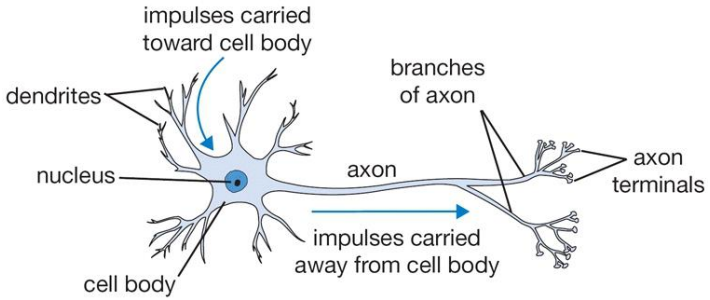
Deep Learning Overview

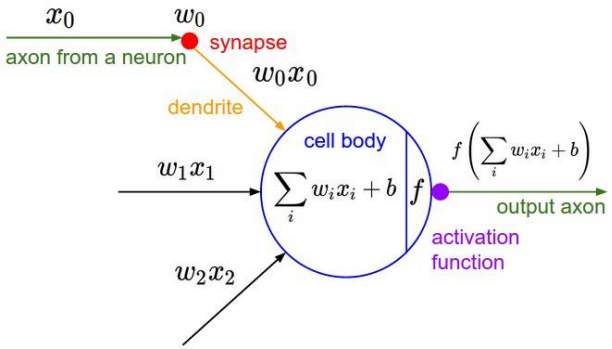
©Dr. Min Chi

mchi@ncsu.edu

AI Academy

Neuron: Biological Inspiration for Computation



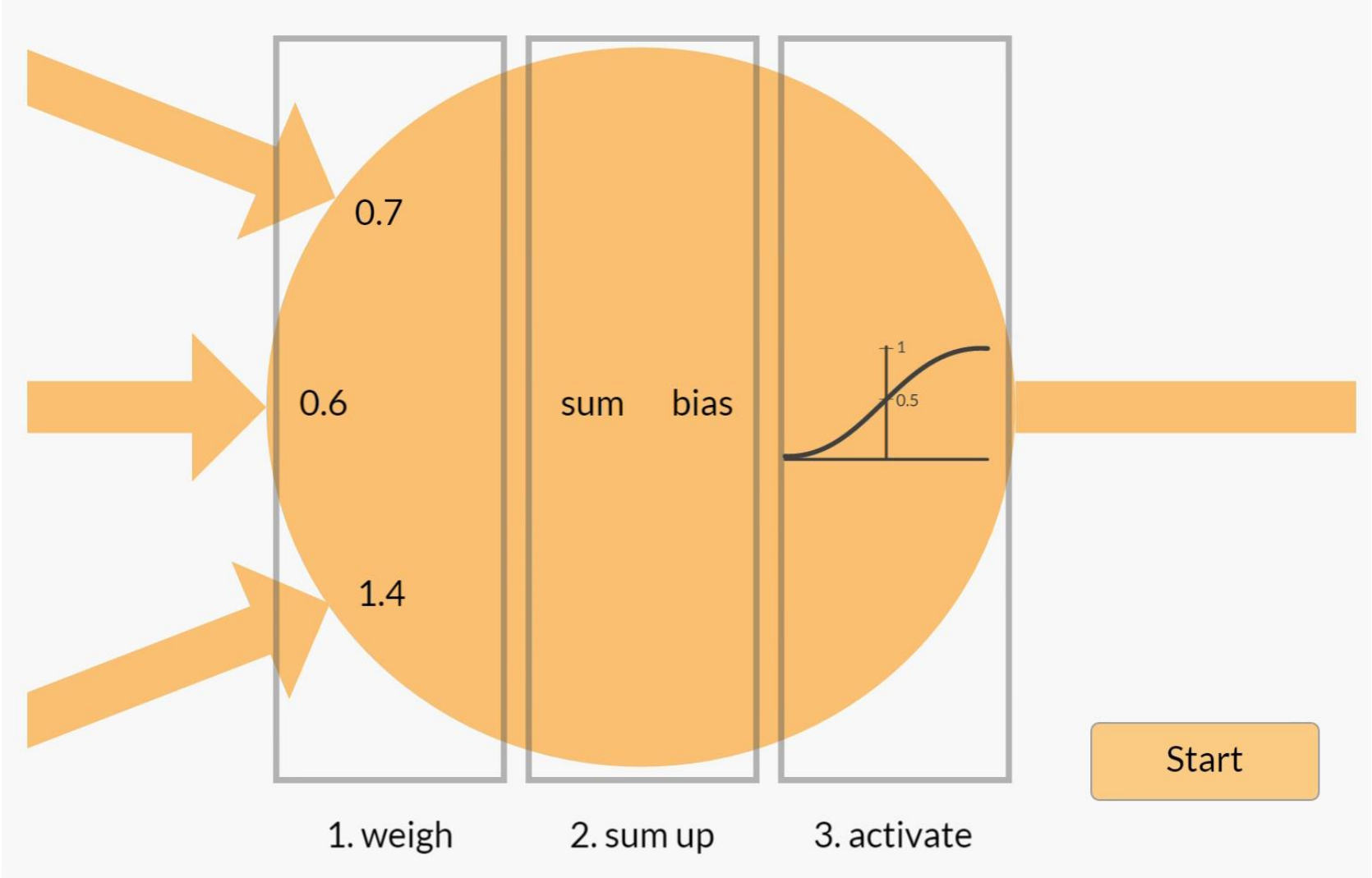


- **Neuron:** computational building block for the brain
- Human brain:
  - ~100-1,000 trillion synapses

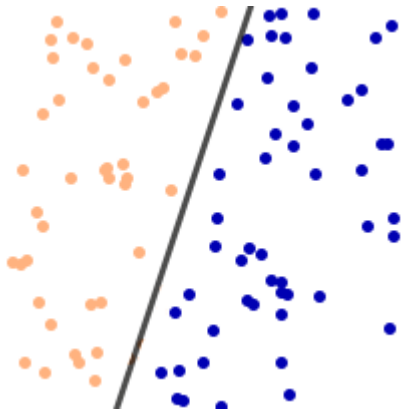
- **(Artificial) Neuron:** computational building block for the "neural network"
- (Artificial) neural network:
  - ~1-10 billion synapses

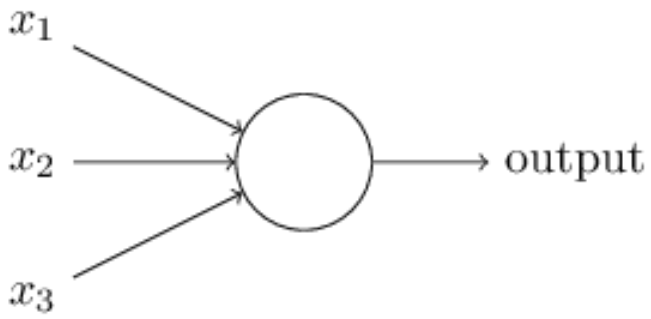
Human brains have ~10,000 computational power than computer brains.

Perceptron: Forward Pass



Perceptron Algorithm

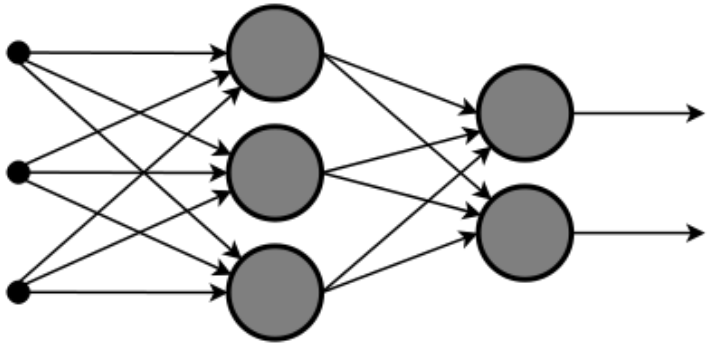




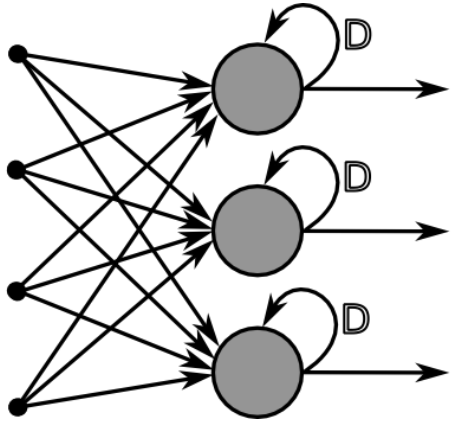
Provide training set of (input, output) pairs and run:

1. Initialize perceptron with random weights
2. For the inputs of an example in the training set, compute the Perceptron's output
3. If the output of the Perceptron does not match the output that is known to be correct for the example:
    1. If the output should have been 0 but was 1, decrease the weights that had an input of 1.
    2. If the output should have been 1 but was 0, increase the weights that had an input of 1.
4. Go to the next example in the training set and repeat steps 2-4 until the Perceptron makes no more mistakes

Combining Neurons into Layers



Feed Forward Neural Network

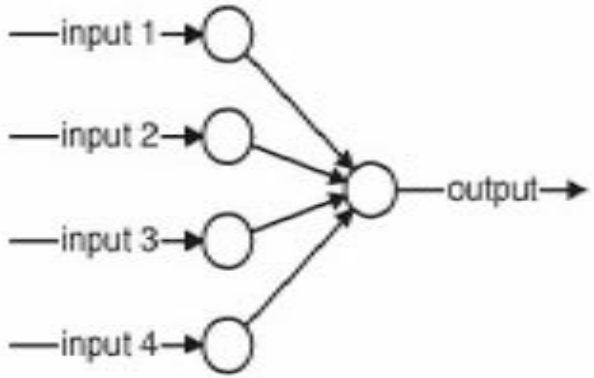


Recurrent Neural Network

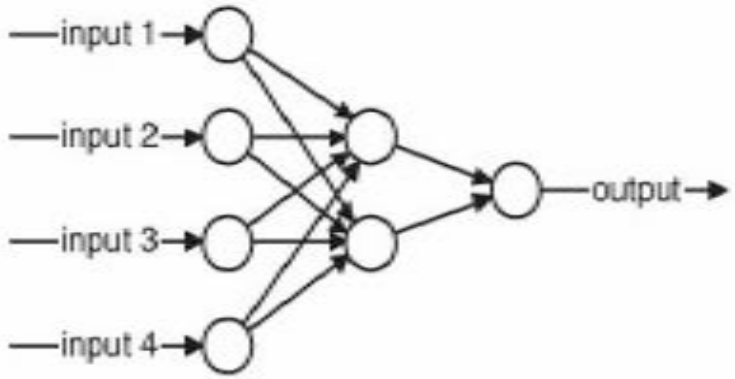
- Have state memory
- Are hard to train

Feed-Forward Neural Net Examples

- One-way flow through the network from inputs to outputs

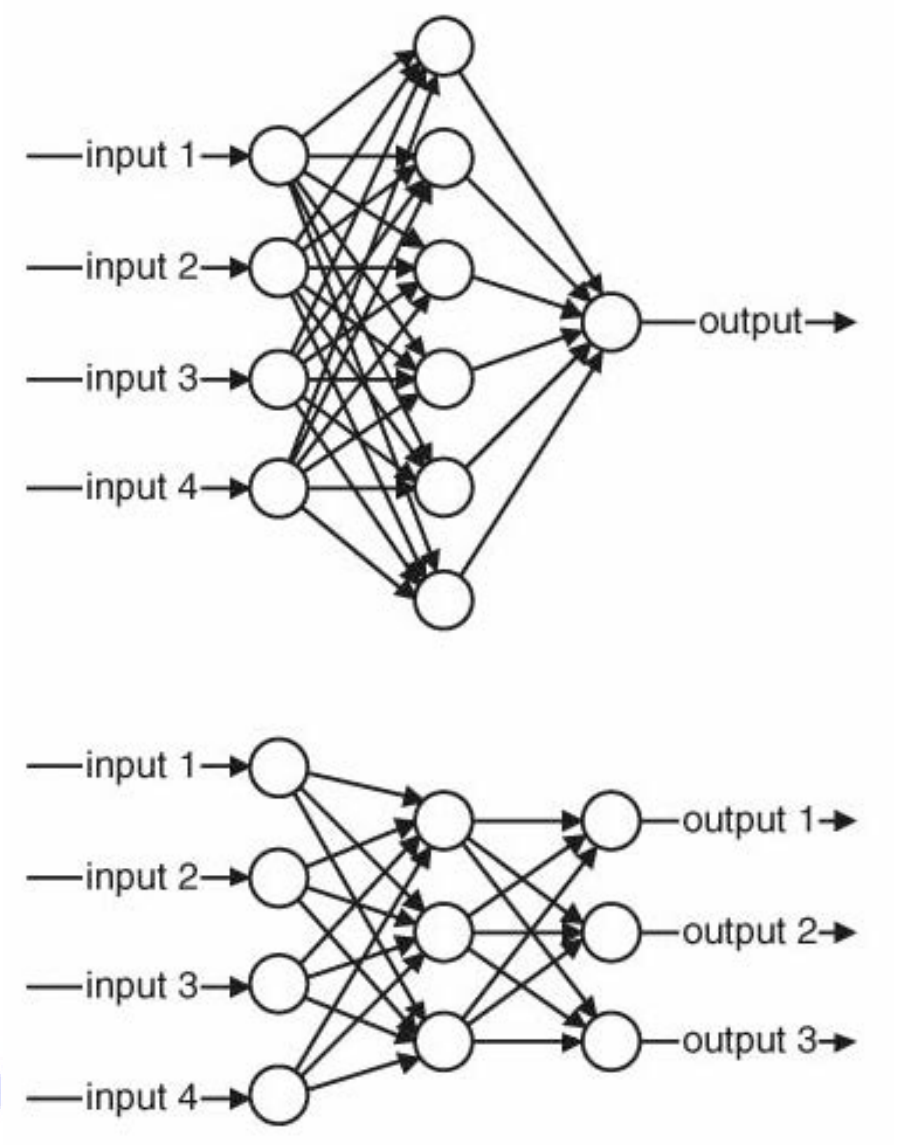


A very simple neural network that takes four inputs and produces one outoput.



This network has a middle layer called the hidden layer, which makes the network more powerful by enabling it to recognize more patterns.

