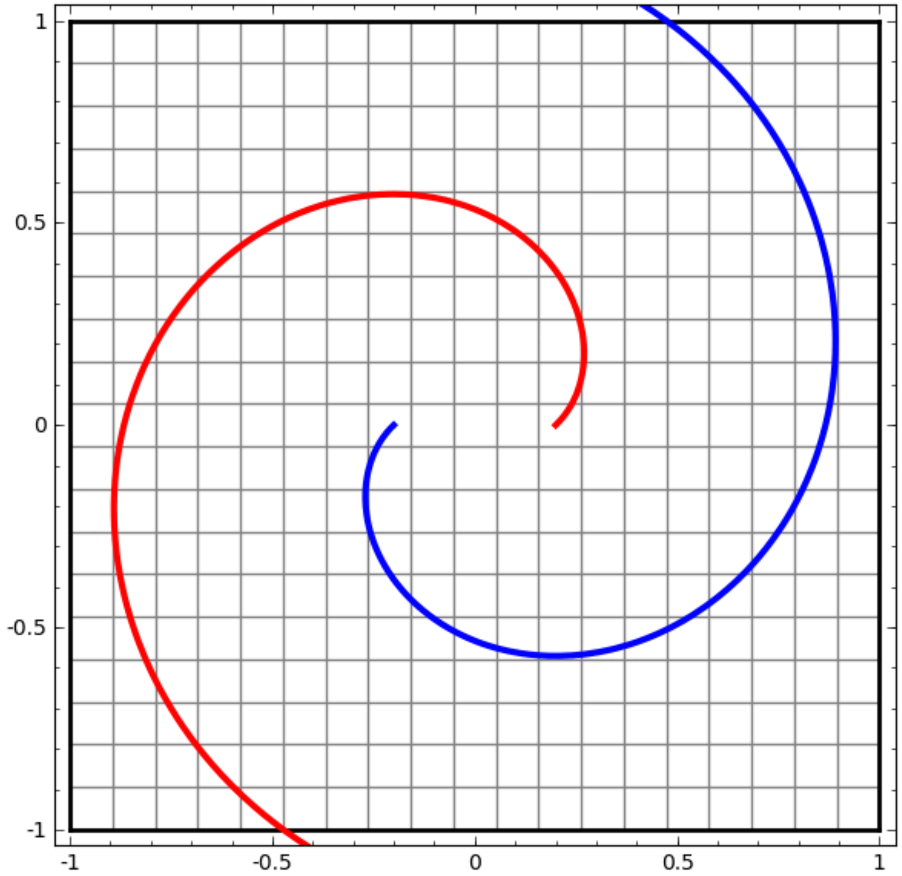
"Feedforward" computation (not recurrent)

Check: what happens if no nonlinearity? More powerful than basic linear models?

$$\mathbf{z} = \mathbf{V}(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{c}$$

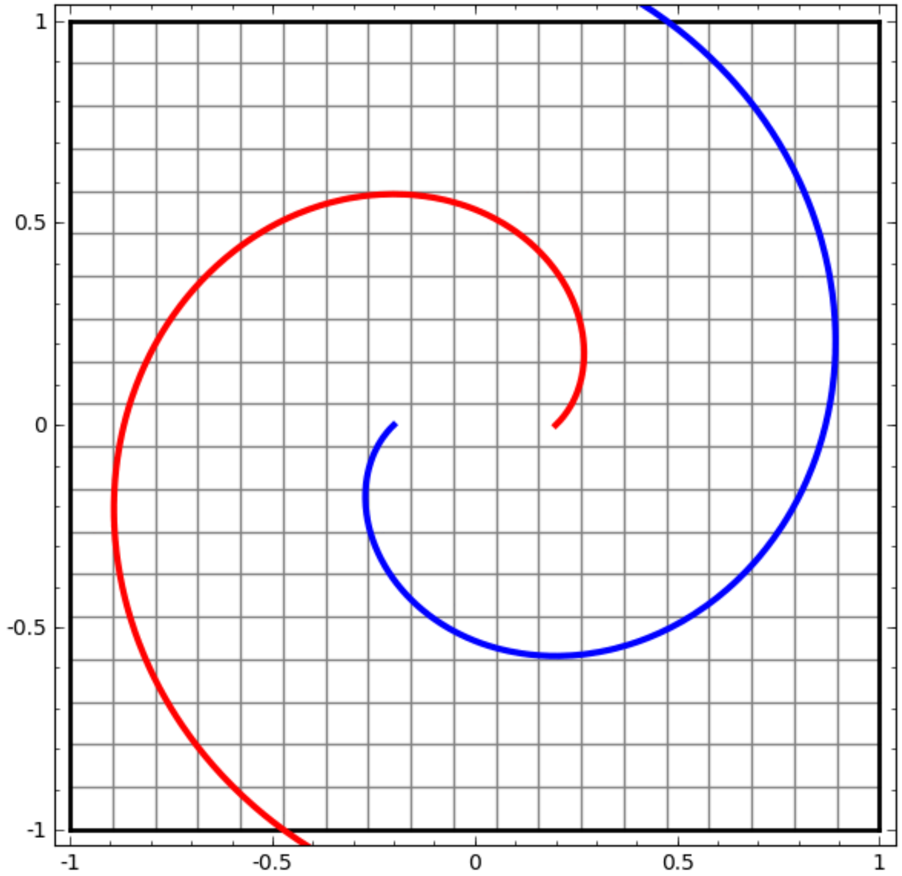
Adopted from Chris Dyer

Deep Neural Networks



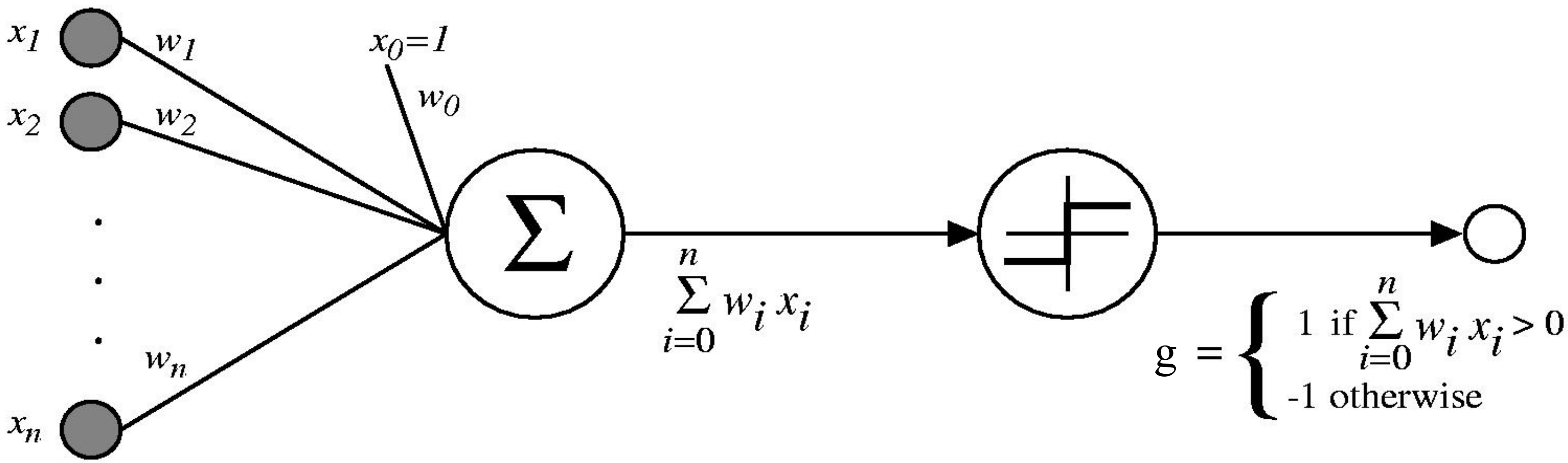
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Deep Neural Networks



Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

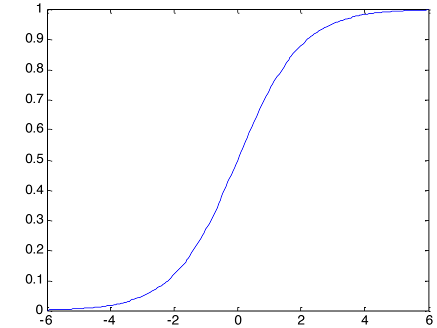
Perceptron as a Neural Network



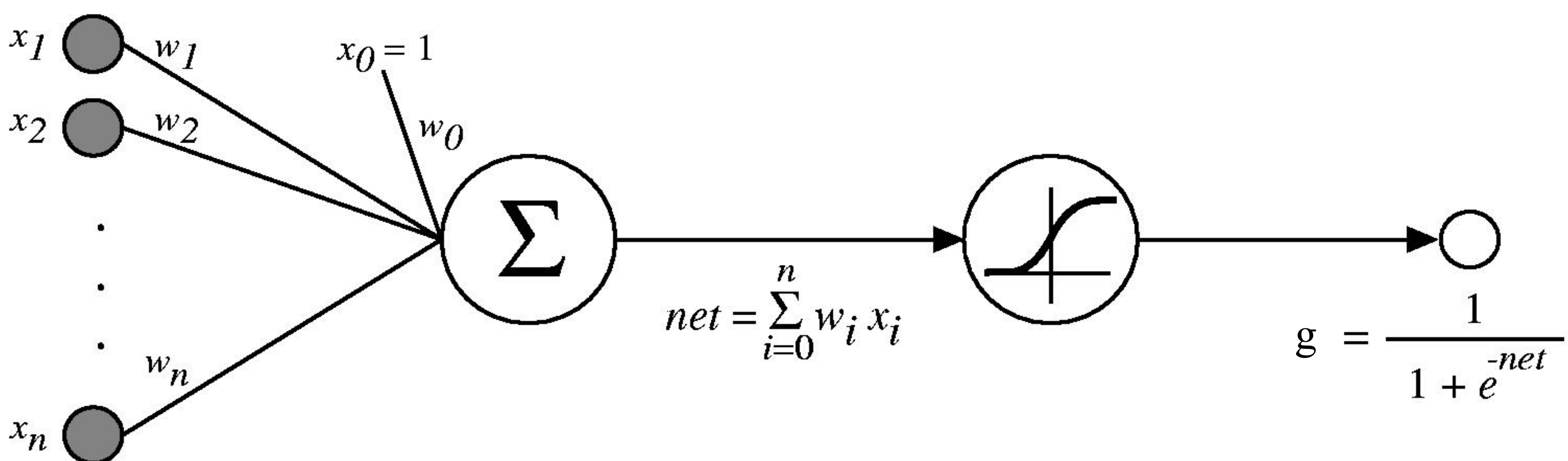
This is one neuron:

- Input edges $x_1 \ldots x_n$, along with basis
- The sum is represented graphically
- Sum passed through an activation function g

Sigmoid Neuron



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$



Just change g!

- Why would be want to do this?
- Notice new output range [0,1]. What was it before?
- Look familiar?

Optimizing a neuron

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$$

We train to minimize sum-squared error

$$\ell(W) = \frac{1}{2}\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]^2$$

$$\frac{\partial l}{\partial w_i} = -\sum_j [y_j - g(w_0 + \sum_i w_i x_i^j)] \frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j)$$

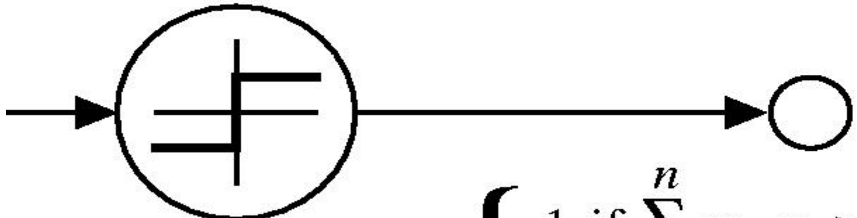
$$\frac{\partial}{\partial w_i} g(w_0 + \sum_i w_i x_i^j) = x_i^j g'(w_0 + \sum_i w_i x_i^j)$$

$$\boxed{\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] \; x_i^j \; g'(w_0 + \sum_i w_i x_i^j)}$$

Solution just depends on g’: derivative of activation function!

Re-deriving the perceptron update

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]\ x_i^j\ g'(w_0 + \sum_i w_i x_i^j)$$



$$g = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]\ x_i^j$$

For a specific, incorrect example:

- $w = w + y^*x$ (our familiar update!)

Re-deriving the perceptron update

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] \; x_i^j \; g'(w_0 + \sum_i w_i x_i^j)$$

=0
everywhere :(

$$g = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ \text{-1 otherwise} \end{cases}$$

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)] \; x_i^j$$

For a specific, incorrect example:

- $w = w + y*x$ (our familiar update!)

Sigmoid units: have to differentiate g

$$\frac{\partial \ell(W)}{\partial w_i} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]\ x_i^j\ g'(w_0 + \sum_i w_i x_i^j)$$

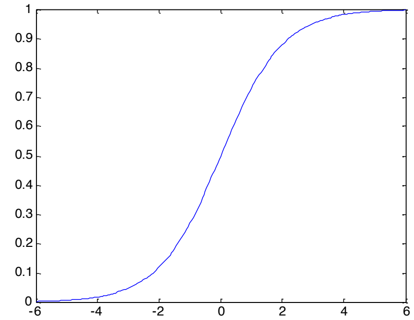
$$g(x) = \frac{1}{1+e^{-x}} \qquad g'(x) = g(x)(1-g(x))$$

$$\boxed{w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j}$$

$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)] g^j (1 - g^j)$$

$$g^j = g(w_0 + \sum_i w_i x_i^j)$$

Aside: Comparison to logistic regression



- P(Y|X) represented by:

$$P(Y = 1 \mid x, W) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$
$$= g(w_0 + \sum_i w_i x_i)$$

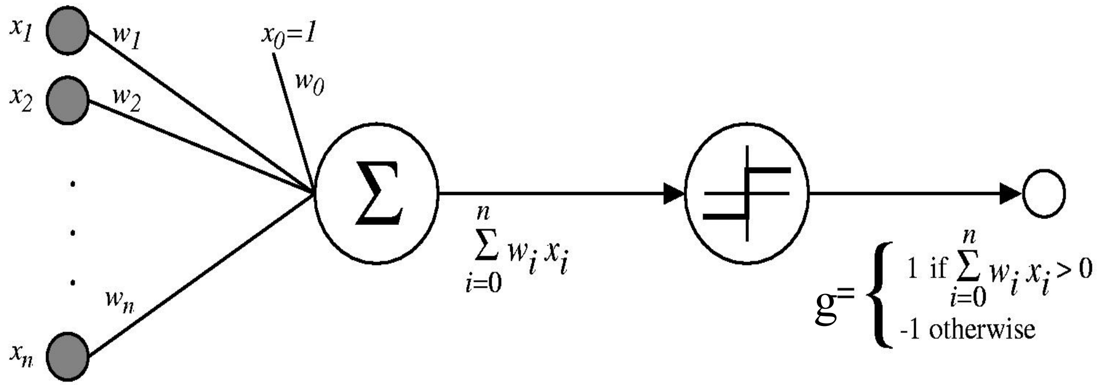
- Learning rule – MLE:

$$\frac{\partial \ell(W)}{\partial w_i} = \sum_j x_i^j [y^j - P(Y^j = 1 \mid x^j, W)]$$
$$= \sum_j x_i^j [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

$$w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j$$

$$\delta^j = y^j - g(w_0 + \sum_i w_i x_i^j)$$

Perceptron, linear classification, Boolean functions: $x_i \in \{0,1\}$



- Can learn $x_1 \vee x_2$?
  - $-0.5 + x_1 + x_2$
- Can learn $x_1 \wedge x_2$?
  - $-1.5 + x_1 + x_2$
- Can learn any conjunction or disjunction?
  - $-0.5 + x_1 + \ldots + x_n$
  - $(n-0.5) + x_1 + \ldots + x_n$
- Can learn majority?
  - $(-0.5*n) + x_1 + \ldots + x_n$
- What are we missing? The dreaded XOR!, etc.

