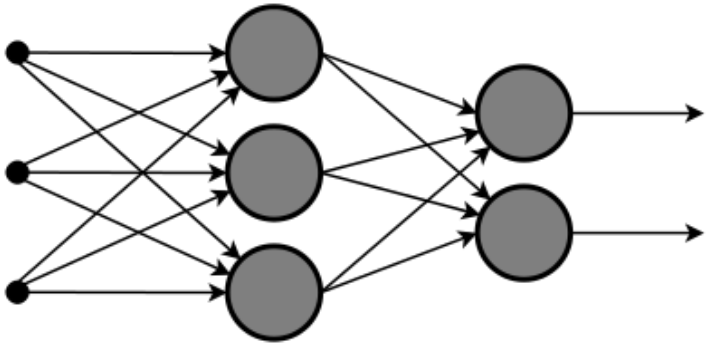
NORTH CAROLINA STATE UNIVERSITY
College of Engineering
Department of Computer Science

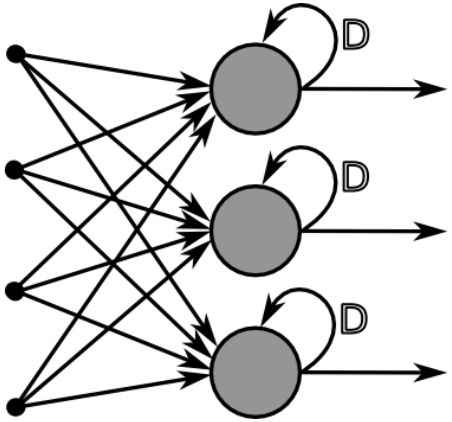
Deep Learning

Dr. Min Chi
Department of Computer Science
mchi@ncsu.edu

Combining Neurons into Layers



Feed Forward Neural Network

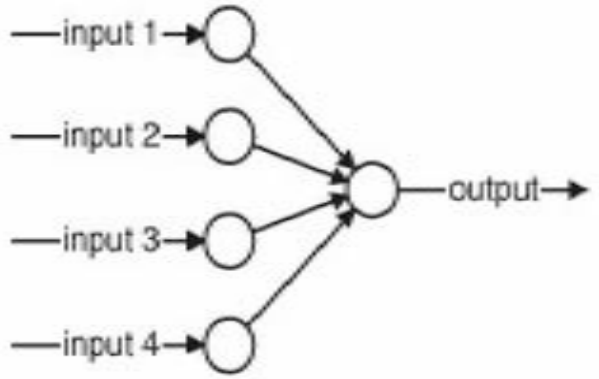


Recurrent Neural Network

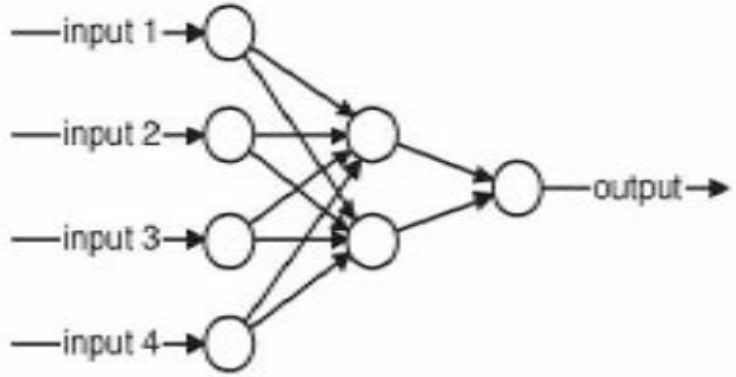
- Have state memory
- Are hard to train

Feed-Forward Neural Net Examples

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

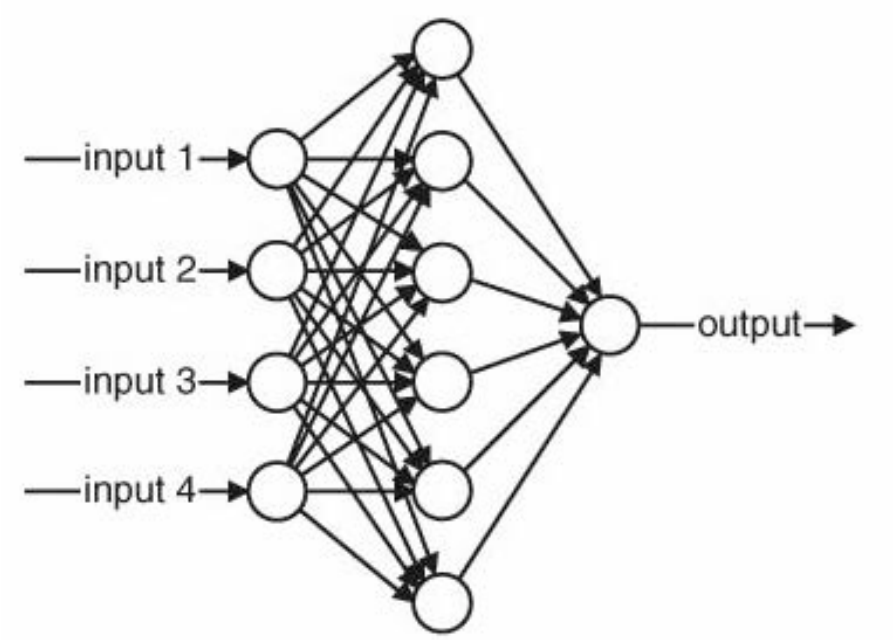


A very simple neural network that takes four inputs and produces one outoput.

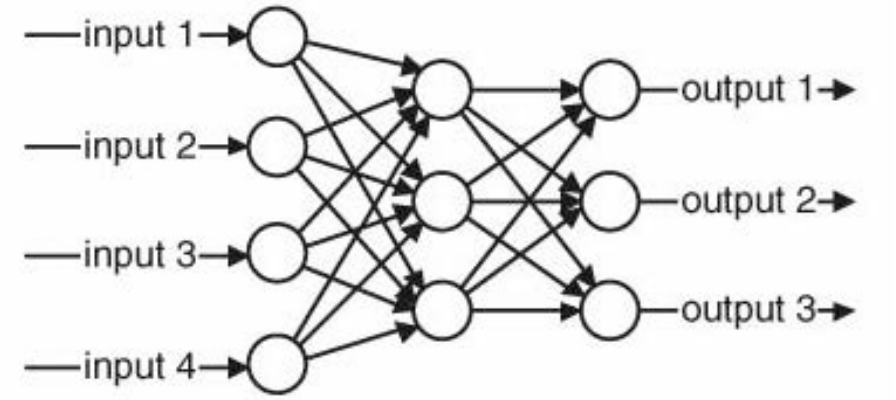


This network has a middle layer called the hidden layer, which makes the network more powerful by enabling it to recognize more patterns.

Feed-Forward Neural Net Examples

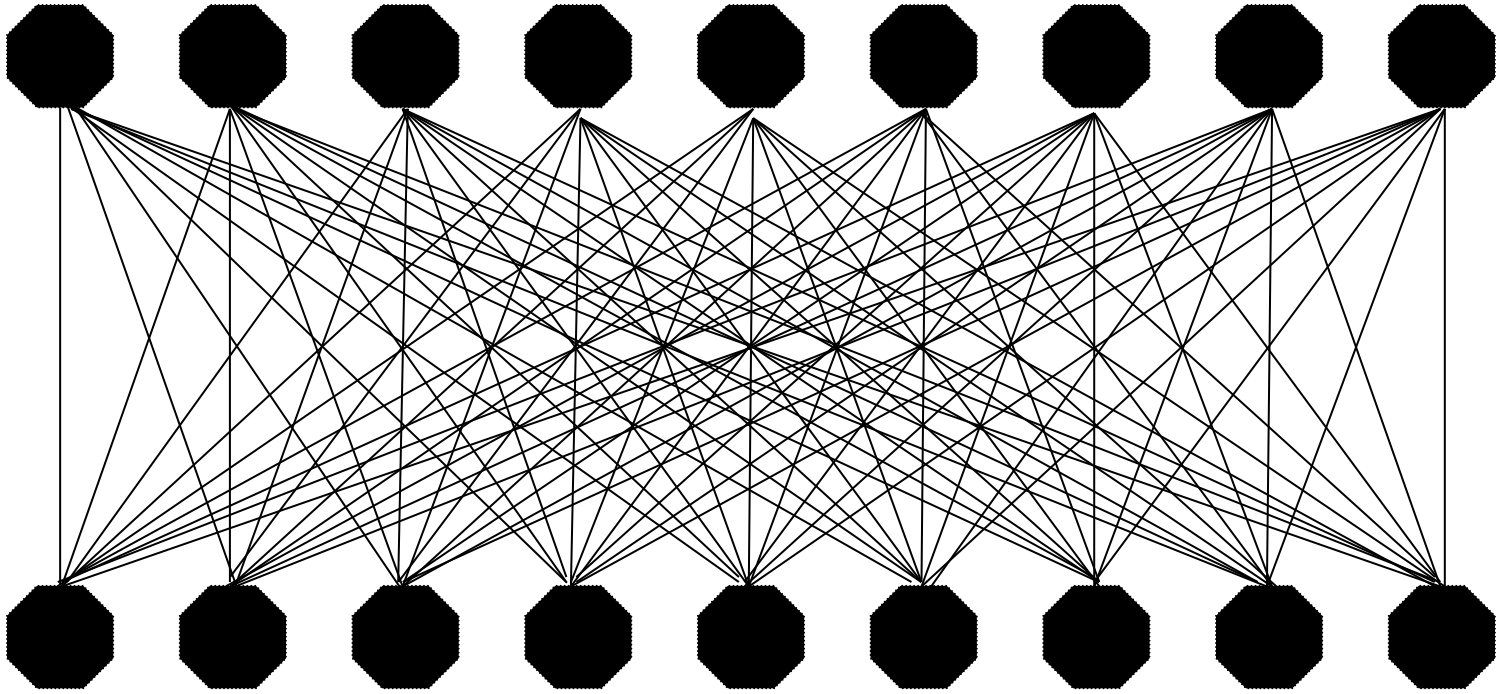


Increasing the size of the hidden layer makes the network more powerful but introduce the risk of overfitting. Usually, only one hiddent layer is needed.



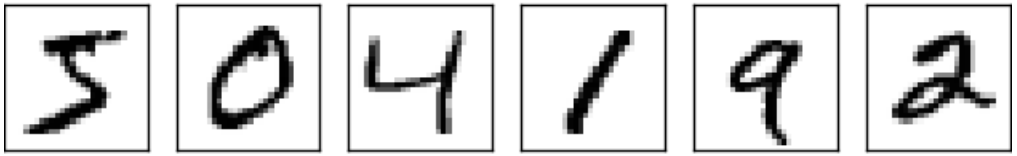
A neural network can produce multiple output values.

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

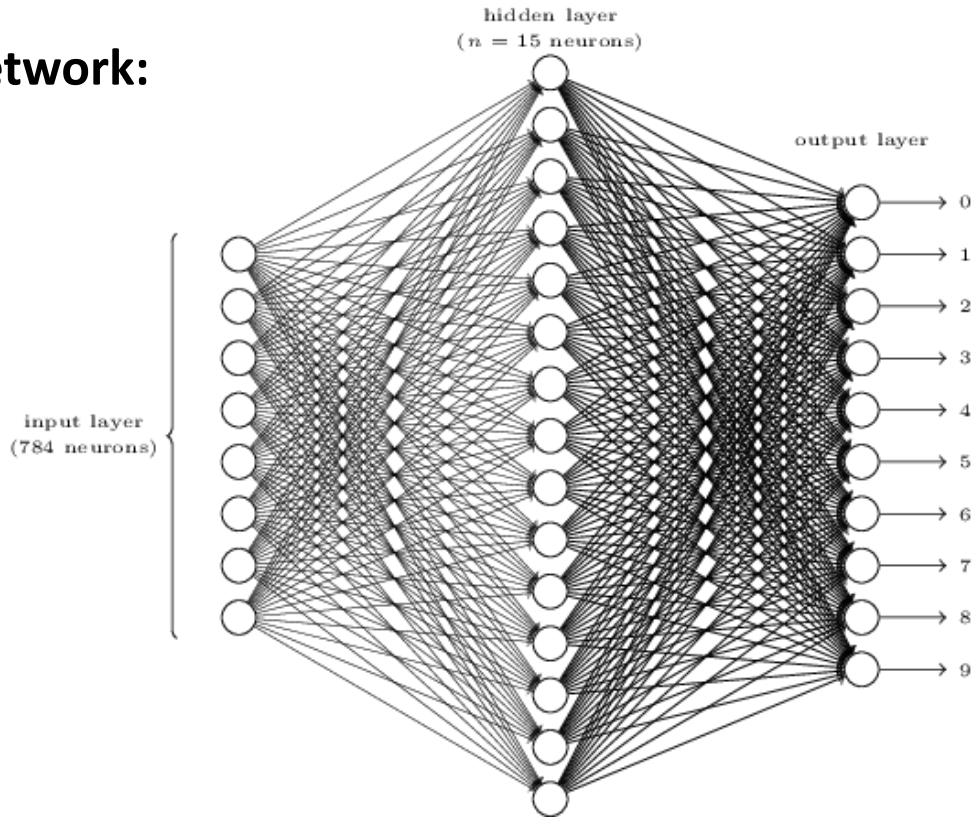


Task: Classify and Image of a Number

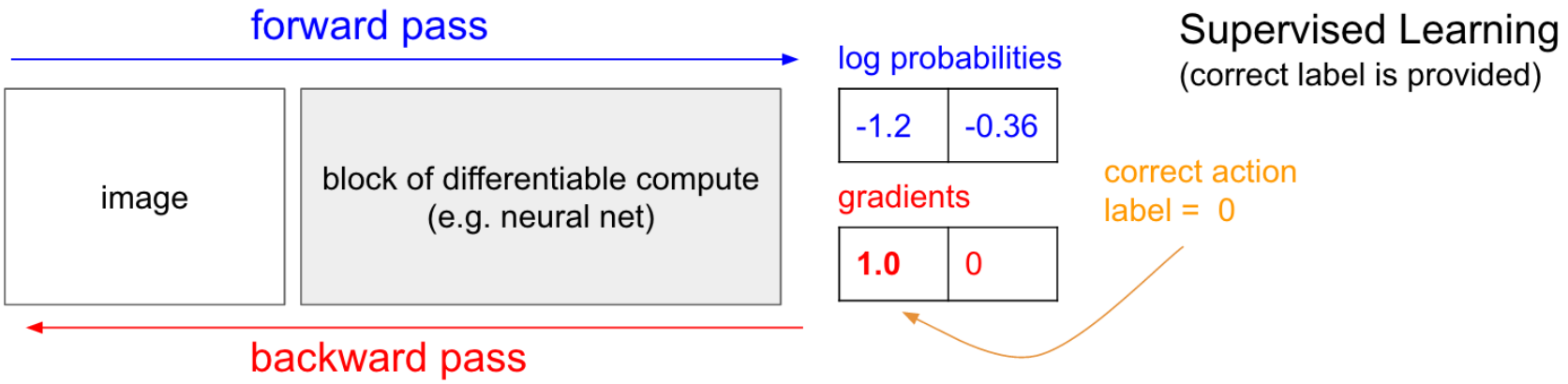
Input:
(28x28)



Network:



Task: Classify and Image of a Number

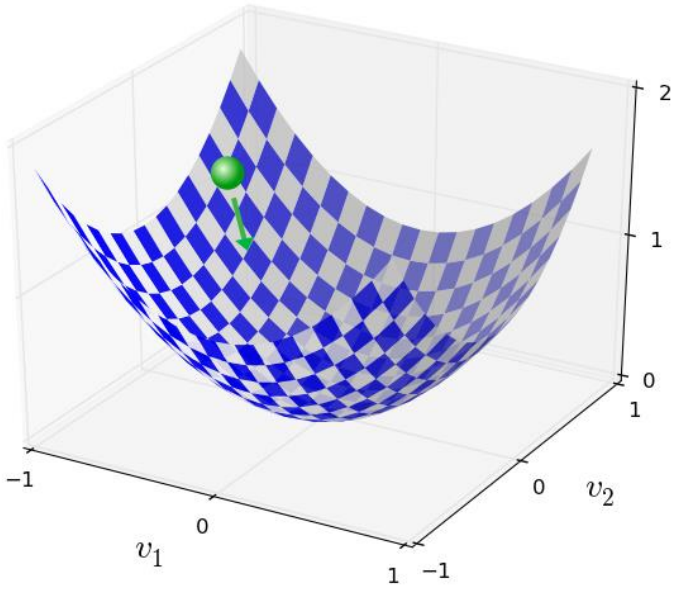


Ground truth for "6":

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

"Loss" function:

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

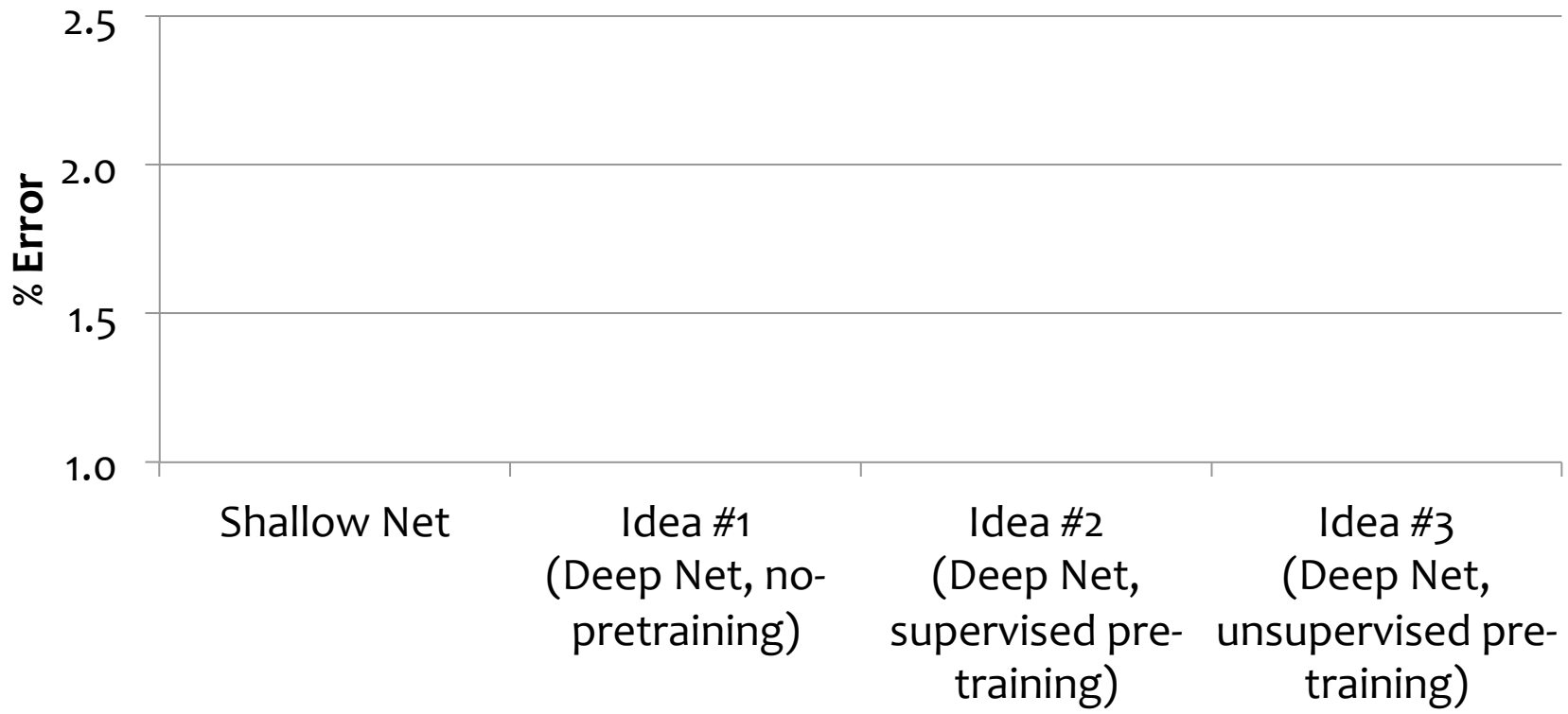


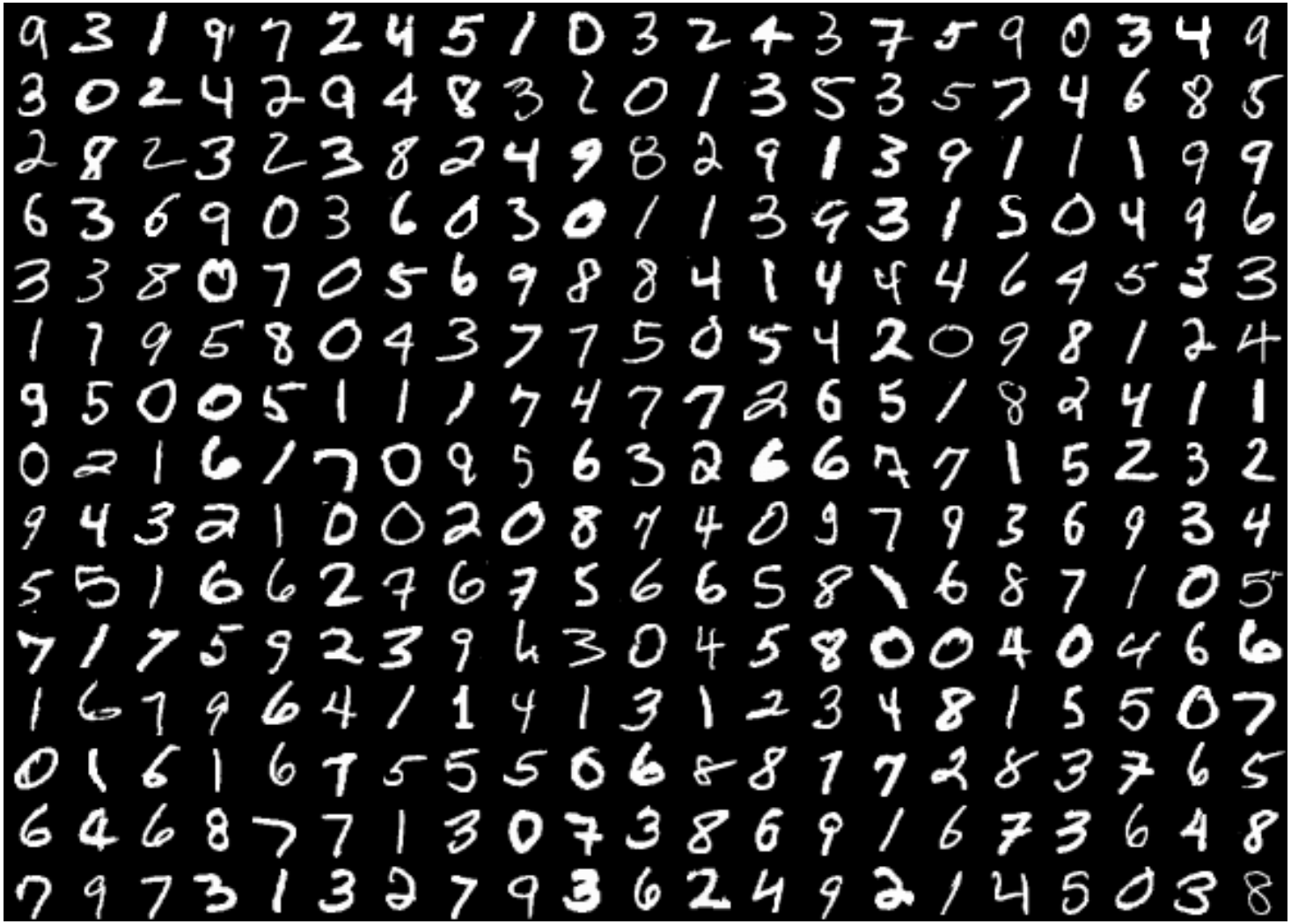
Historical problems

- 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)

Training

Idea #1

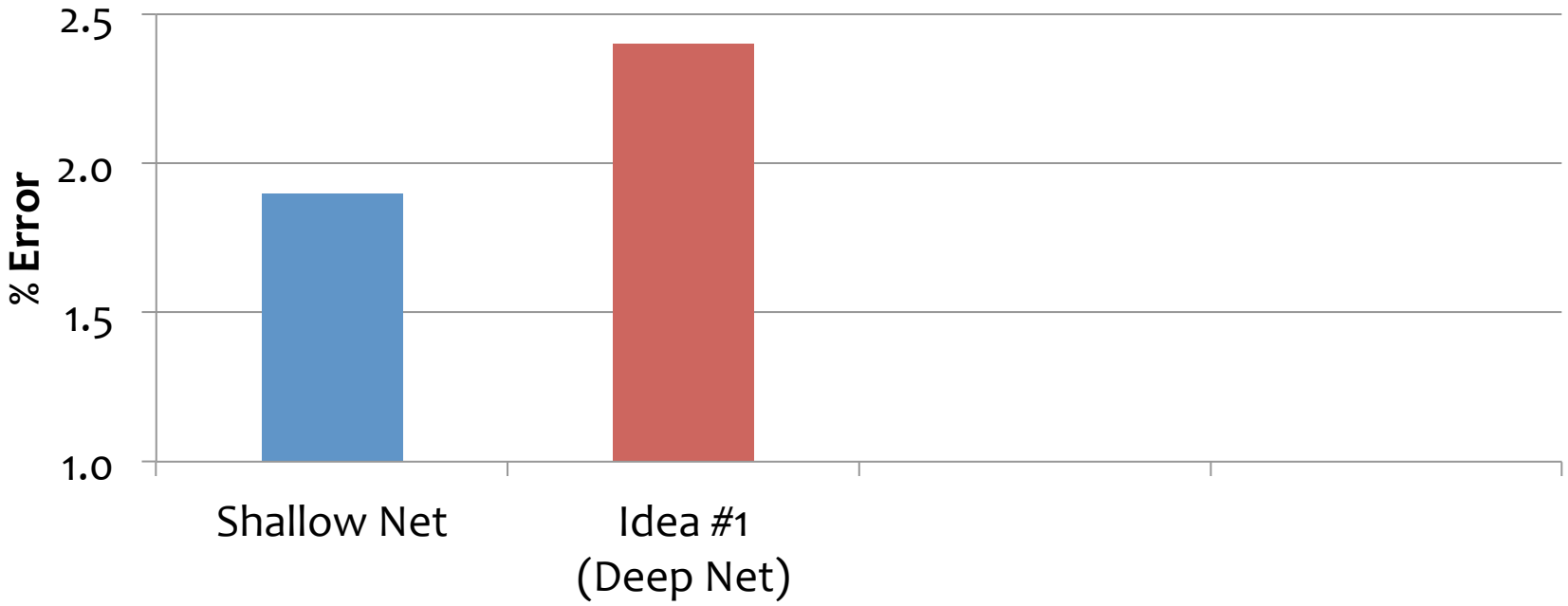
Idea #1: (Just like a shallow network)

Compute the supervised gradient by backpropagation.

Take small steps in the direction of the gradient

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.

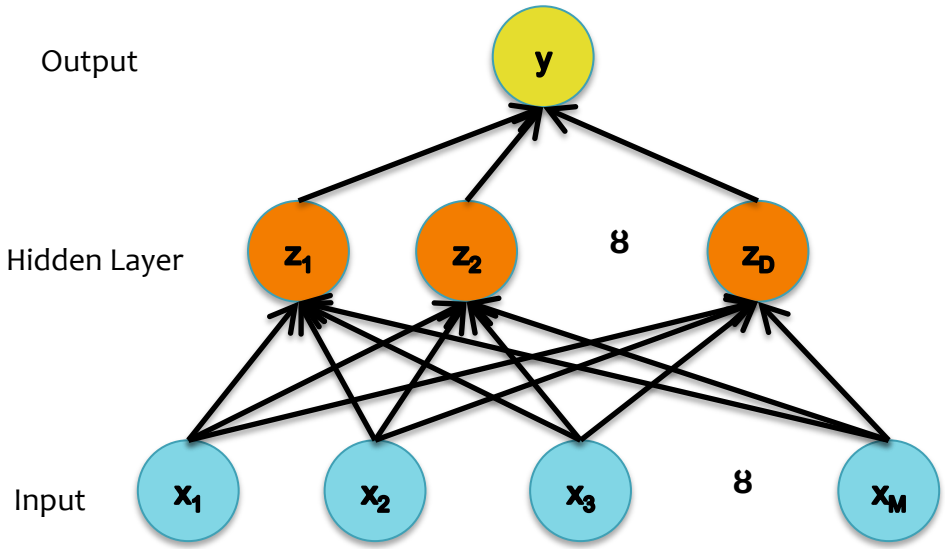
Take small steps in the direction of the gradient (SGD)

- 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



(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 $x_i, \ \forall i$

Neural Network Training

- Back-Propagation (BP)
- Gradient descent
  - A routine to compute gradient
  - Use chain rule of derivative

General Cycle of Machine Learning

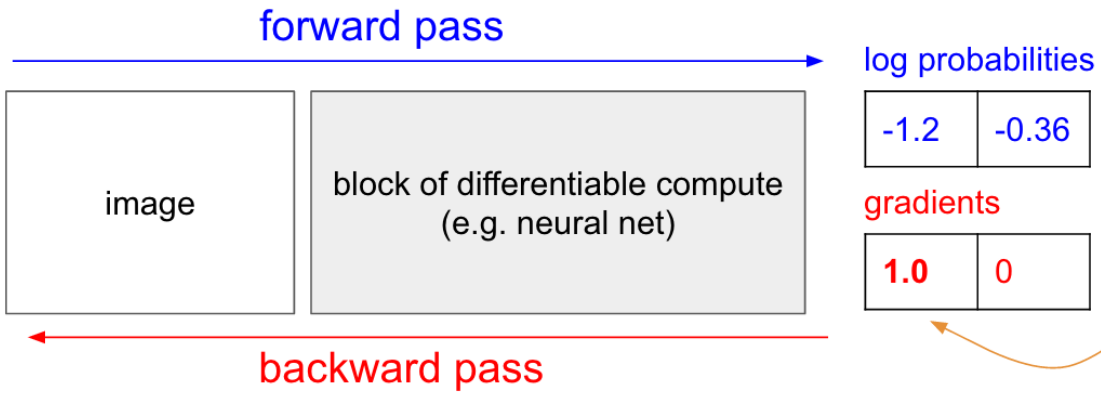
1. Given training data: $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$