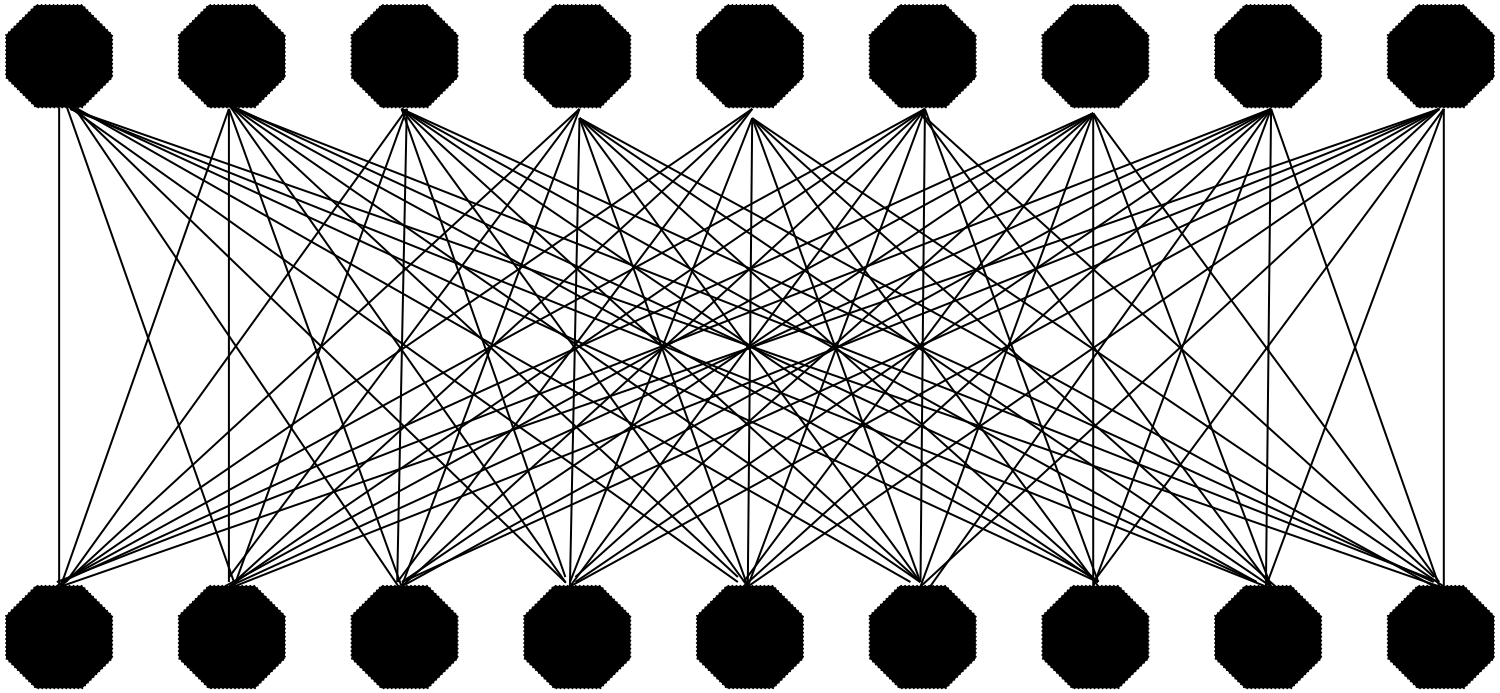
Feed-Forward Neural Net Examples



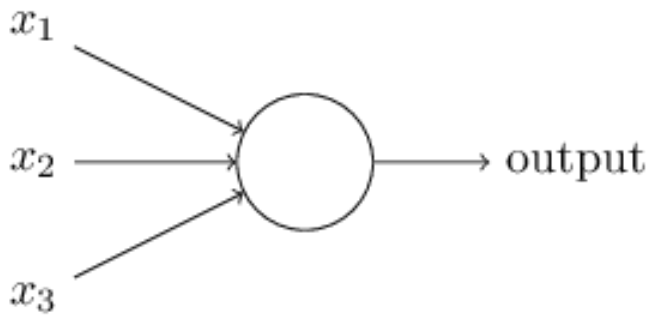
Increasing the size of the hidden layer makes the network more powerful but introduce the risk of overfitting. Usually, only one hiddent layer is needed.

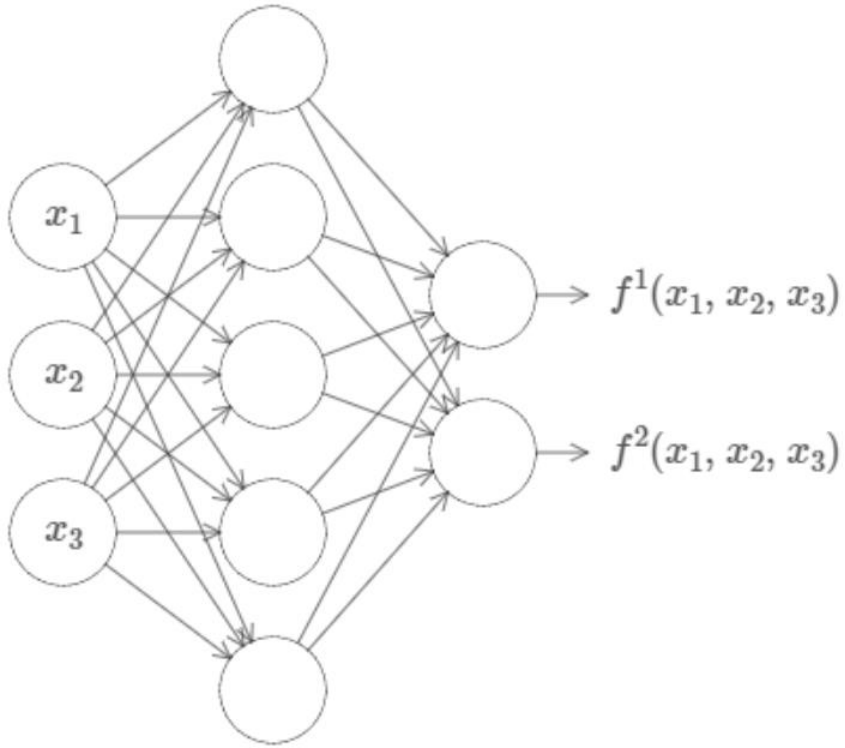
A neural network can produce multiple output values.

Two-layer ANN: 9 units in each layer, fully connected network



Neural Networks are Amazing



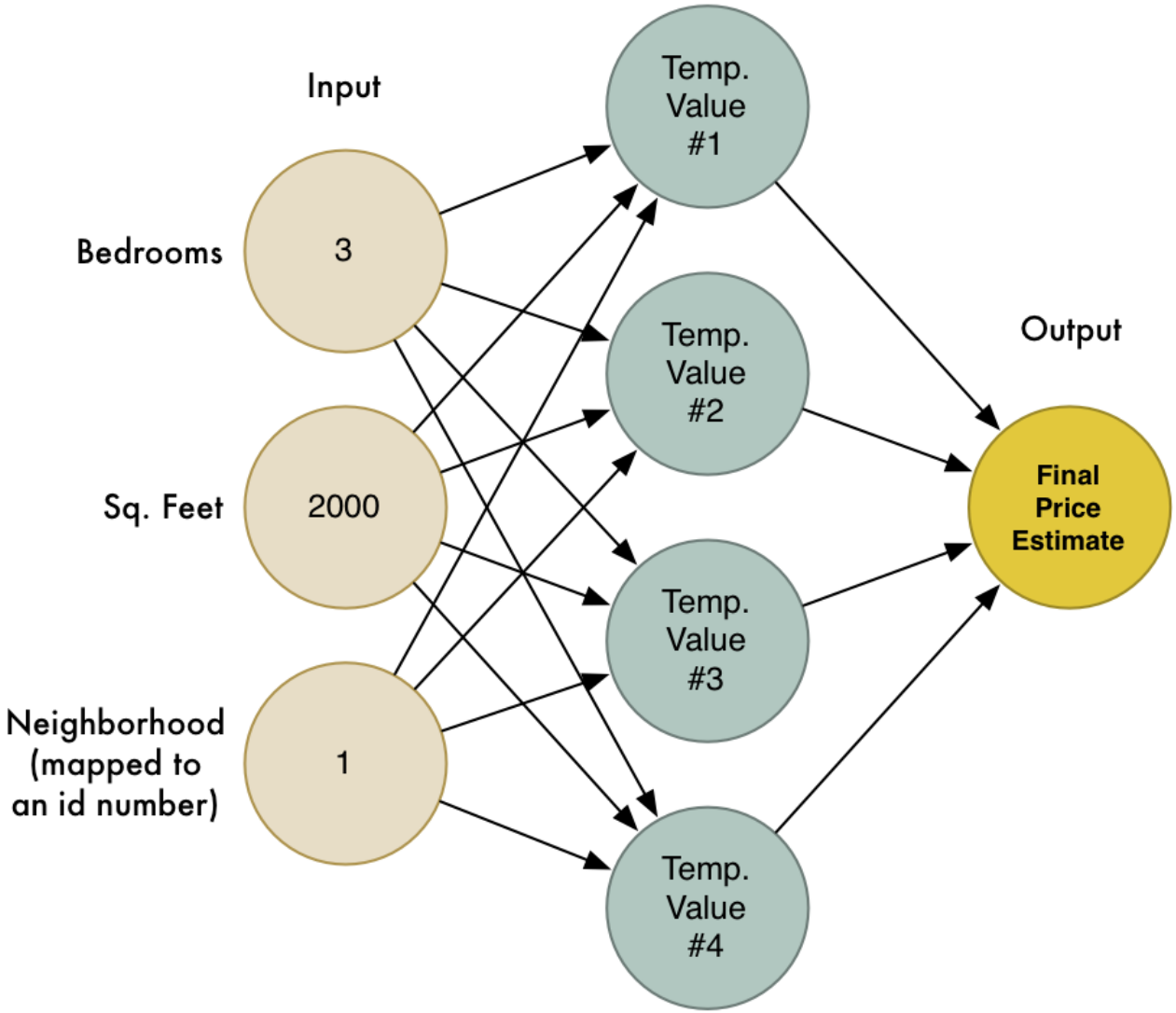


Universality: For any arbitrary function f(x), there exists a neural network that closely approximate it for any input x

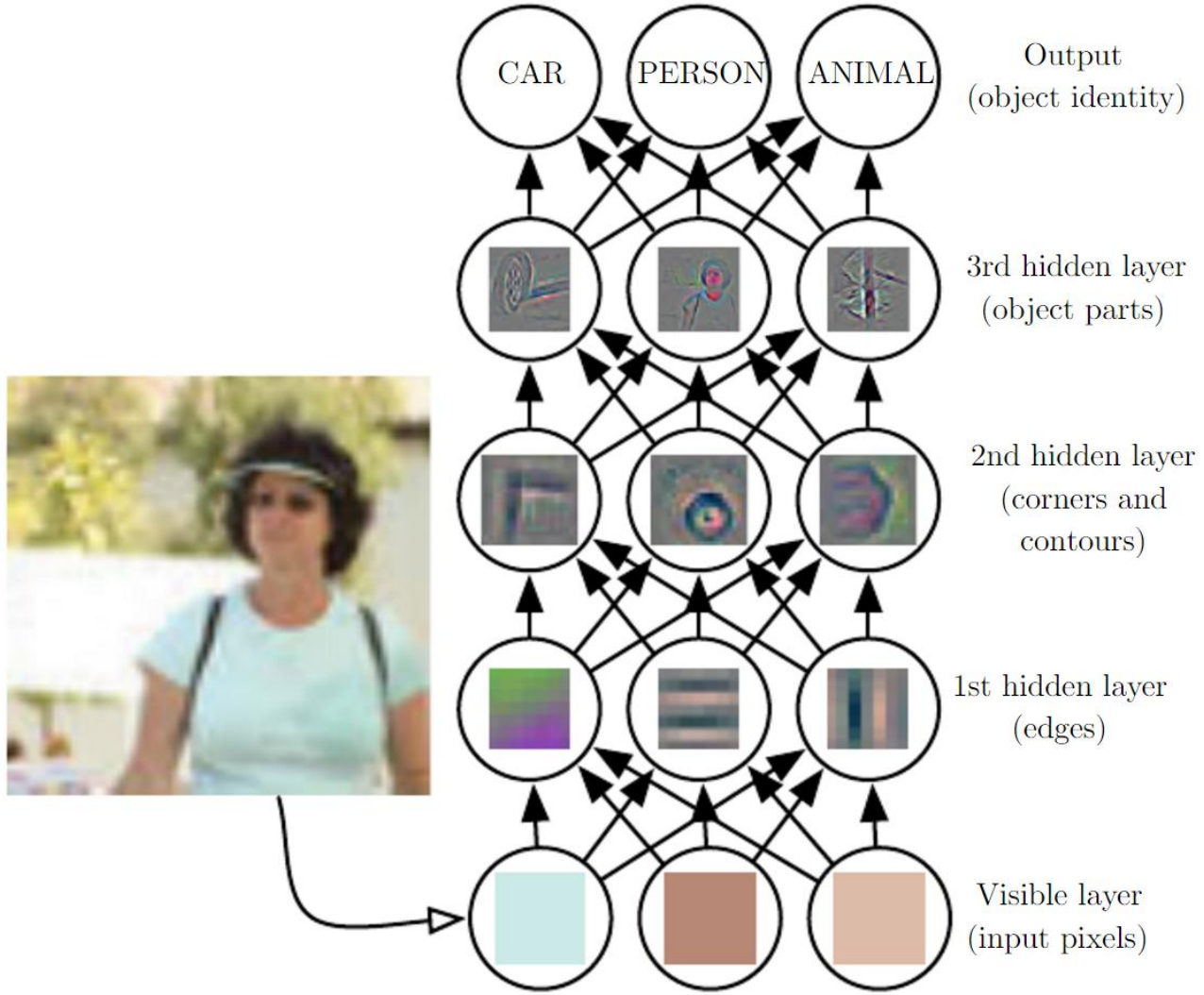
Universality is an incredible property!* And it holds for just 1 hidden layer.

* Given that we have good algorithms for training these networks.