Going beyond linear classification

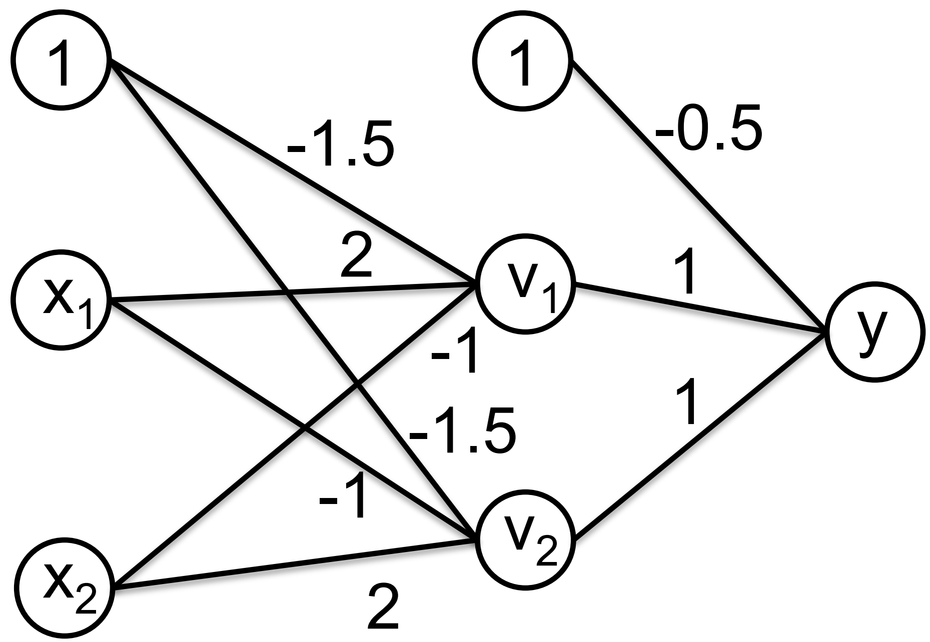
Solving the XOR problem

$y = x_1 \text{ XOR } x_2 = (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_1)$

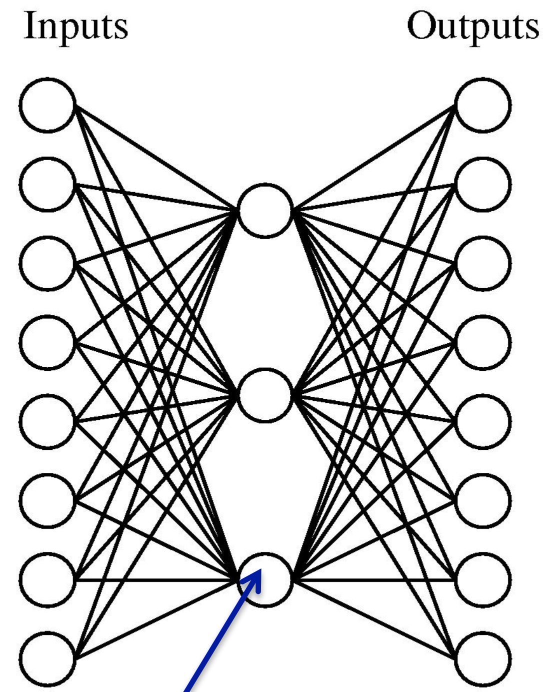
$v_1 = (x_1 \wedge \neg x_2)$
$\quad = -1.5 + 2x_1 - x_2$

$v_2 = (x_2 \wedge \neg x_1)$
$\quad = -1.5 + 2x_2 - x_1$

$y = v_1 \vee v_2$
$\quad = -0.5 + v_1 + v_2$



Hidden layer



- Single unit:

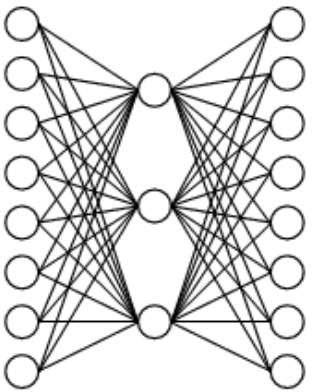
$$out(\mathbf{x}) \;=\; g(w_0 + \sum_i w_i x_i)$$

- 1-hidden layer:

$$out(\mathbf{x}) \;=\; g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

- No longer convex function!

Example data for NN with hidden layer

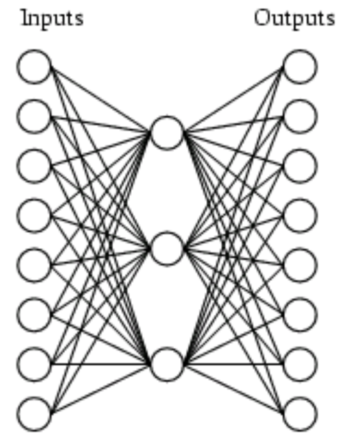


A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned??

Example data for NN with hidden layer



A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

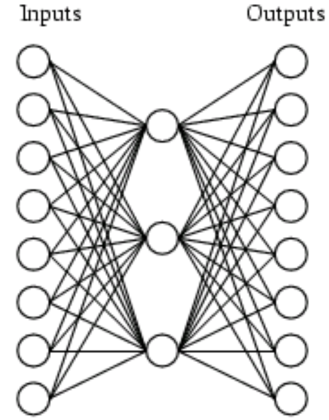
Autoencoder
(kind of unsupervised learning)

Dimensionality Reduction

Can this be learned??

Learned weights for hidden layer

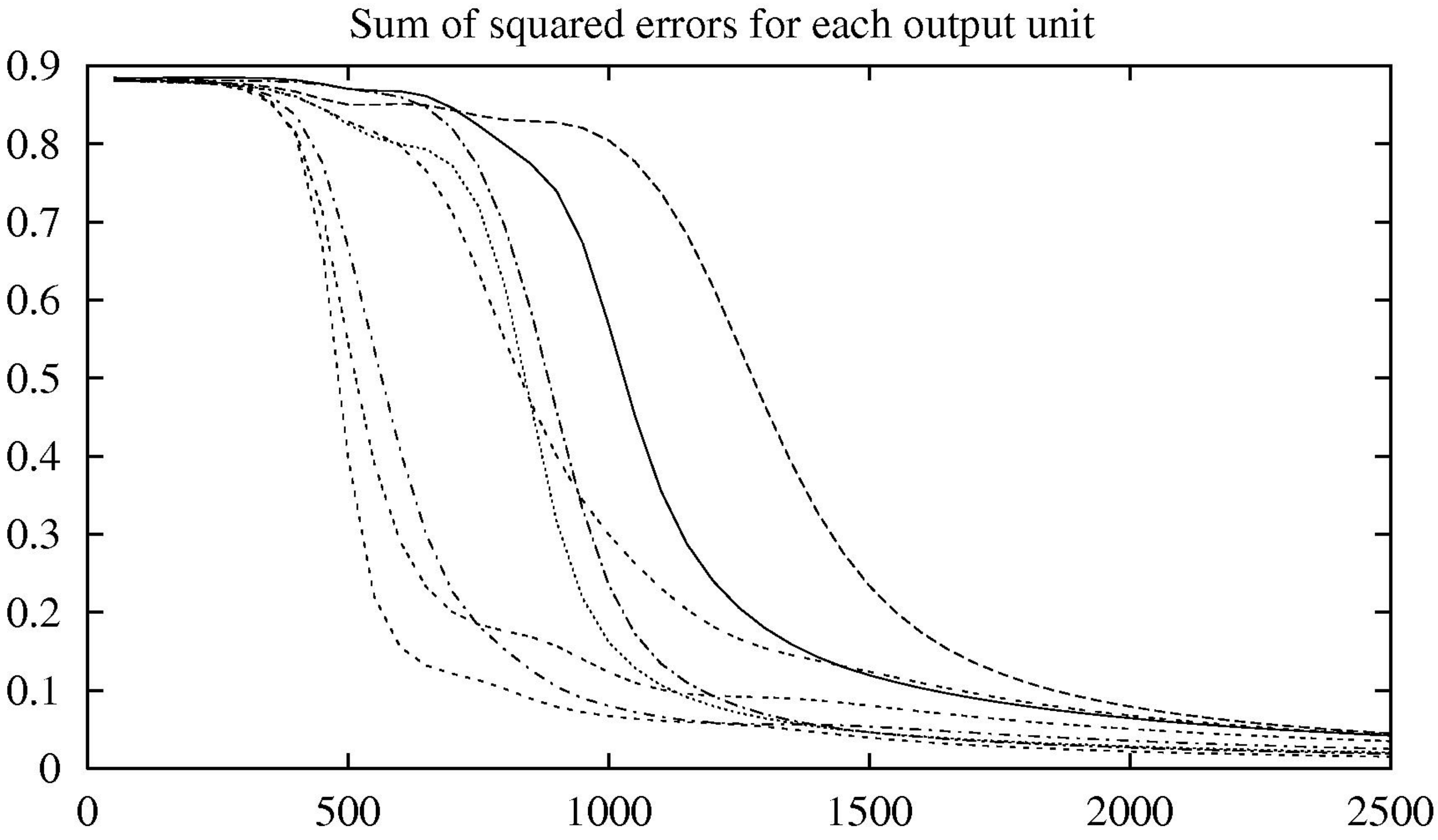
A network:



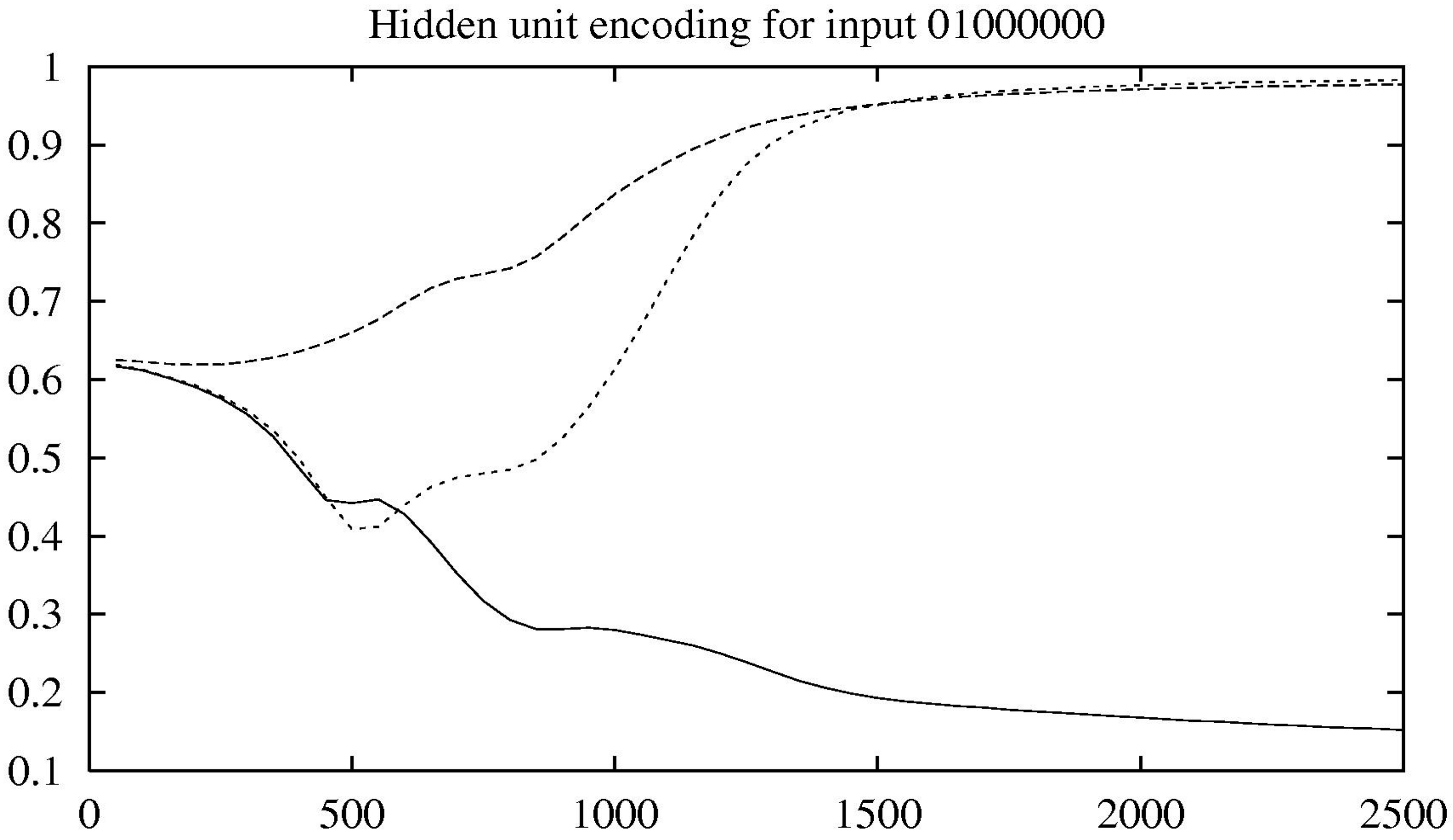
Learned hidden layer representation:

Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$	00001000
00000100	$\rightarrow$	.22	.99	.99	$\rightarrow$	00000100
00000010	$\rightarrow$	.80	.01	.98	$\rightarrow$	00000010
00000001	$\rightarrow$	.60	.94	.01	$\rightarrow$	00000001

