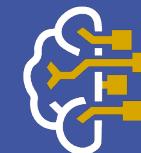


Deep Learning Overview

©Dr. Min Chi

mchi@ncsu.edu

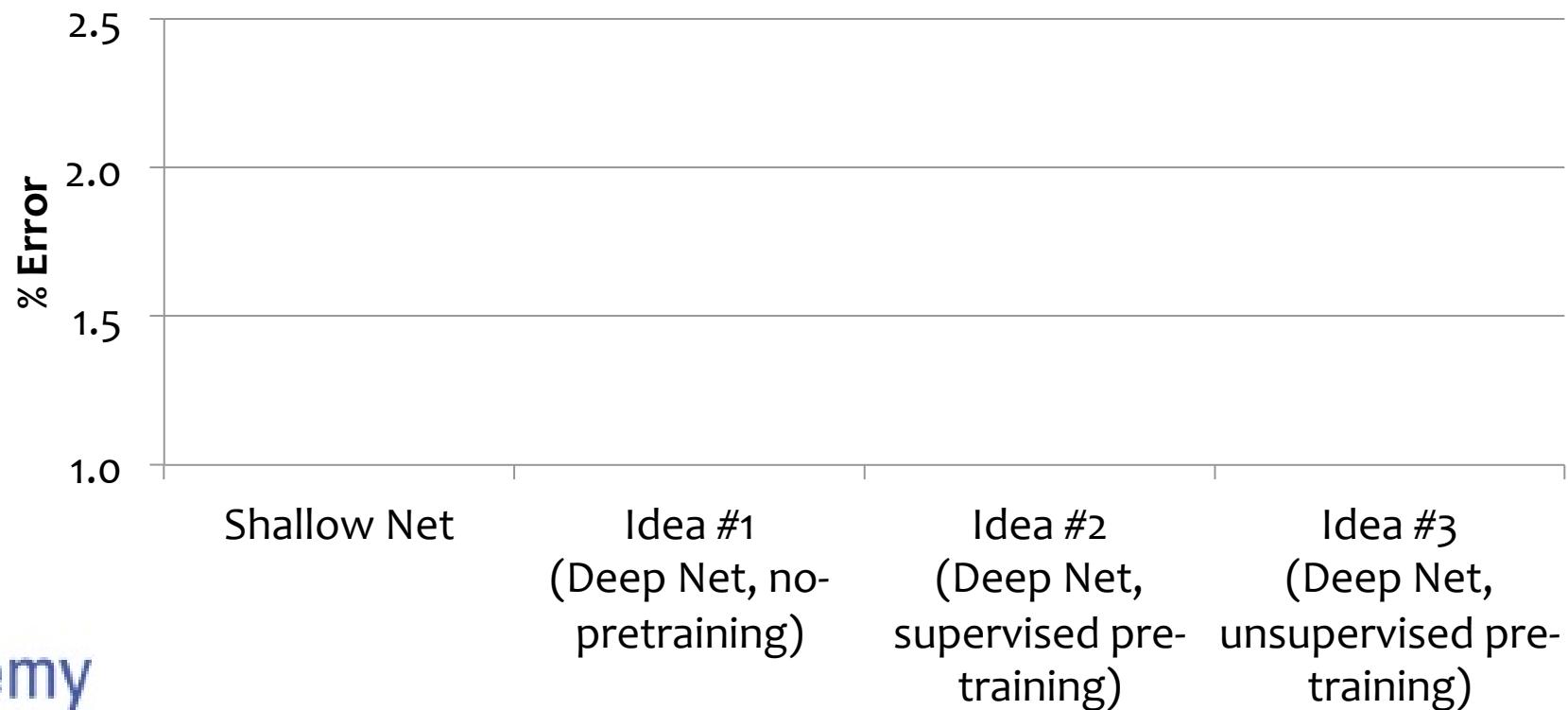


AI Academy

The materials on this course website
are only for use of students enrolled
AIA and must not be retained or
disseminated to others or Internet.

Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)

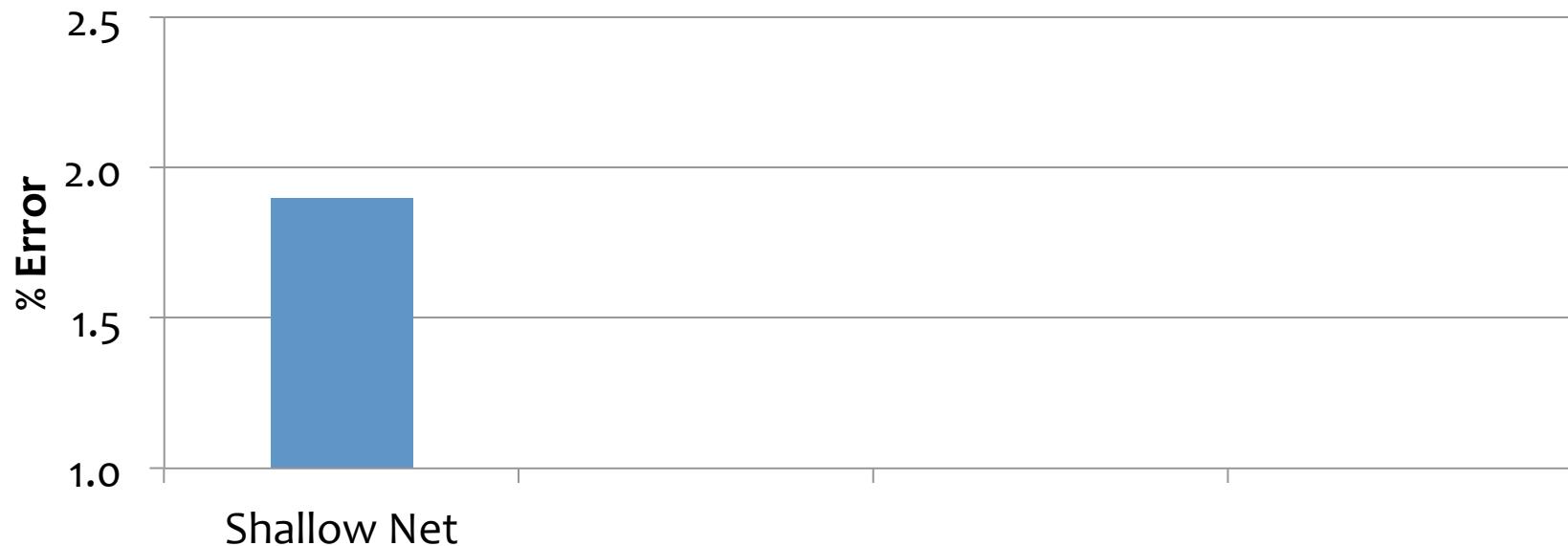


9	3	1	9	7	2	4	5	1	0	3	2	4	3	7	5	9	0	3	4	9
3	0	2	4	2	9	4	8	3	2	0	1	3	5	3	5	7	4	6	8	5
2	8	2	3	2	3	8	2	4	9	8	2	9	1	3	9	1	1	1	9	9
6	3	6	9	0	3	6	0	3	0	1	1	3	9	3	1	5	0	4	9	6
3	3	8	0	7	0	5	6	9	8	8	4	1	4	4	4	6	4	5	3	3
1	7	9	5	8	0	4	3	7	7	5	0	5	4	2	0	9	8	1	2	4
9	5	0	0	5	1	1	1	7	4	7	7	2	6	5	1	8	2	4	1	1
0	2	1	6	1	7	0	9	5	6	3	2	6	6	4	7	1	5	2	3	2
9	4	3	2	1	0	0	2	0	8	7	4	0	9	7	9	3	6	9	3	4
5	5	1	6	6	2	7	6	7	5	6	6	5	8	1	6	8	7	1	0	5
7	1	7	5	9	2	3	9	4	3	0	4	5	8	0	0	4	0	4	6	6
1	6	7	9	6	4	1	1	4	1	3	1	2	3	4	8	1	5	5	0	7
0	1	6	1	6	7	5	5	5	6	6	8	8	7	7	2	8	3	7	6	5
6	4	6	8	7	7	1	3	0	7	3	8	6	9	1	6	7	3	6	4	8
7	9	7	3	1	3	2	7	9	3	6	2	4	9	2	1	4	5	0	3	8

MNIST: handwritten digits

Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Training

Idea #1

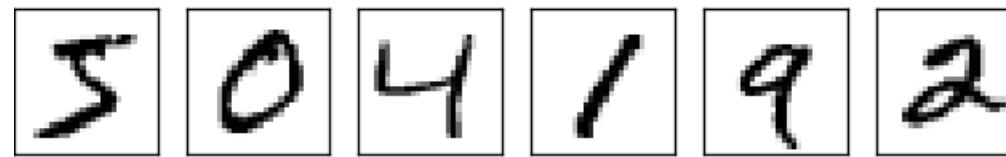
Idea #1: (Just like a shallow network)

Compute the supervised gradient by backpropagation.

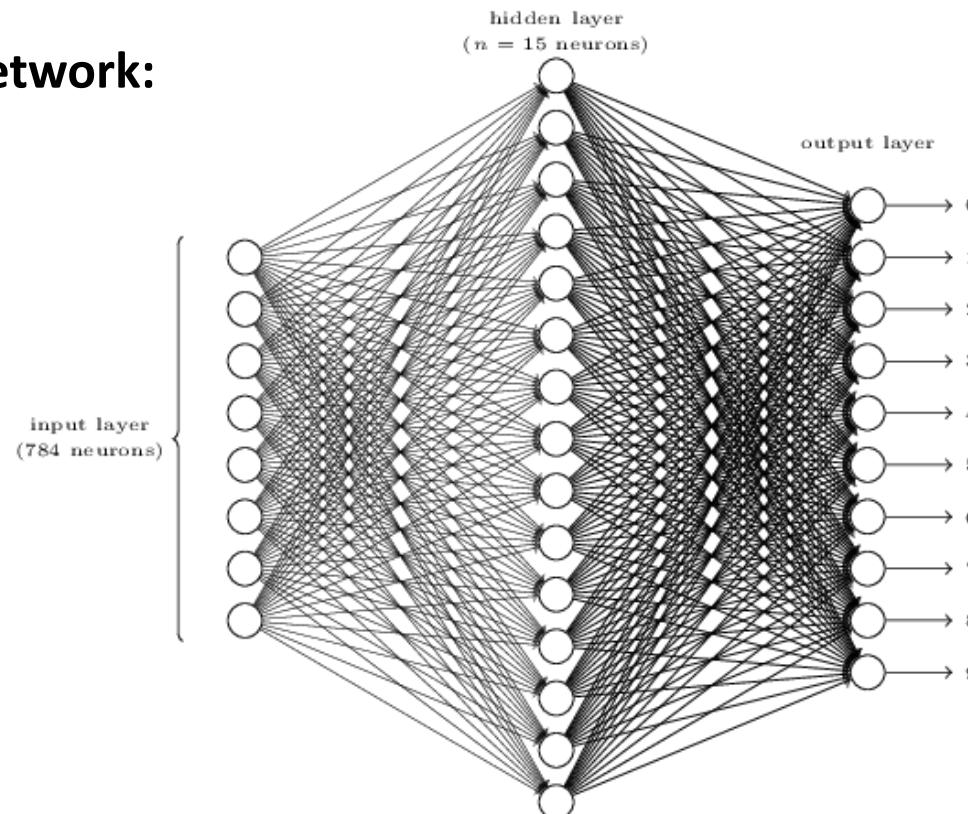
Take small steps in the direction of the gradient

Task: Classify and Image of a Number

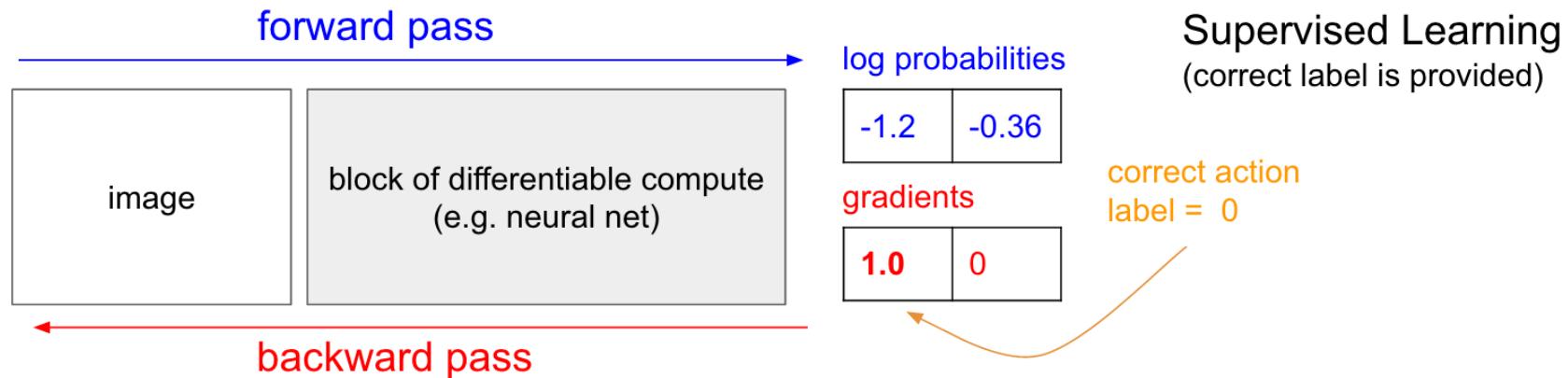
Input:
(28x28)



Network:



Task: Classify and Image of a Number

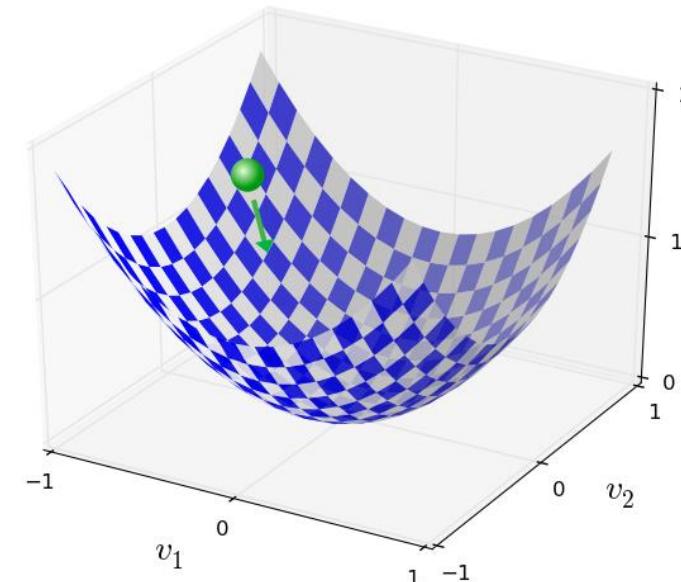


Ground truth for “6”:

$$y(x) = (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$$

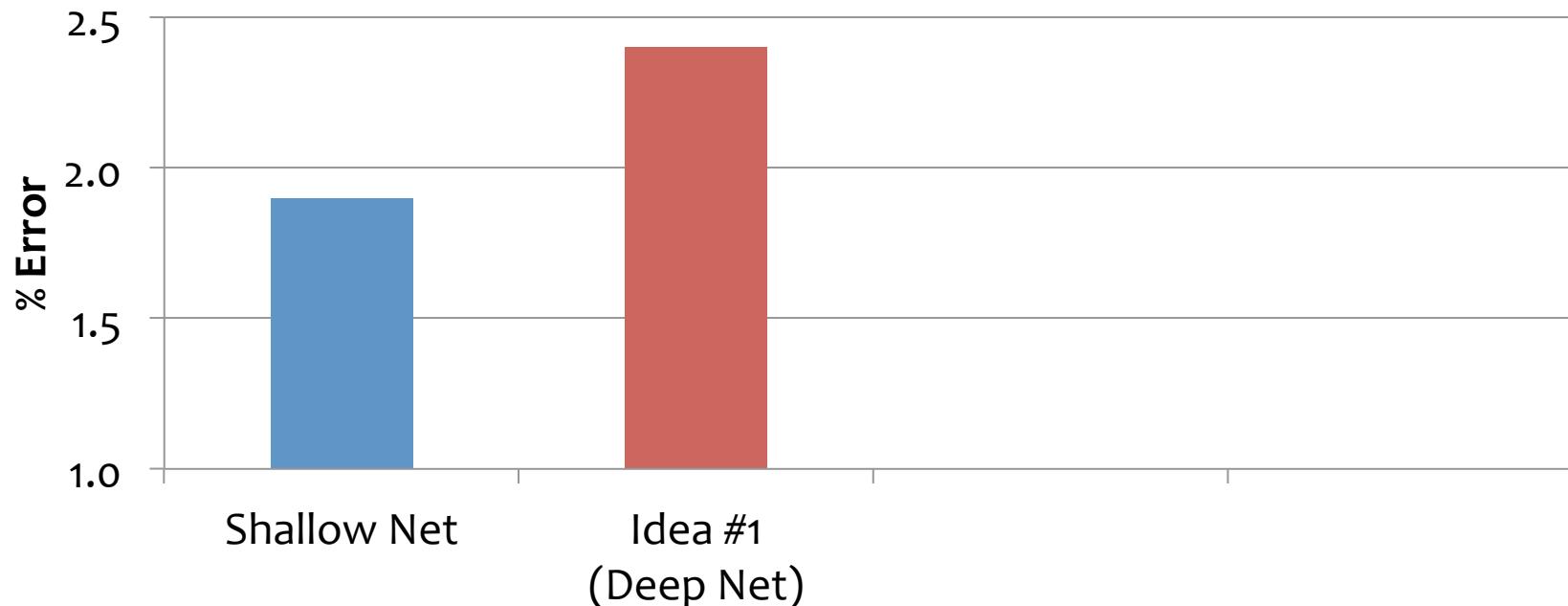
“Loss” function:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$



Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



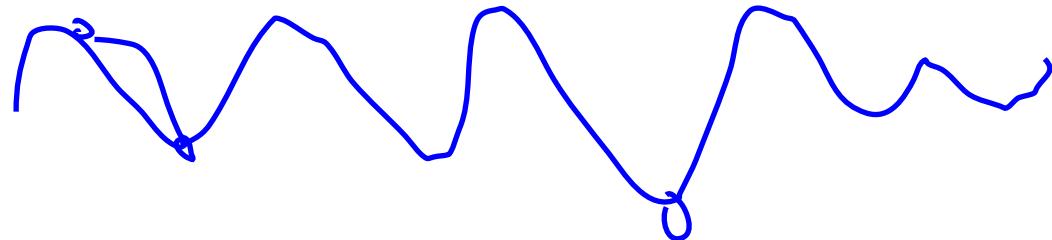
Idea #1: Deep Net

Idea #1: (Just like a shallow network)

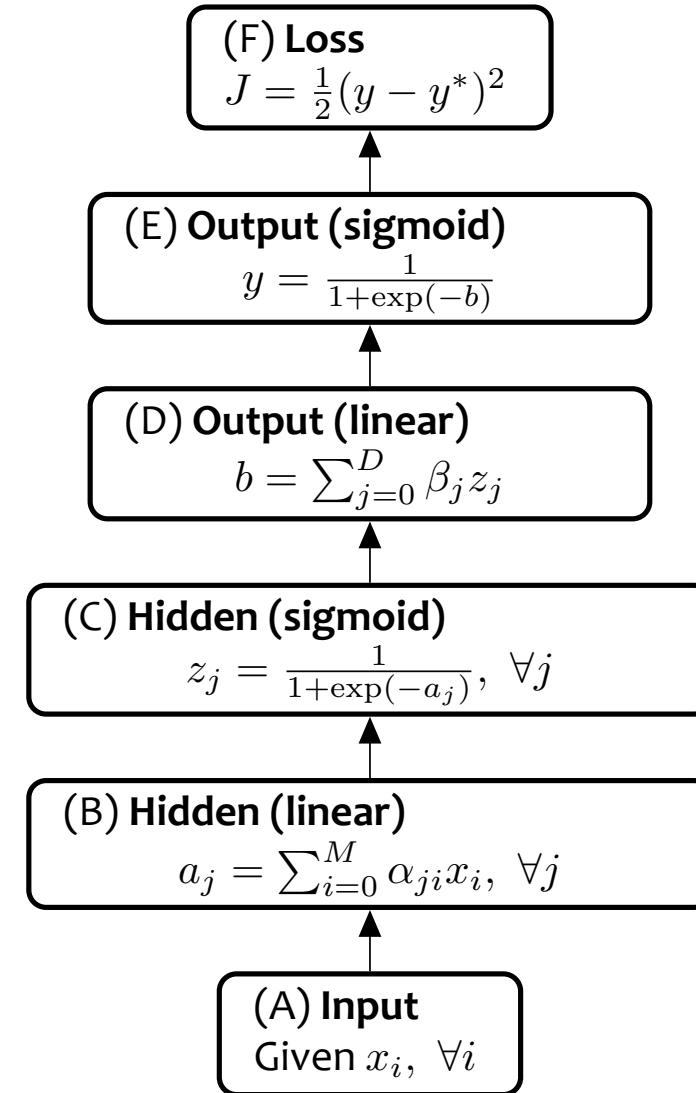
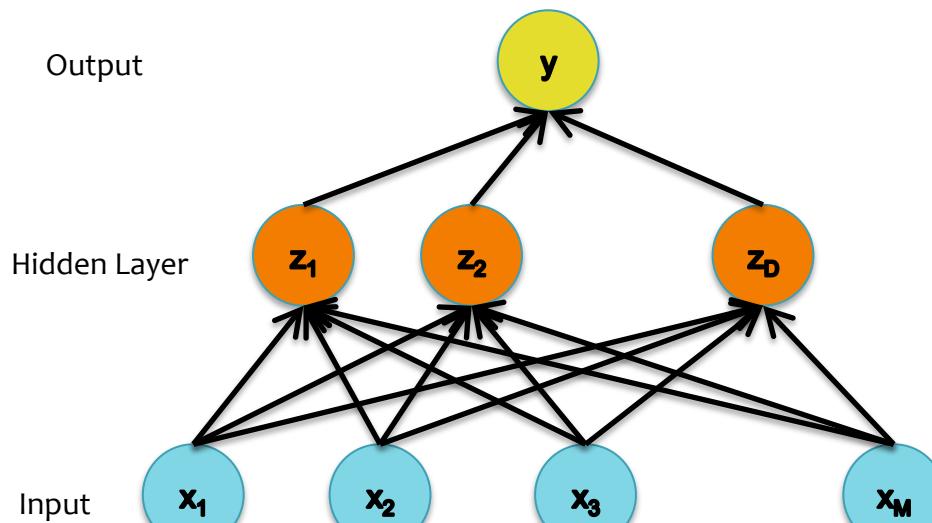
Compute the supervised gradient by backpropagation.

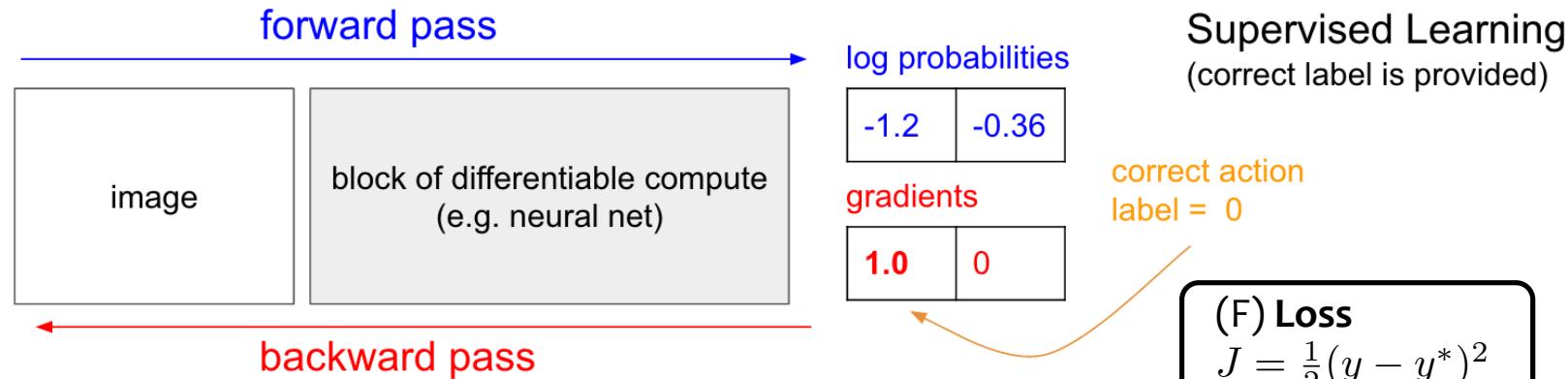
Take small steps in the direction of the gradient (SGD)

- What goes wrong?
 - A. Gets stuck in local optima
 - Nonconvex objective
 - Usually start at a random (bad) point in parameter space
 - B. Gradient is progressively getting more dilute
 - “Vanishing gradients”



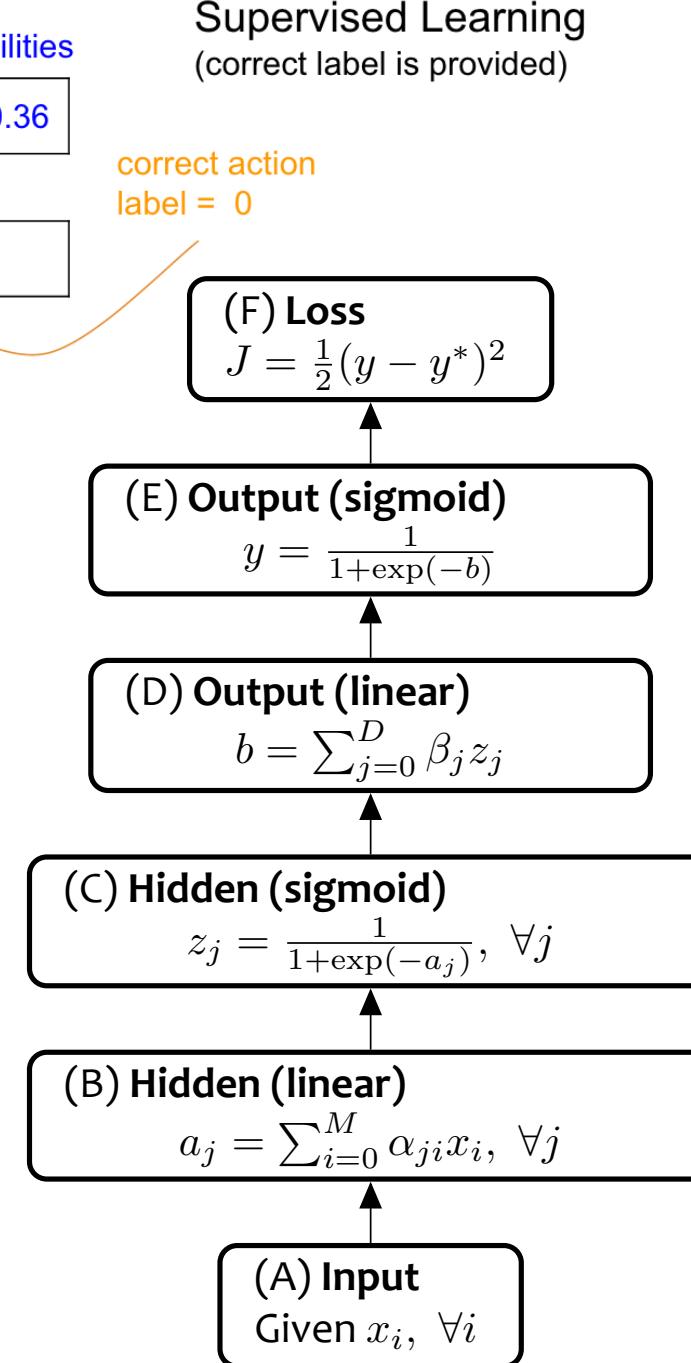
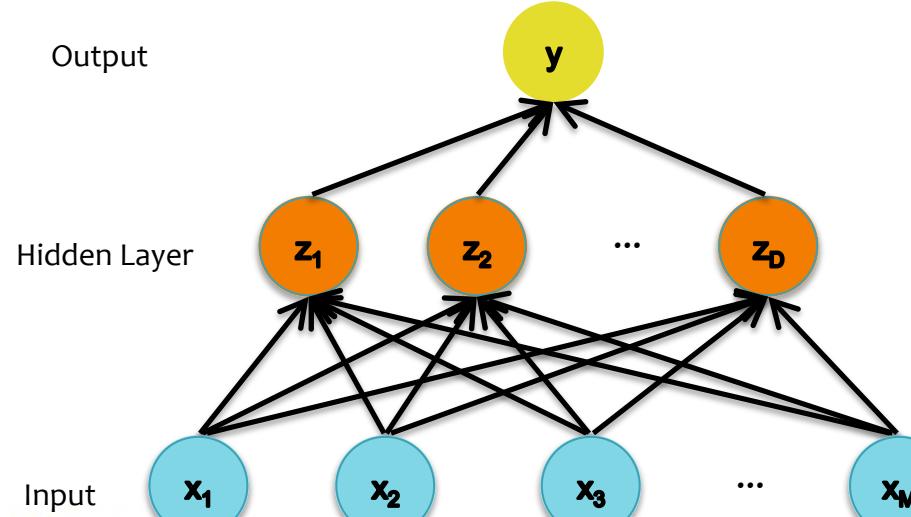
Neural Network with sigmoid activation functions





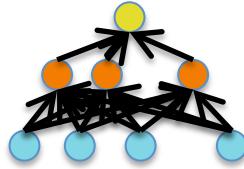
Back to Basics: Backpropagation

Adjust the weights to reduce the error:



Backpropagation

Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

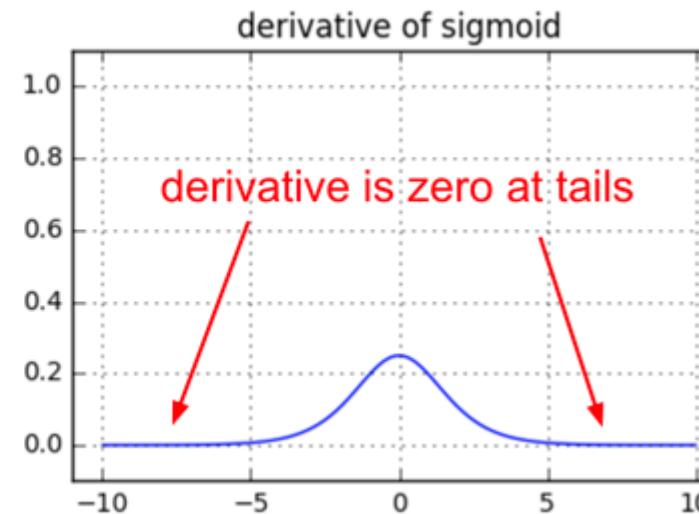
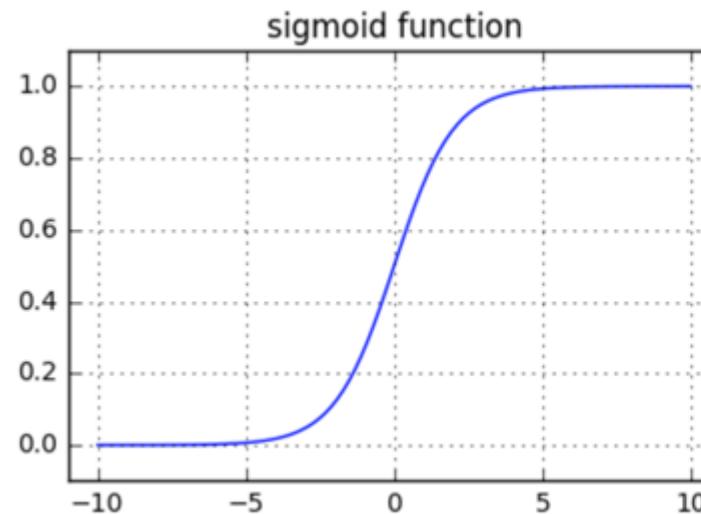
$$\frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$$

Backpropagation

	Forward	Backward
Module 5	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{1 - y}$
Module 4	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Module 3	$b = \sum_{j=0}^D \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$
Module 2	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Module 1	$a_j = \sum_{i=0}^M \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \alpha_{ji}$

Optimization is Hard: Vanishing Gradients

14



$$\frac{d\sigma(x)}{dx} = (1 - \sigma(x)) \sigma(x)$$

Partial derivatives are small = Learning is slow

Saturating Neurons with Vanishing Gradients:
Zero-ish gradients drives gradients in earlier layers to zero.

Training

Idea #2: Supervised Pre-training

Idea #2: (Two Steps)

Train each level of the model in a **greedy** way

Then use our **original idea**

1. Supervised Pre-- training
 - Use **labeled** data
 - Work bottom-- up
 - Train hidden layer 1. Then fix its parameters.
 - Train hidden layer 2. Then fix its parameters.
 - ...
 - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-- tuning
 - Use **labeled** data to train following “Idea #1”
 - Refine the features by backpropagation so that they become tuned to the end-- task

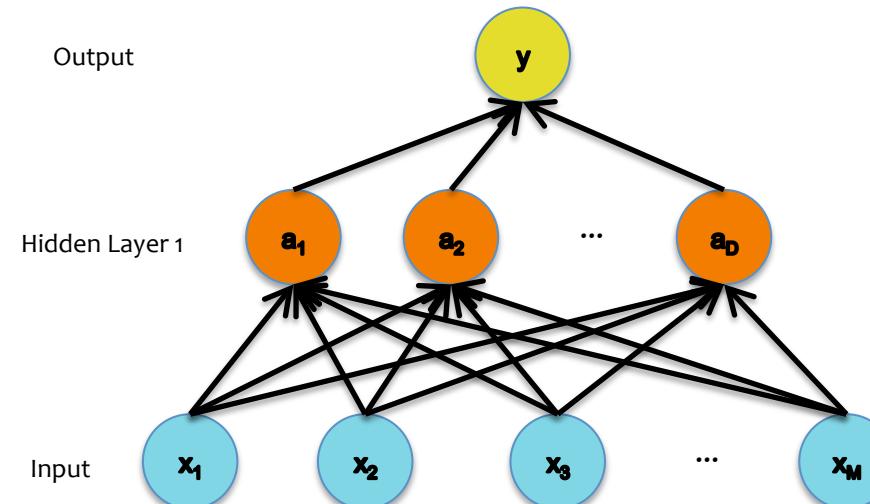
Training

Idea #2: Supervised Pre-training

Idea #2: (Two Steps)

Train each level of the model in a greedy way

Then use our original idea



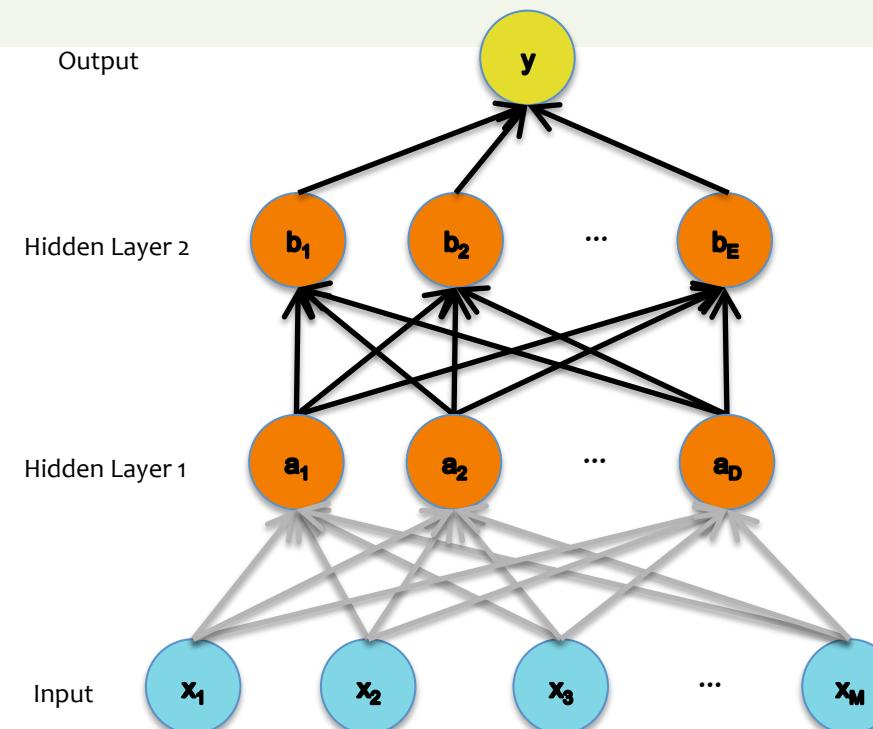
Training

Idea #2: Supervised Pre-training

Idea #2: (Two Steps)

Train each level of the model in a greedy way

Then use our original idea

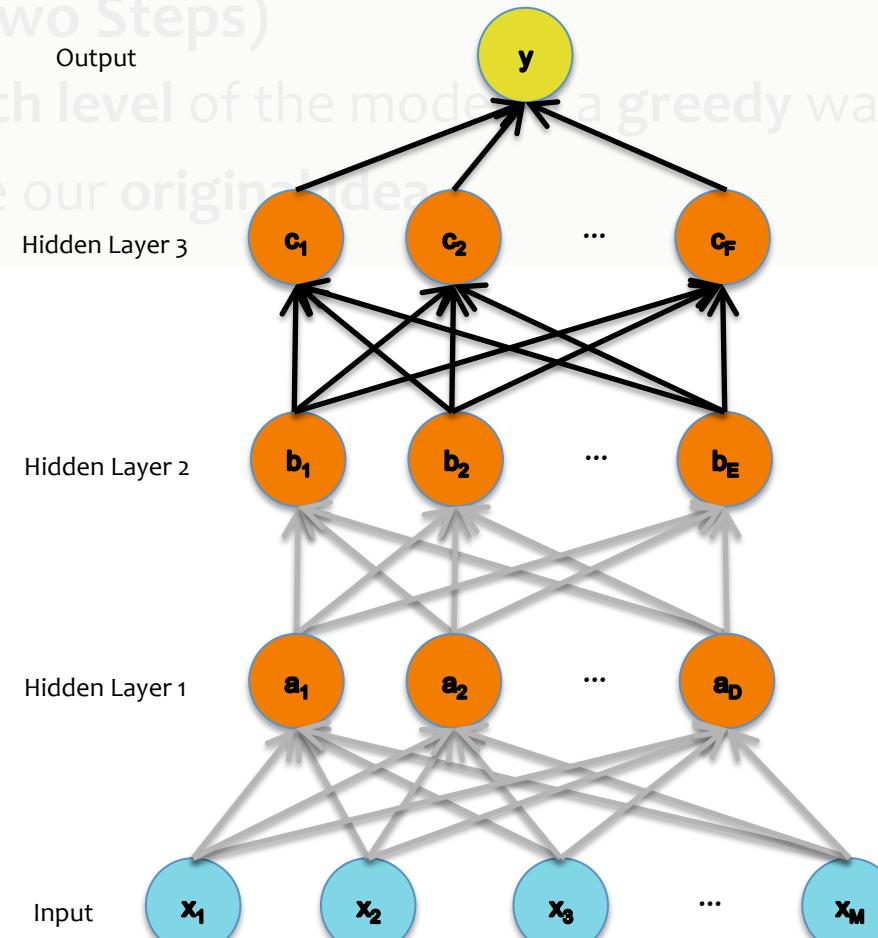


Training

Idea #2: Supervised Pre-training

- Idea #2: (Two Steps)

- Train each level of the model in a greedy way
- Then use our original idea

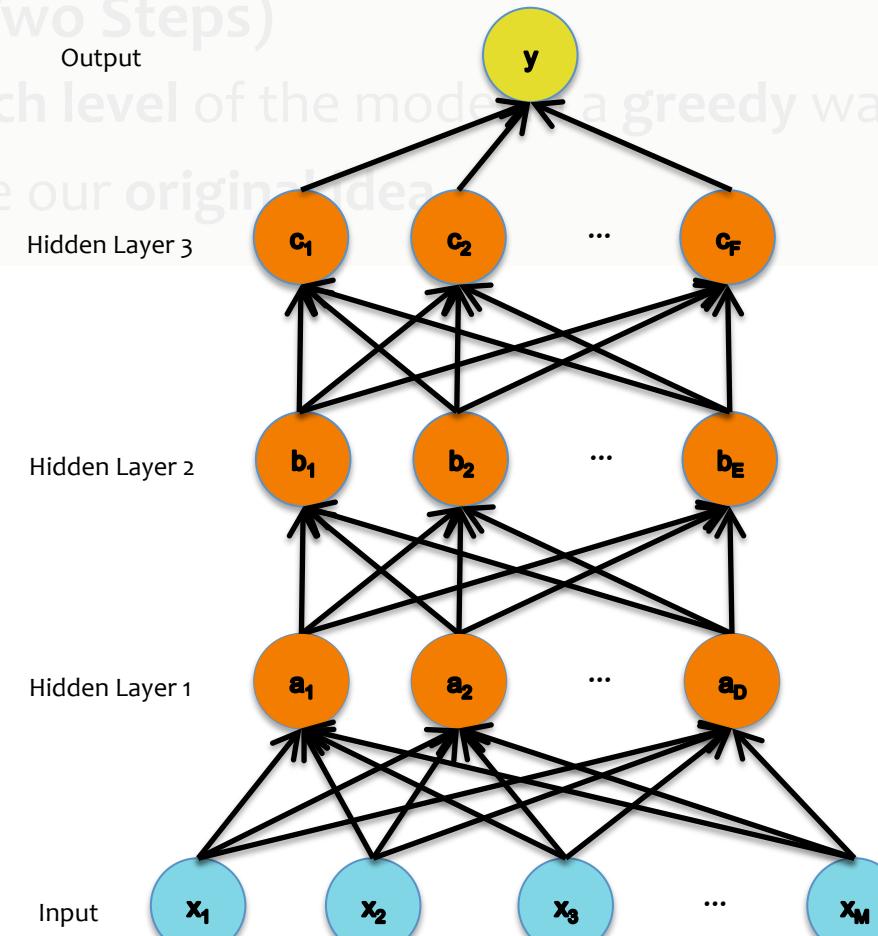


Training

Idea #2: Supervised Pre-training

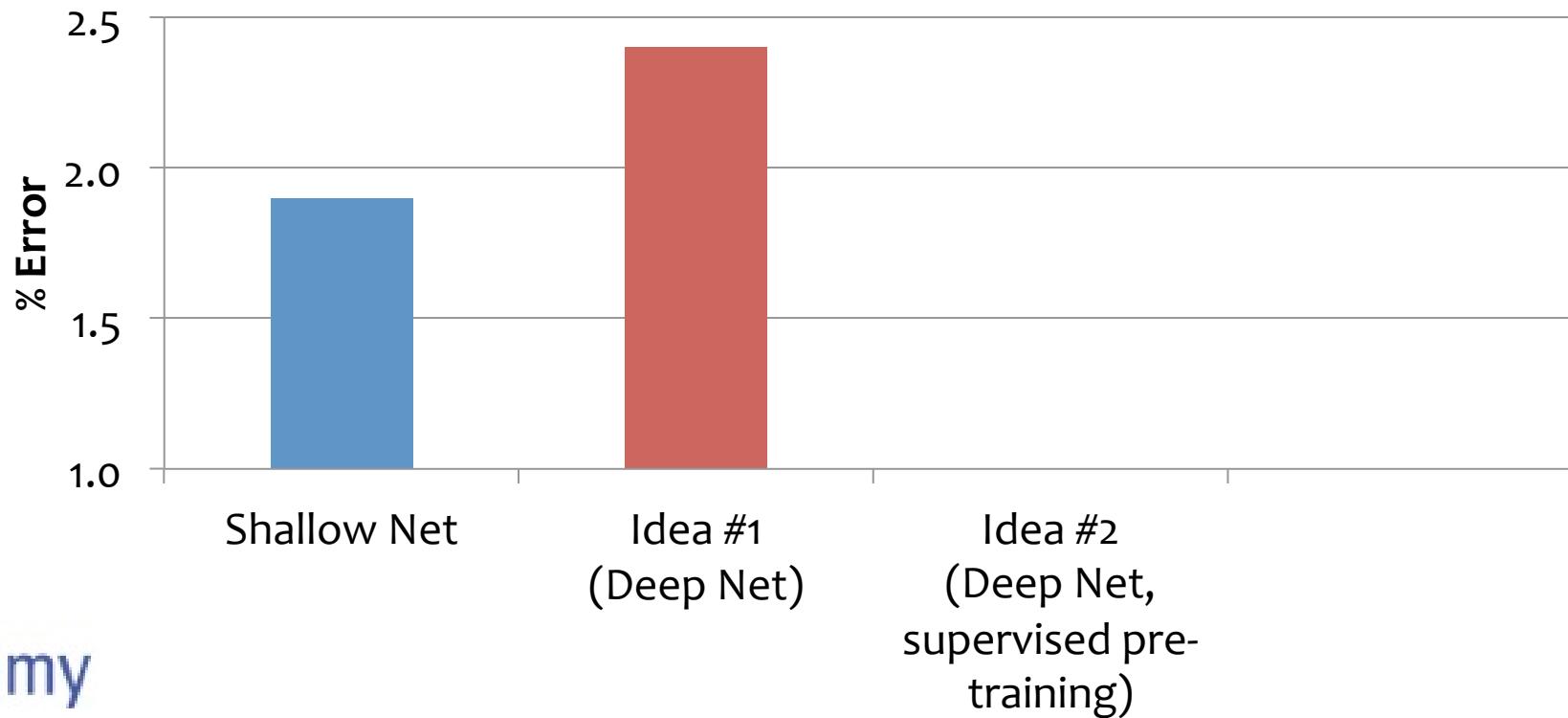
- Idea #2: (Two Steps)