Special Purpose Intelligence

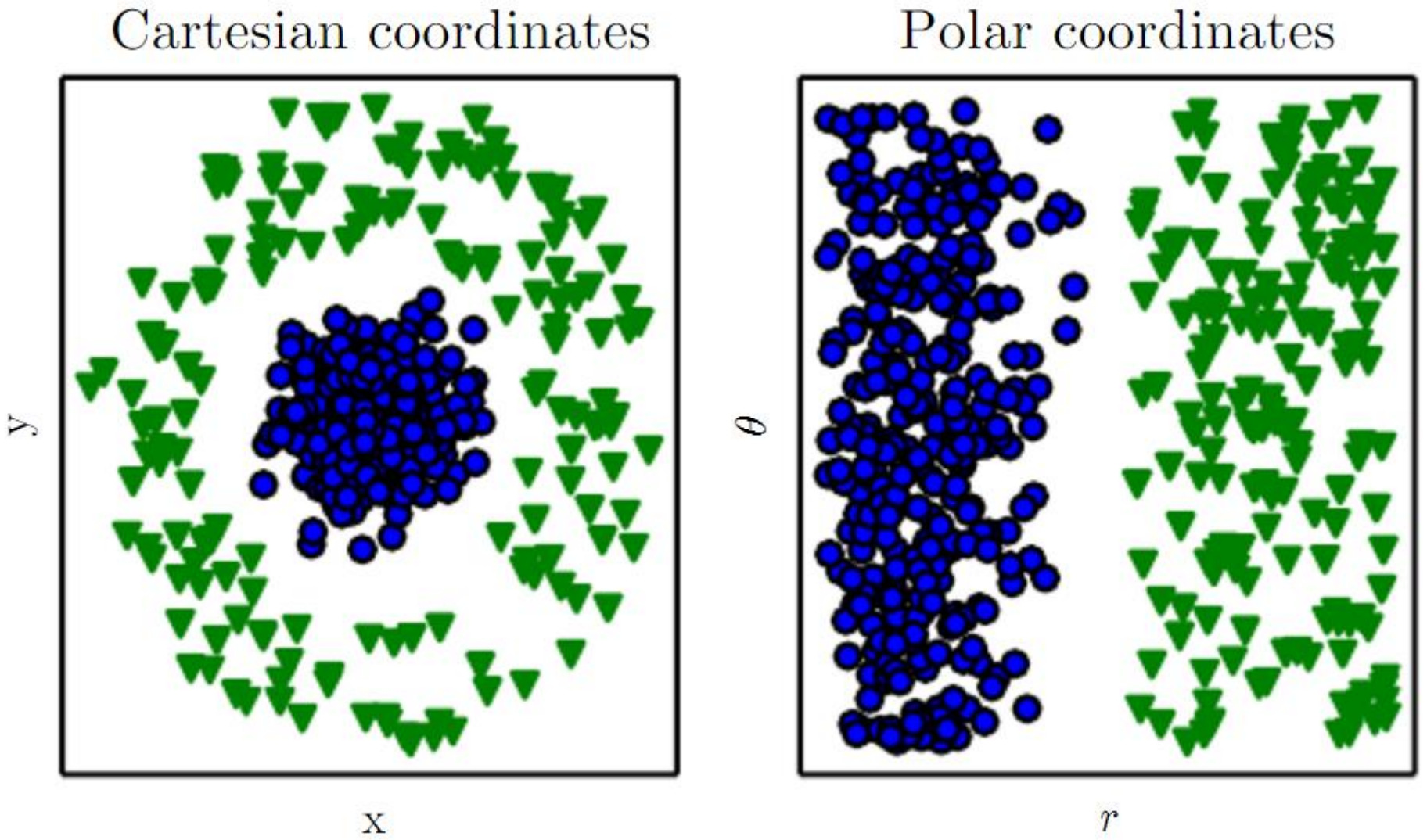


Deep Learning is Representation Learning



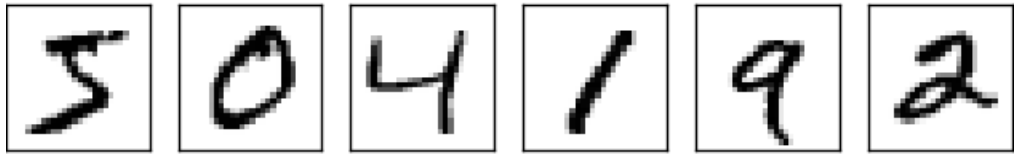
[20] Goodfellow et al. "Deep learning." (2017).

Representation Matters

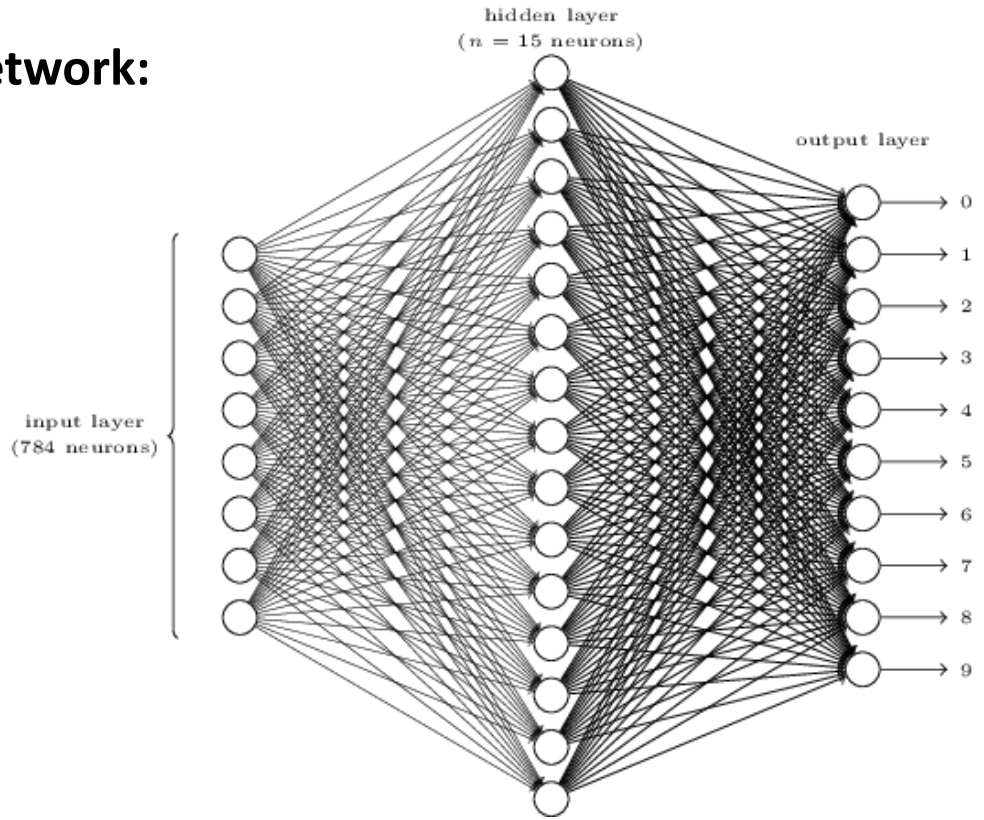


Task: Classify and Image of a Number

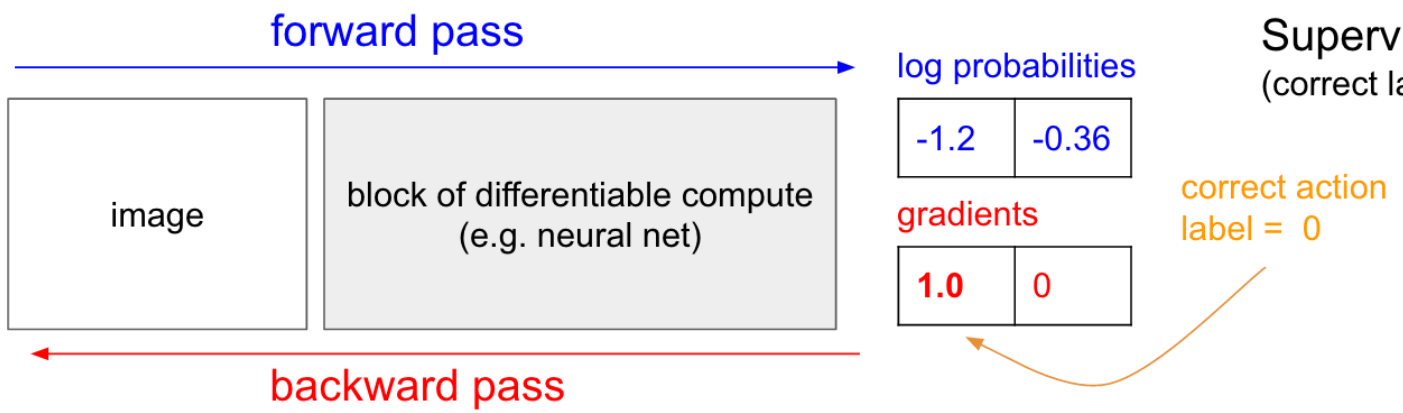
Input:
(28x28)



Network:

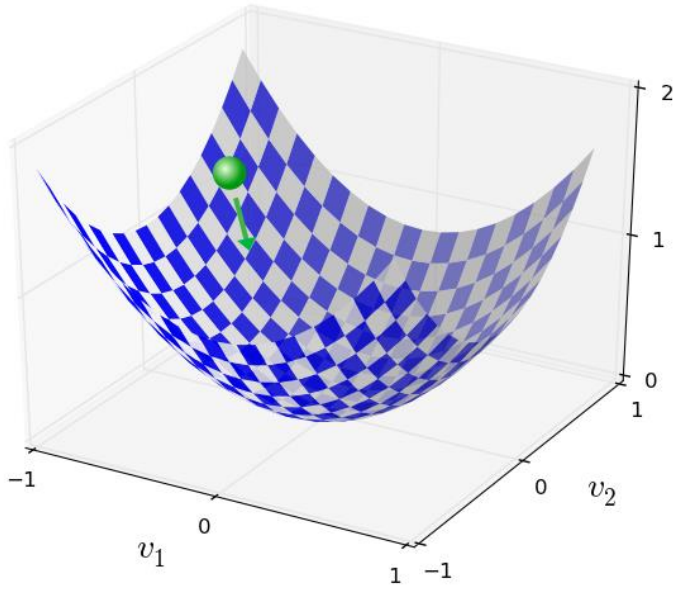


Task: Classify and Image of a Number



Ground truth for "6":

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



"Loss" function:

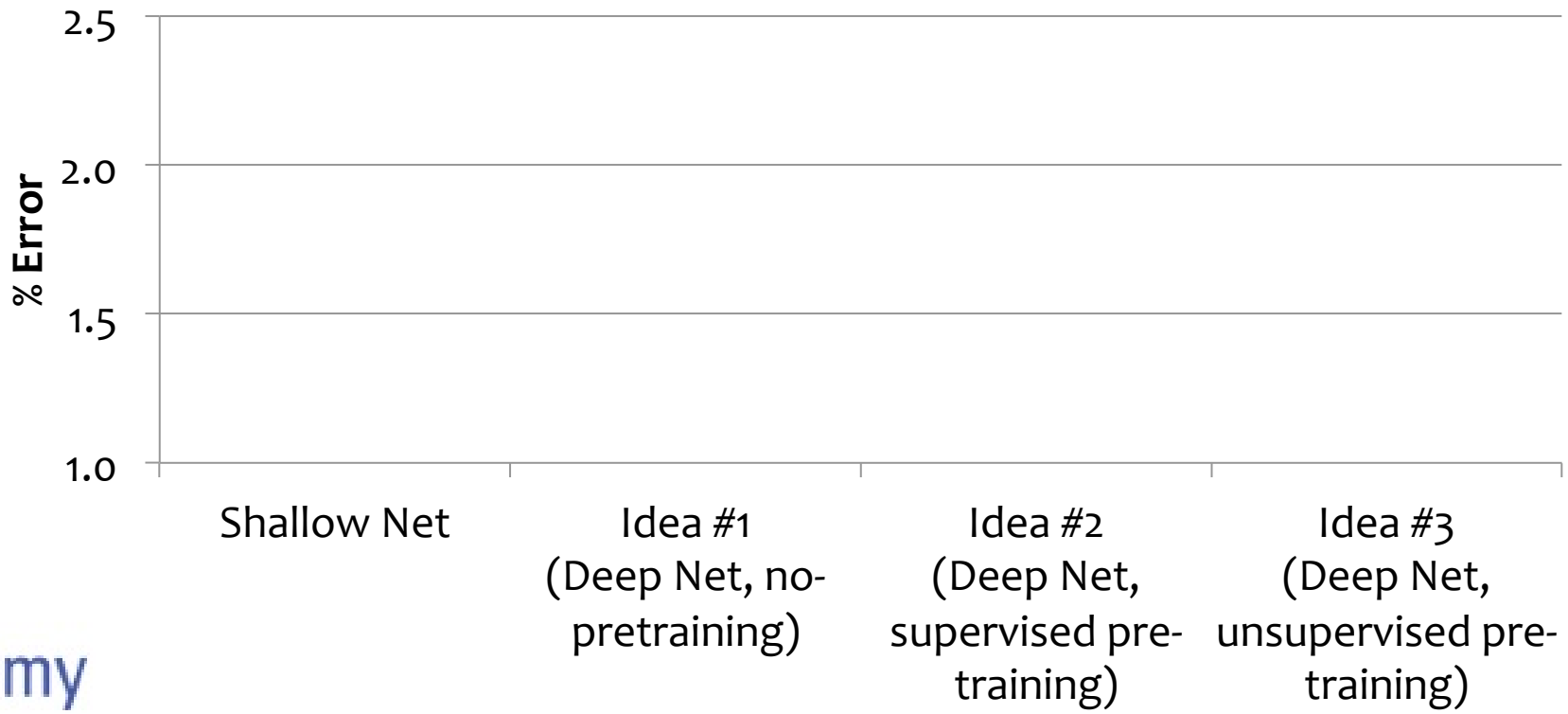
$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