2. Choose metrics: Loss function $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$

3. Define goal: $$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

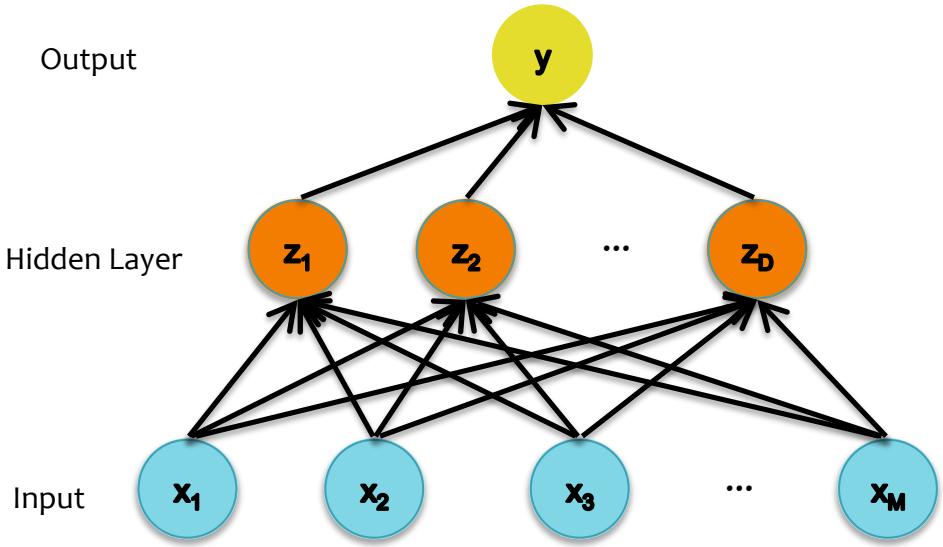
4. Train with SGD: (take small steps opposite the gradient)

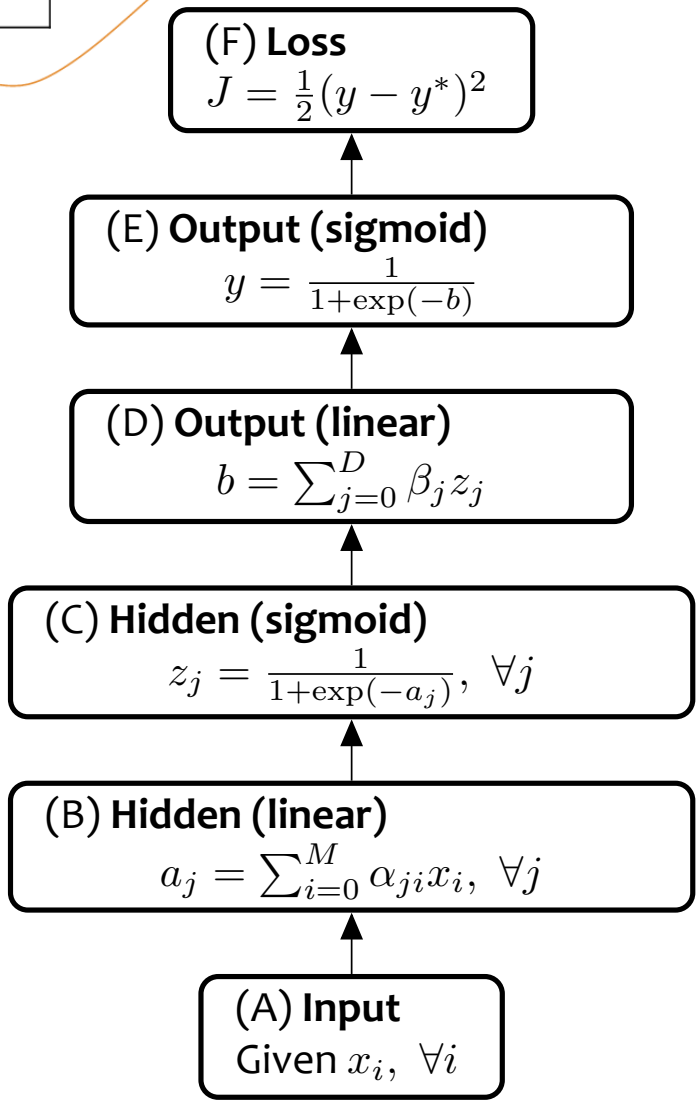
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$



Back to Basics: Backpropagation

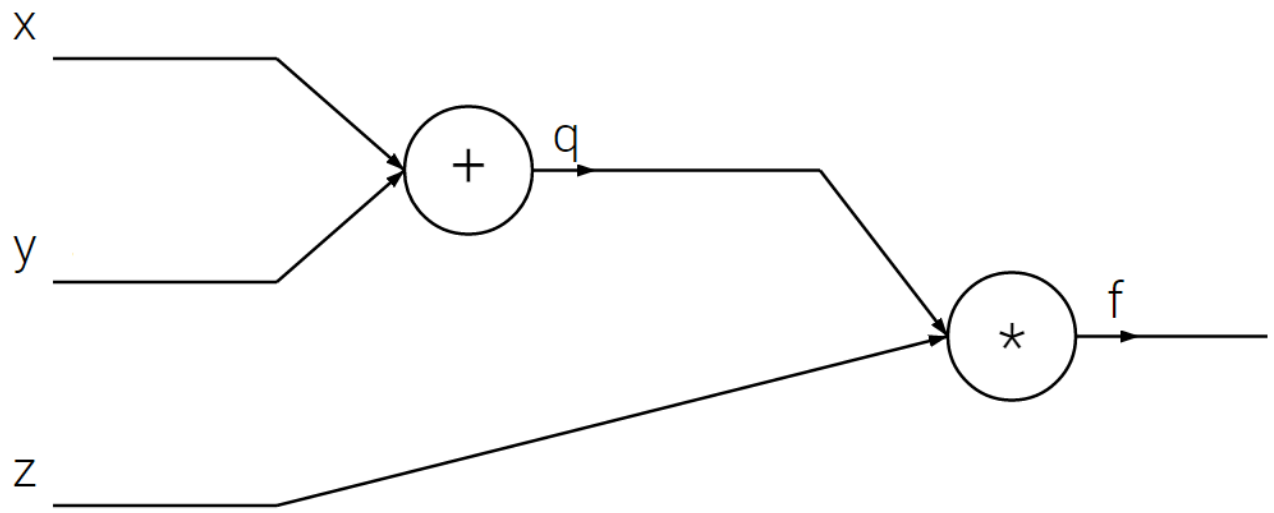
Adjust the weights to reduce the error:





Backpropagation: By Example

$$f(x, y, z) = (x + y)z$$

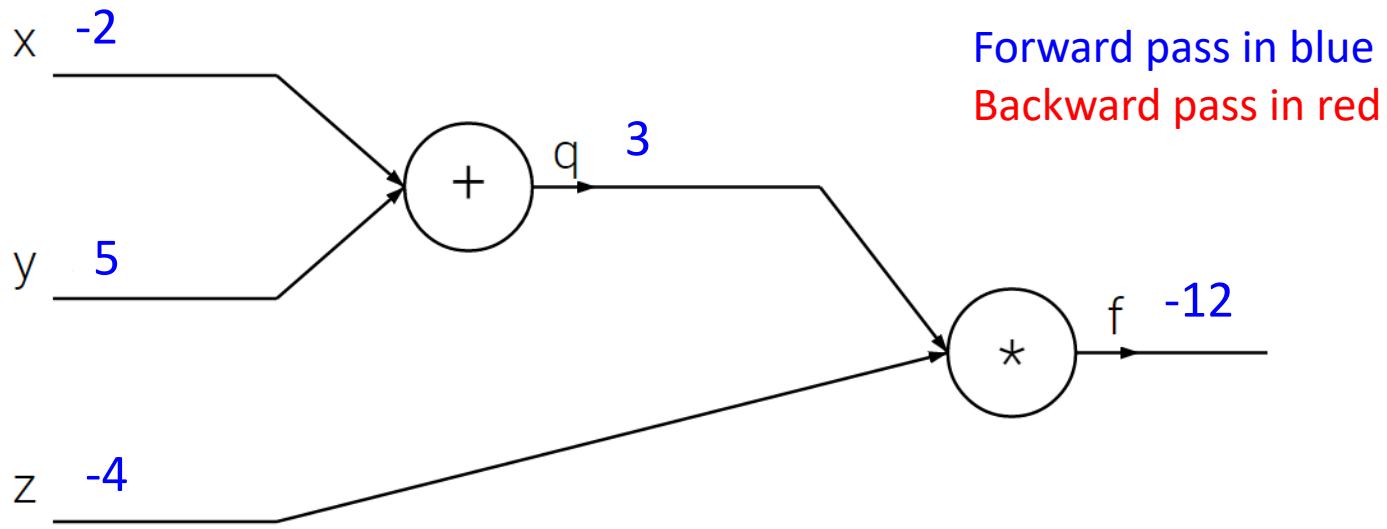


Modularity: We compute an arbitrary function locally in stages

$$q = x + y \qquad f = qz$$

Backpropagation: Forward Pass

$$f(x, y, z) = (x + y)z$$

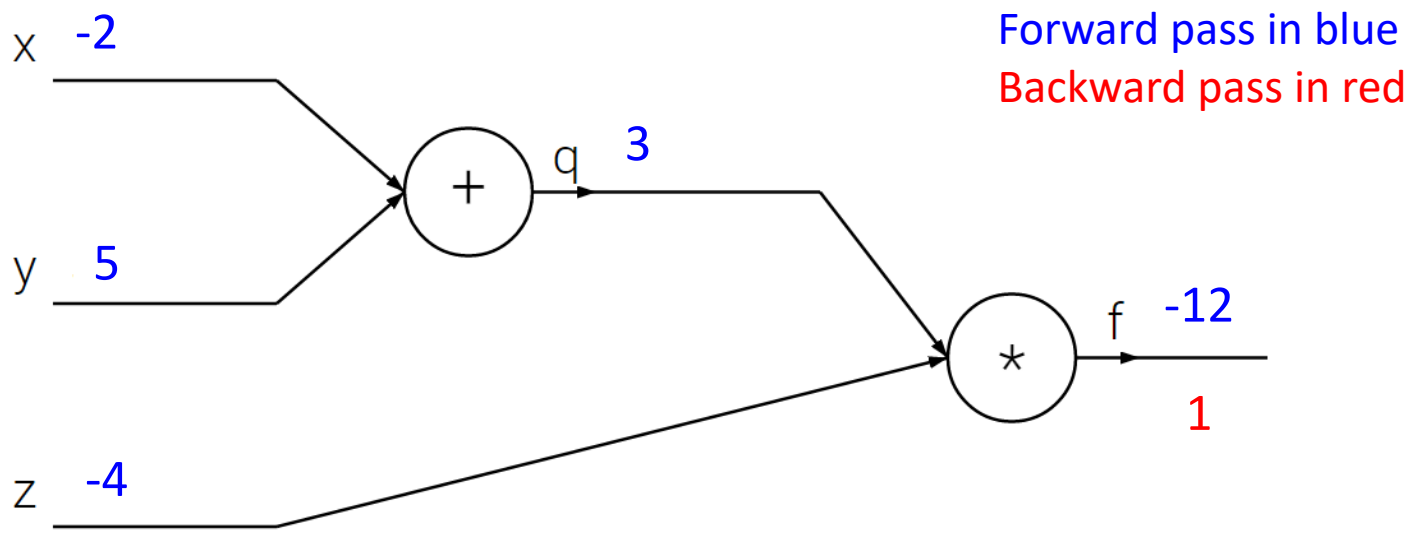


f(x, y, z) is “happy” when the output is positive.

How do we “teach” it to produce a higher output?

Backpropagation: Forward Pass

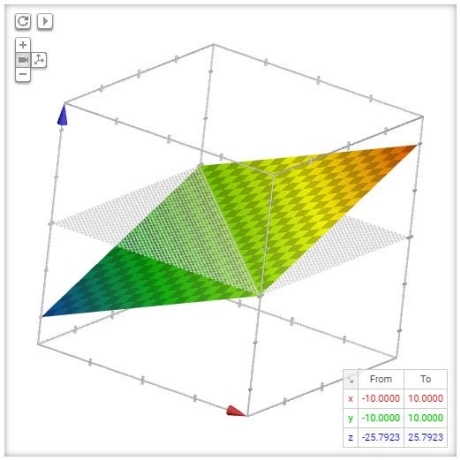
$$f(x, y, z) = (x + y)z$$



f(x, y, z) is “happy” when the output is positive.

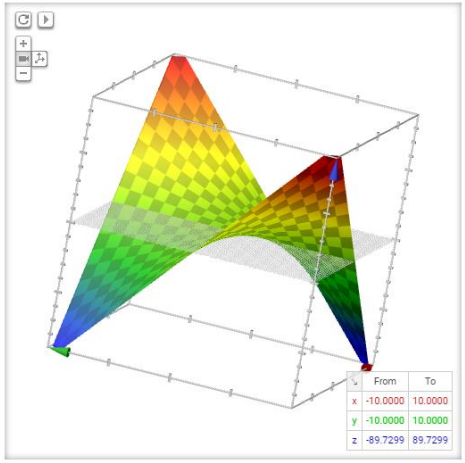
How do we “teach” it to produce a higher output?

Backpropagation: By Example



Addition:

$$f(x,y) = x + y \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

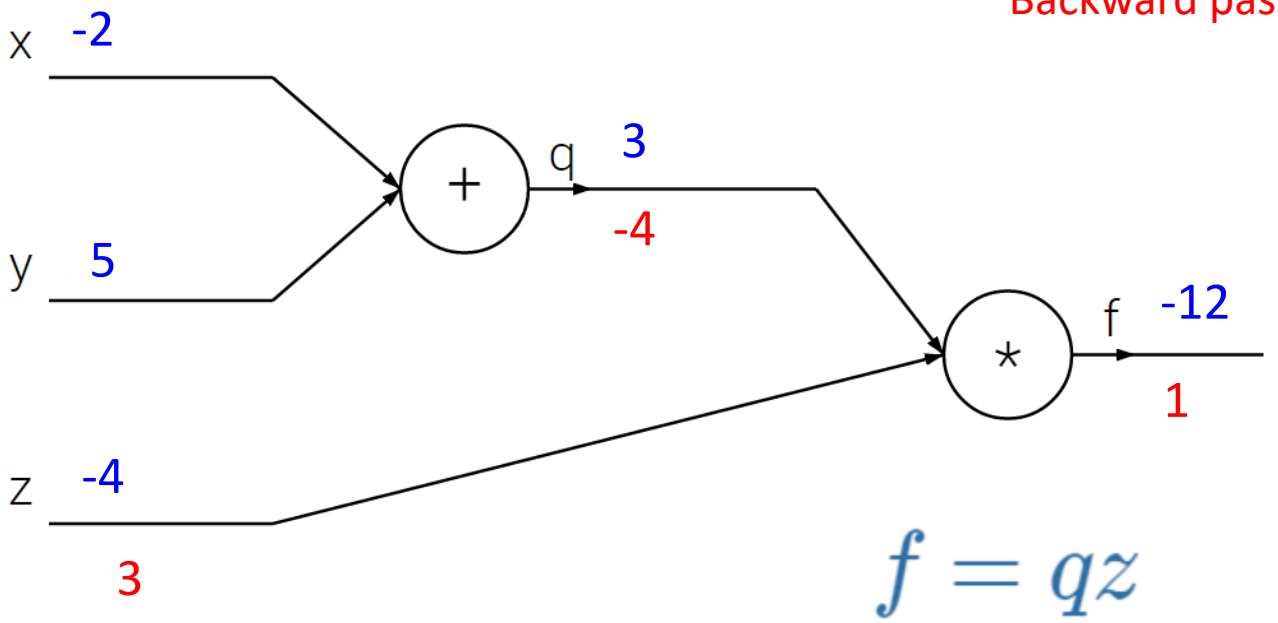


Multiplication:

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

Backpropagation: Backward Pass

Forward pass in blue
Backward pass in red



Let's compute the local gradient on *f*:

$$\frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

Modular Magic: Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

So, instead of computing the gradient of this:

$$f(x, y, z) = (x + y)z$$

We compute the gradients of these:

$$q = x + y \qquad\qquad f = qz$$

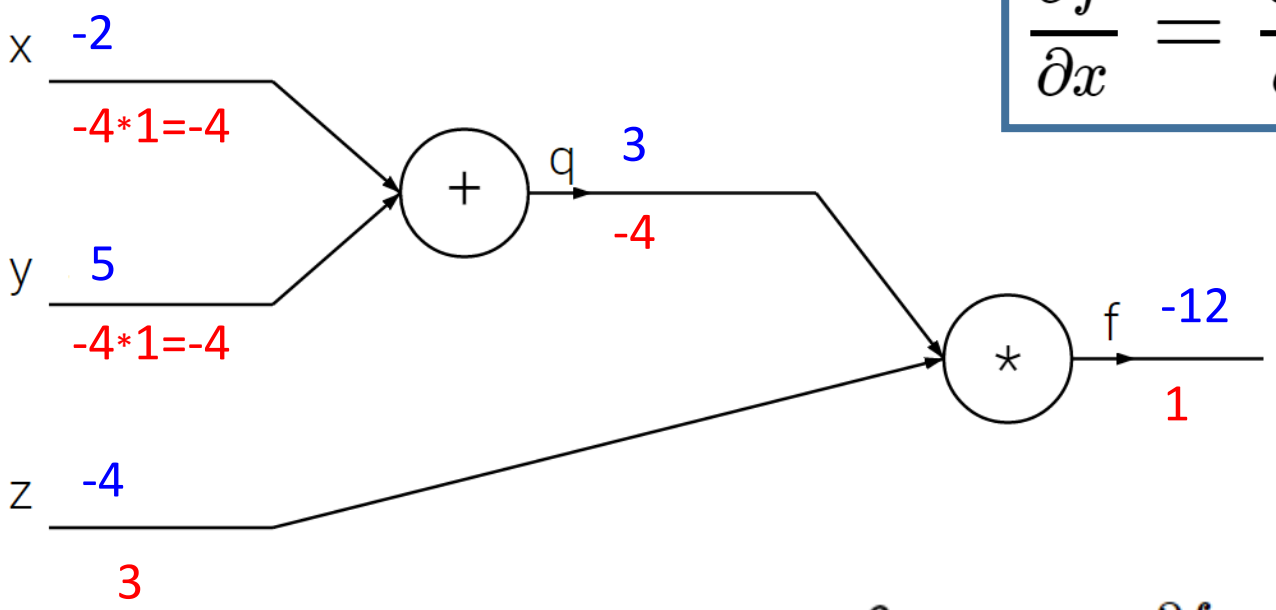
Backpropagation: Backward Pass

Forward pass in blue

Backward pass in red

Modular Magic: Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



$$q = x + y \qquad f = qz \quad \frac{\partial f}{\partial q} = z$$

Let's compute the local gradient on *q*:

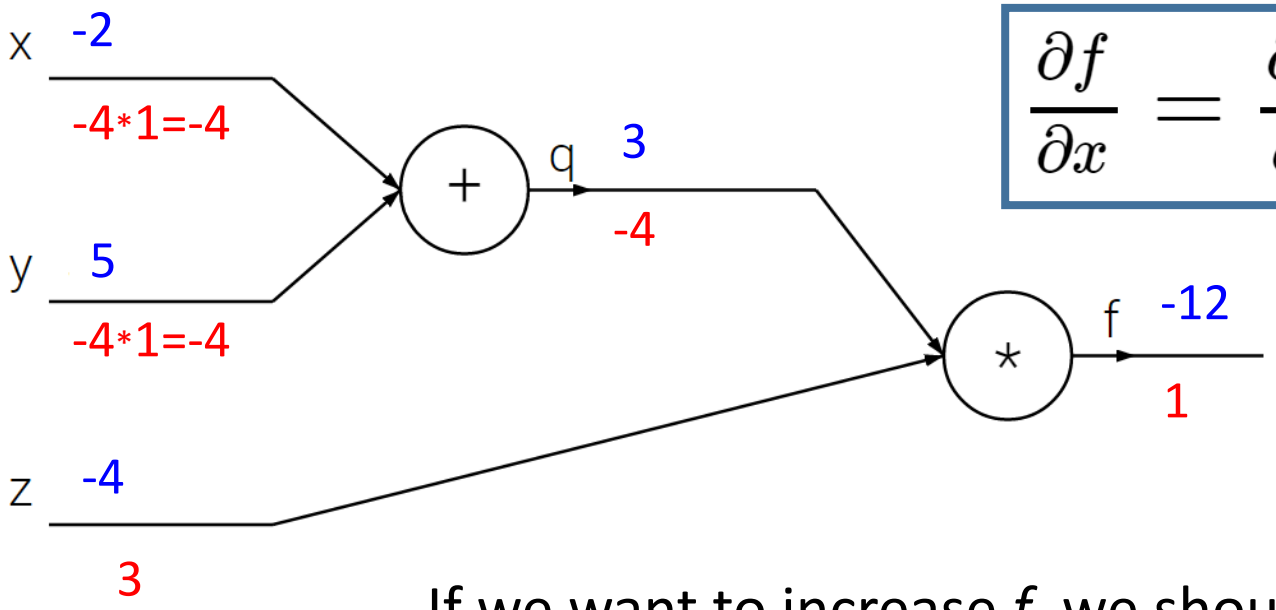
$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

Forward pass in blue
Backward pass in red

Interpreting Gradients

Modular Magic: Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



If we want to increase *f*, we should:

- Decrease *q*
- Decrease *x*
- Decrease *y*
- Increase *z*

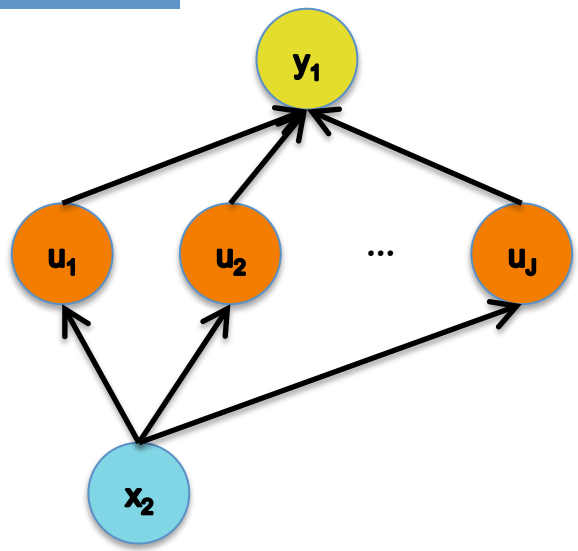
Every local gradient is a local worker in a global chase for greater *f*.

Backpropogation

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$

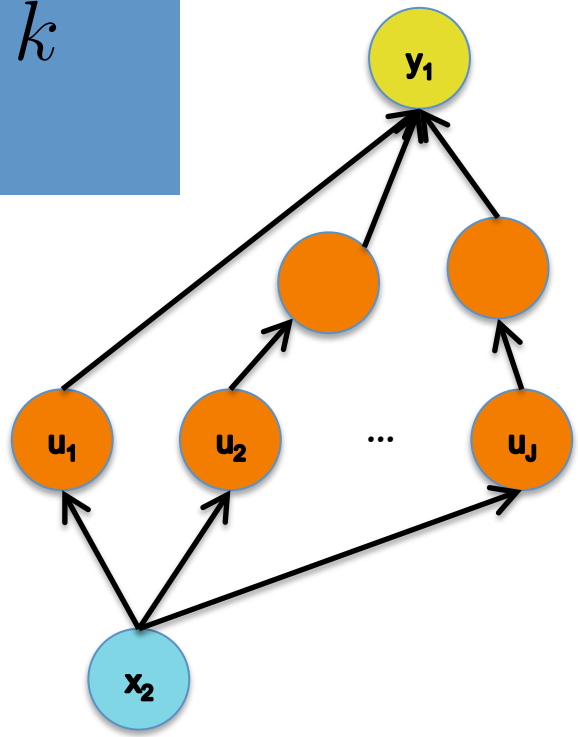


Backpropogation

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \qquad \forall i, k$$



Backpropogation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$J = cos(u)$

$u = u_1 + u_2$

$u_1 = sin(t)$

$u_2 = 3t$

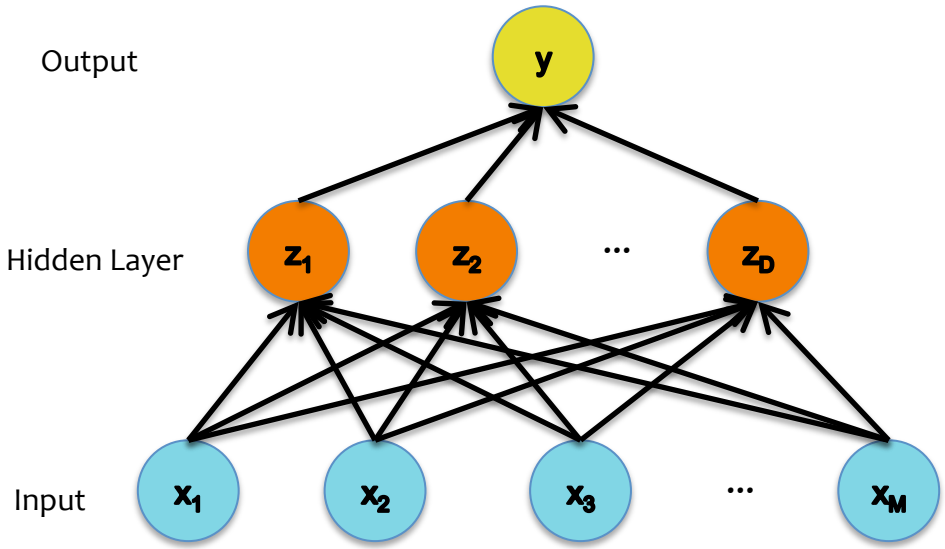
$t = x^2$

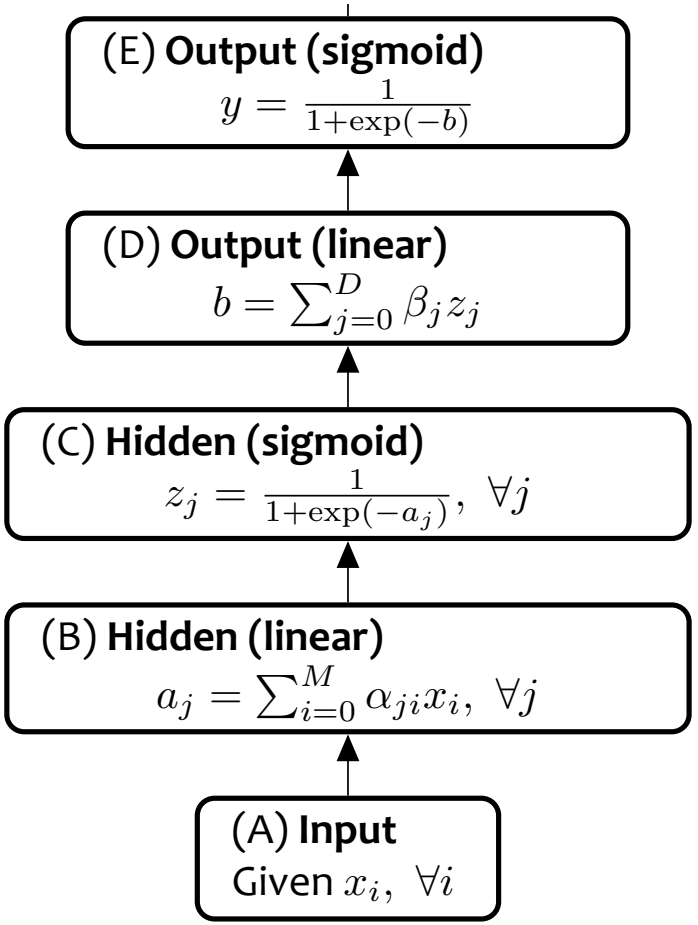
Backpropogation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

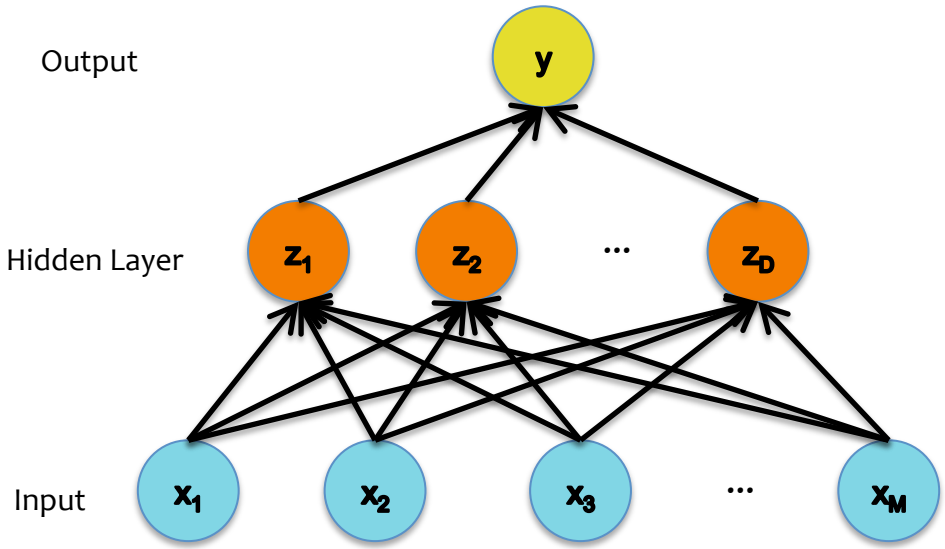
Forward	Backward
$J = cos(u)$	$\frac{dJ}{du}$ += $-sin(u)$
$u = u_1 + u_2$	$\frac{dJ}{du_1}$ += $\frac{dJ}{du}\frac{du}{du_1}$, $\frac{du}{du_1} = 1$ $\quad$ $\frac{dJ}{du_2}$ += $\frac{dJ}{du}\frac{du}{du_2}$, $\frac{du}{du_2} = 1$
$u_1 = sin(t)$	$\frac{dJ}{dt}$ += $\frac{dJ}{du_1}\frac{du_1}{dt}$, $\frac{du_1}{dt} = \cos(t)$
$u_2 = 3t$	$\frac{dJ}{dt}$ += $\frac{dJ}{du_2}\frac{du_2}{dt}$, $\frac{du_2}{dt} = 3$
$t = x^2$	$\frac{dJ}{dx}$ += $\frac{dJ}{dt}\frac{dt}{dx}$, $\frac{dt}{dx} = 2x$