- Train each level of the model in a greedy way
- Then use our original idea



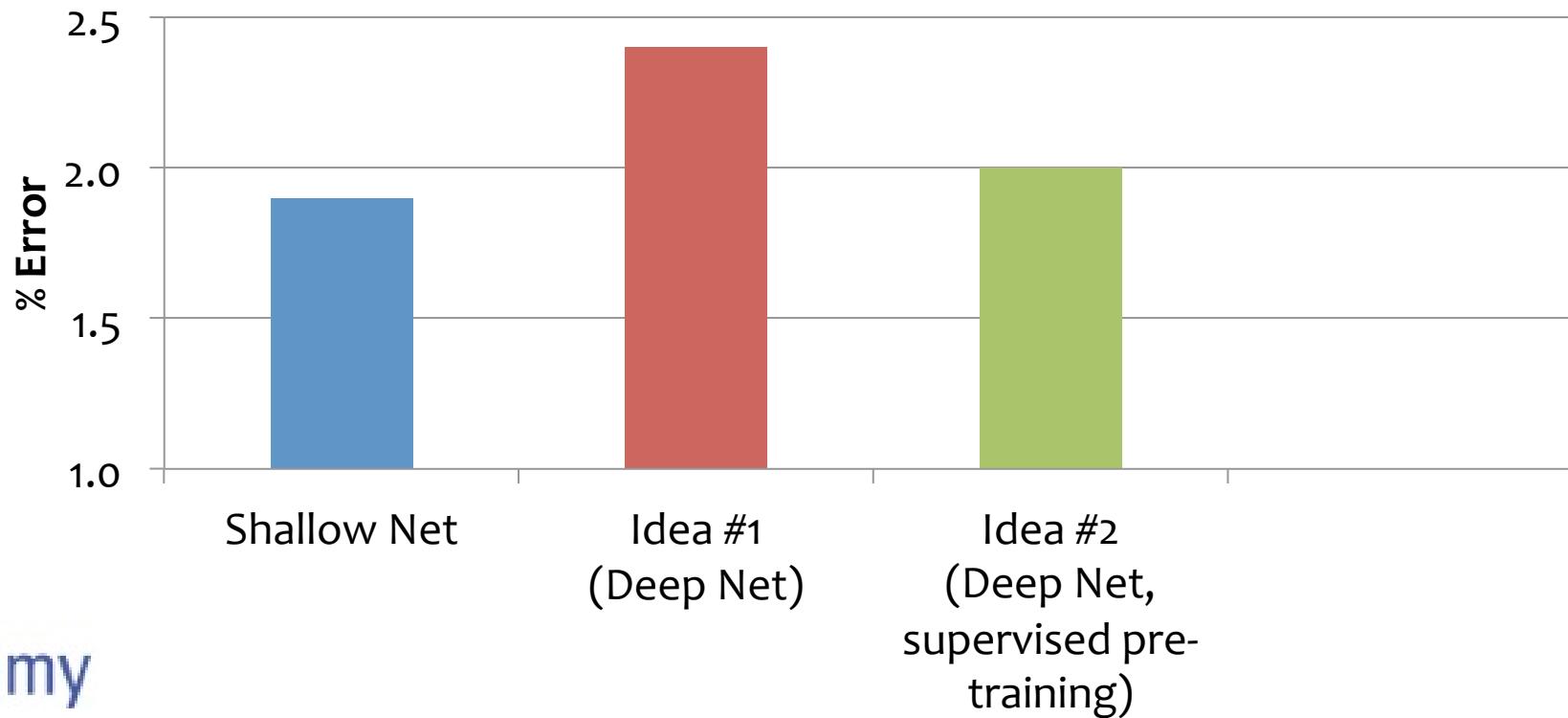
Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Training

Idea #3: Unsupervised Pre-training

Idea #3: (Two Steps)

Use our original idea, but **pick a better starting point**

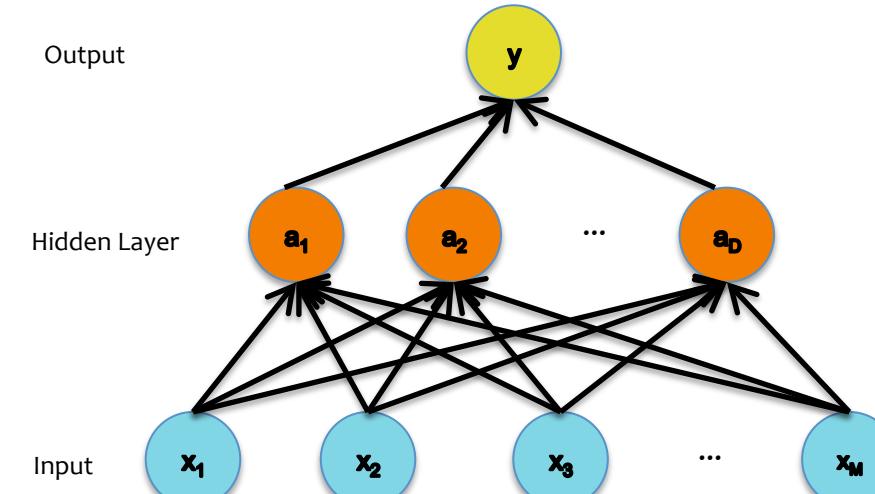
Train each level of the model in a **greedy way**

1. Unsupervised Pre-- training
 - Use unlabeled data
 - Work bottom-- up
 - Train hidden layer 1. Then fix its parameters.
 - Train hidden layer 2. Then fix its parameters.
 - ...
 - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-- tuning
 - Use **labeled** data to train following “Idea #1”
 - Refine the features by backpropagation so that they become tuned to the end-- task

The solution: *Unsupervised pre-training*

Unsupervised pre-- training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

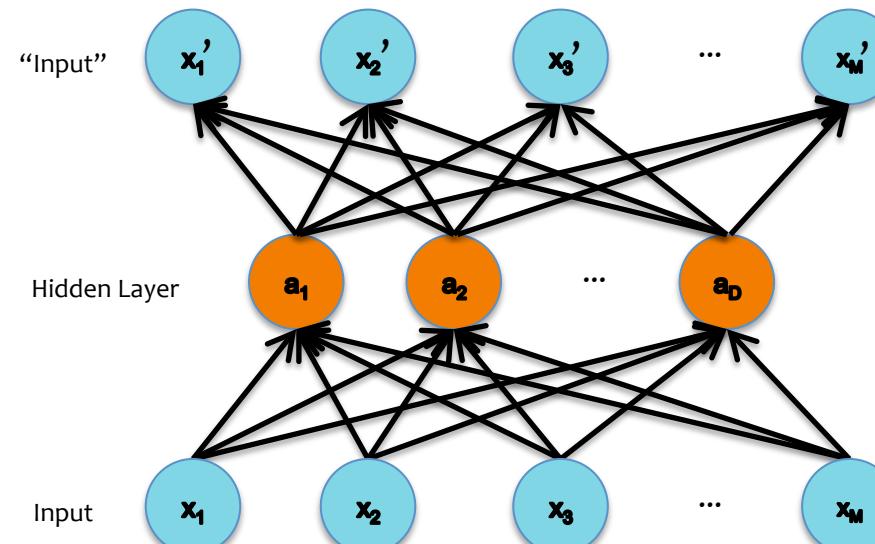


The solution: *Unsupervised pre-training*

Unsupervised pre-- training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

This topology defines an Auto-- encoder.



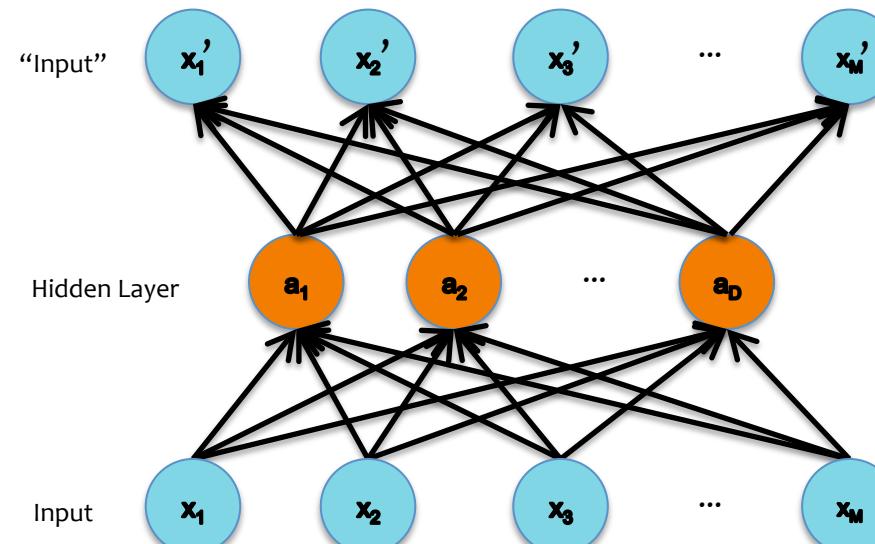
Auto-Encoders

Key idea: Encourage z to give small reconstruction error:

- x' is the reconstruction of x
- Loss = $\| x - \text{DECODER}(\text{ENCODER}(x)) \|_2^2$
- Train with the same backpropagation algorithm for 2-- layer Neural Networks with x_m as both input and output.

$$\text{DECODER: } x' = h(W'z)$$

$$\text{ENCODER: } z = h(Wx)$$

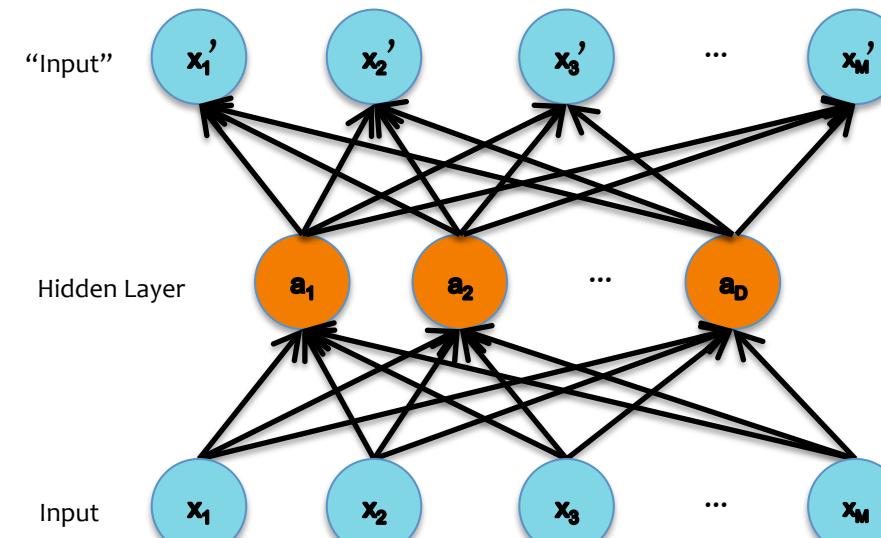


The solution:

Unsupervised pre- training

Unsupervised pre-- training

- Work bottom-- up
 - Train hidden layer 1.
Then fix its parameters.
 - Train hidden layer 2.
Then fix its parameters.
 - ...
 - Train hidden layer n.
Then fix its parameters.

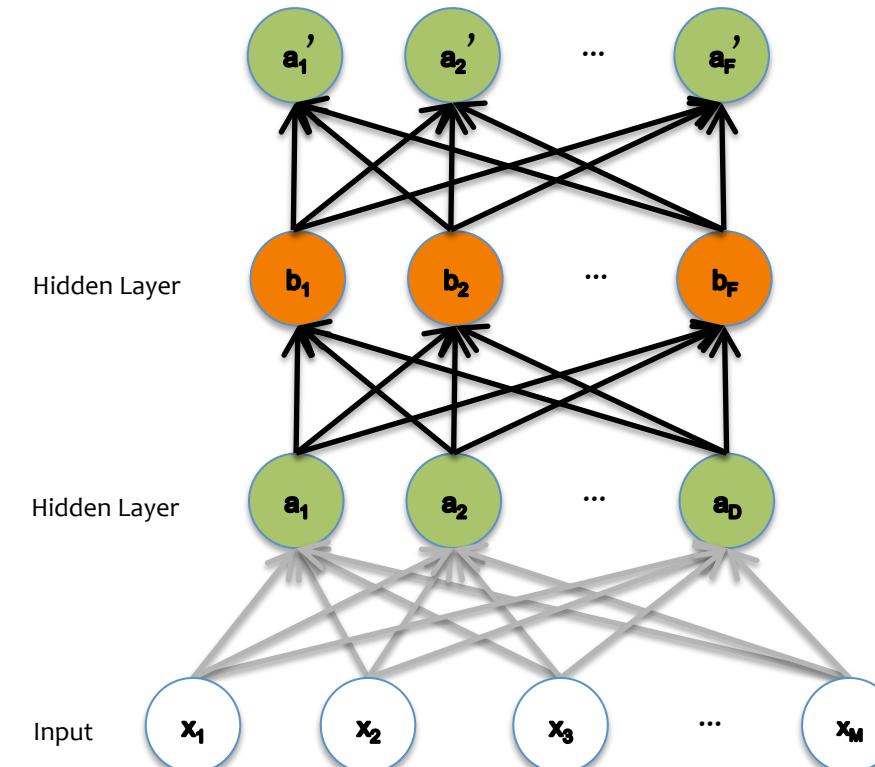


The solution:

Unsupervised pre- training

Unsupervised pre-- training

- Work bottom-- up
 - Train hidden layer 1.
Then fix its parameters.
 - Train hidden layer 2.
Then fix its parameters.
 - ...
 - Train hidden layer n.
Then fix its parameters.