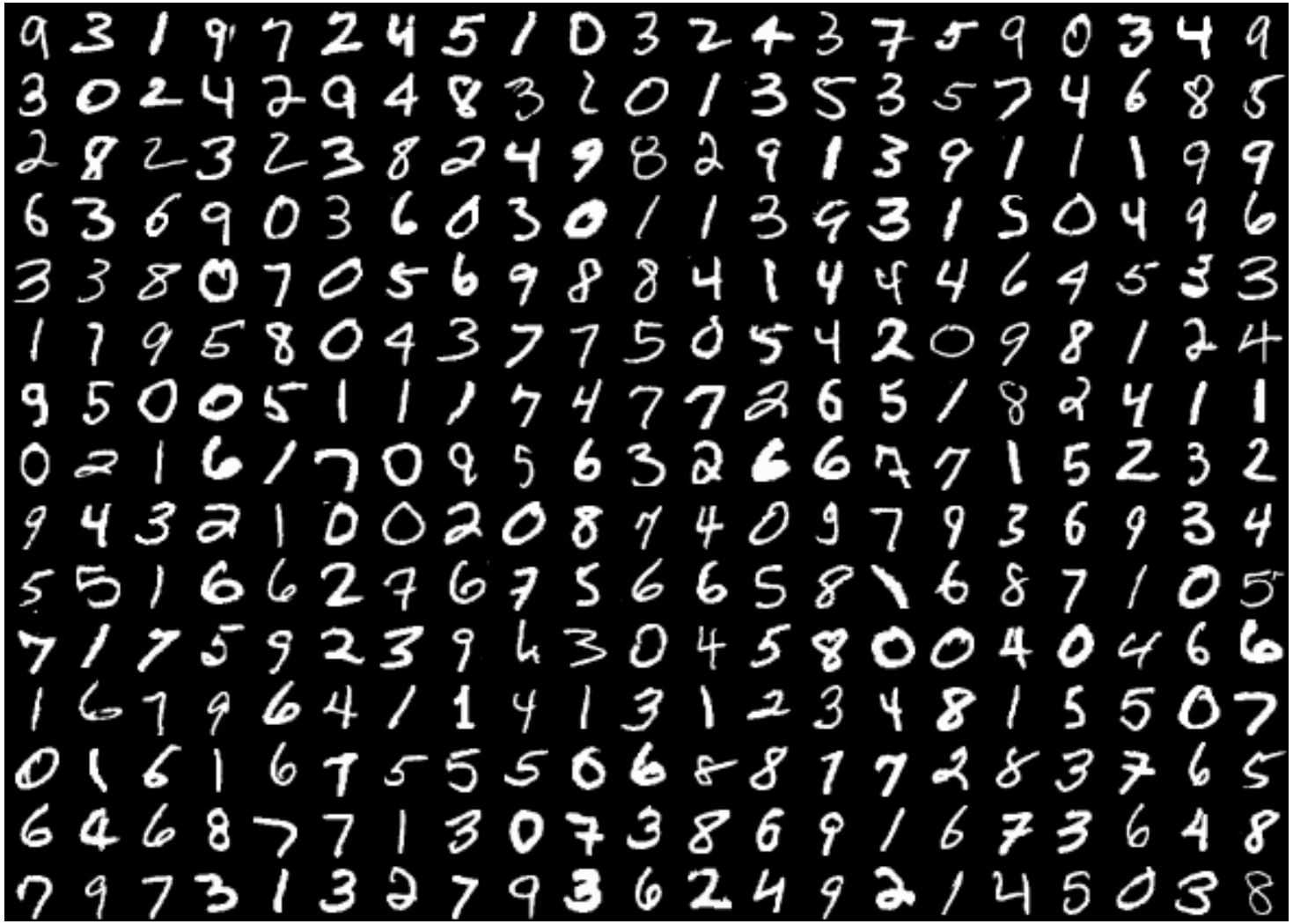
Historical problems

- Early work focused on linear threshold recognizers**: single-layer neural nets called perceptrons.**
- Lots of terrible work and false claims
- Minsky & Papert's discouraging news
  - Many important patterns not linearly recognizable
  - Expensive training times
  - Enormous recognition parameters
- Multilayer nonlinear networks improve the outlook

Comparison on MNIST

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





MNIST: handwritten digits

Comparison on MNIST

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

Idea #1

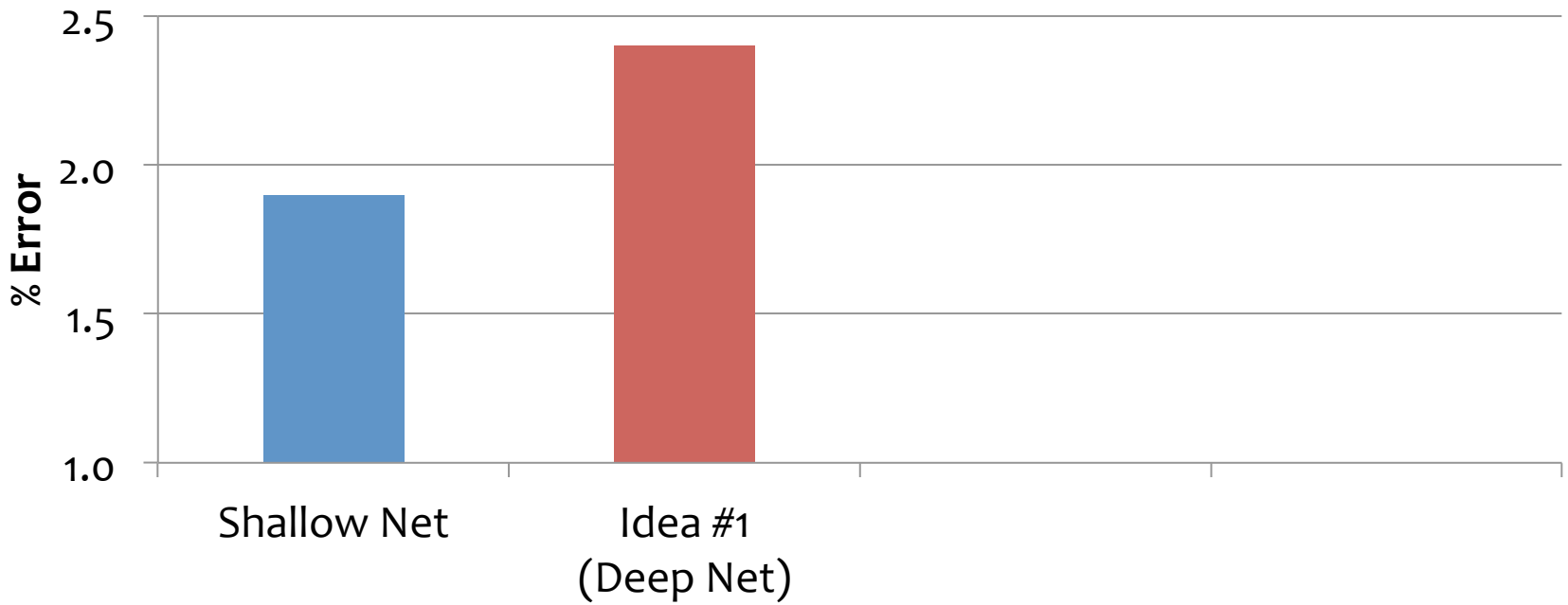
Idea #1: (Just like a shallow network)

Compute the supervised gradient by backpropagation.

Take small steps in the direction of the gradient

Comparison on MNIST

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



Idea #1: Deep Net

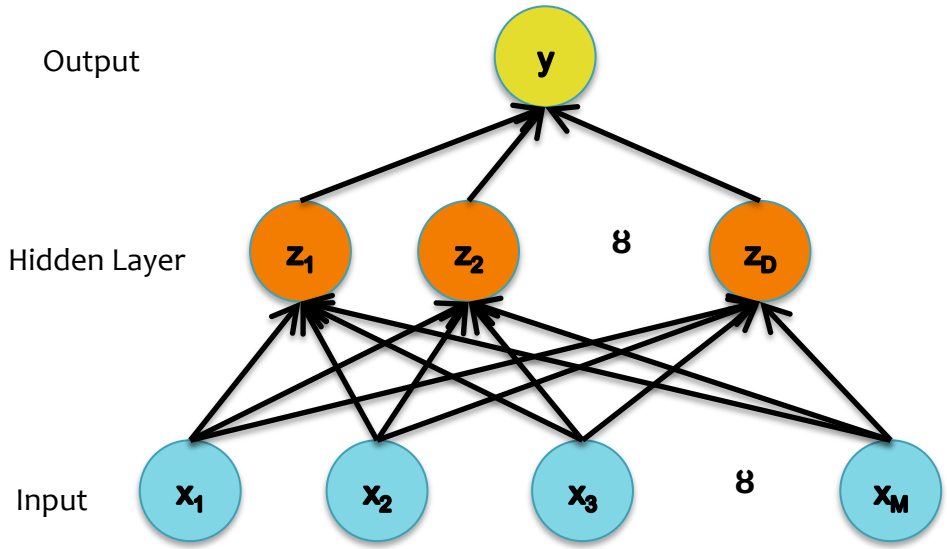
Idea #1: (Just like a shallow network)

Compute the supervised gradient by backpropagation.

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

- What goes wrong?

Neural Network with sigmoid activation functions



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

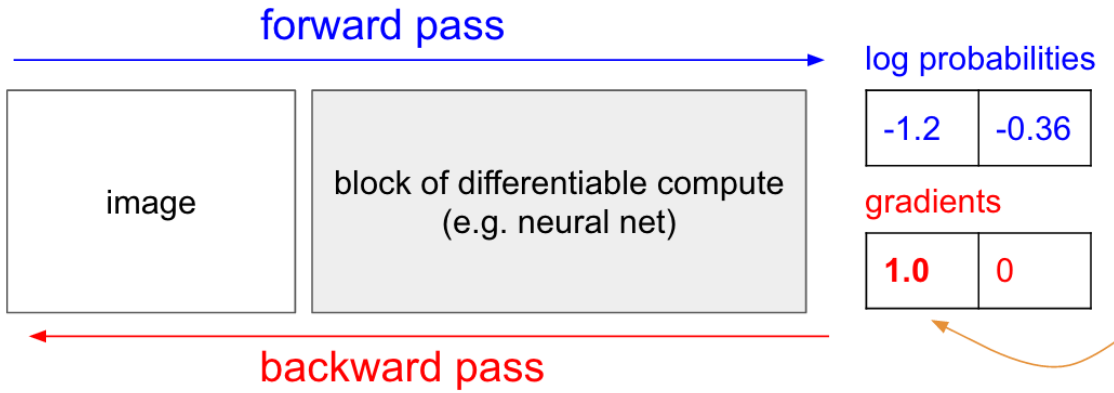
(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

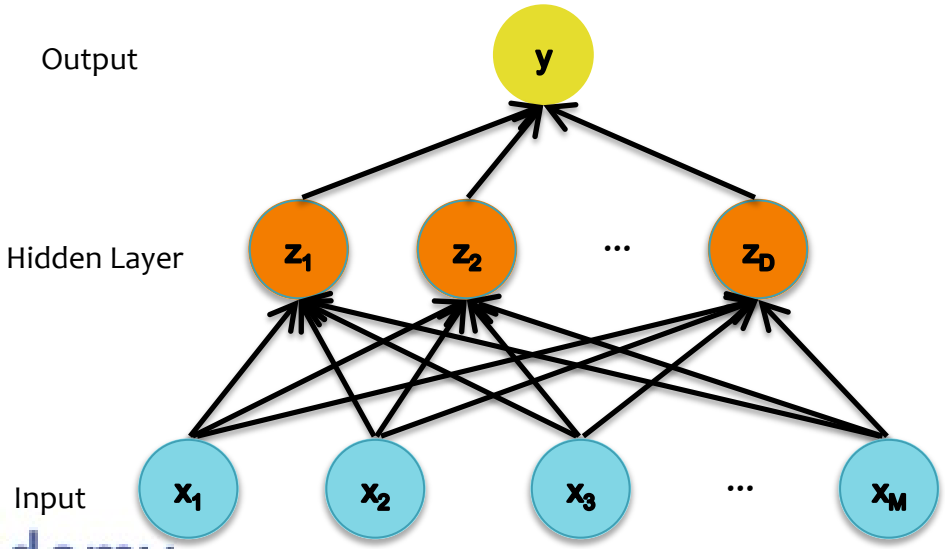


Supervised Learning
(correct label is provided)

correct action
label = 0

Back to Basics: Backpropagation

Adjust the weights to reduce the error:



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

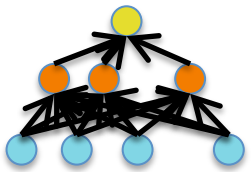
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

Backpropogation

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

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

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

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

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

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

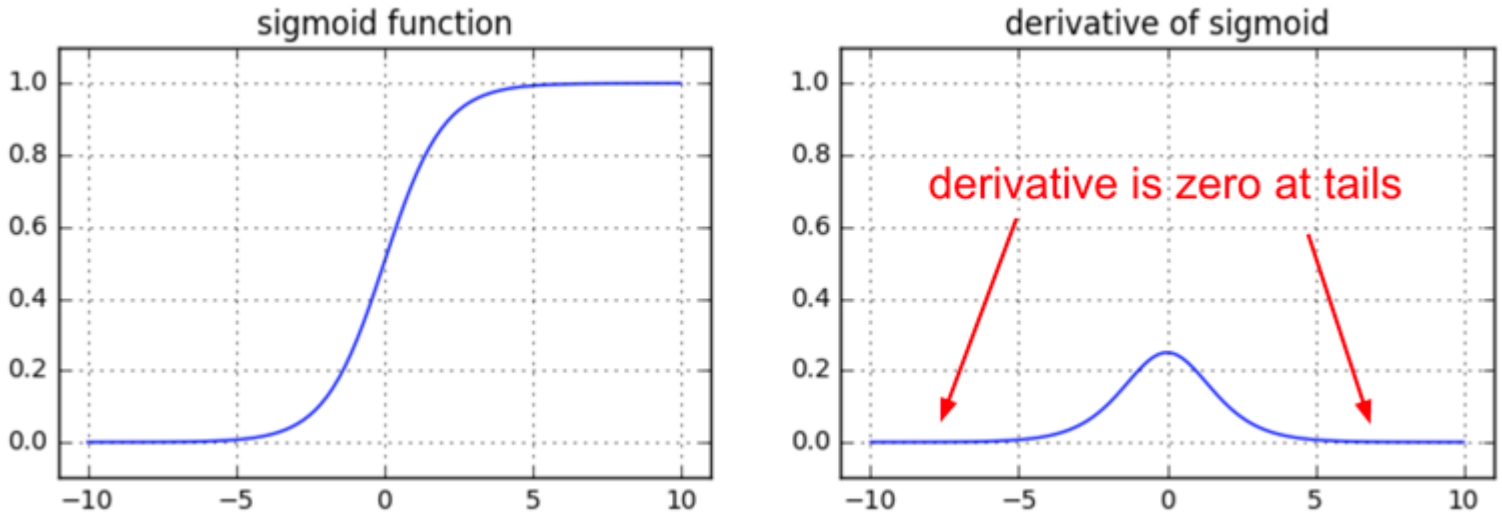
$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