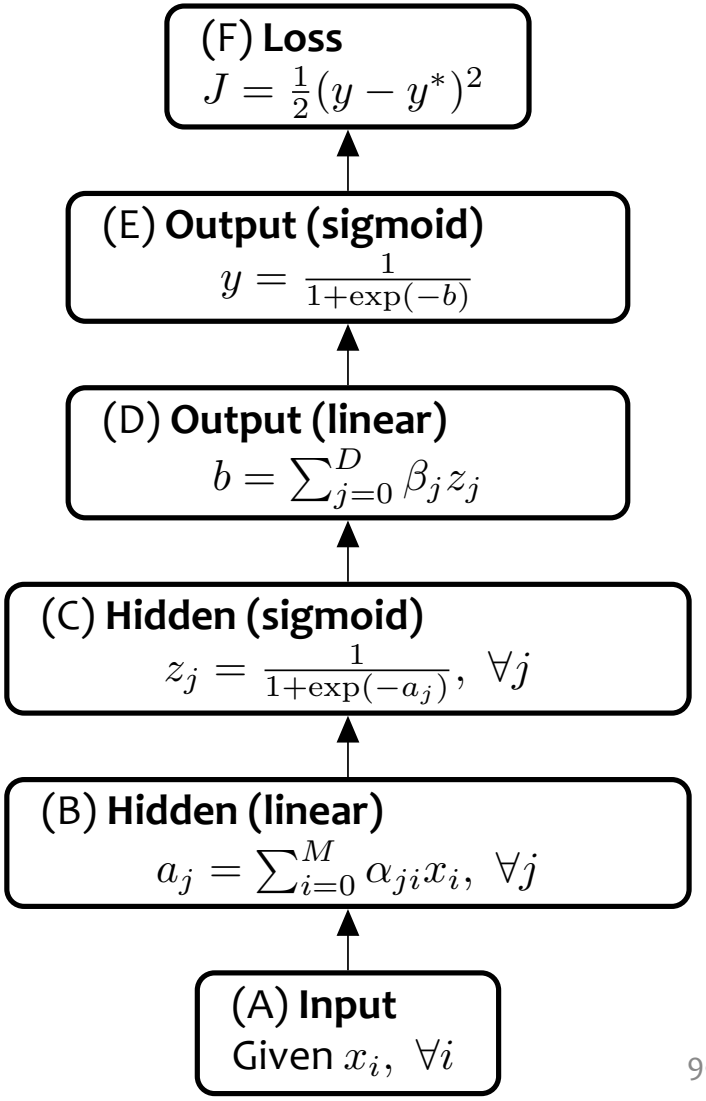
Backpropogation





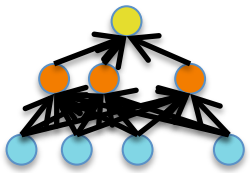
Backpropogation





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

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

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

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

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

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

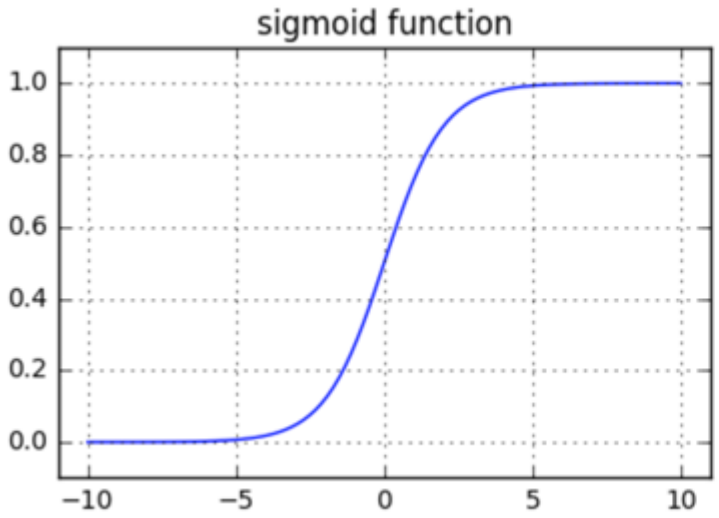
$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$$

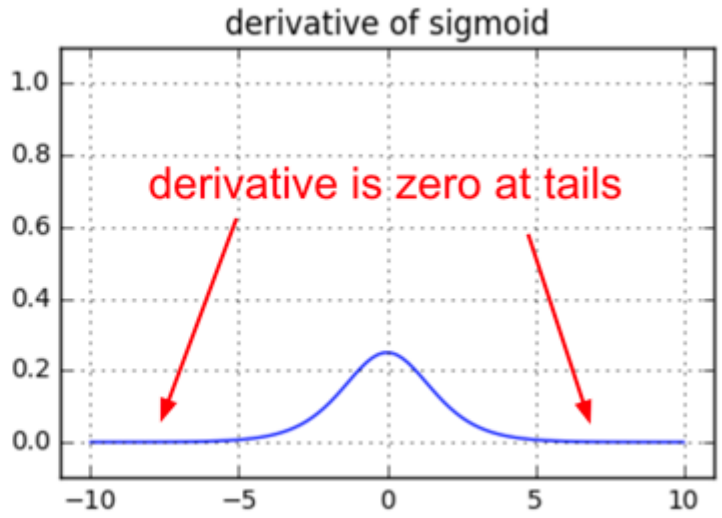
$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

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





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

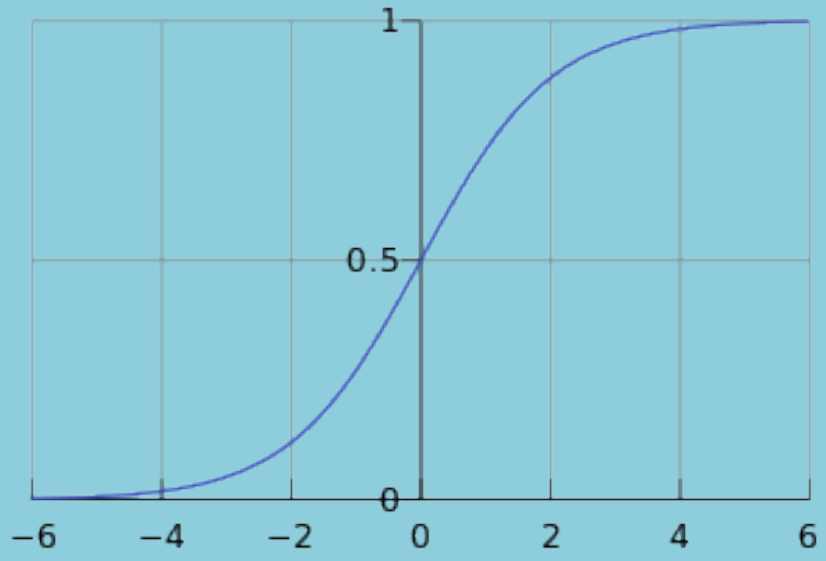
Partial derivatives are small = Learning is slow

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

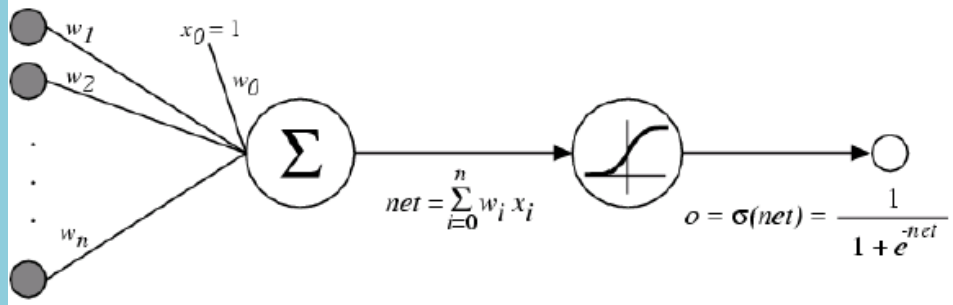
Activation Functions

Sigmoid / Logistic Function

$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$



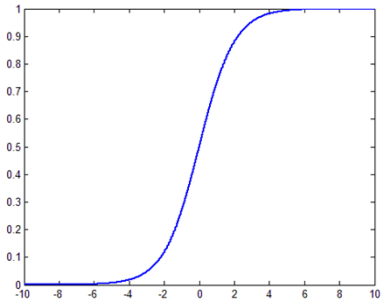
So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…



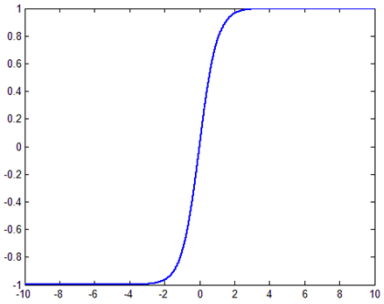
Activation Functions

- Applied on the hidden units
- Achieve nonlinearity
- Popular activation functions

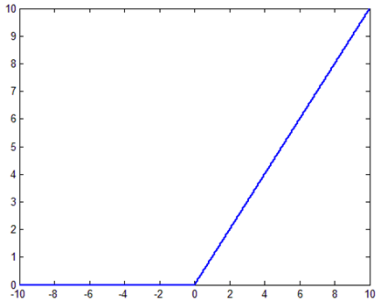
Sigmoid



Tanh

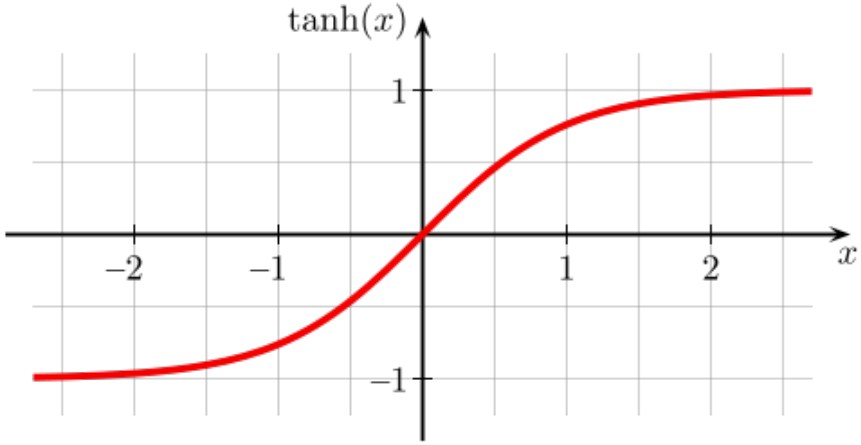


Rectified Linear



Activation Functions

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs

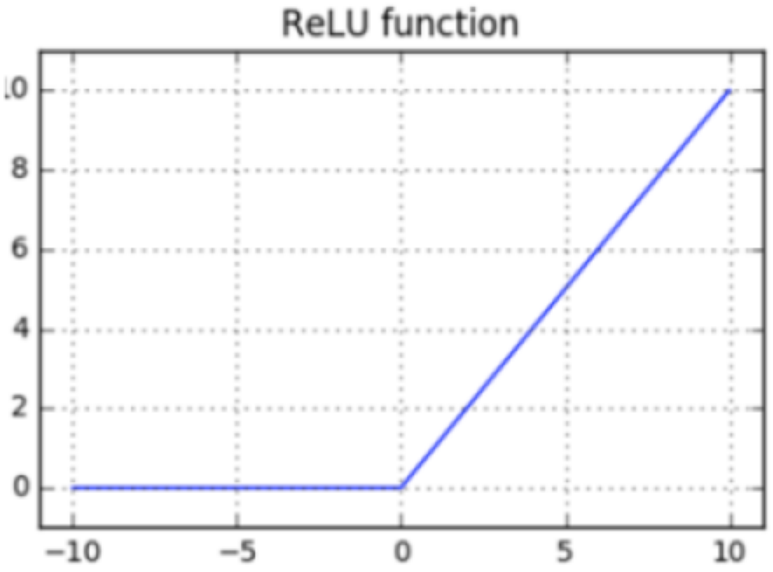


Alternate :
tanh

Like logistic function but shifted to range [-1, +1]

Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks



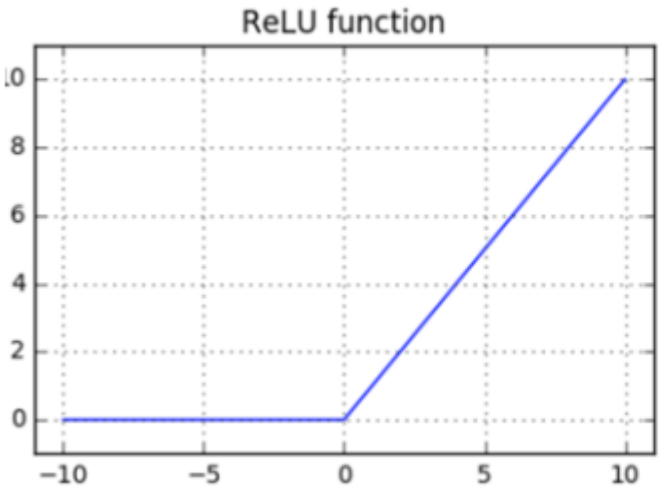
$$\max(0, w \cdot x + b).$$

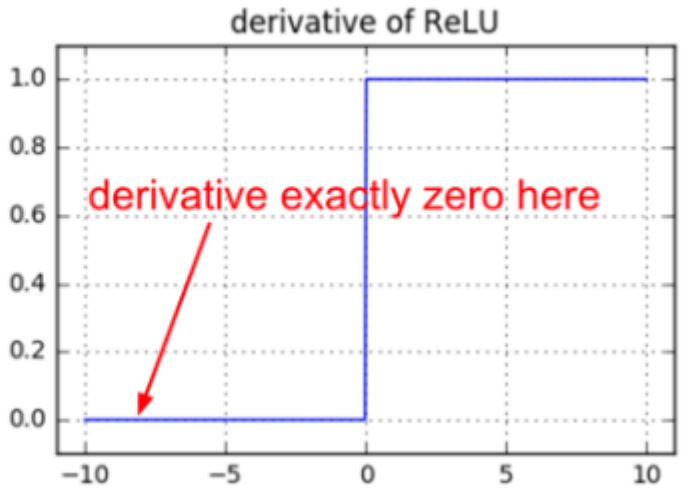
Alternate: rectified linear unit

Linear with a cutoff at zero

(Implementation: clip the gradient when you pass zero)