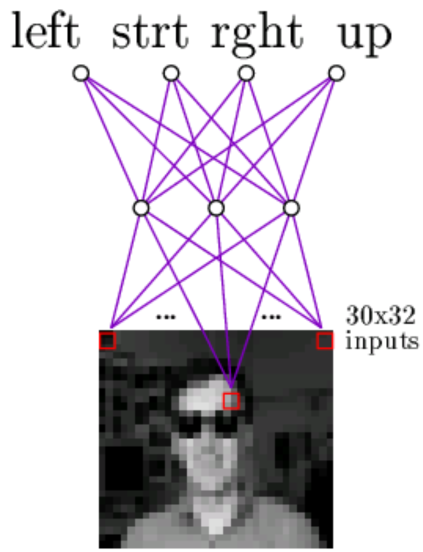
Learning the weights



Learning an encoding



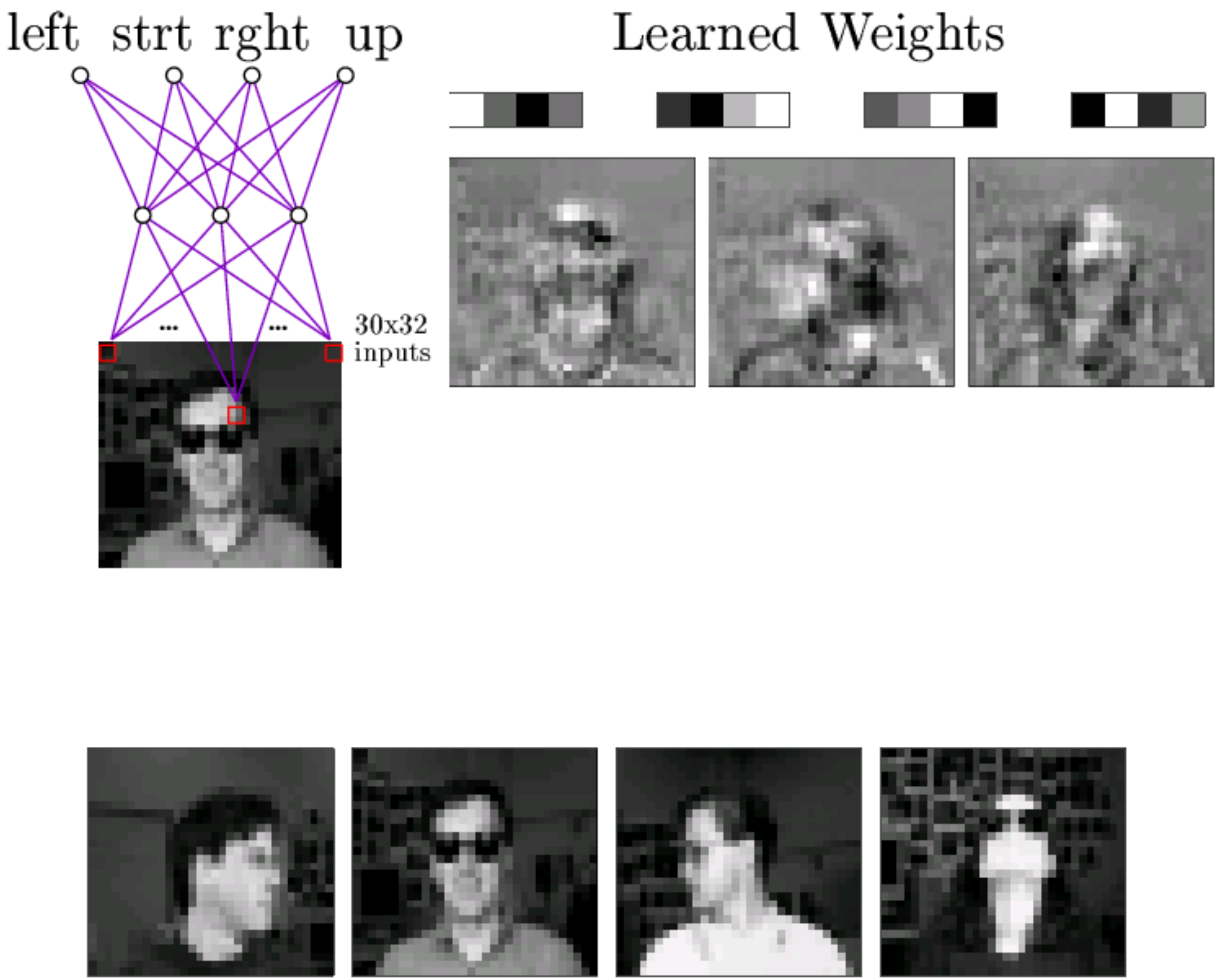
NN for images




Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Weights in NN for images



Typical input images

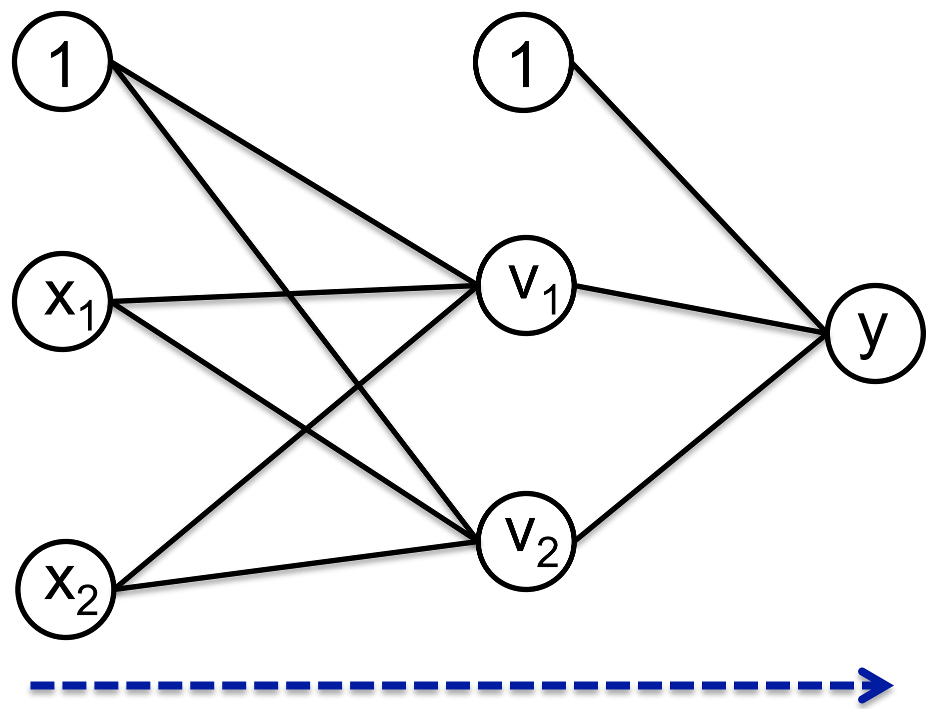
Forward propagation

1-hidden layer:

$$out(\mathbf{x}) \;=\; g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

Compute values left to right

1. Inputs: $x_1, \ldots, x_n$
2. Hidden: $v_1, \ldots, v_n$
3. Output: y



Forward propagation

1-hidden layer:

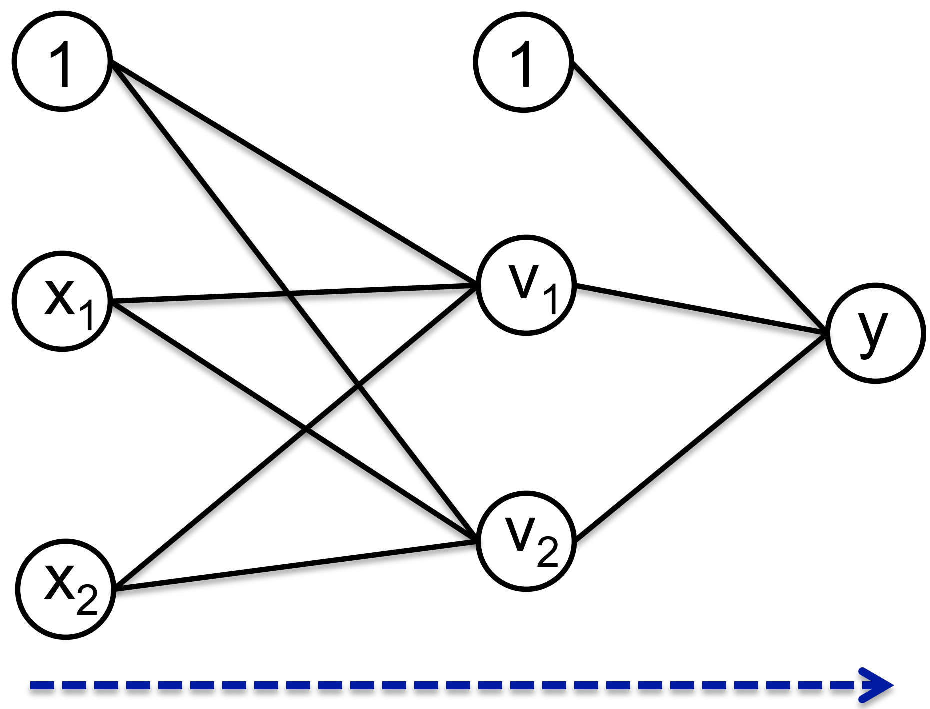
k sums over hidden layer

i sums over input layer

$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

Compute values left to right

1. Inputs: $x_1, \ldots, x_n$
2. Hidden: $v_1, \ldots, v_n$
3. Output: y



Gradient descent for 1-hidden layer

$$\boxed{\frac{\partial \ell(W)}{\partial w_k}}$$

Dropped w_0 to make derivation simpler

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right)$$

$$\boxed{v_k^j = g\left(\sum_{i'} w_{i'}^{k'} x_{i'}\right)}$$

$$\frac{\partial \ell(W)}{\partial w_k} = \sum_{j=1}^{m} -[y^j - out(\mathbf{x}^j)]\frac{\partial out(\mathbf{x}^j)}{\partial w_k}$$

$$out(x) = g\left(\sum_{k'} w_{k'} v_{k'}^j\right)$$

$$\frac{\partial out(\mathbf{x})}{\partial w_k} = v_k^j g'\left(\sum_{k'} w_{k'} v_{k'}^j\right)$$

Gradient for last layer same as the single node case, but with hidden nodes v as input!

Gradient descent for 1-hidden layer

$$\boxed{\frac{\partial \ell(W)}{\partial w_i^k}}$$

Dropped w_0 to make derivation simpler

$$\boxed{\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)}$$

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right)$$

$$\frac{\partial \ell(W)}{\partial w_i^k} = \sum_{j=1}^{m} -[y - out(\mathbf{x}^j)]\frac{\partial out(\mathbf{x}^j)}{\partial w_i^k}$$

$$\frac{\partial out(\mathbf{x})}{\partial w_i^k} = g'\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right) \frac{\partial}{\partial w_i^k} g\left(\sum_{i'} w_{i'}^{k'} x_{i'}\right) w_k$$

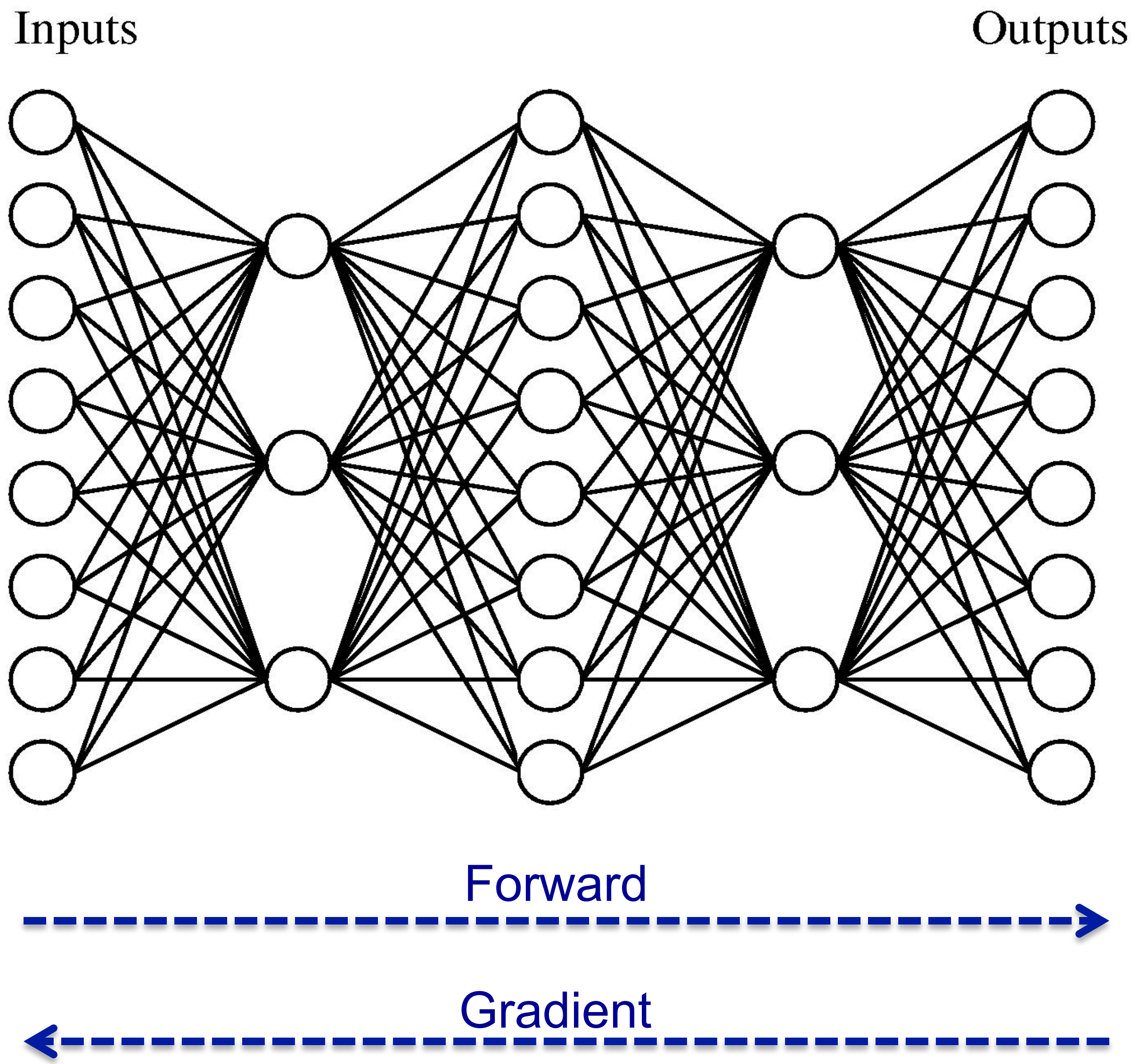
For hidden layer, two parts:

- Normal update for single neuron
- Recursive computation of gradient on output layer

Multilayer neural networks

Inference and Learning:

- Forward pass: left to right, each hidden layer in turn
- Gradient computation: right to left, propagating gradient for each node



Forward propagation – prediction

- Recursive algorithm
- Start from input layer
- Output of node V_k with parents $U_1, U_2, \ldots$:

$$\boxed{V_k \; = \; g\left(\sum_i w_i^k U_i\right)}$$

Back-propagation – learning

- Just gradient descent!!!
- Recursive algorithm for computing gradient
- For each example
  - Perform forward propagation
  - Start from output layer
    - Compute gradient of node V_k with parents $U_1, U_2, \ldots$
    - Update weight w_i^k
    - Repeat (move to preceding layer)

Back-propagation – pseudocode

Initialize all weights to small random numbers

- Until convergence, do:
  - For each training example x,y:
    1. Forward propagation, compute node values V_k
    2. For each output unit o (with labeled output y):

       $$\delta_o = V_o(1-V_o)(y-V_o)$$

    3. For each hidden unit h:

       $$\delta_h = V_h(1-V_h) \sum_{k \text{ in } output(h)} w_{h,k}\delta_k$$

    4. Update each network weight $w_{i,j}$ from node i to node j

       $$w_{i,j} = w_{i,j} + \eta\delta_j x_{i,j}$$

Convergence of backprop

- Perceptron leads to convex optimization
  - Gradient descent reaches **global minima**
- Multilayer neural nets **not convex**
  - Gradient descent gets stuck in local minima
  - Selecting number of hidden units and layers = fuzzy process
  - NNs have made a HUGE comeback in the last few years!!!
    - Neural nets are back with a new name!!!!
      - Deep belief networks
      - Huge error reduction when trained with lots of data on GPUs

Weight Initialization

» Don’t just initialize weights to 0!
- (like we did with linear models)
- Bad local optima

» Initial weights = 0
- => all hidden units = 0
- => Gradient on output layer will be 0
- => Gradients of weights on each hidden unit will be the same.
- => Values of hidden units always the same

Breadth vs. Depth

» 2 layer networks can represent any function

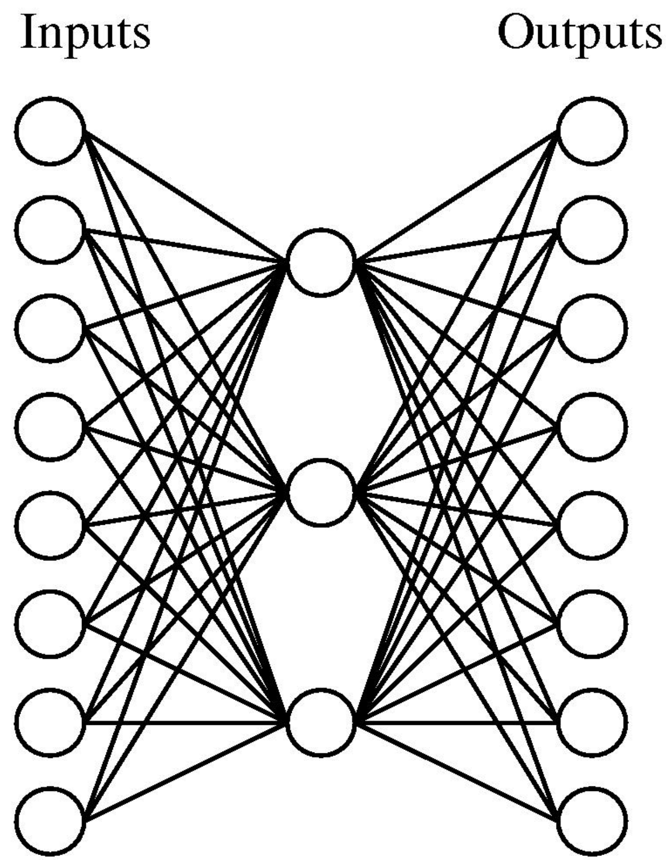
» But, can require exponentially many hidden units

» Analogy to circuit complexity

- Parity function
- Easy to define O(log(D)) depth circuit
- Exponential # of gates required for constant depth

Overfitting in NNs

- Are NNs likely to overfit?
  – Yes, they can represent arbitrary functions!!!
- Avoiding overfitting?
  – More training data
  – Fewer hidden nodes / better topology
  – Regularization
  – Early stopping



Disadvantages of Neural Networks

» Lots of hyperparameters!

- Learning rate
- Early stopping
- How many hidden units?
- How many layers?
- …

Disadvantages of Neural Networks

» Vanishing Gradients

- gradients shrink during backprop
- At beginning of deep network, changing one weight doesn’t have much effect on the output
- Derivatives are small

Computation Graphs

- » Arbitrary graphs of function composition
- » Choose whatever activation functions
- » Choose any loss function
- » Computing gradients is all very mechanical
- » Also automatically parallelize on GPUs

Computation Graphs

- Arbitrary graphs of function composition
- Choose whatever activation functions
- Choose any loss function
- Computing gradients is all very mechanical
- Also automatically parallelize on GPUs

theano

Computation Graphs

» Arbitrary graphs of function composition
» Choose whatever activation functions
» Choose any loss function
» Computing gradients is all very mechanical
» Also automatically parallelize on GPUs

theano

TensorFlow

Computation Graphs

» Arbitrary graphs of function composition
» Choose whatever activation functions
» Choose any loss function
» Computing gradients is all very mechanical
» Also automatically parallelize on GPUs

TensorFlow

theano

Keras == simple
10 lines of code

```
input_encoder = Sequential()
input_encoder.add(Embedding(input_dim=vocab_size, output_dim=64))
input_encoder.add(LSTM(64))

question_encoder = Sequential()
question_encoder.add(Embedding(input_dim=vocab_size, output_dim=64))
question_encoder.add(LSTM(64))

model = Sequential()
model.add(Merge([input_encoder, question_encoder], mode='concat'))
model.add(Dense(128))
model.add(Activation('softmax'))
```

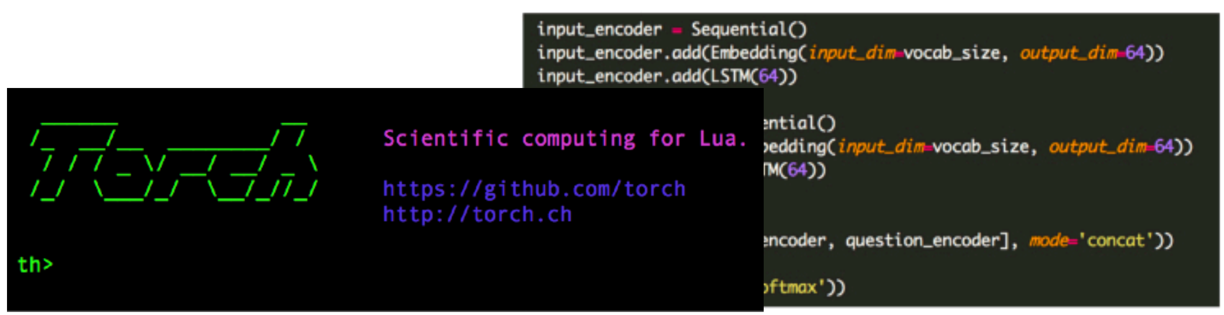
Computation Graphs

» Arbitrary graphs of function composition
» Choose whatever activation functions
» Choose any loss function
» Computing gradients is all very mechanical
» Also automatically parallelize on GPUs