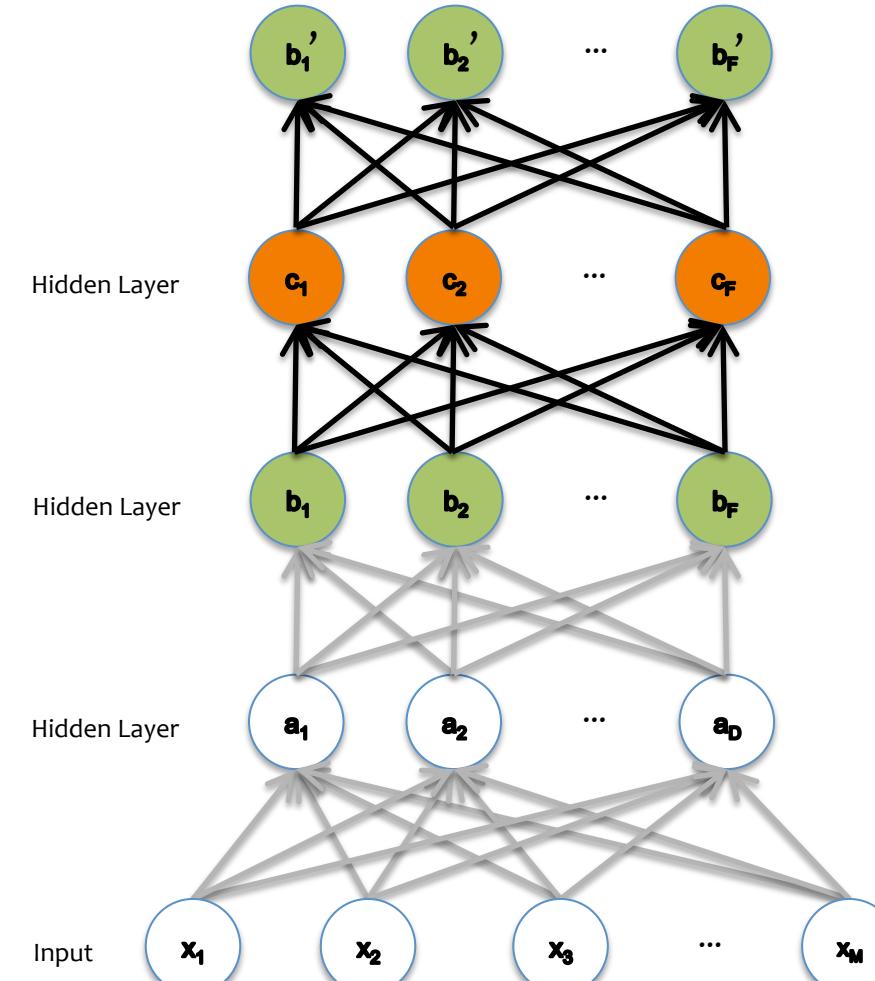


The solution:

Unsupervised pre-training

Unsupervised pre- training

- Work bottom-- up
 - Train hidden layer 1.
Then fix its parameters.
 - Train hidden layer 2.
Then fix its parameters.
 - ...
 - Train hidden layer n.
Then fix its parameters.



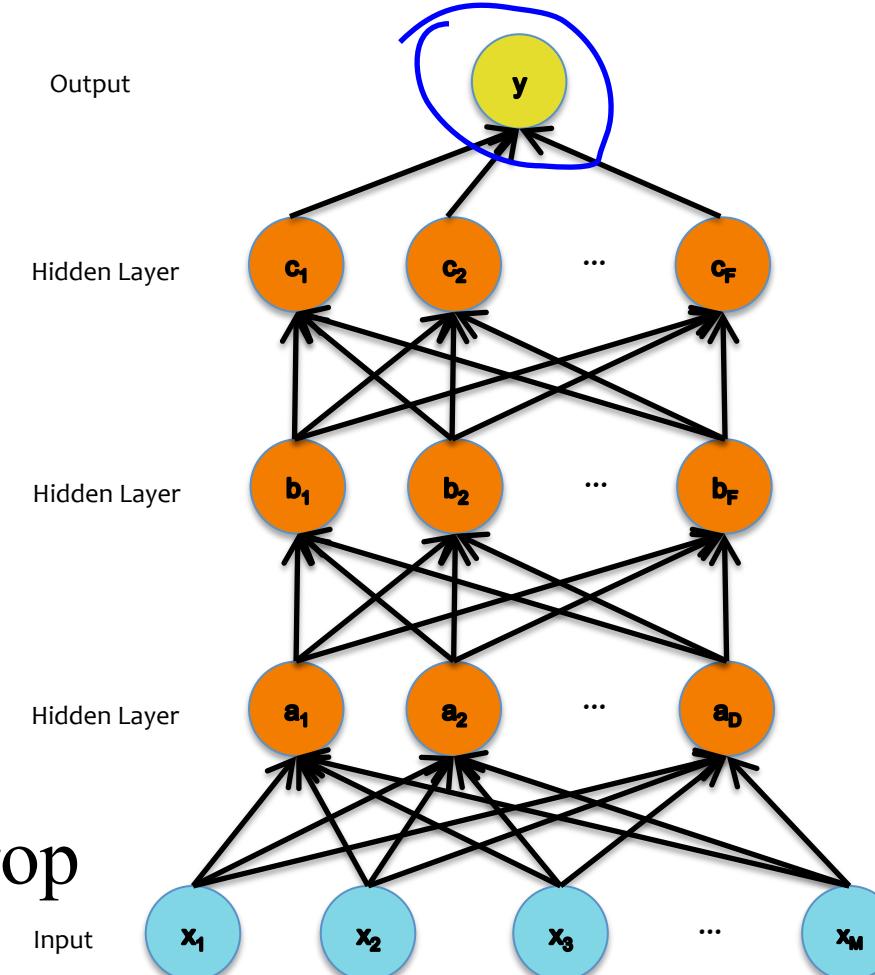
The solution:

Unsupervised pre-training

Unsupervised pre-training

- Work bottom-- up
 - Train hidden layer 1.
Then fix its parameters.
 - Train hidden layer 2.
Then fix its parameters.
 - ...
 - Train hidden layer n.
Then fix its parameters.

Supervised fine-tuning Backprop
and update all parameters



Deep Network Training

Idea #1:

1. Supervised fine-- tuning only

Idea #2:

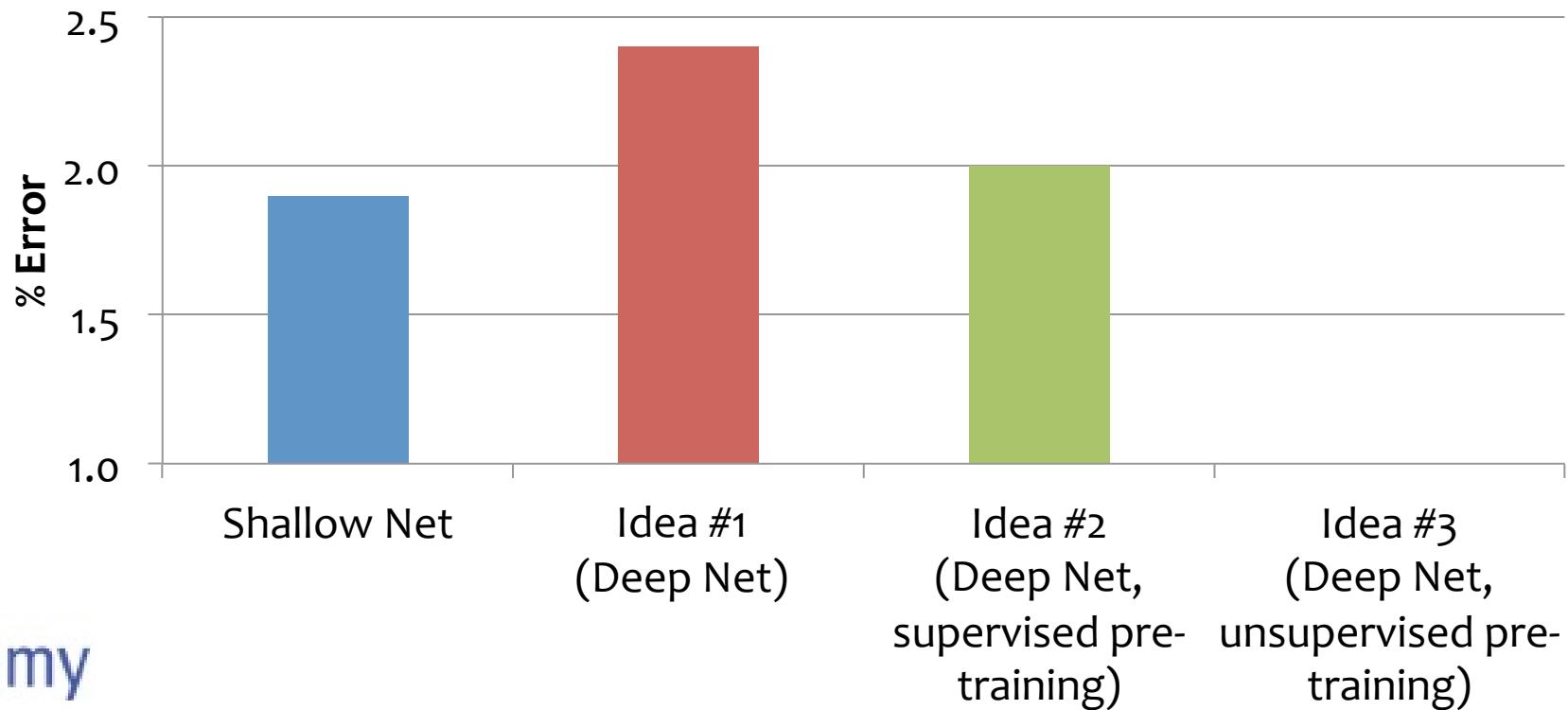
1. Supervised layer-- wise pre-- training
2. Supervised fine-- tuning

Idea #3:

1. Unsupervised layer-- wise pre-- training
2. Supervised fine-- tuning

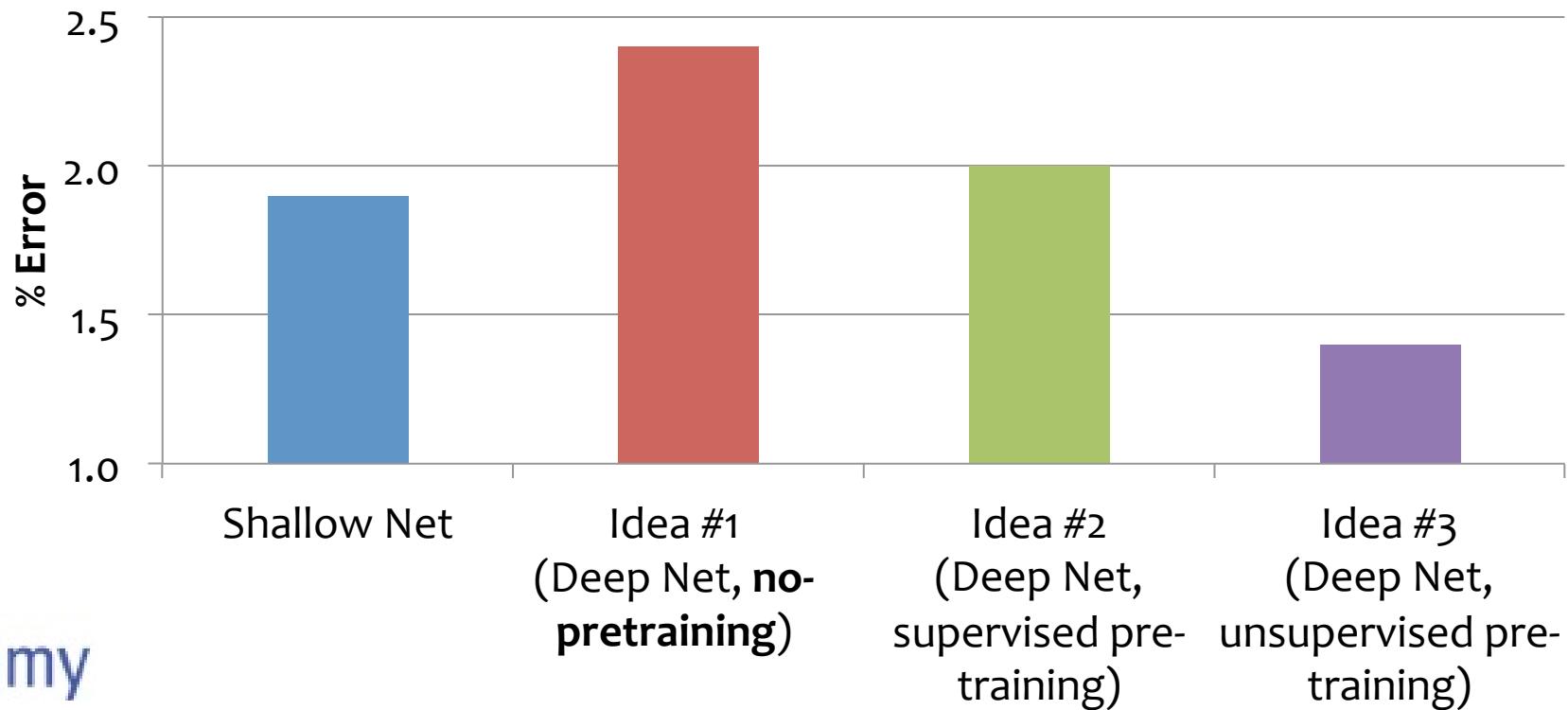
Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)