Optimization is Hard: Vanishing Gradients



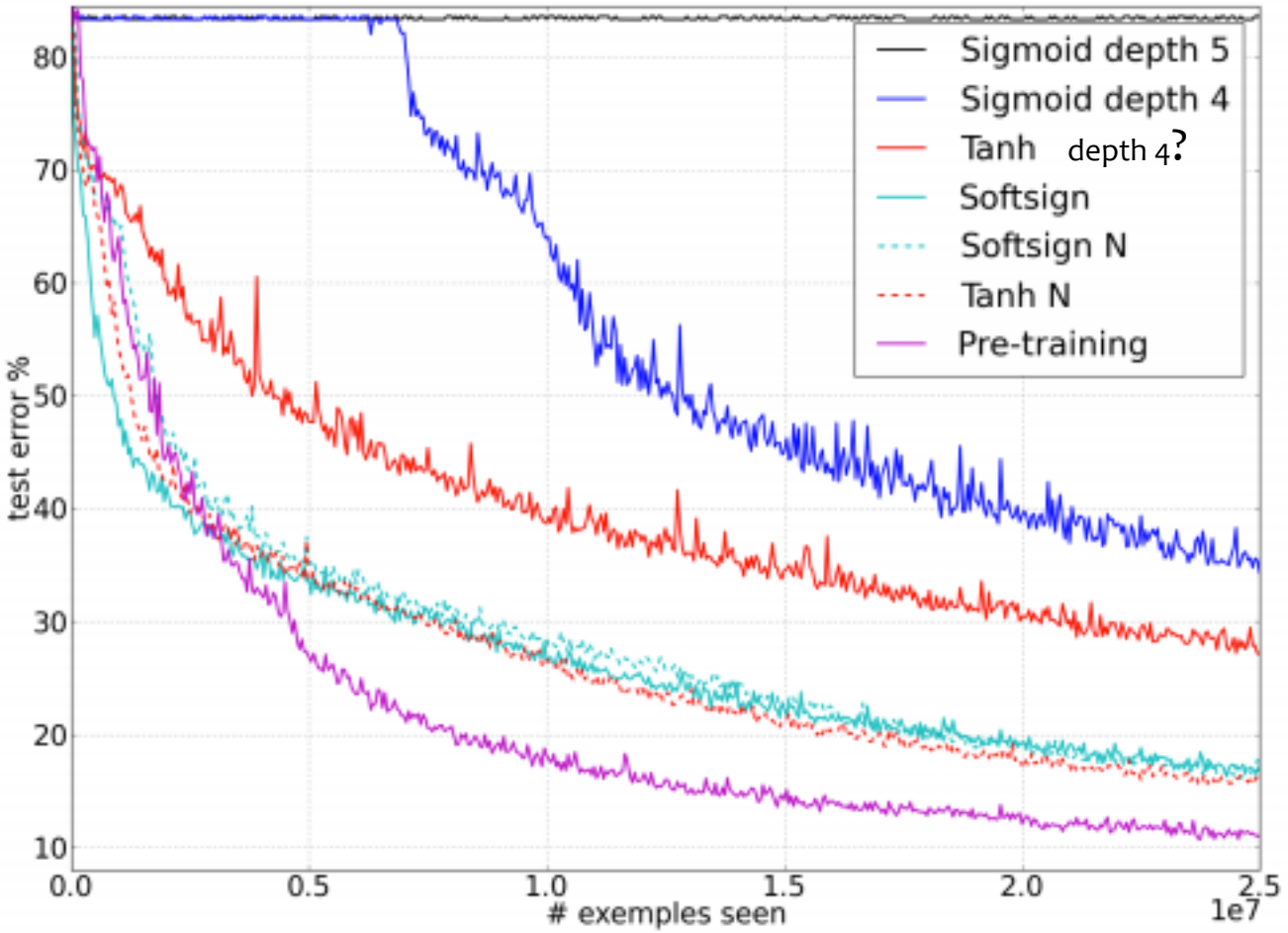


- If a neuron is initialized poorly, it might not fire for entire training dataset.
- Large parts of your network could be dead ReLUs!

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

Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

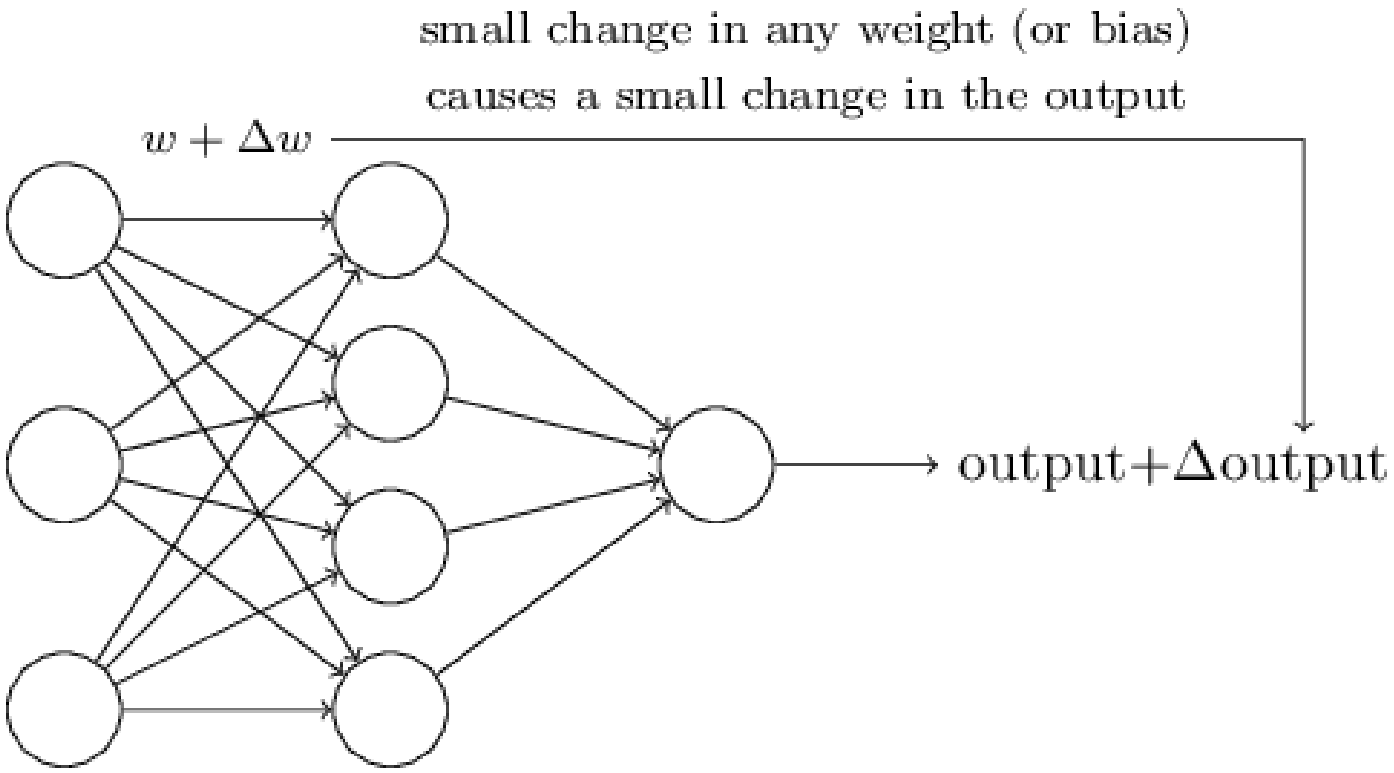


sigmoid vs. tanh

Figure from Glorot & Bentio (2010)

The Process of Learning:

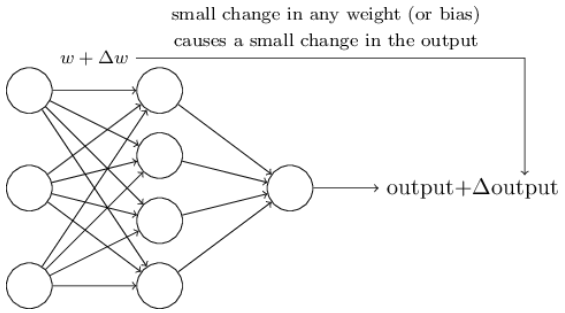
Small Change in Weights →Small Change in Output

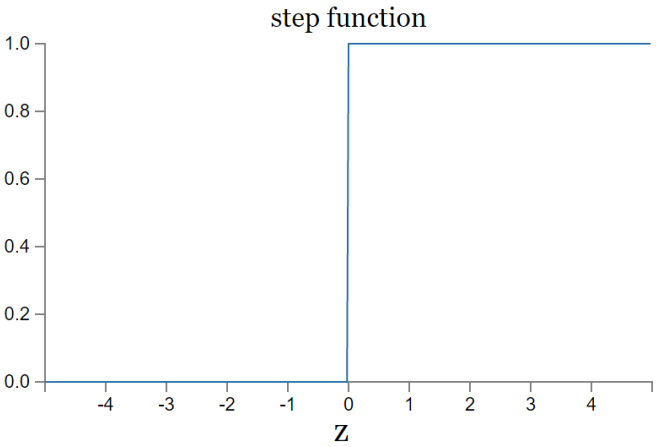


The Process of Learning:

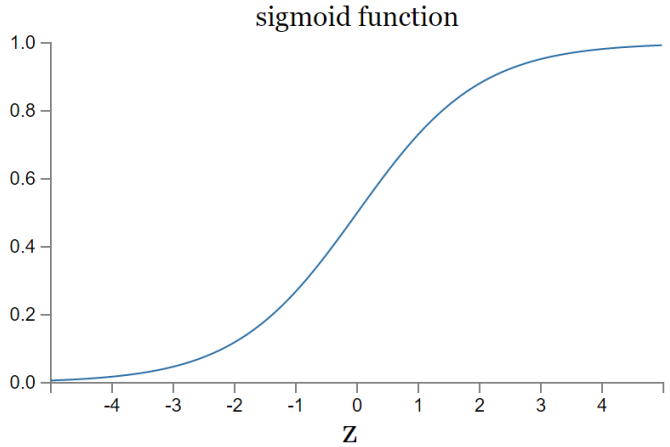
Small Change in Weights →Small Change in Output

This requires a “smoothness”





Perceptron

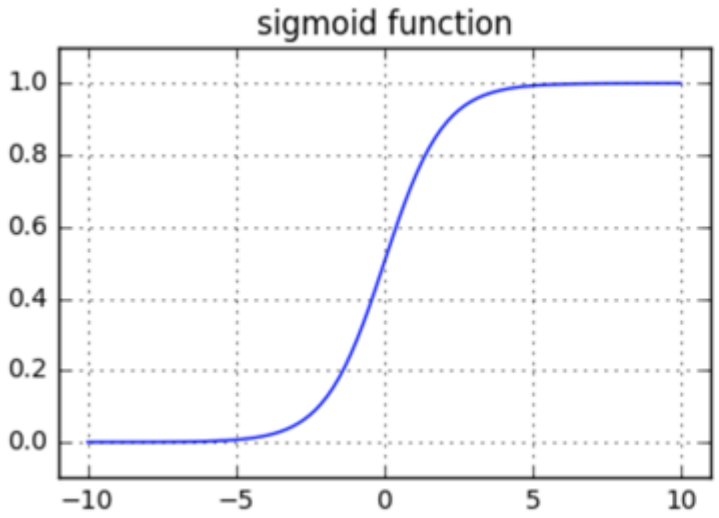


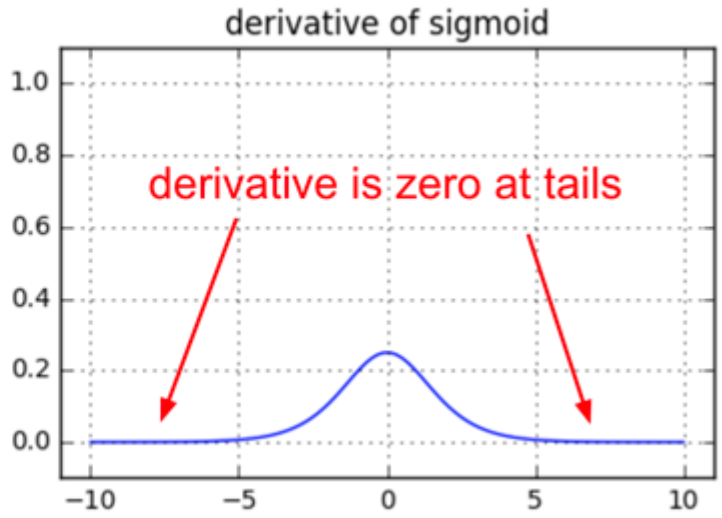
Neuron

Smoothness of activation function means: **the Δoutput is a linear function of the Δweights and Δbias**

Learning is the process of gradually adjusting the weights to achieve any gradual change in the output.

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.

Idea #2: Supervised Pre-training

Idea #2: (Two Steps)

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

Then use our **original idea**

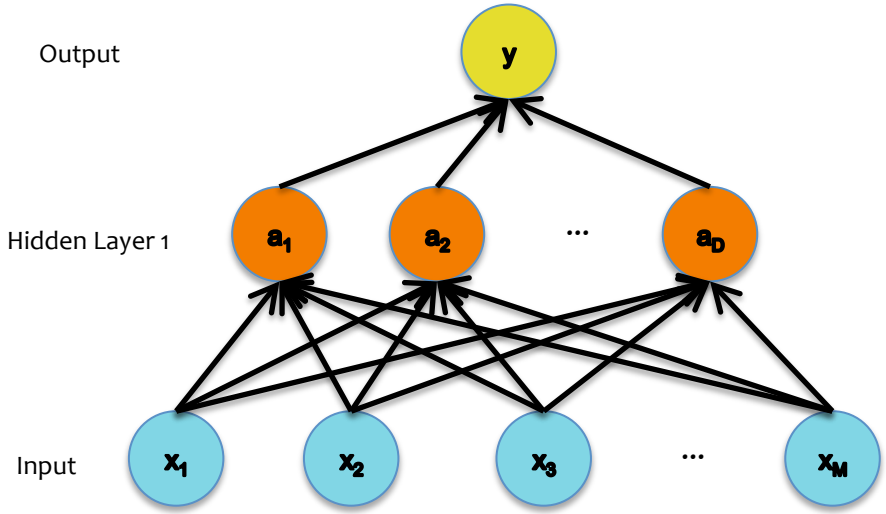
1. Supervised Pre-- training
   - Use **labeled** data
   - Work bottom-- up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-- tuning
   - Use **labeled** data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

Idea #2: Supervised Pre-training

Idea #2: (Two Steps)

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

Then use our **original idea**



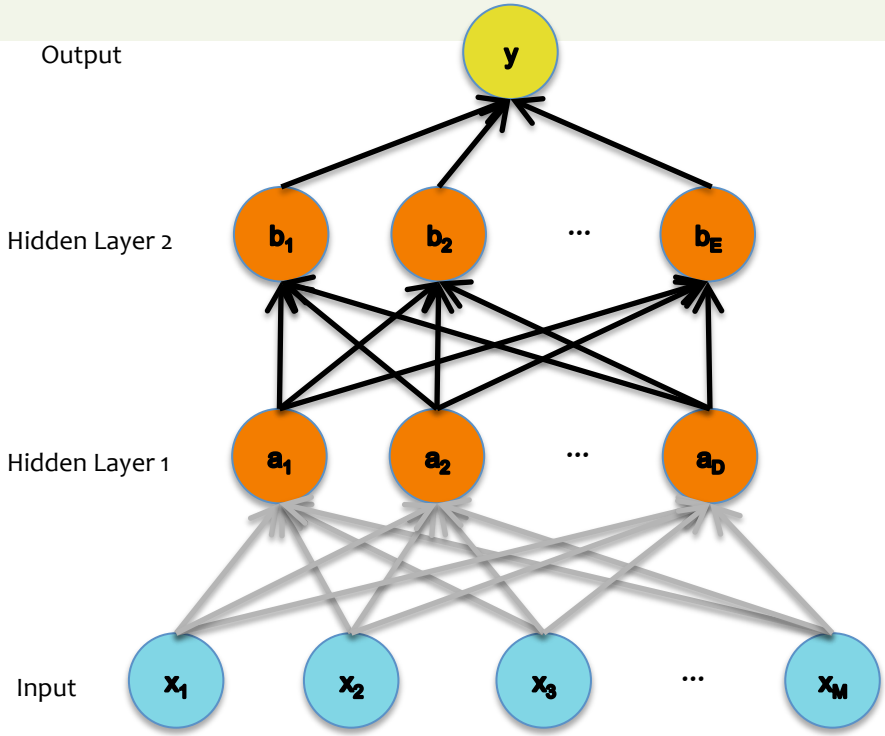
Training

Idea #2: Supervised Pre-training

Idea #2: (Two Steps)