Backpropogation

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

Optimization is Hard: Vanishing Gradients



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

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

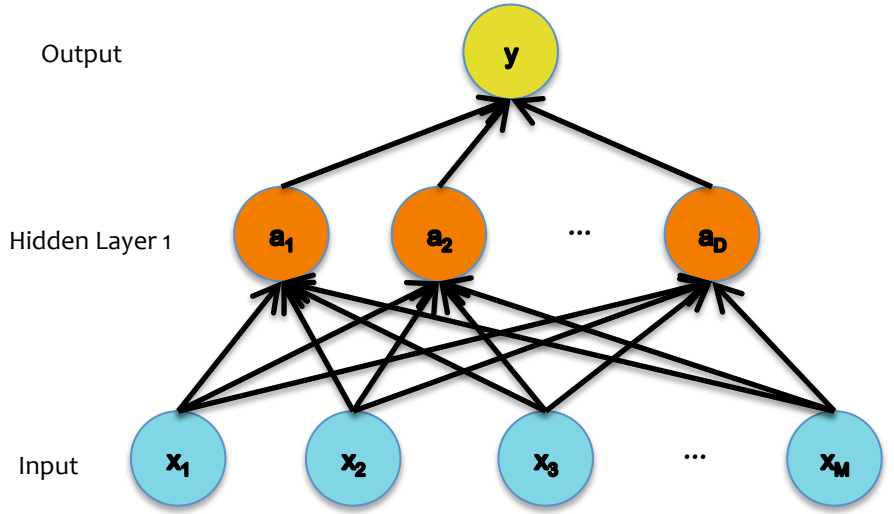
Training

Idea #2: Supervised Pre-training

Idea #2: (Two Steps)

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

Then use our **original idea**



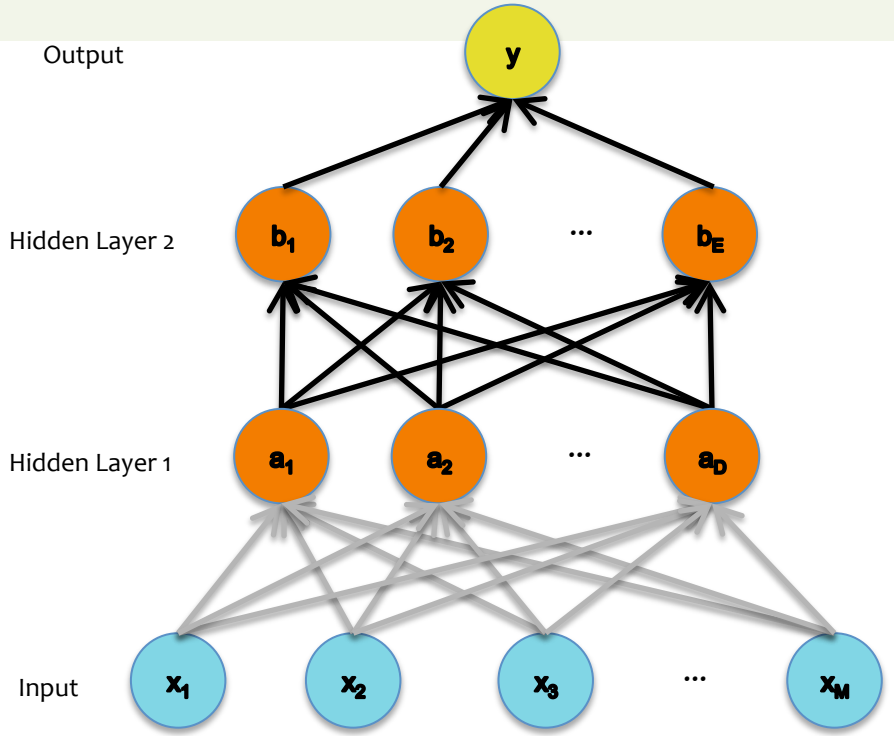
Idea #2: Supervised Pre-training

Training

Idea #2: (Two Steps)

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

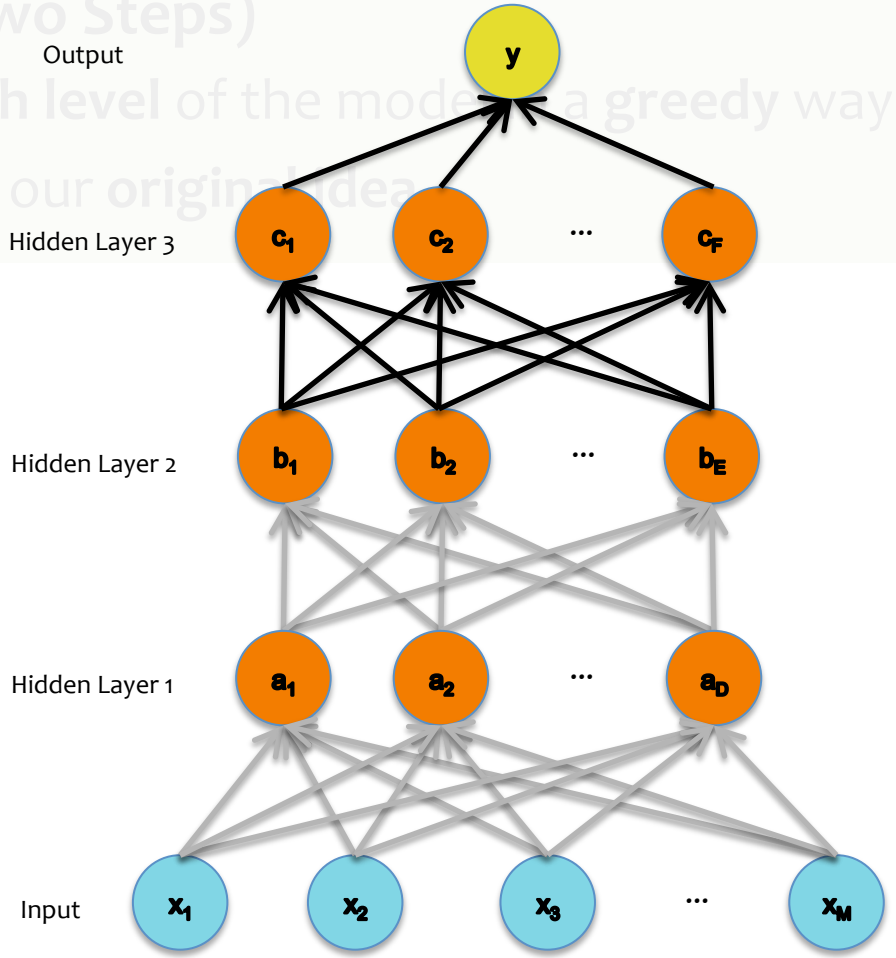
Then use our **original idea**



Training

Idea #2: Supervised Pre-training

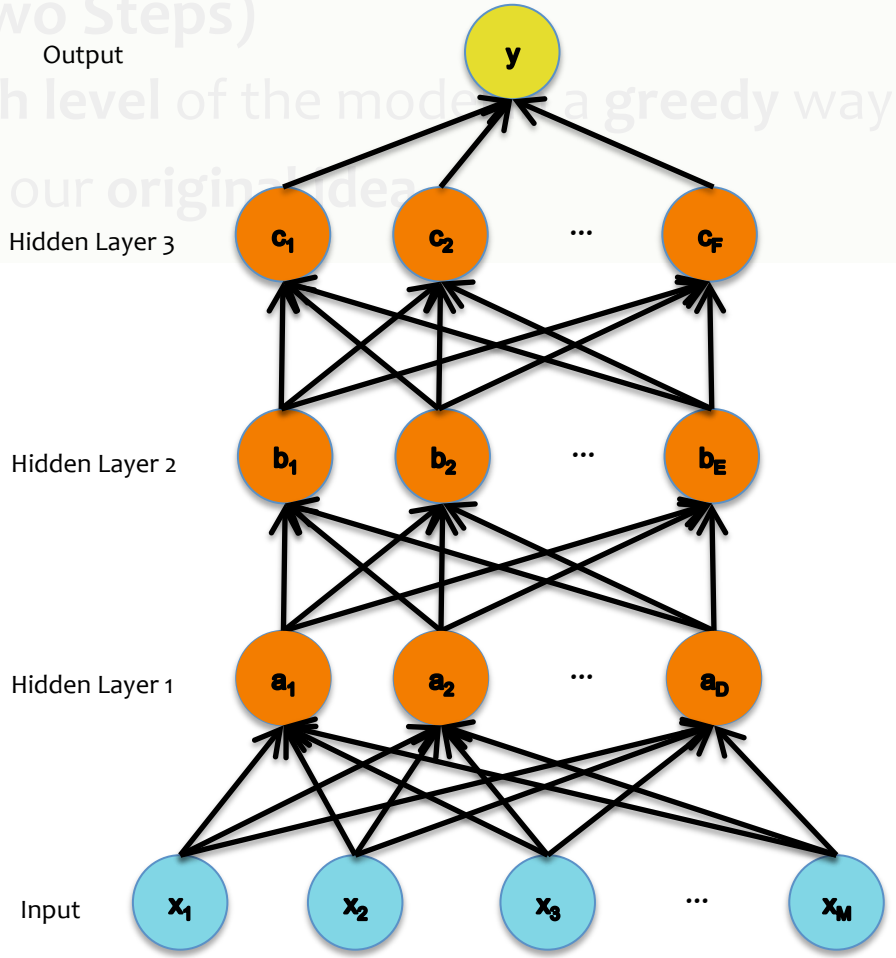
- Idea #2: (Two Steps)
  - Train each level of the model in a greedy way
  - Then use our original idea



Training

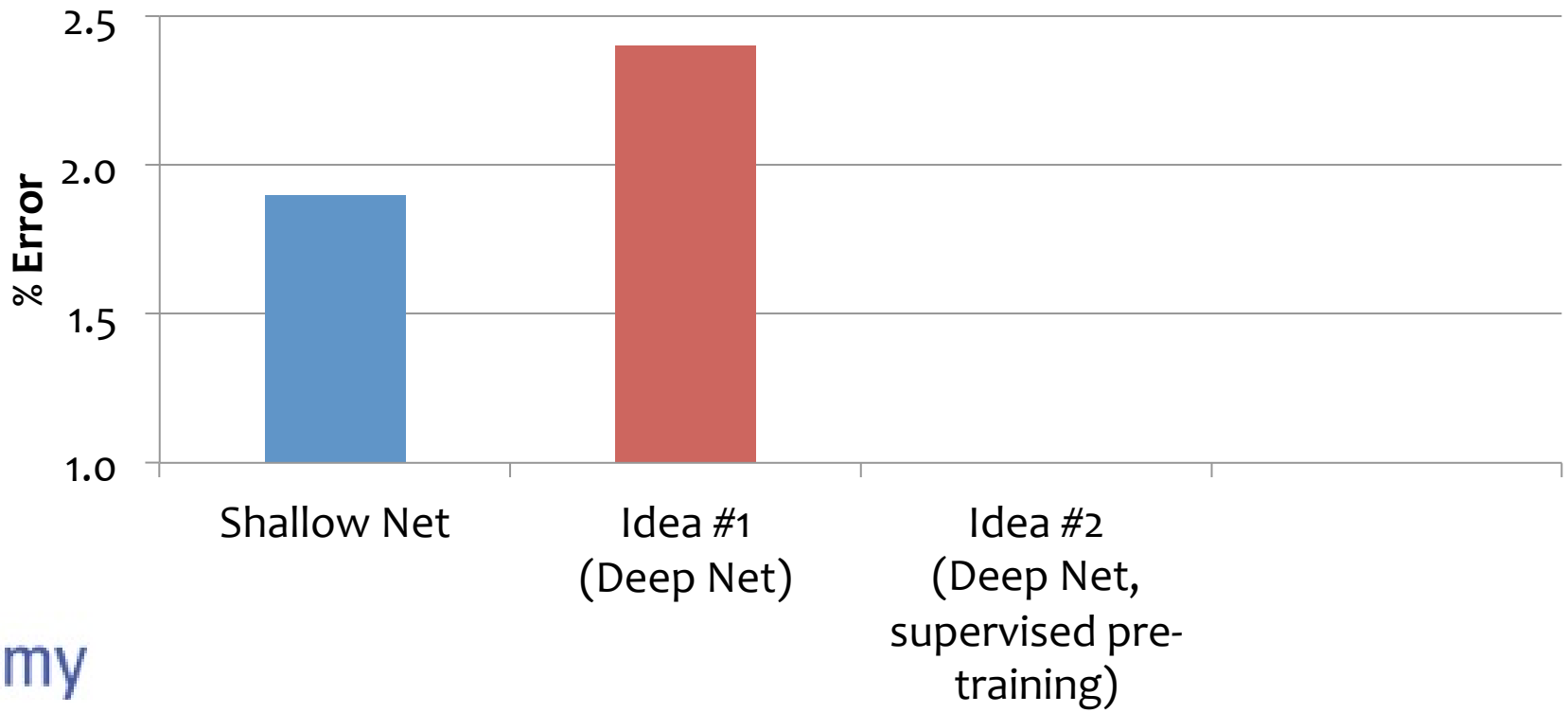
Idea #2: Supervised Pre-training

- Idea #2: (Two Steps)
  - Train each level of the model in a greedy way
  - Then use our original idea