TensorFlow

theano

Keras == simple
10 lines of code



Computation Graphs

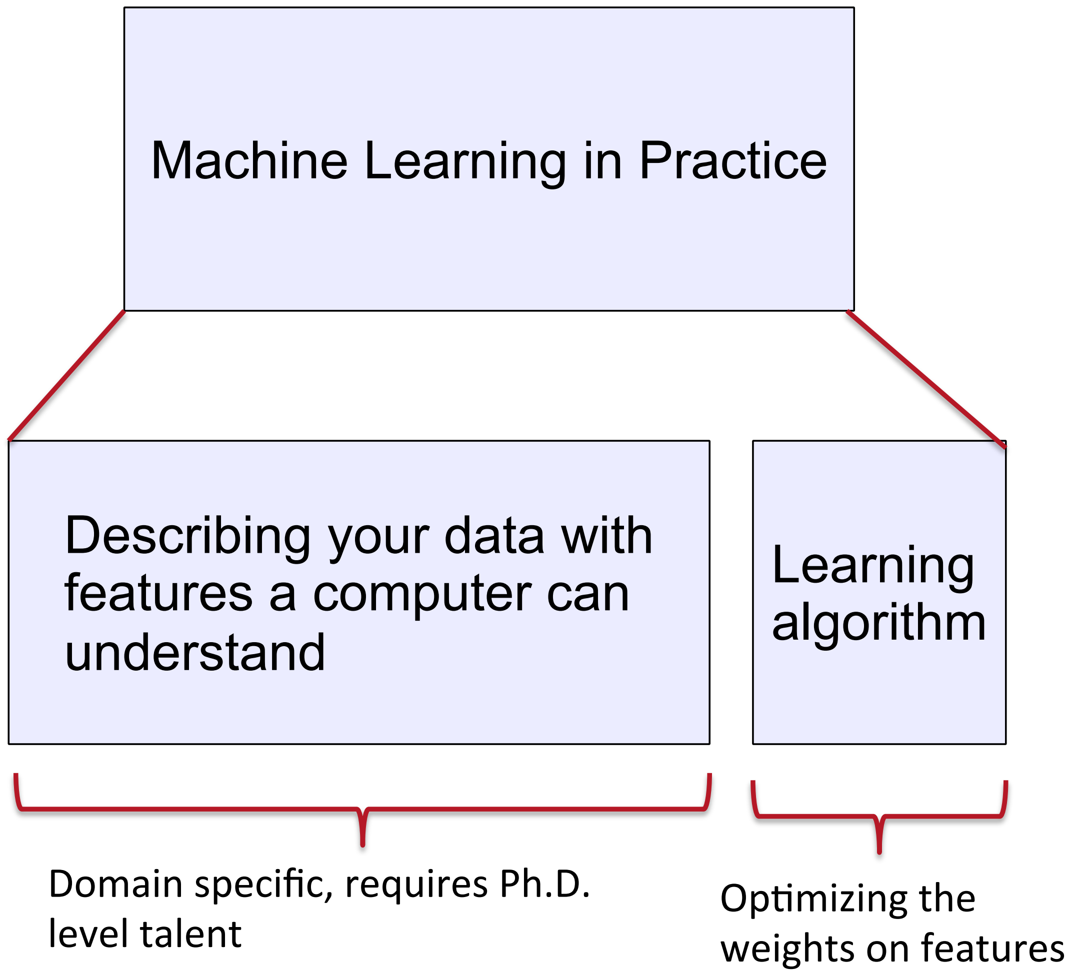
» Arbitrary graphs of function composition
» Choose whatever activation functions
» Choose any loss function
» Computing gradients is all very mechanical
» Also automatically parallelize on GPUs

TensorFlow

theano

Keras == simple
10 lines of code

Machine Learning vs Deep Learning



Reasons for Exploring Deep Learning

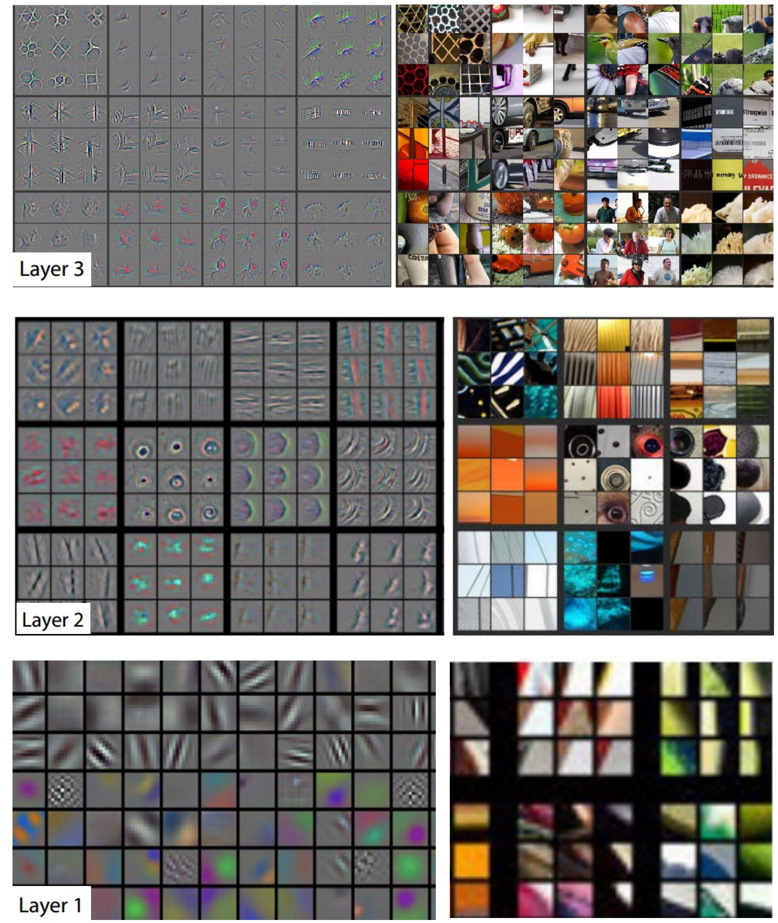
- In 2006 **deep** learning techniques started outperforming other machine learning techniques. Why now?

- DL techniques benefit more from a lot of data
- Faster machines and multicore CPU/GPU help DL
- New models, algorithms, ideas

→ **Improved performance** (first in speech and vision, then NLP)

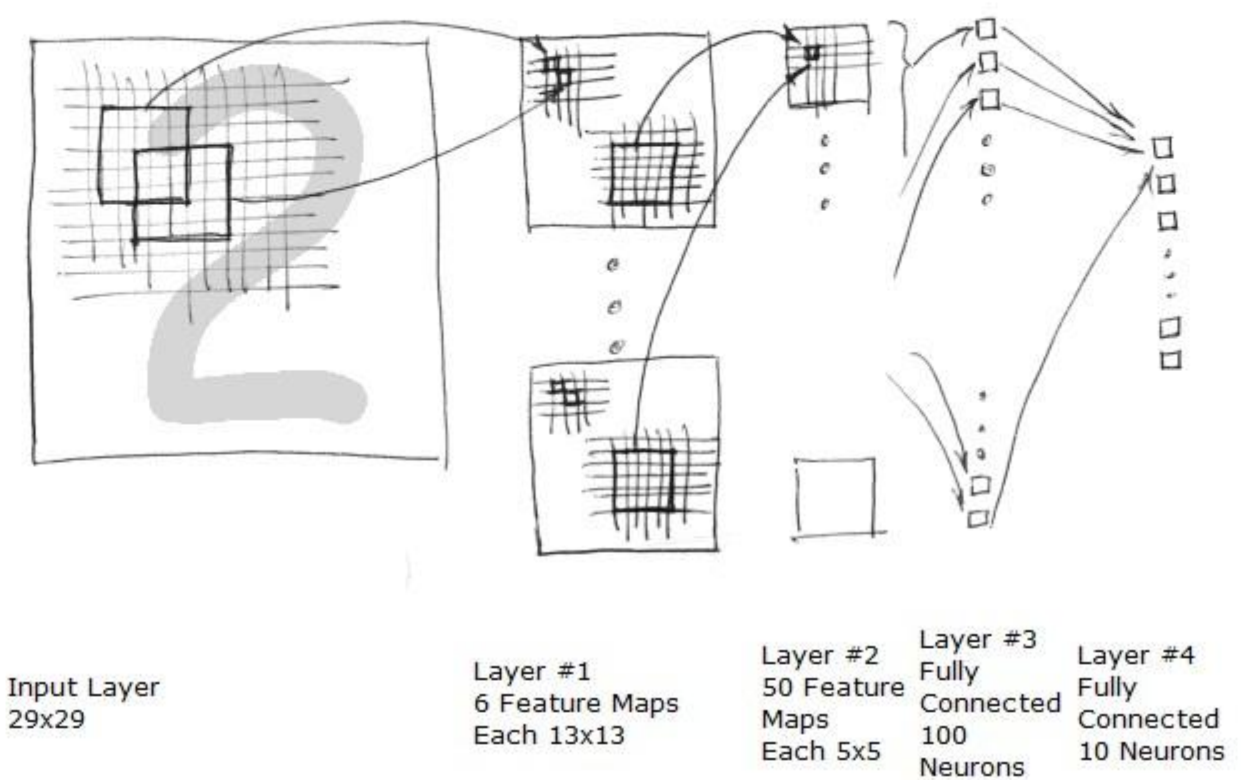
Deep Learning for Computer Vision

- Most deep learning groups have (until recently) largely focused on computer vision
- Break through paper: ImageNet Classification with Deep Convolutional Neural Networks by Krizhevsky et al. 2012