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

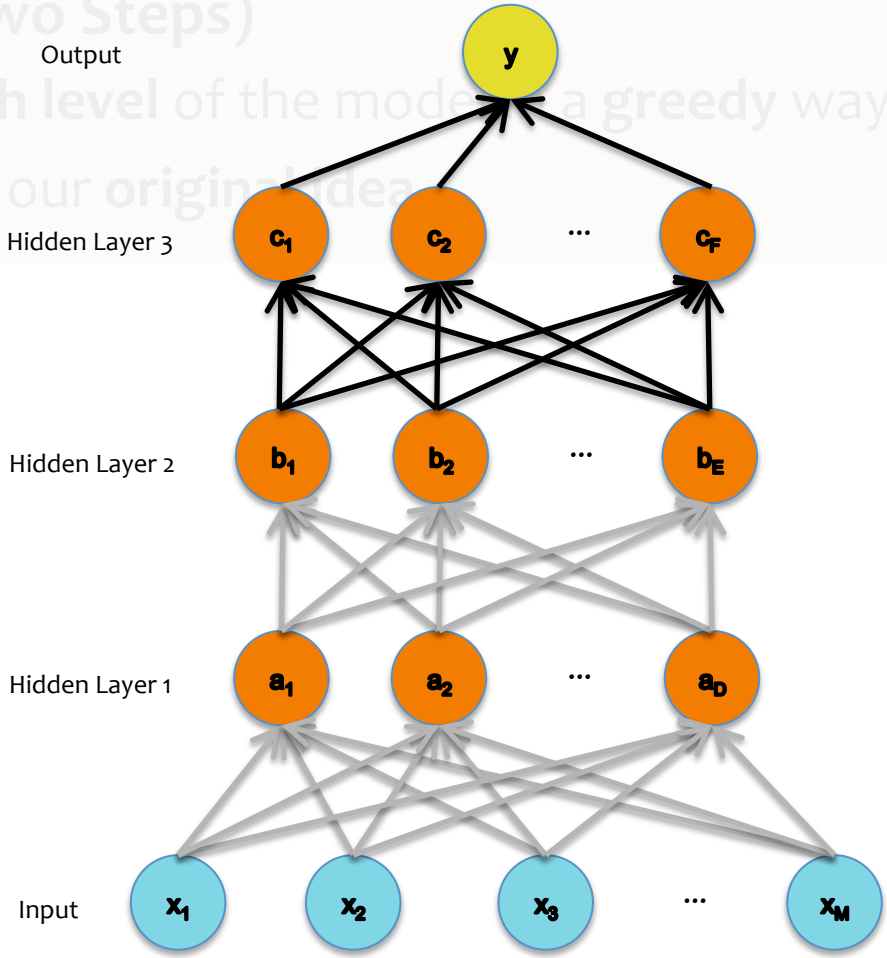
Then use our **original idea**



Training

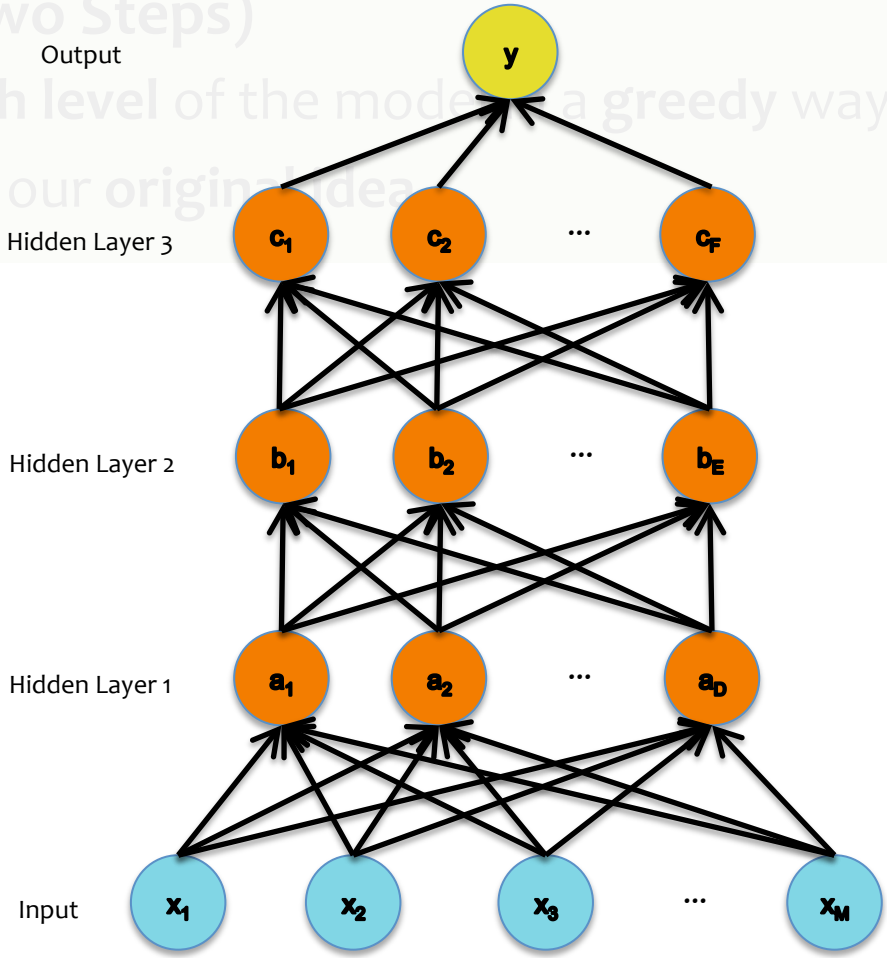
Idea #2: Supervised Pre-training

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



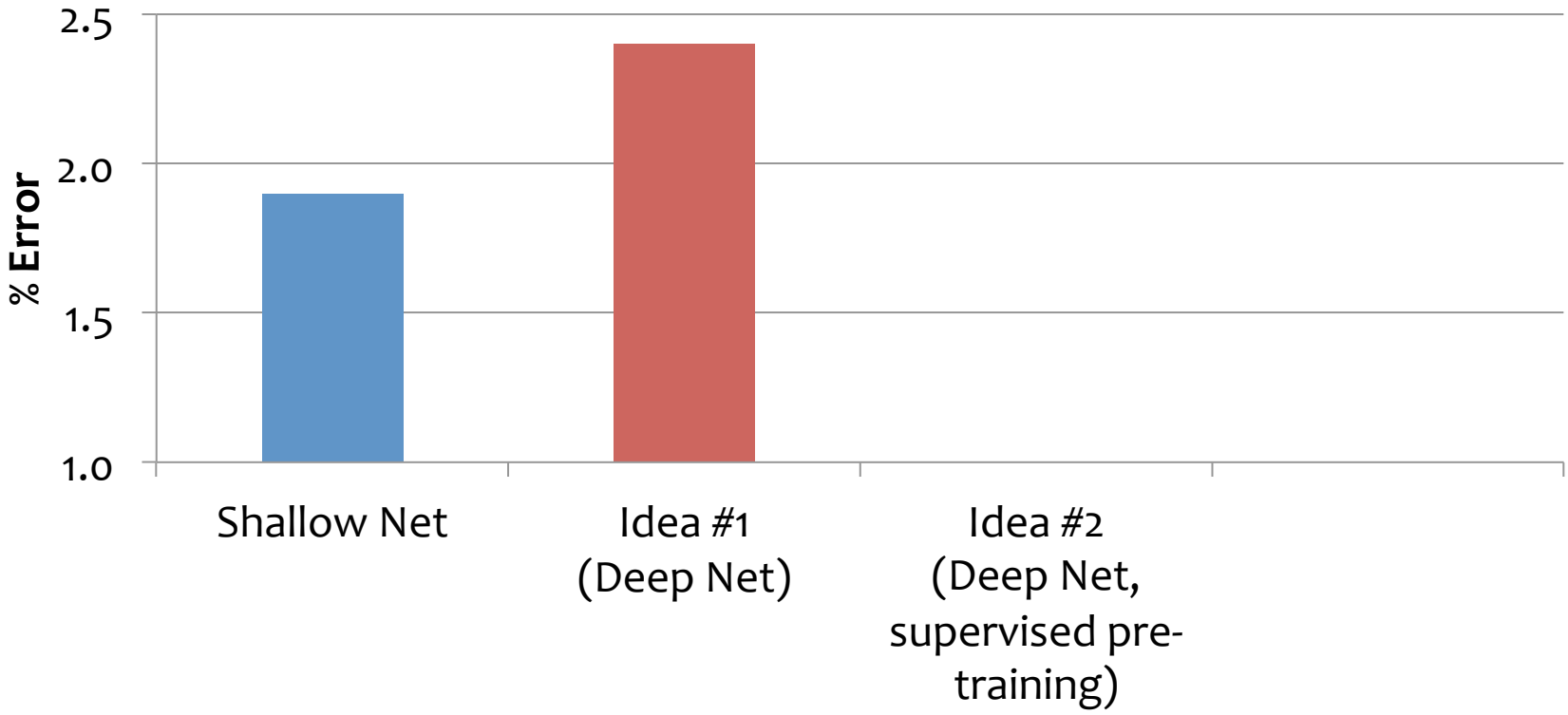
Training

Idea #2: Supervised Pre-training



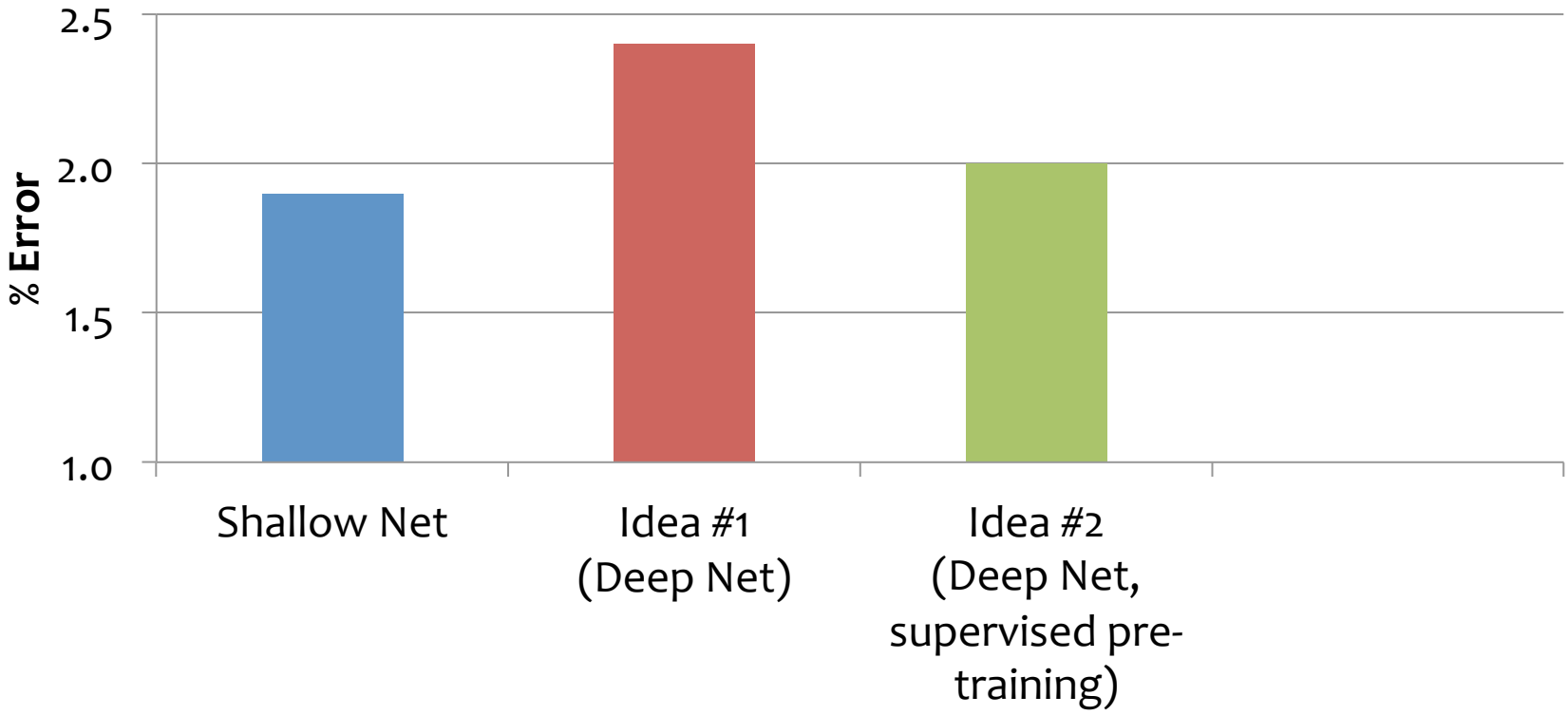
Comparison on MNIST

- 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**

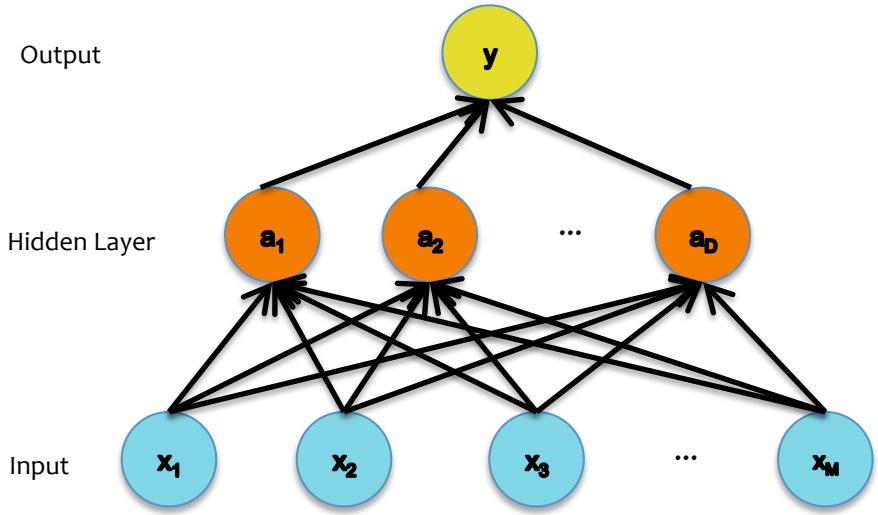
Train each level of the model in a **greedy** way

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!**

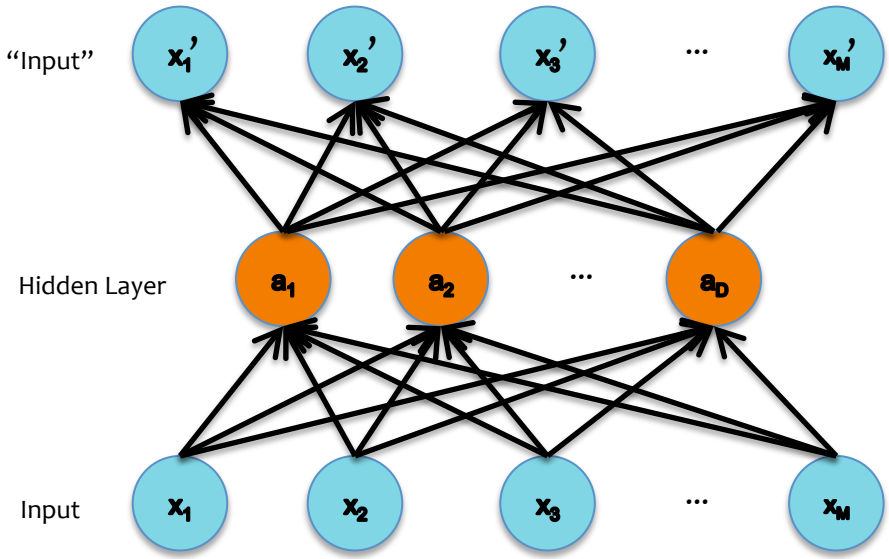


The solution: *Unsupervised pre-training*

Unsupervised pre-- training of the first layer:

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

This topology defines an Auto-- encoder.