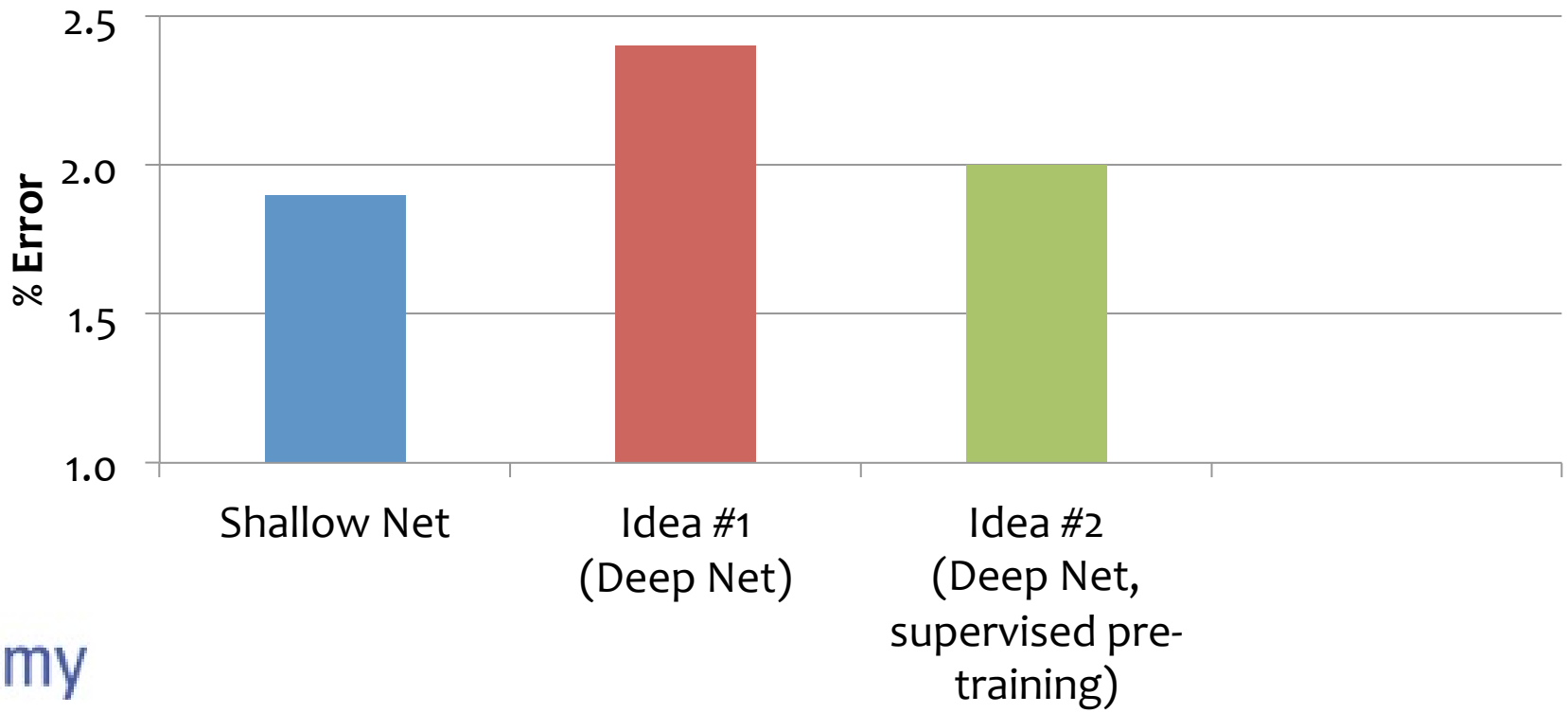
Comparison on MNIST

- 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)



Idea #3: Unsupervised Pre-training

Idea #3: (Two Steps)

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

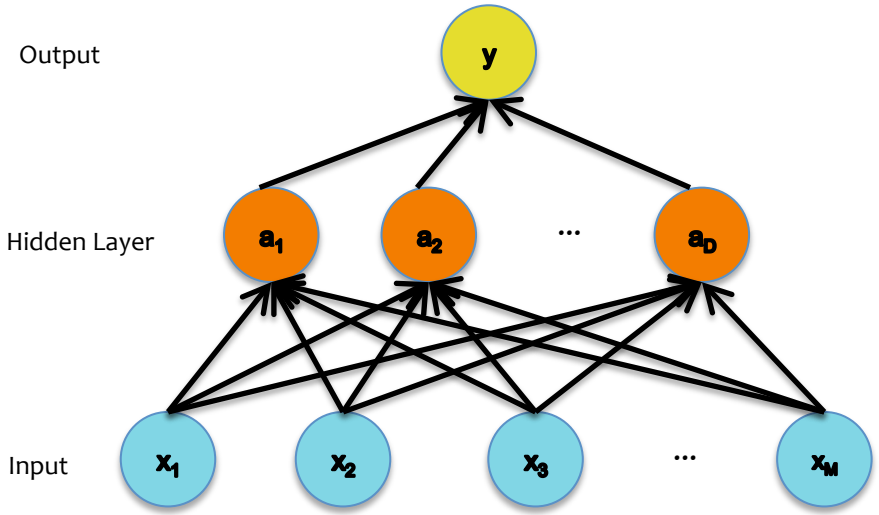
Train each level of the model in a **greedy** way

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!**

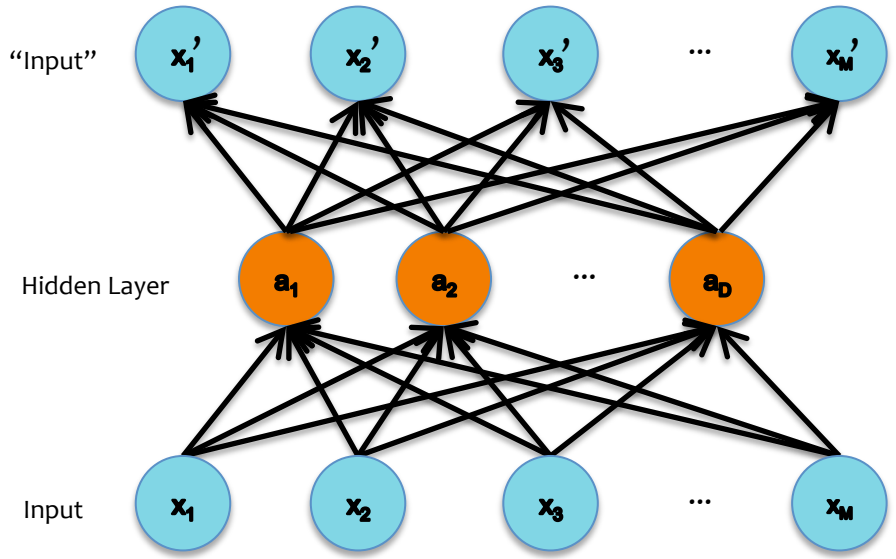


The solution: *Unsupervised pre-training*

Unsupervised pre-- training of the first layer:

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

This topology defines an Auto-- encoder.



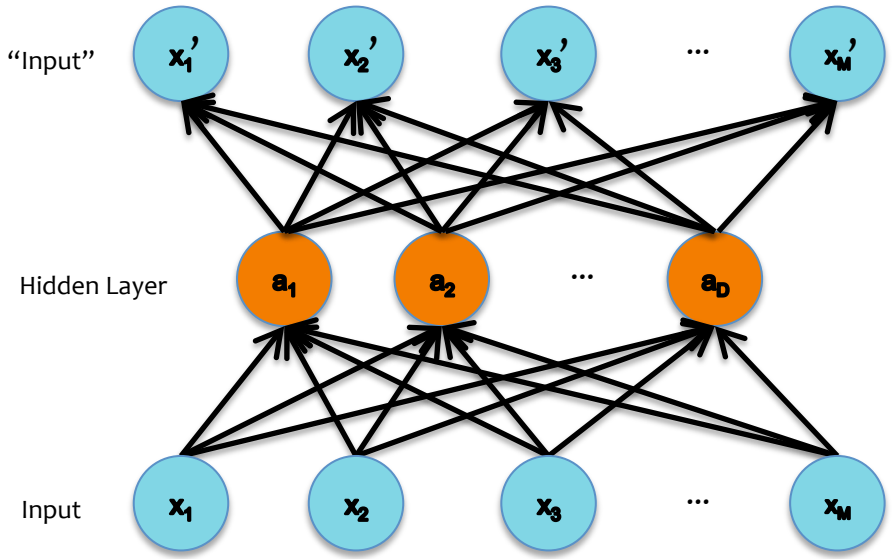
Auto--Encoders

Key idea: Encourage z to give small reconstruction error:

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

DECODER: $x' = h(W'z)$

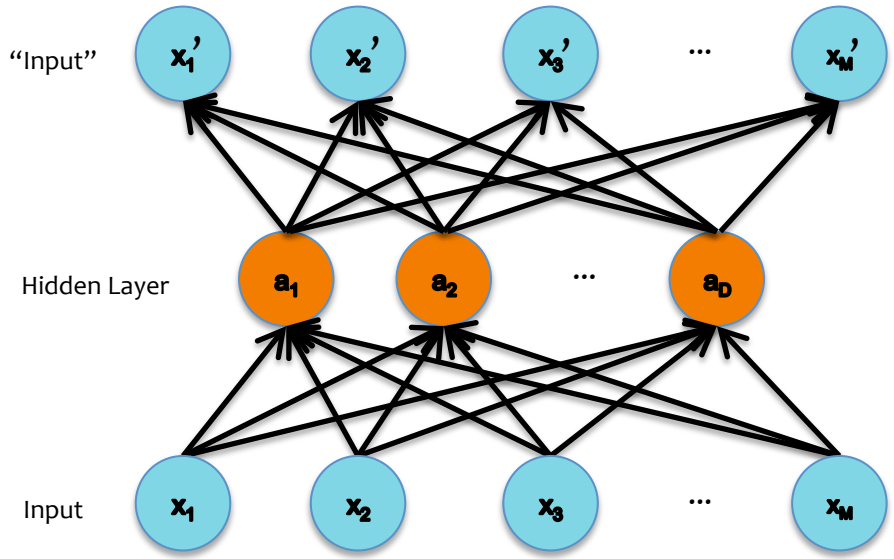
ENCODER: $z = h(Wx)$



The solution: *Unsupervised pre- training*

Unsupervised pre-- training

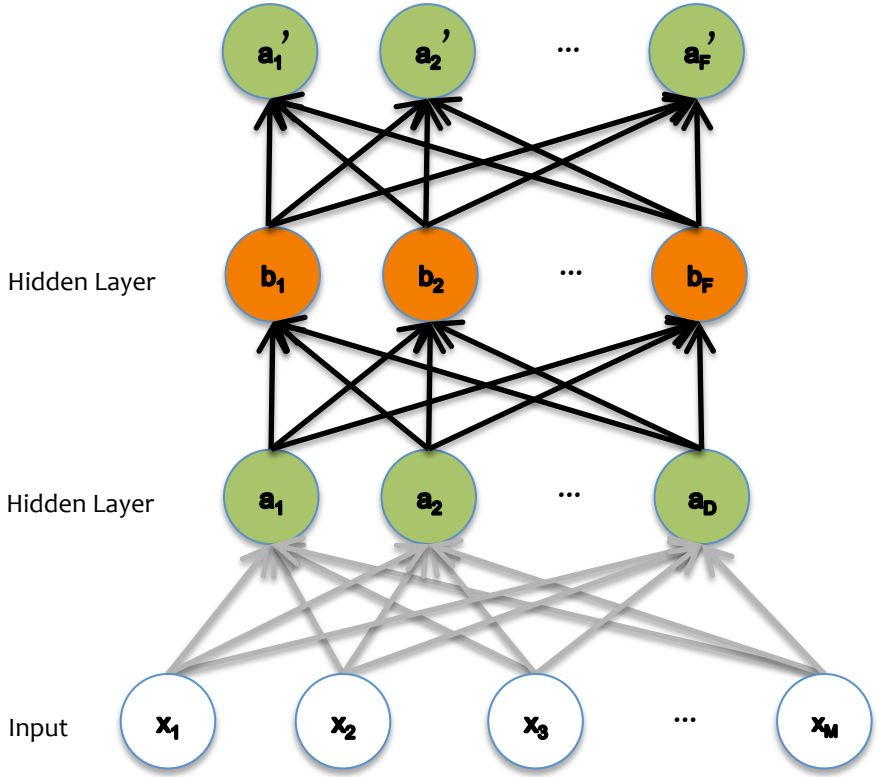
- Work bottom-- up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution: *Unsupervised pre- training*

Unsupervised pre-- training

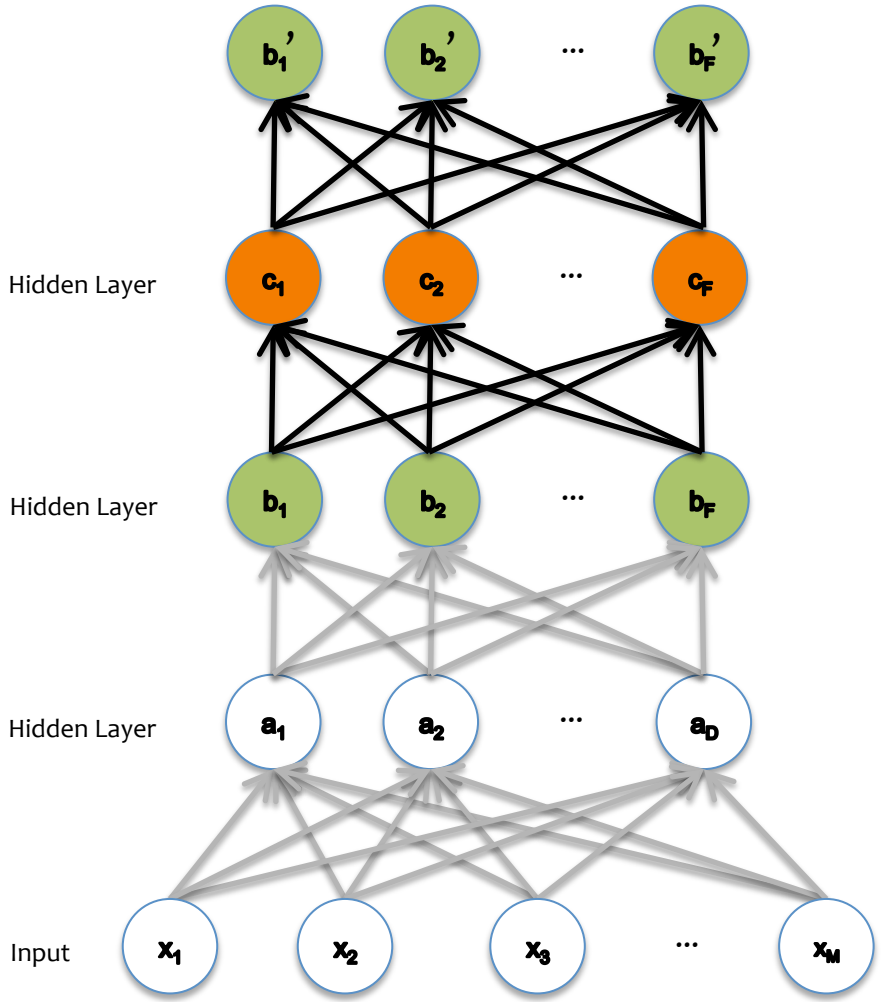
- Work bottom-- up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



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.