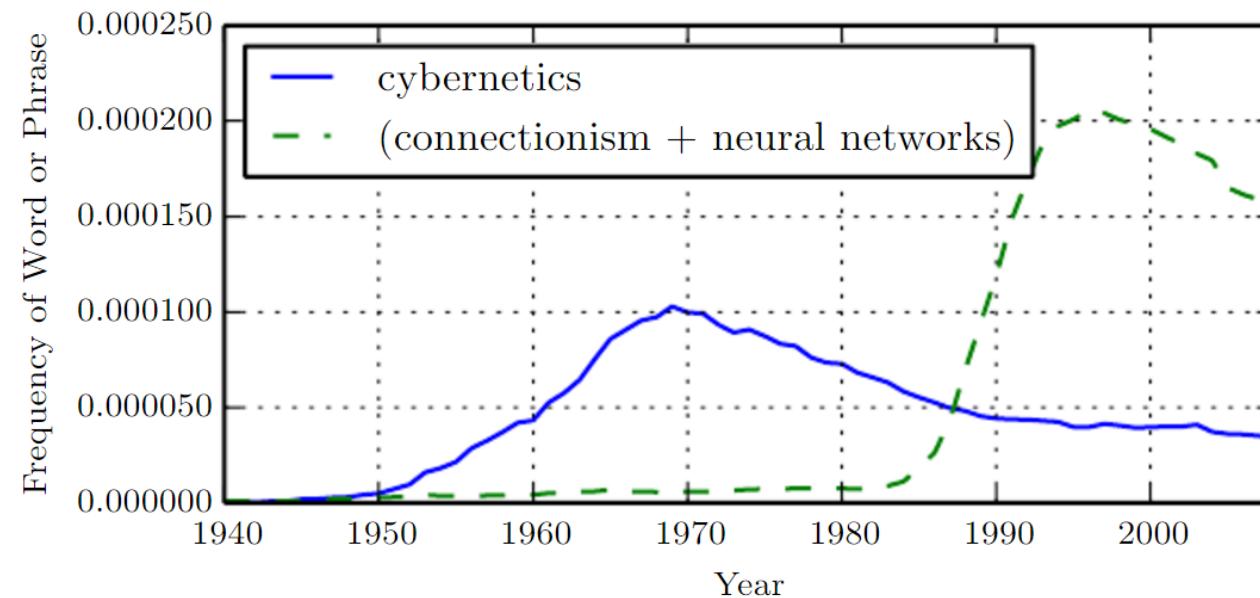


Deep Learning

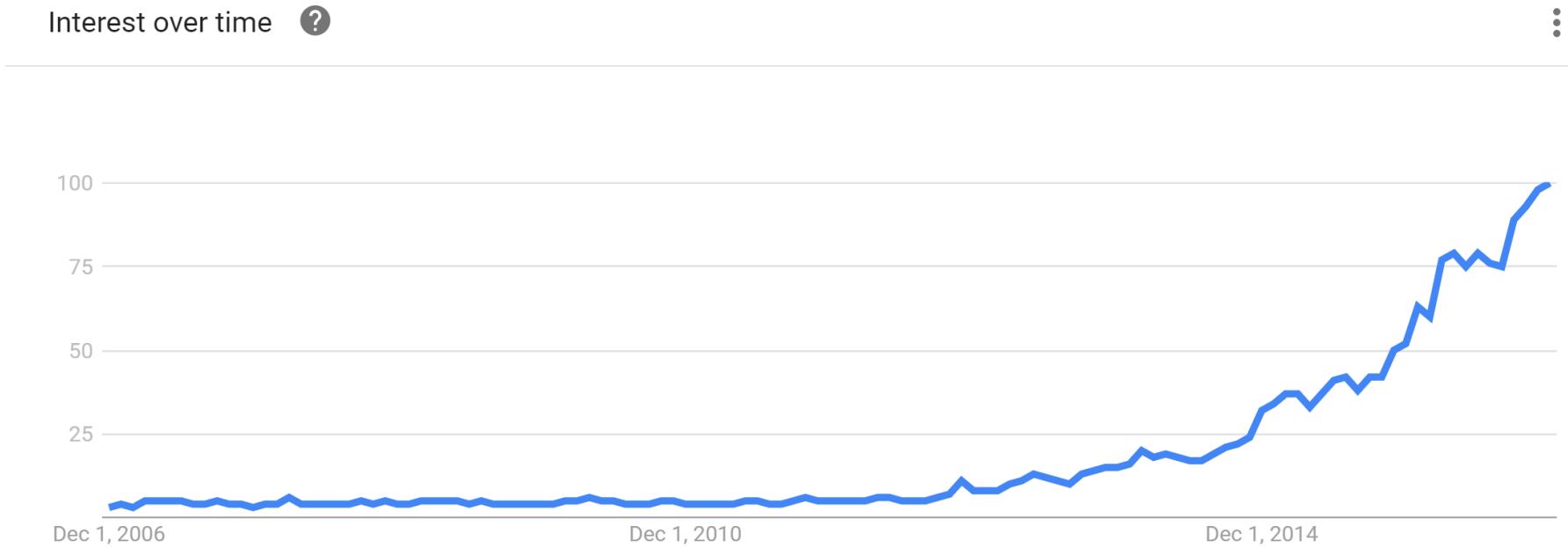
- Goal: learn features at different levels of abstraction
- Training can be tricky due to...
 - Local Optimal
 - Vanishing gradients
- Unsupervised layer-wise pre-training can help with both!

The Seasons of Deep Learning

- 1940s-1960s: Cybernetics
 - Biological learning (1943)
 - Perceptron (1958)
- 1980s-1990s: Connectionism
 - Backpropagation
- 2006-: Deep Learning

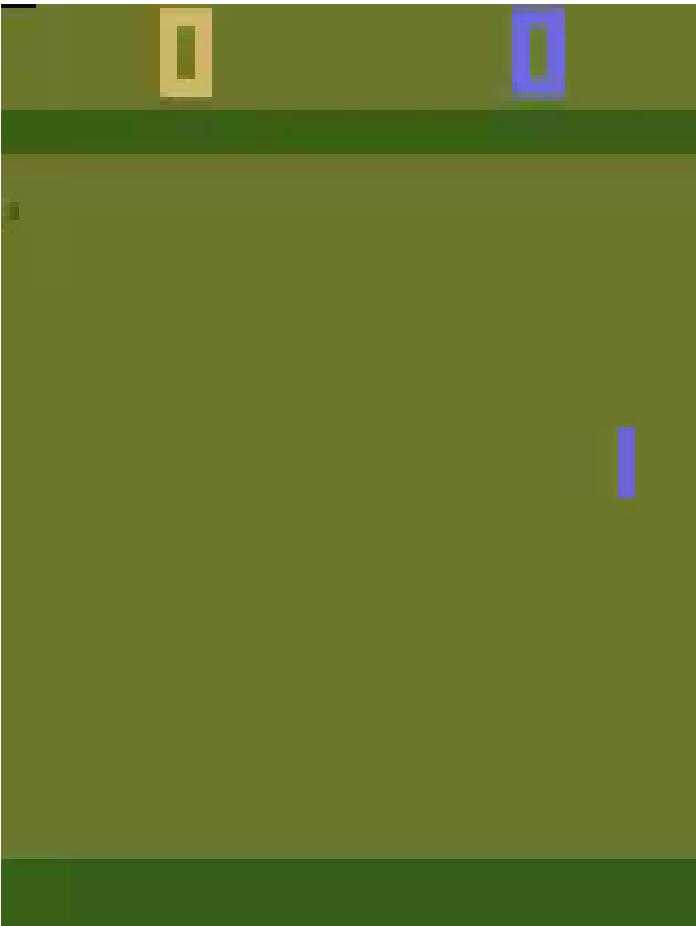


3rd Summer of Deep Learning

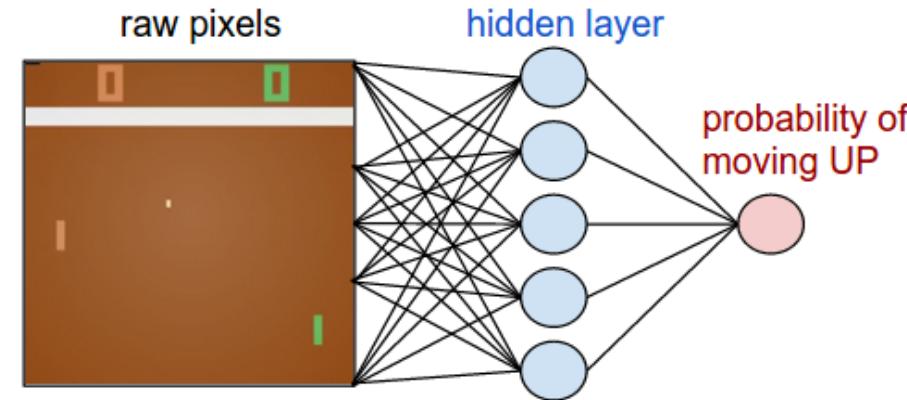


Google trends: “Deep Learning”

Neural Networks are Amazing: General Purpose Intelligence



Policy Network:

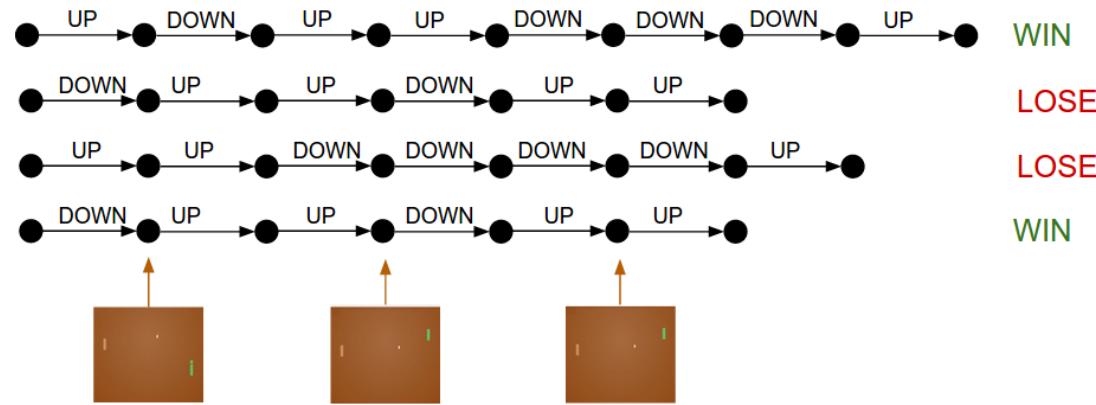
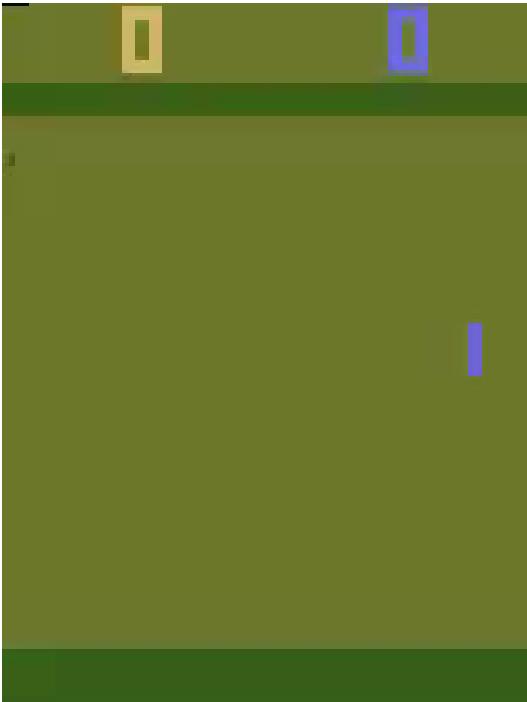


- 80x80 image (difference image)
- 2 actions: up or down
- 200,000 Pong games

**This is a step towards general purpose
artificial intelligence!**

Andrej Karpathy. "Deep Reinforcement Learning: Pong from Pixels." 2016.

Neural Networks are Amazing: General Purpose Intelligence



- Every (state, action) pair is **rewarded** when the final result is a **win**.
- Every (state, action) pair is **punished** when the final result is a **loss**.

The fact that this works at all is amazing!

It could be called “general intelligence” but not yet “human-level” intelligence...

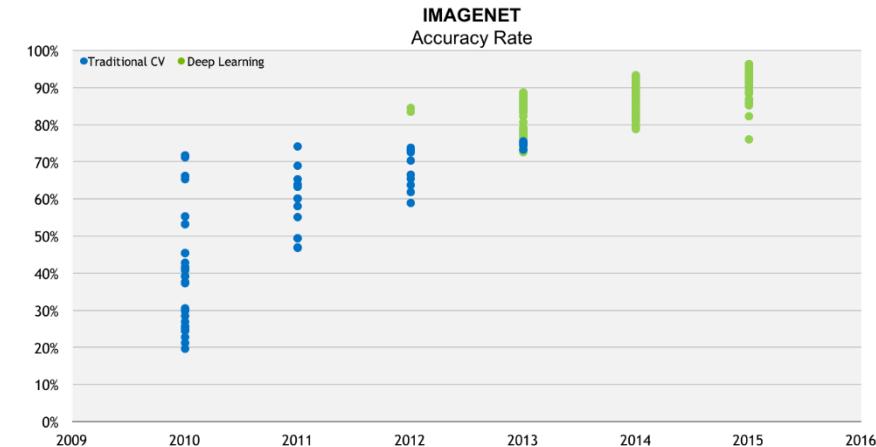
Current Drawbacks

38

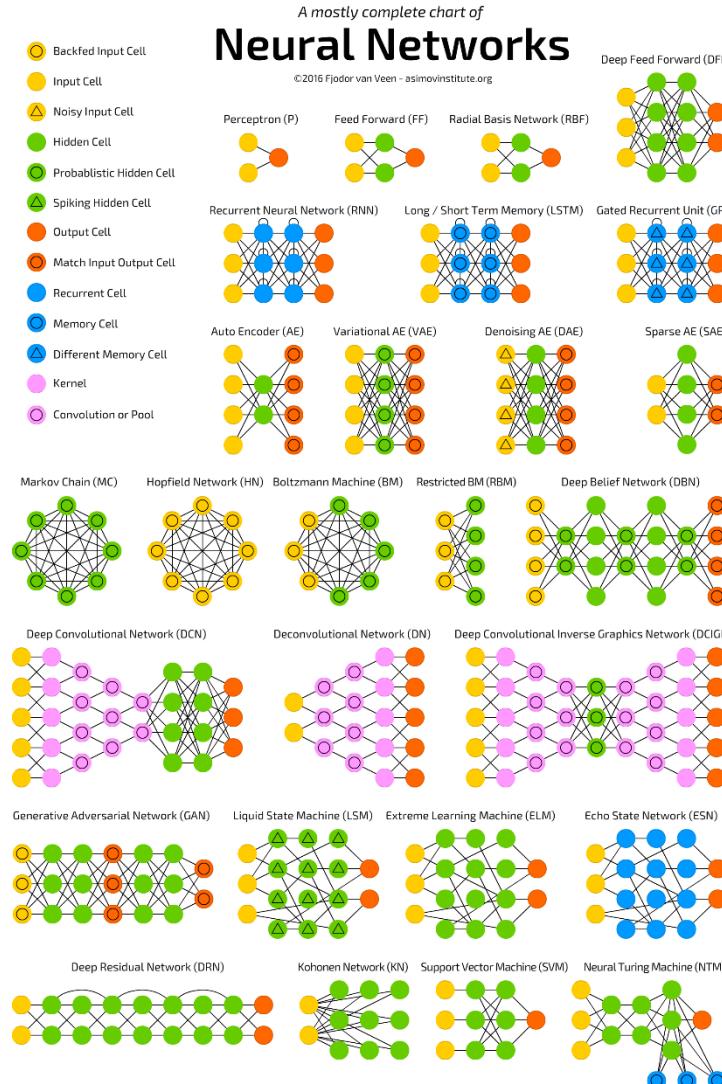
- Lacks Reasoning:
 - Humans only need simple instructions:
“You’re in **control** of a paddle and you can move it up and down, and your task is to bounce the ball past the other player controlled by AI.”
- Requires **big** data: inefficient at learning from data
- Requires **supervised** data: costly to annotate real-world data
- Need to manually select network structure
- Needs hyperparameter tuning for training:
 - Learning rate
 - Loss function
 - Mini-batch size
 - Number of training iterations
 - Momentum: gradient update smoothing
 - Optimizer selection
- Defining a good reward function is difficult...

Deep Learning Breakthroughs: What Changed³⁹

- **Compute**
CPUs, GPUs, ASICs
- **Organized large(-ish) datasets**
Imagenet
- **Algorithms and research:**
Backprop, CNN, LSTM
- **Software and Infrastructure**
Git, ROS, PR2, AWS, Amazon Mechanical Turk, TensorFlow, ...
- **Financial backing of large companies**
Google, Facebook, Amazon, ...