Zeiler and Fergus (2013)

Convolutional Neural Networks



AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

AlexNet

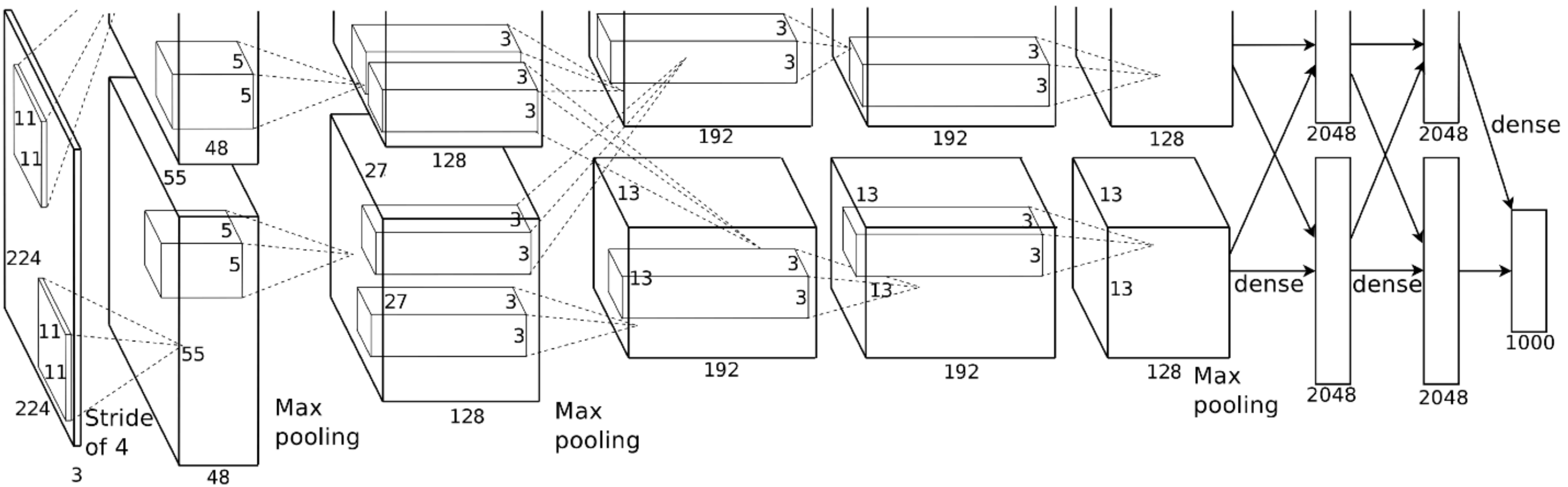
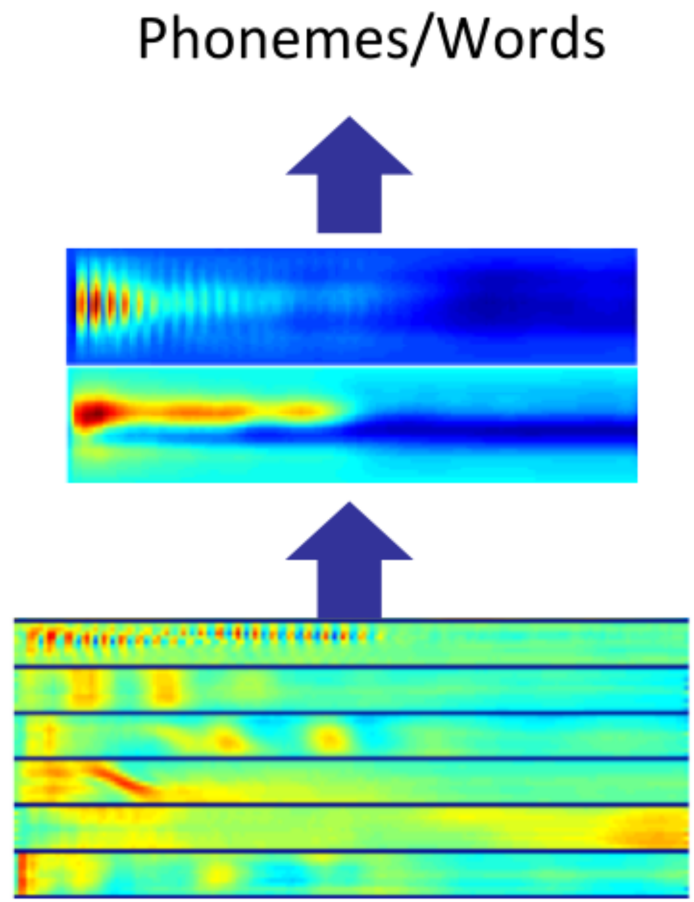


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Deep Learning for Speech

- The first breakthrough results of "deep learning" on large datasets happened in speech recognition
- Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition Dahl et al. (2010)



Acoustic model	Recog \ WER	RT03S FSH	Hub5 SWB
Traditional features	1-pass −adapt	**27.4**	**23.6**
Deep Learning	1-pass −adapt	**18.5** (−33%)	**16.1** (−32%)

Machine Translation

- Many levels of translation have been tried in the past:

- Traditional MT systems are very large complex systems

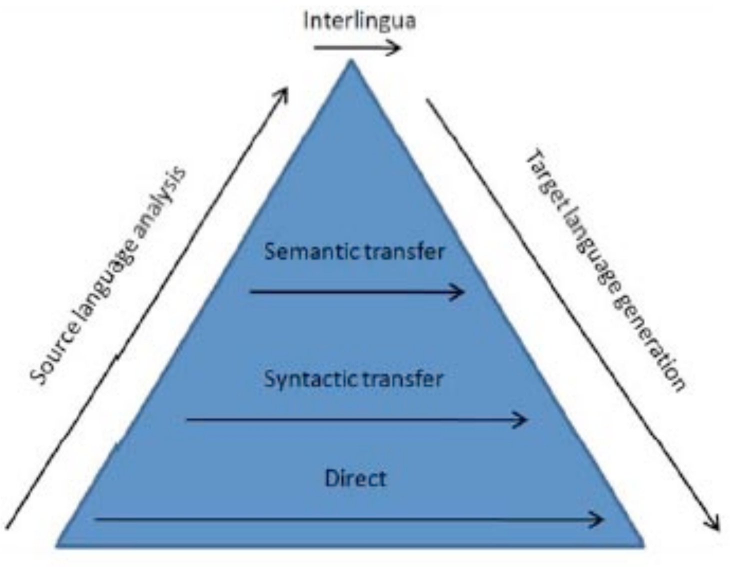
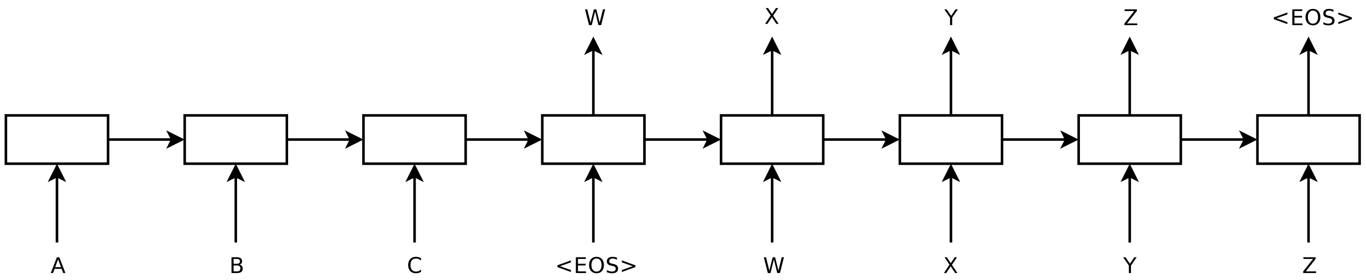


Figure 1: The Vauquois triangle

- What do you think is the interlingua for the DL approach to translation?

Machine Translation

- Source sentence mapped to vector, then output sentence generated.



- Sequence to Sequence Learning with Neural Networks by Sutskever et al. 2014

What you need to know about neural networks

- Perceptron:
  - Relationship to general neurons
- Multilayer neural nets
  - Representation
  - Derivation of backprop
  - Learning rule
- Overfitting