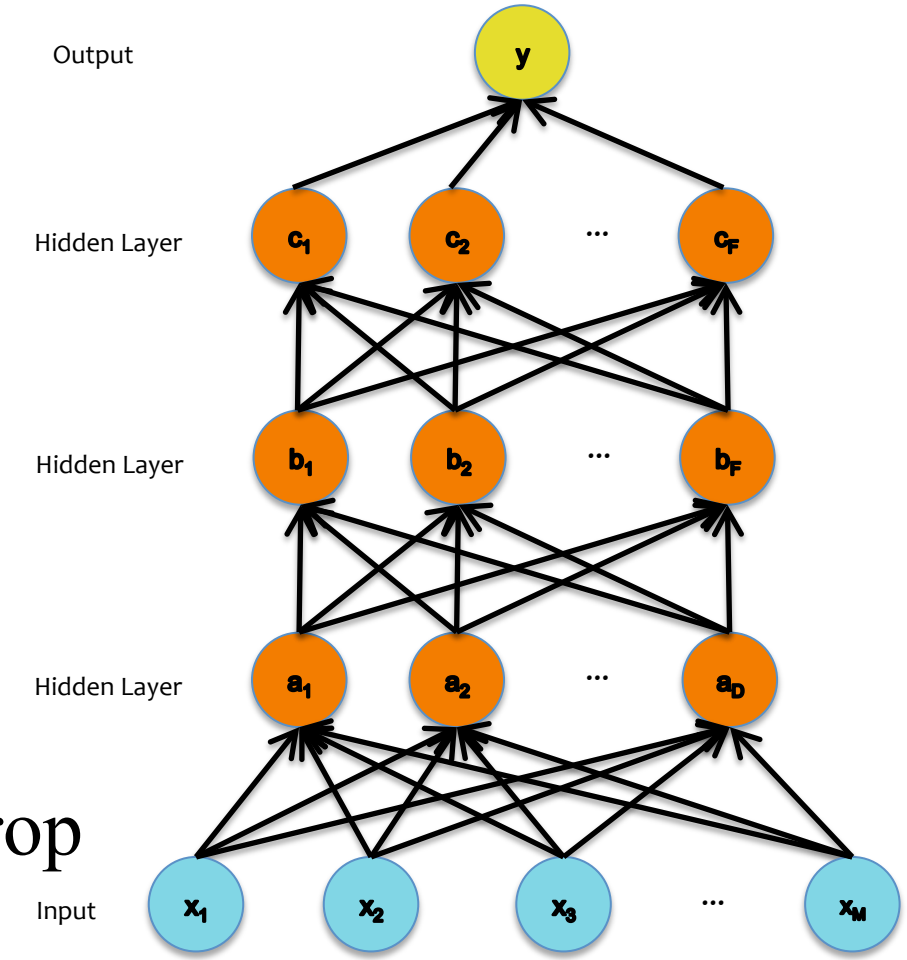


The solution: *Unsupervised pre-training*

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:

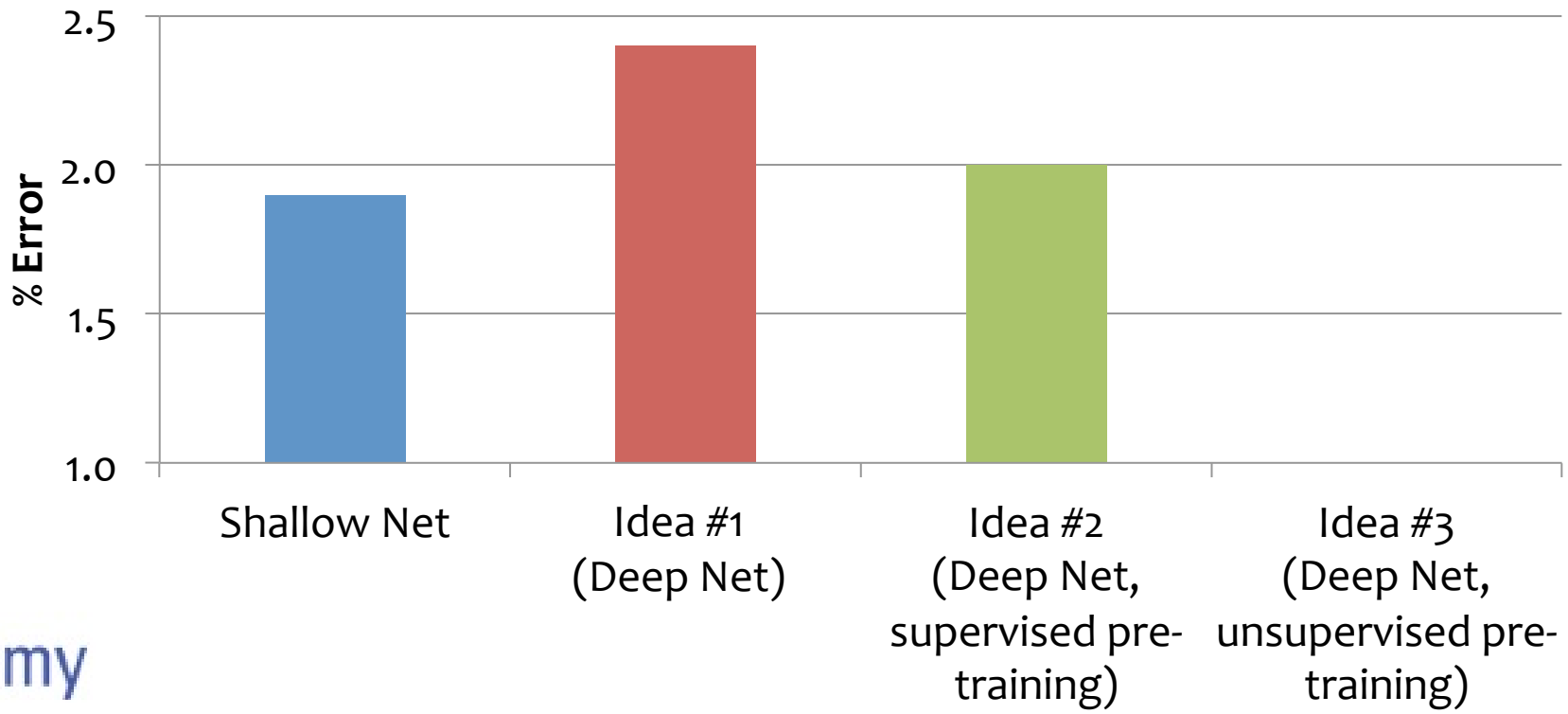
1. Supervised layer-- wise pre-- training
2. Supervised fine-- tuning

Idea #3:

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

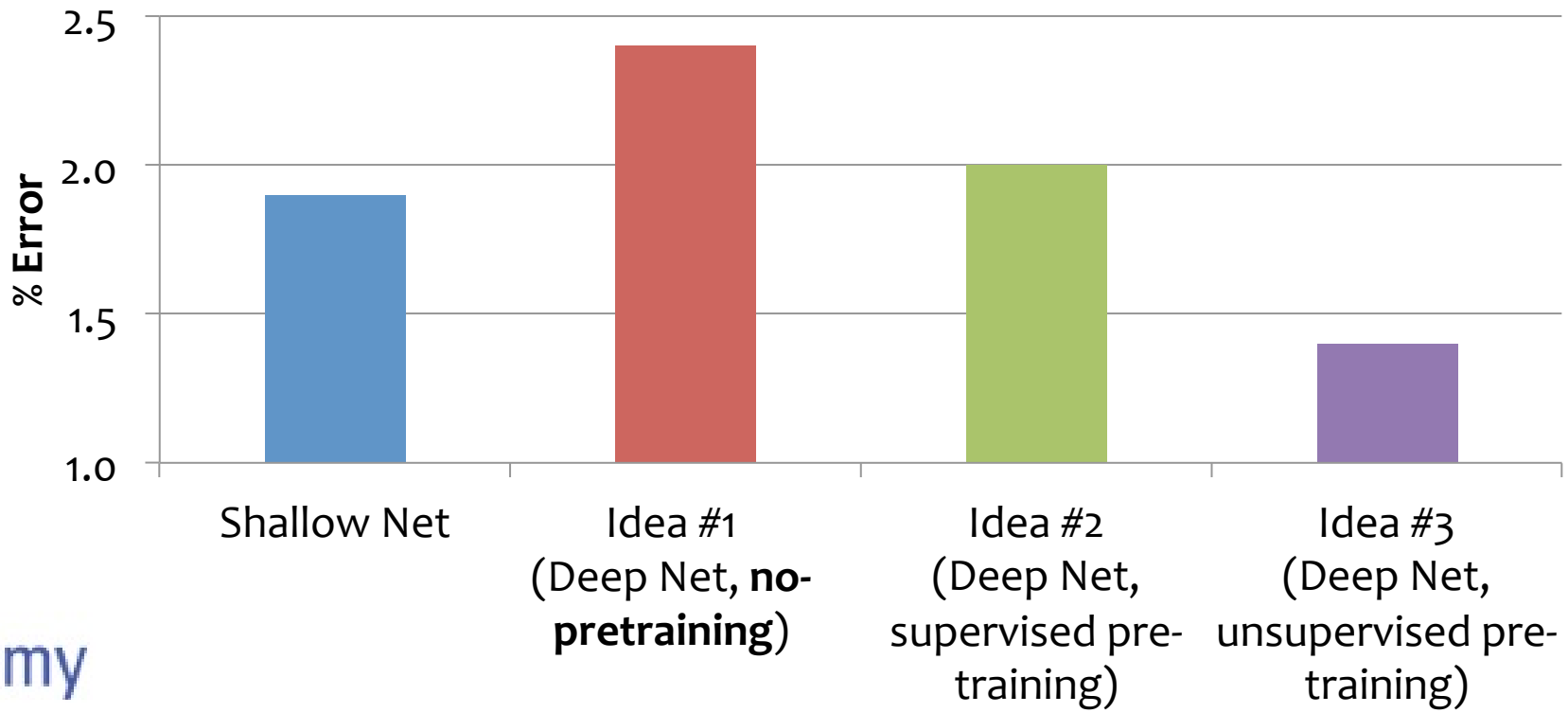
Comparison on MNIST

- 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)

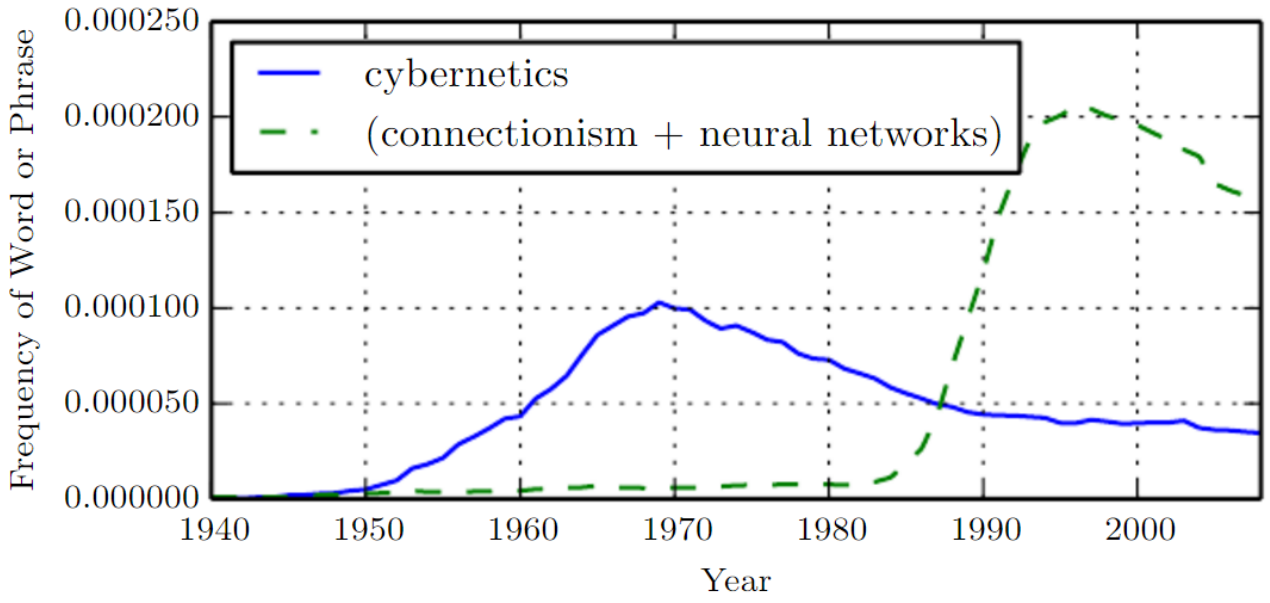


Deep Learning

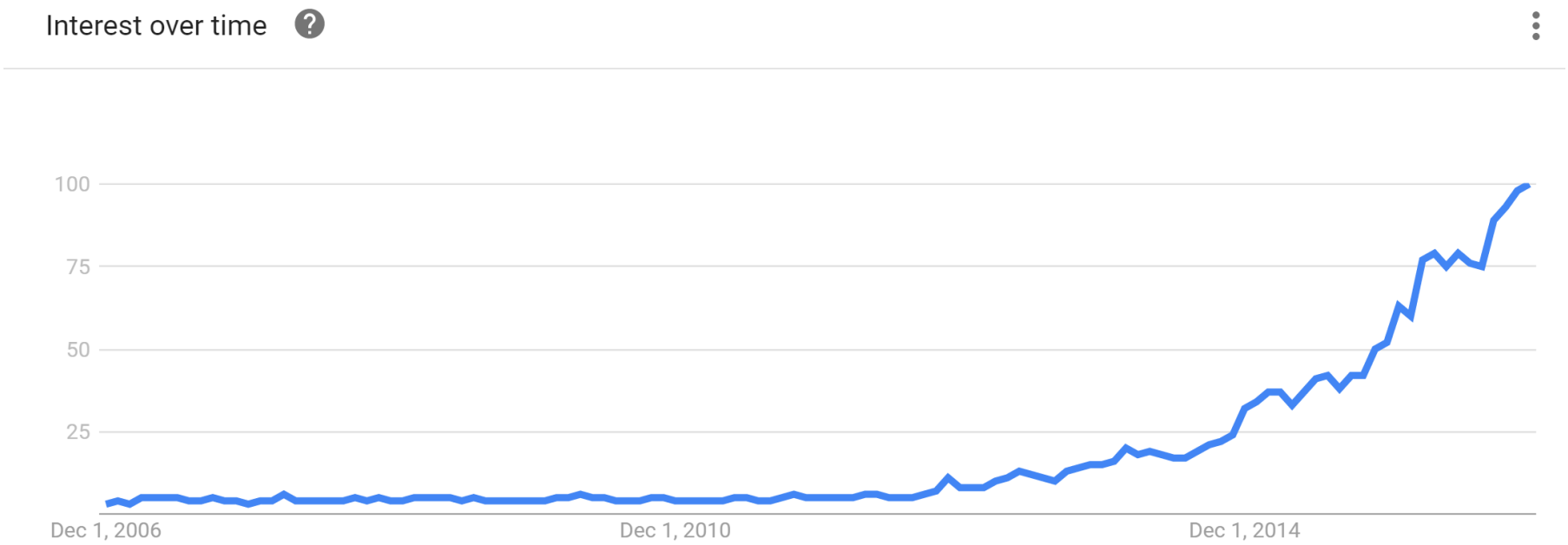
- Goal: learn features at different levels of abstraction
- Training can be tricky due to…
  - Local Optimal
  - Vanishing gradients
- Unsupervised layer-wise pre-training can help with both!