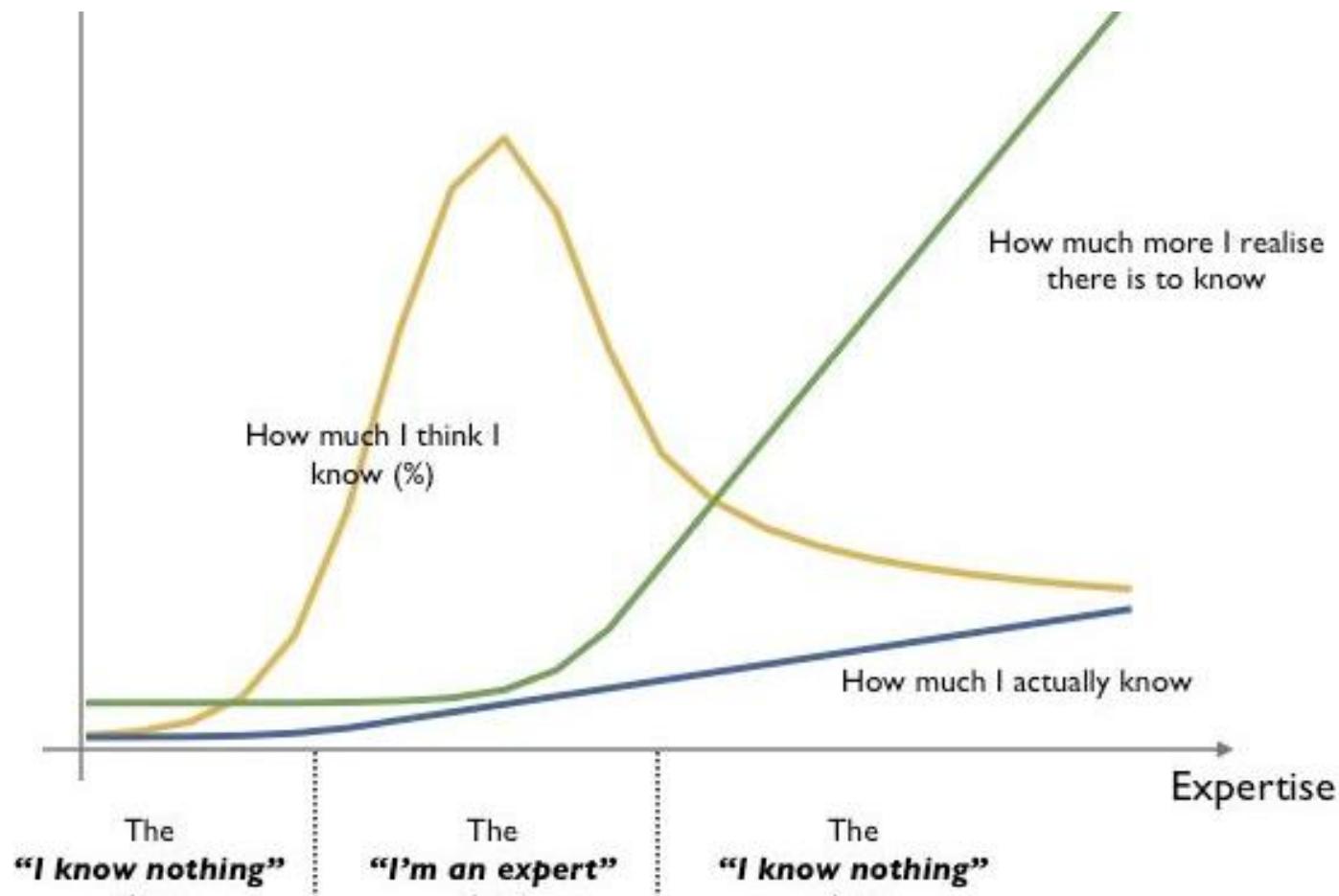
Useful Deep Learning Terms



- Basic terms:
 - **Deep Learning = Neural Networks**
 - **Deep Learning** is a subset of **Machine Learning**
- Terms for neural networks:
 - **MLP**: Multilayer Perceptron
 - **DNN**: Deep neural networks
 - **RNN**: Recurrent neural networks
 - **LSTM**: Long Short-Term Memory
 - **CNN or ConvNet**: Convolutional neural networks
 - **DBN**: Deep Belief Networks
- Neural network operations:
 - Convolution
 - Pooling
 - Activation function
 - Backpropagation

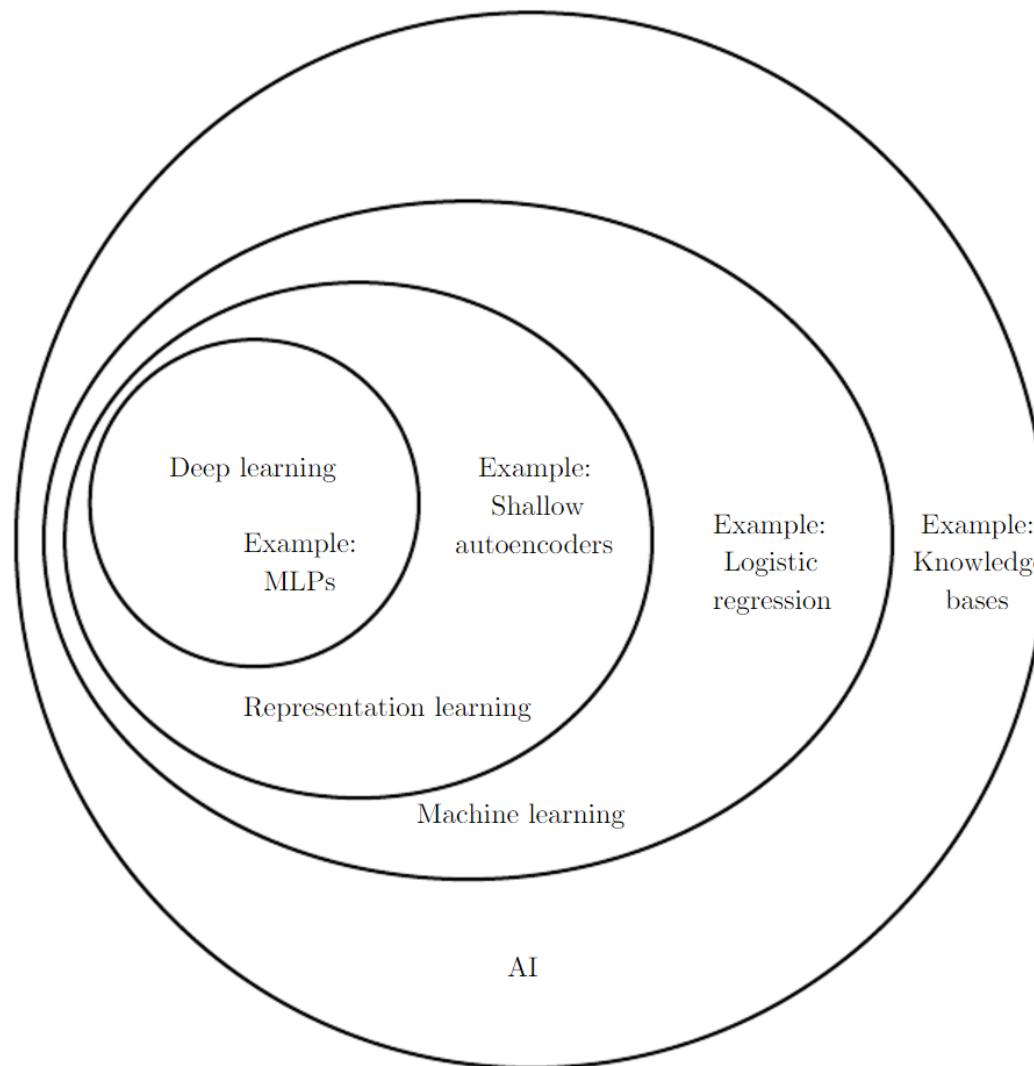
Asimov Institute. "A mostly complete chart of neural networks."

Neural Networks: Proceed with Caution

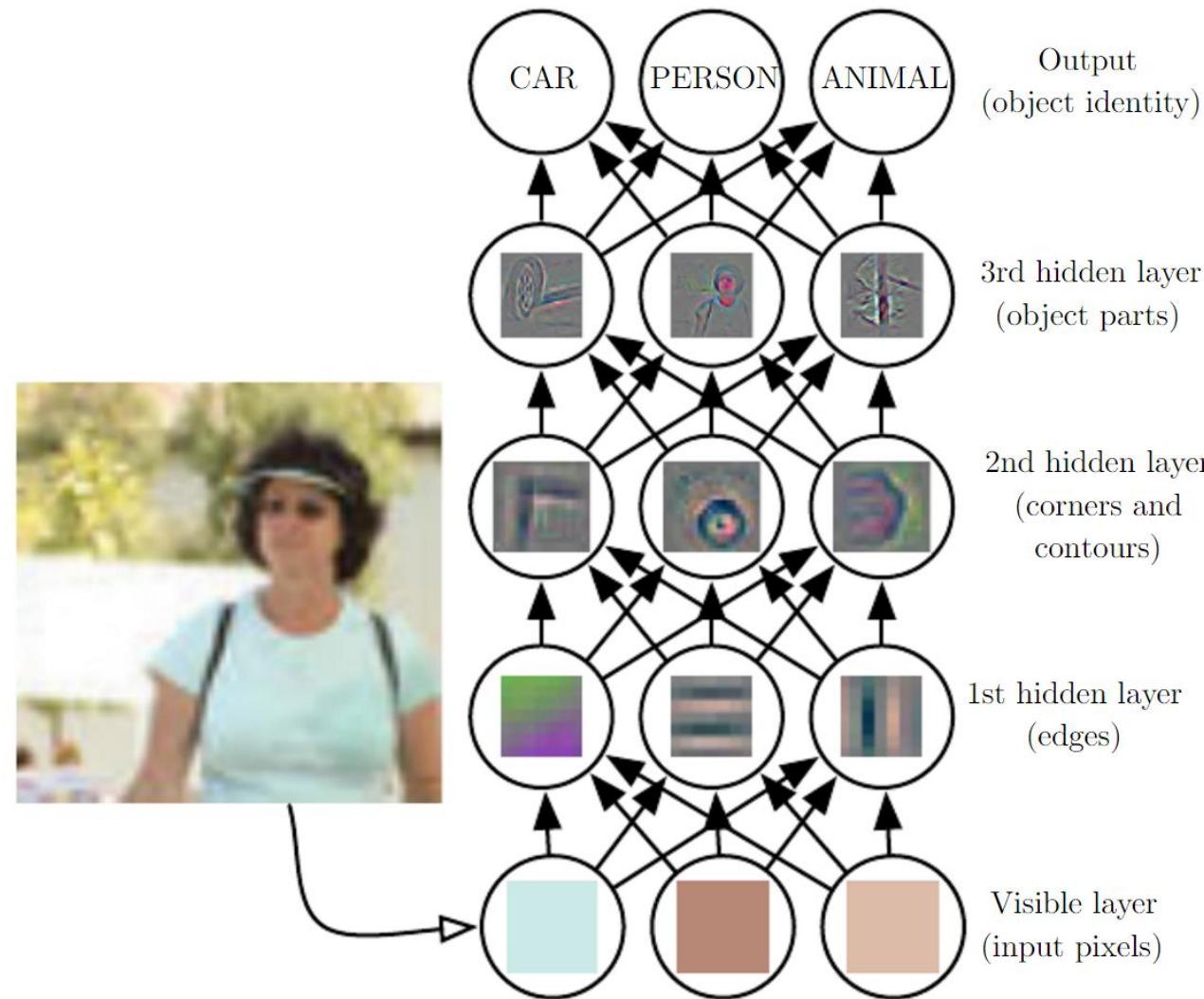


Deep Learning is Representation Learning

42

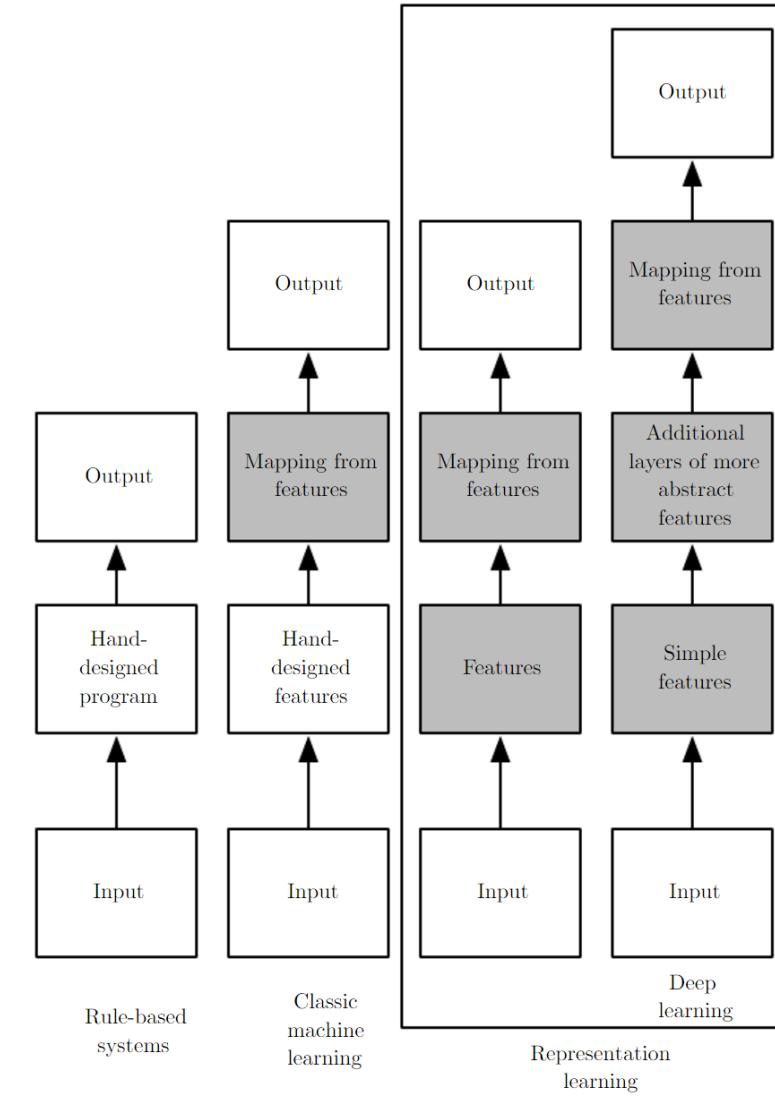
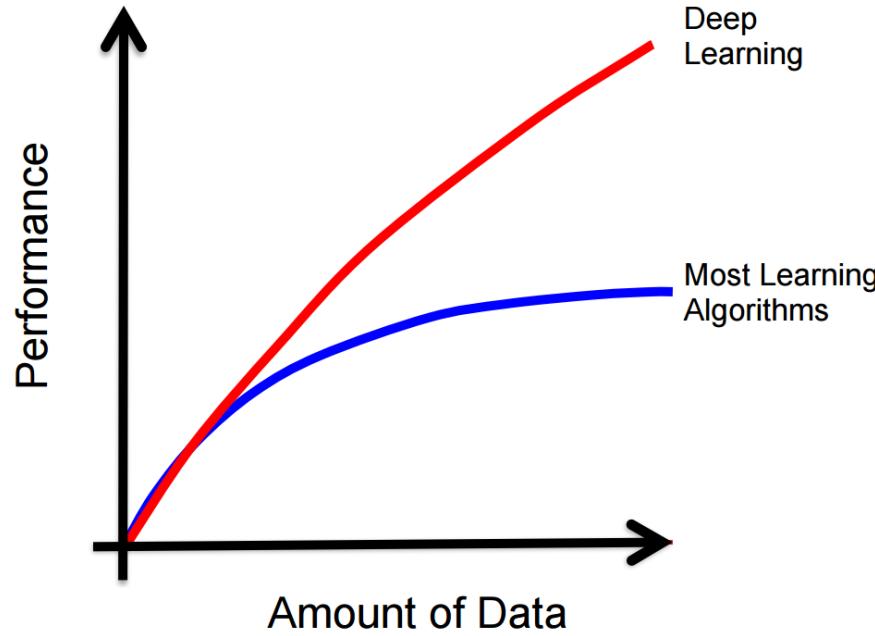


Deep Learning is Representation Learning

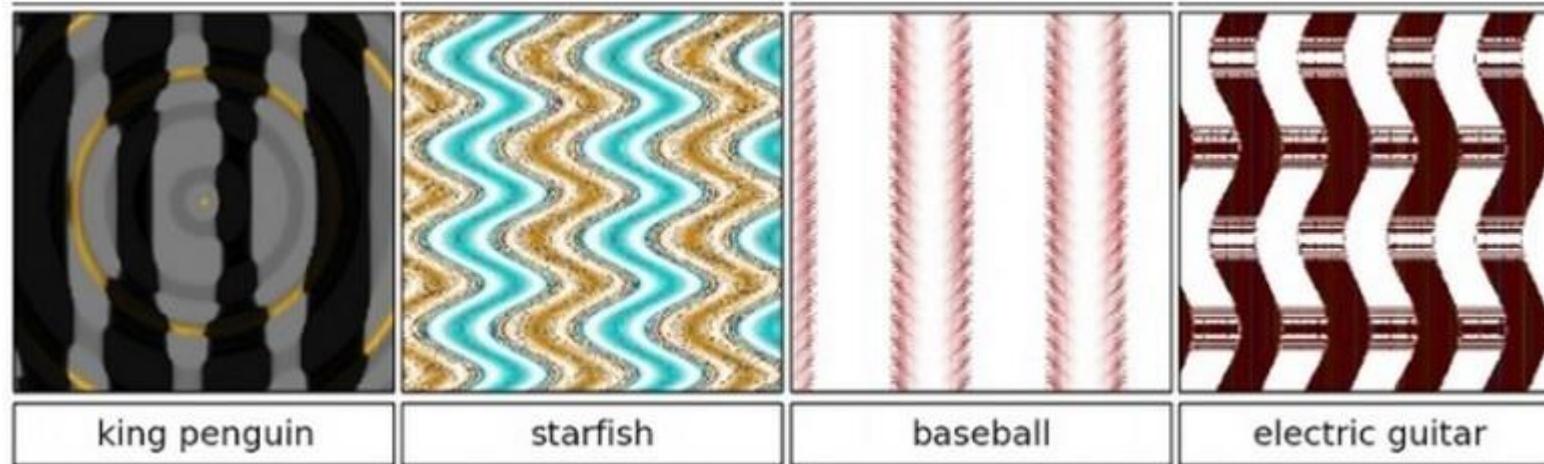
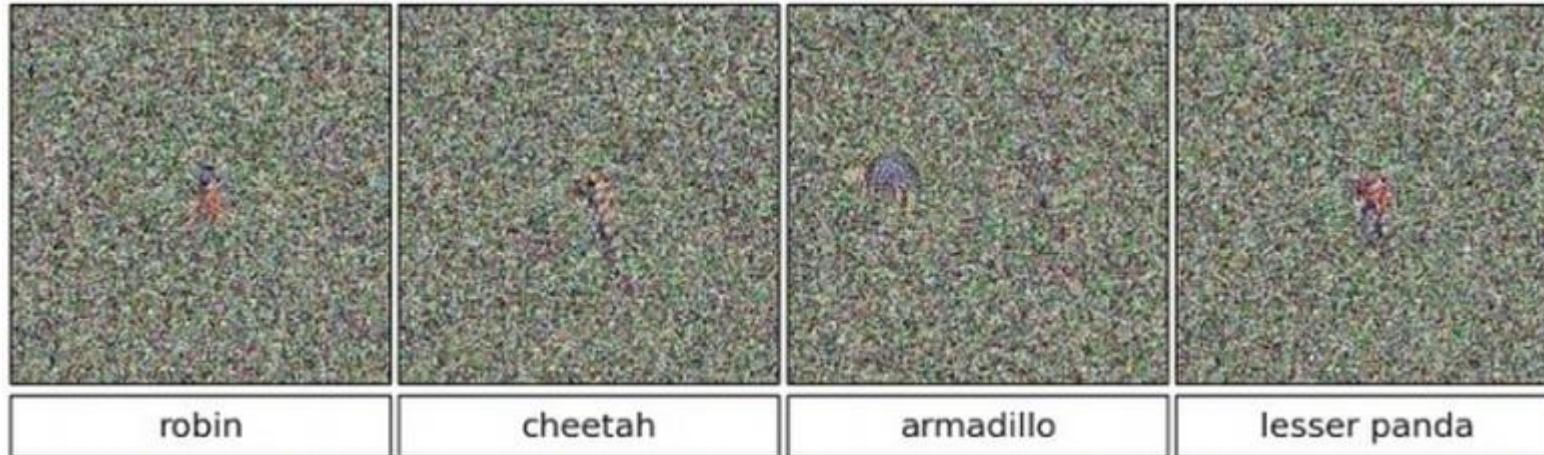


[20] Goodfellow et al. "Deep learning." (2017).

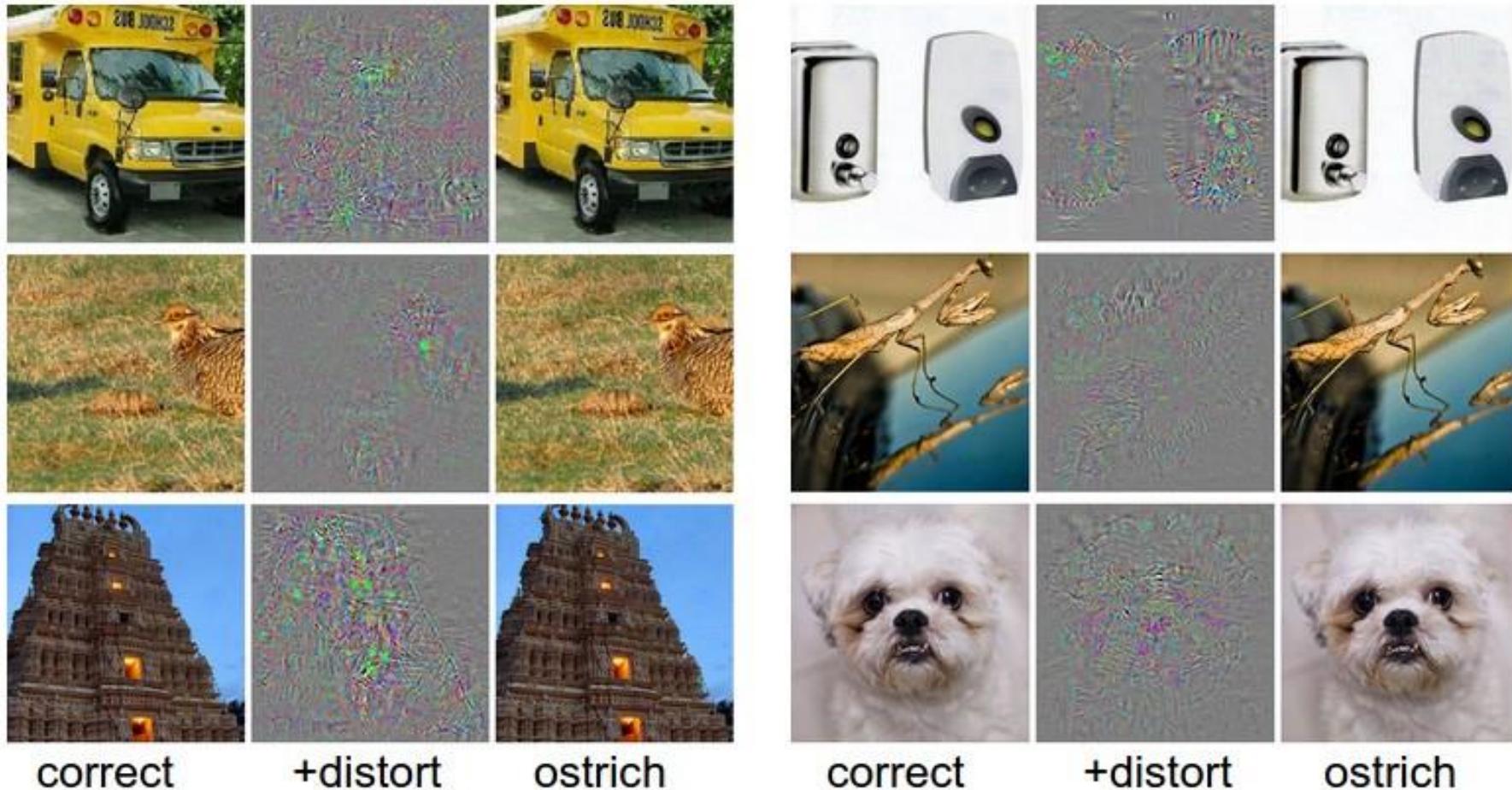
Deep Learning: Scalable Machine Learning



>99.6% Confidence in the Wrong Answer



Robustness: Fooled by a Little Distortion



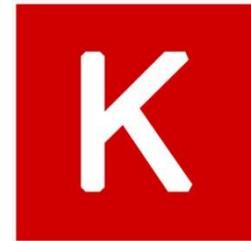


- Interface: Python, (C++)
- Automatic Differentiation
- Multi GPU, Cluster Support
- Currently most popular

The Google logo, featuring the word "Google" in its signature multi-colored font: blue, red, yellow, green, and red again.

Keras

- On top of Tensorflow (and Theano)
 - Interface: Python
 - Goal: provide a simplified interface
-
- **Also:** TF Learn, TF Slim





Torch & PyTorch

- Used by researchers doing lower level (**closer to the details**) neural net work
- Interface: Lua
- Fragmented across different plugins

facebook

theano

- Interface: Python (tight NumPy integration)
- One of the earlier frameworks with GPU support
- Encourages low-level tinkering



cuDNN



- The library that most frameworks use for doing the actual computation
- Implements primitive neural network functions in CUDA on the GPU



- Multi GPU Support (scales well)
- Interface: Python, R, Julia, Scala, Go, Javascript ...



Caffe

- Interface: C++, Python
- One of the earliest GPU supported
- Initial focus on computer vision (and CNNs)

Berkeley
Artificial Intelligence Research Laboratory

Microsoft Cognitive Toolkit (CNTK)

54

- Interface: Custom Language (BrainScript), Python, C++, C#
- Multi GPU Support (scales very well)
- Mostly used at MS Research

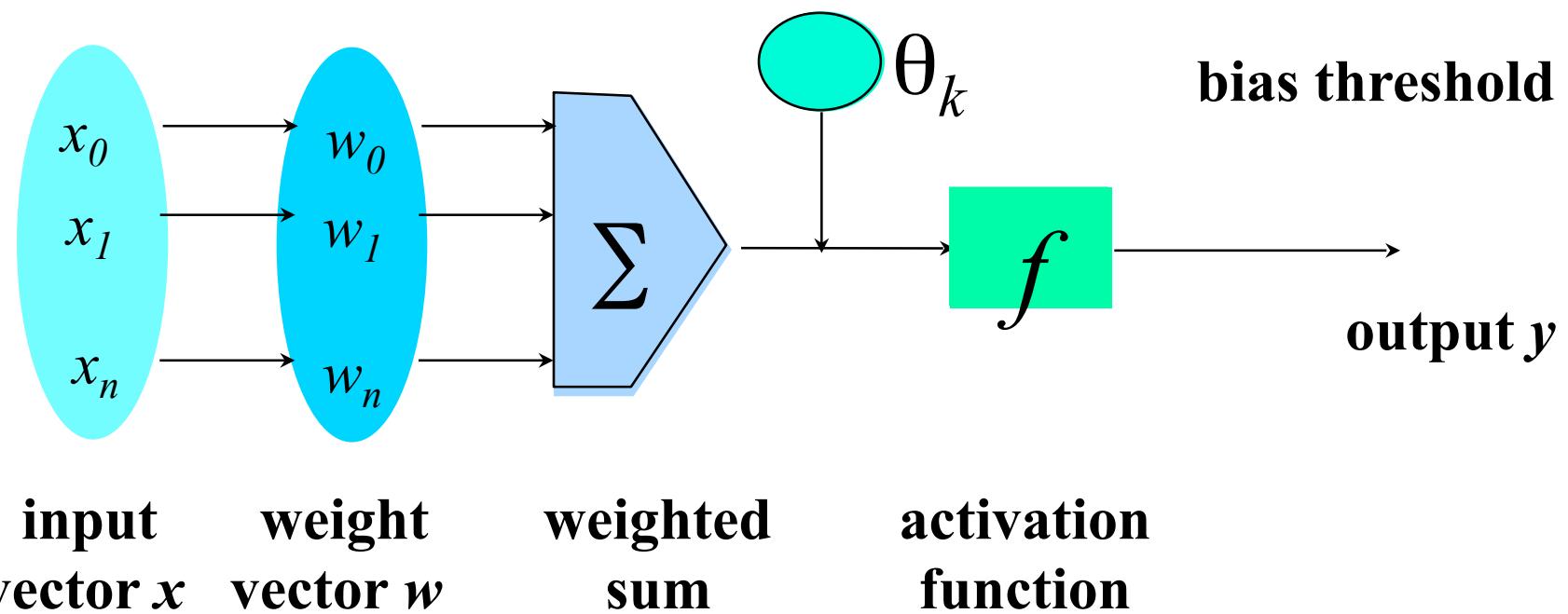


In the Browser

- Keras.js
 - GPU Support
 - Full sized networks
 - Can use trained Keras models
- ConvNetJS
 - Built by Andrej Karpathy
 - Good for explaining neural network concepts
 - Fun to play around with
 - Very few requirements
 - Full CNN, RNN, Deep Q Learning

Summary: Artificial neurons

- Nonlinear functions mapping n-dimensional input vectors x into outputs y by linear weighting, thresholding, and nonlinear transformation



Activation functions

- Linear thresholds
- Sigmoid function
- Radial basis functions
- Other function bases

Advantages

- High prediction accuracy possible
 - With choice of suitable network topology
 - With choice of suitable learning method
- Discrete, continuous, or vector outputs
- Fast classification once trained

Disadvantages

- Empirical search for suitable networks
- Overfitting still a possibility
- No explicit knowledge
- Incorporating domain knowledge is awkward

Network construction

- Network topology selection
 - Number of hidden layers Number
 - of nodes in hidden layers
 - Interconnections of nodes
- Network training
 - Find weights that yield the best classification given topology
- Network topology editing
 - Find topology that yields best classification
 - Find topology that yields clearest interpretation

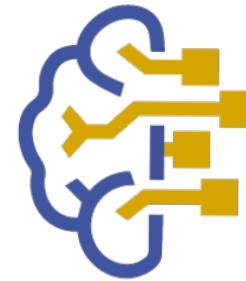
Stay Connected

Dr. Min Chi

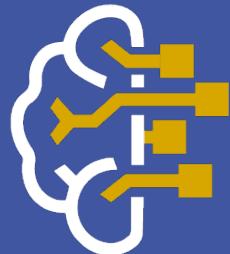
Associate Professor

mchi@ncsu.edu

(919) 515-7825



AI Academy



AI Academy

go.ncsu.edu/aiacademy

NC STATE UNIVERSITY