The Seasons of Deep Learning

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

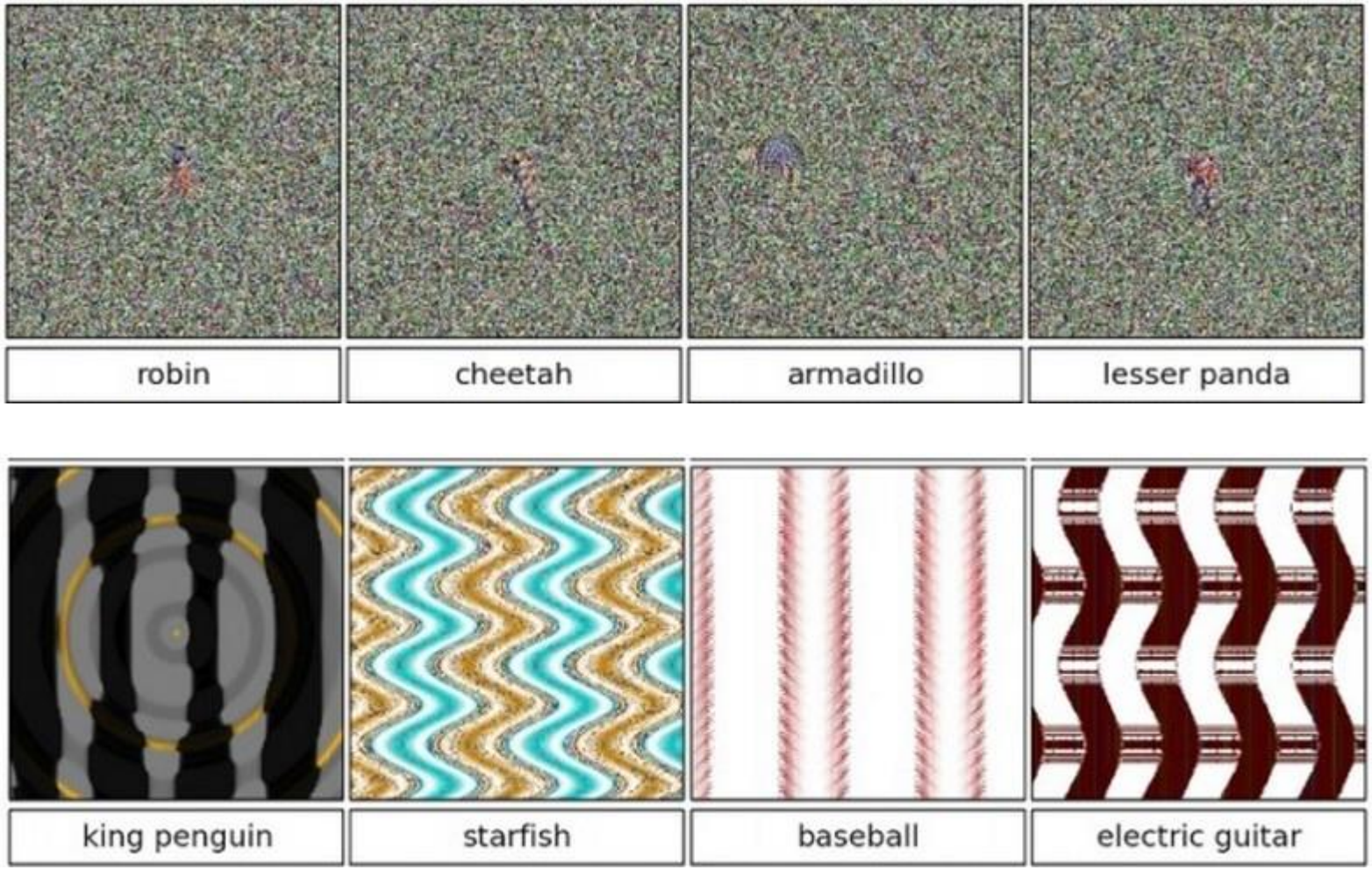


3rd Summer of Deep Learning

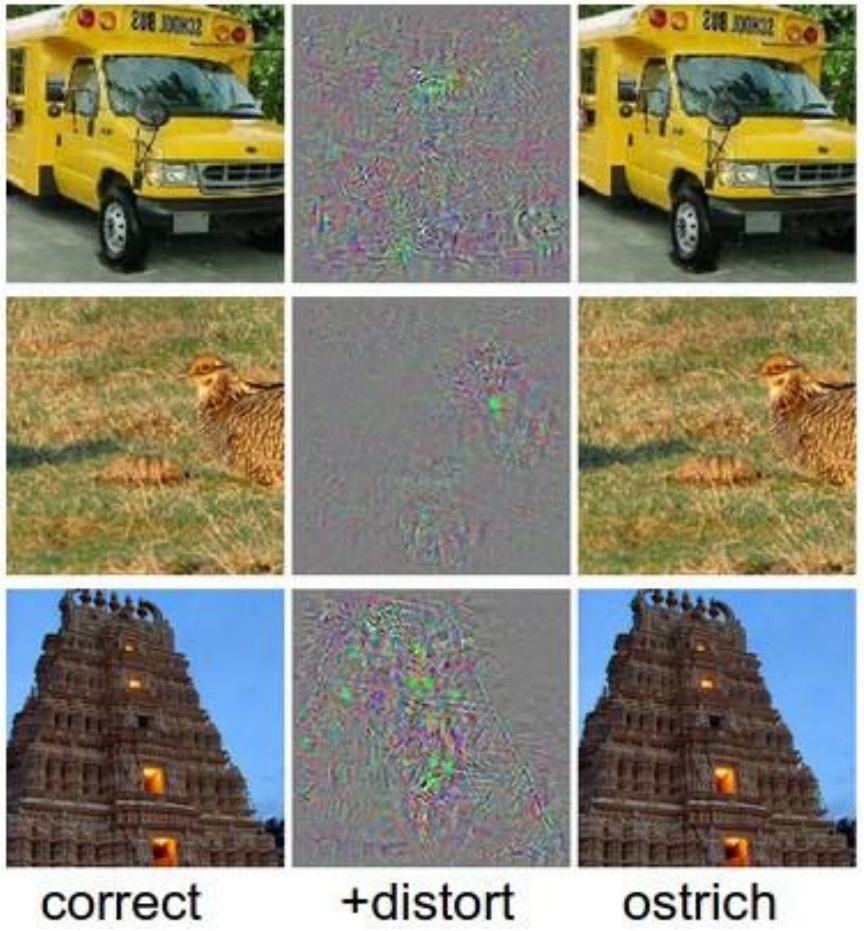


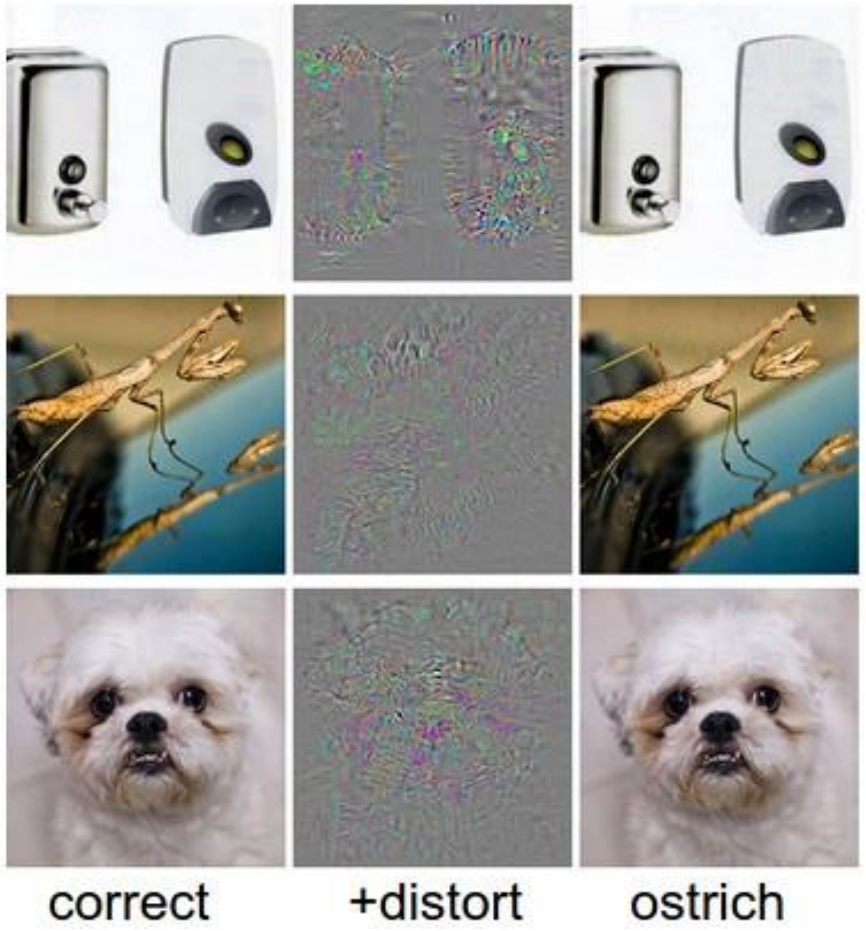
Google trends: “Deep Learning”

Robustness:
>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

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

Université de Montréal | Montreal Institute for Learning Algorithms

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 …

amazon

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)

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

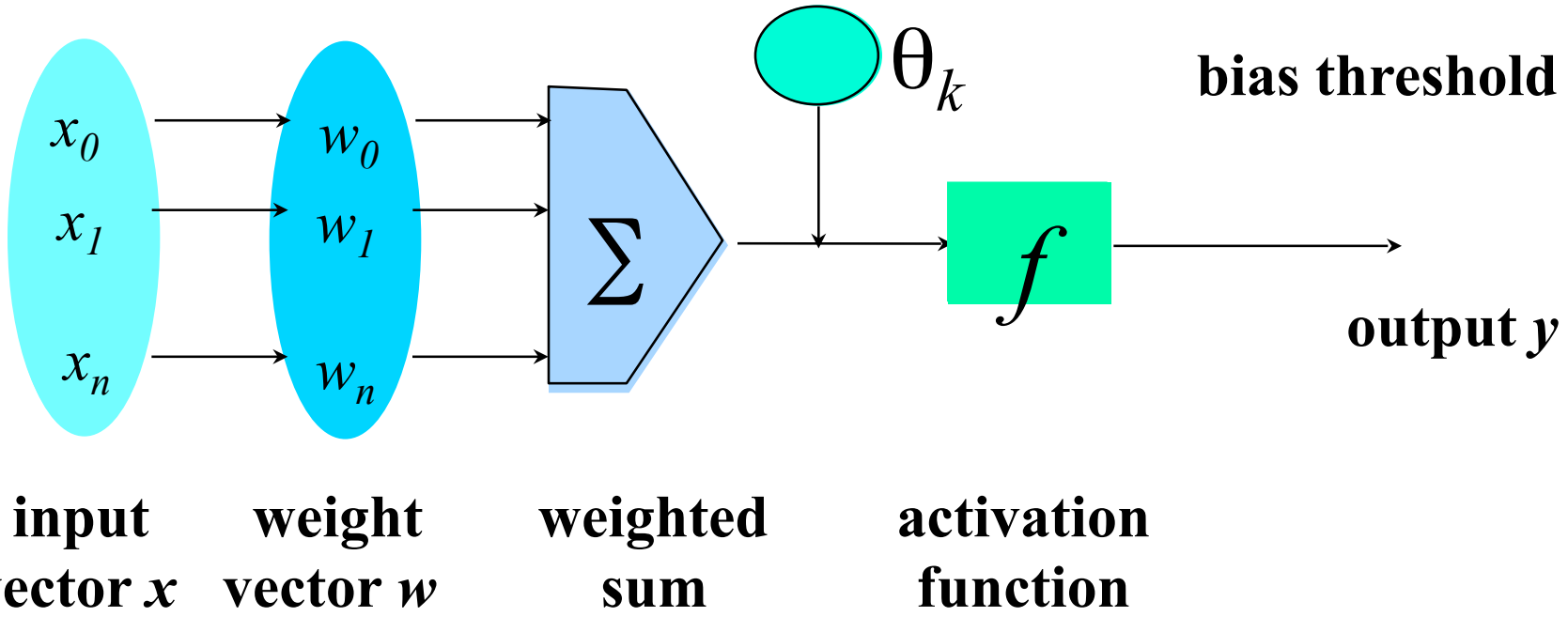
In the Browser

- Keras.js
  - GPU Support
  - Full sized networks
  - Can use trained Keras models

- ConvNetJS
  - Built by a 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

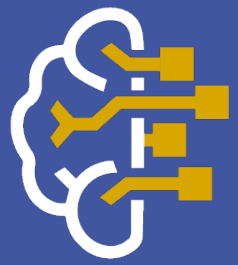
- Network topology selection
  - Number of hidden layers Number
  - of nodes in hidden layers
  - Interconnections of nodes
- Network training
  - Find weights that yield the best classification given topology
- Network topology editing
  - Find topology that yields best classification
  - Find topology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