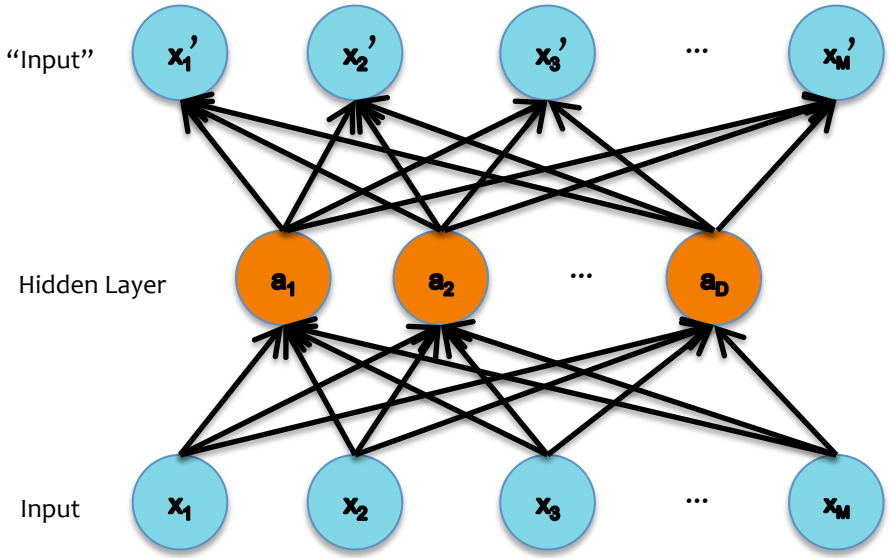
Auto--Encoders

Key idea: Encourage z to give small reconstruction error:

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

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

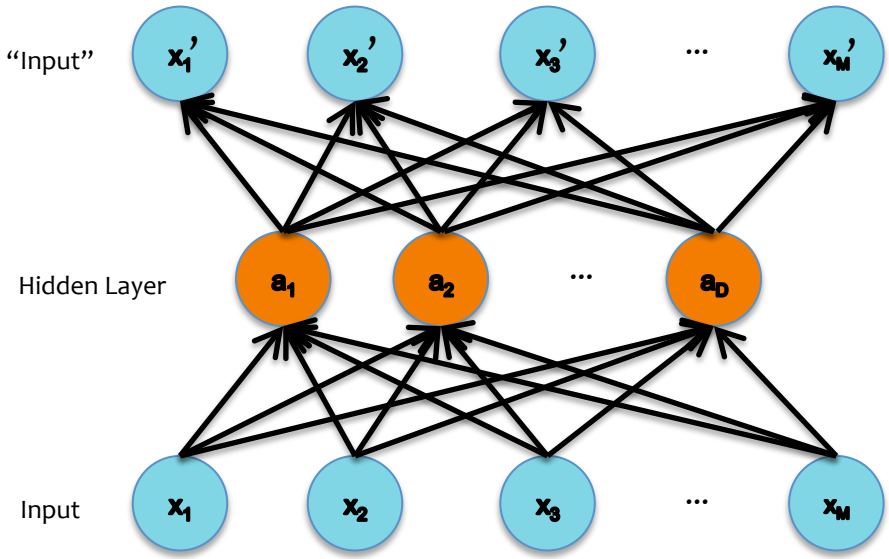
ENCODER: $z = h(Wx)$



The solution: *Unsupervised pre- training*

Unsupervised pre-- training

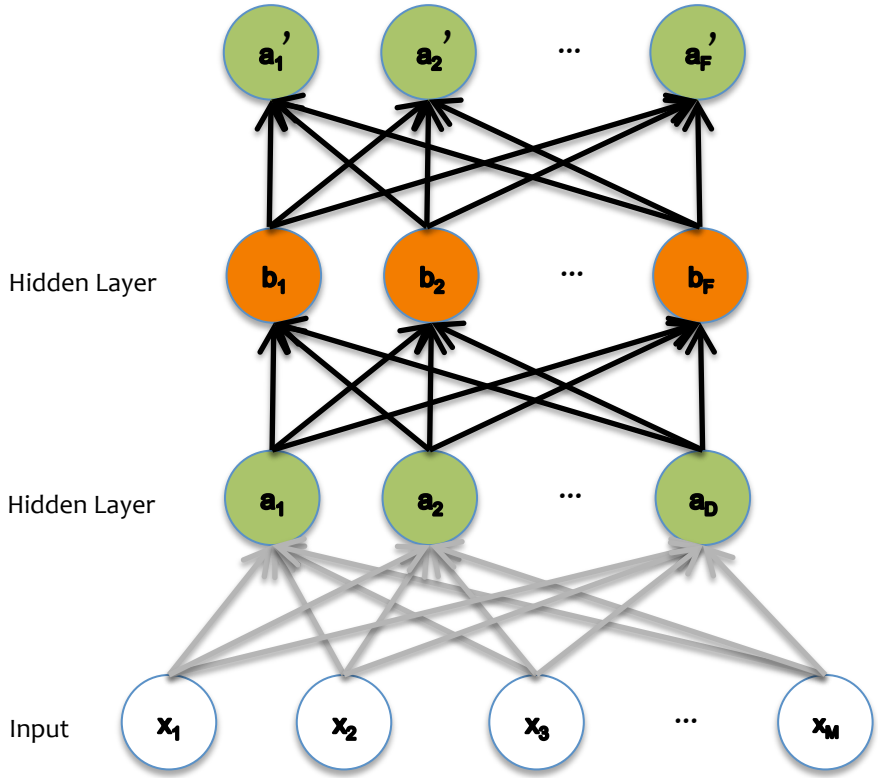
- Work bottom-- up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution: *Unsupervised pre- training*

Unsupervised pre-- training

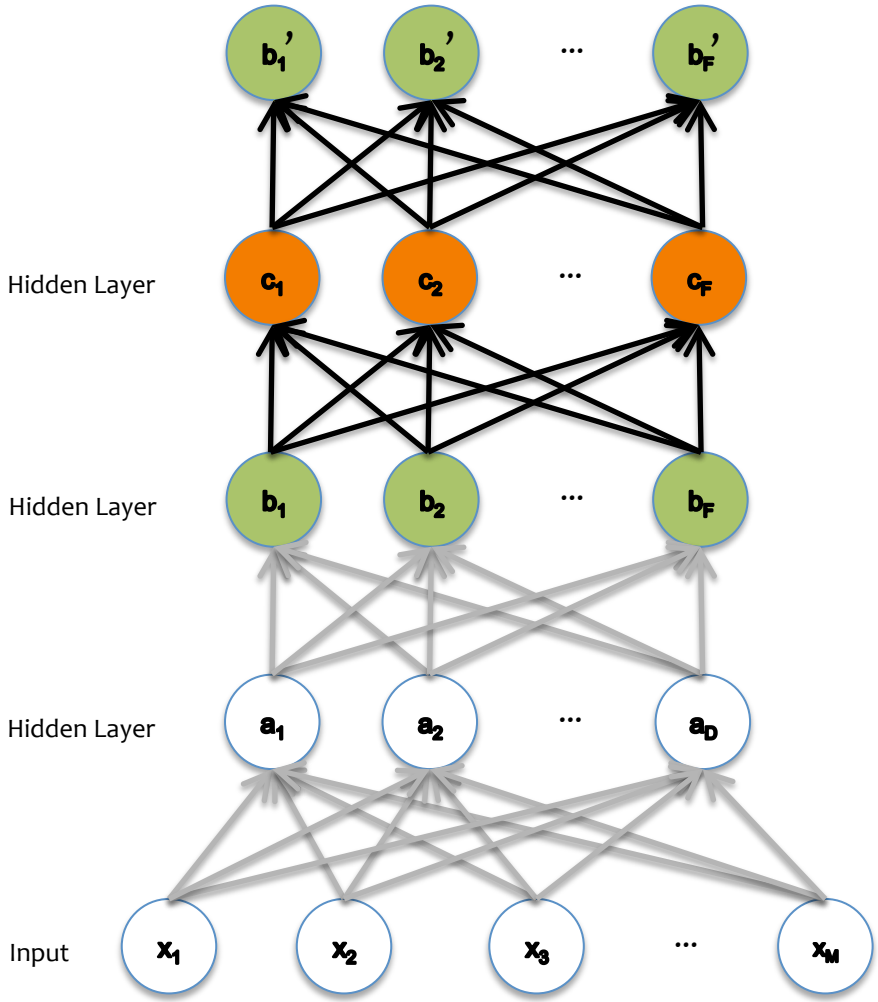
- Work bottom-- up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



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.

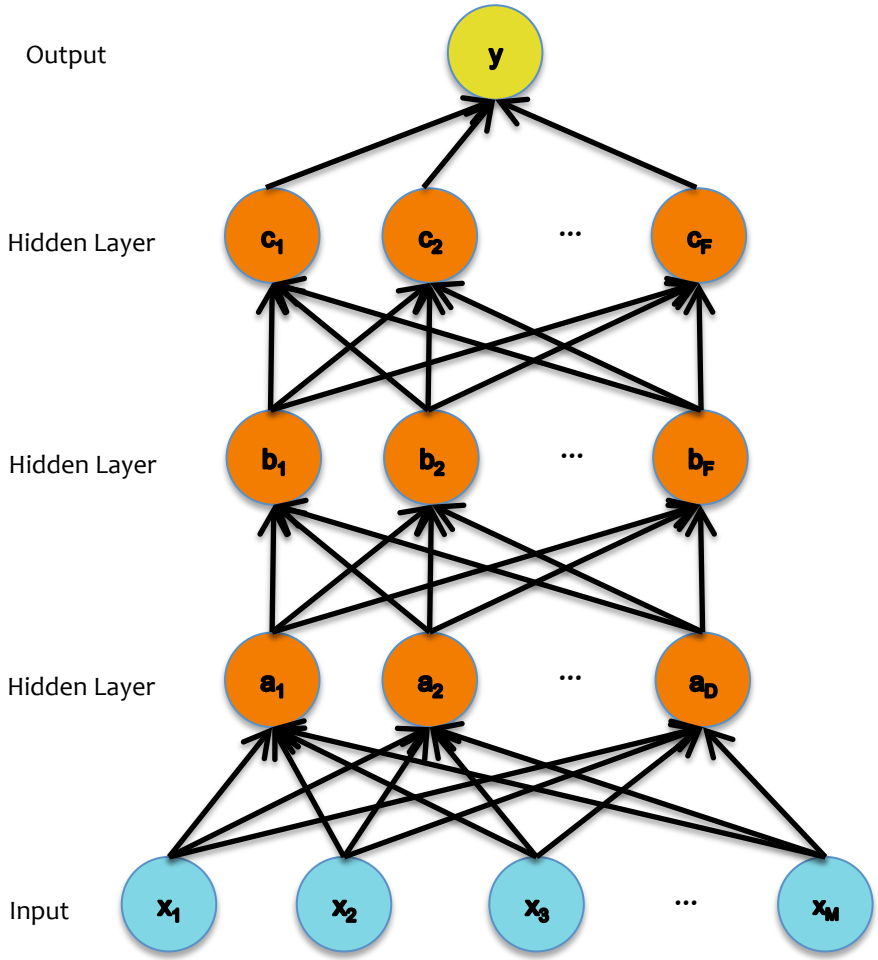


The solution: *Unsupervised pre-training*

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:

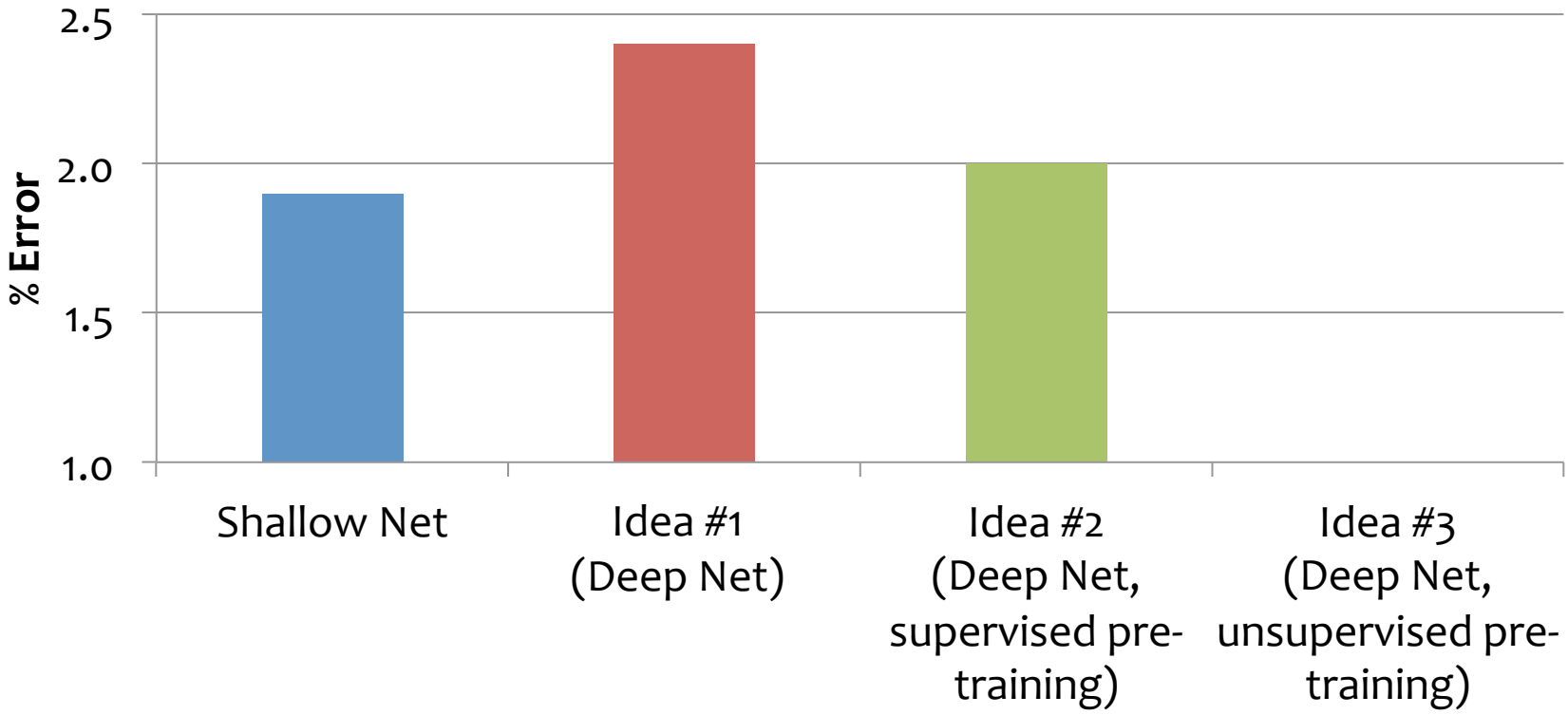
1. Supervised layer-- wise pre-- training
2. Supervised fine-- tuning

Idea #3:

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

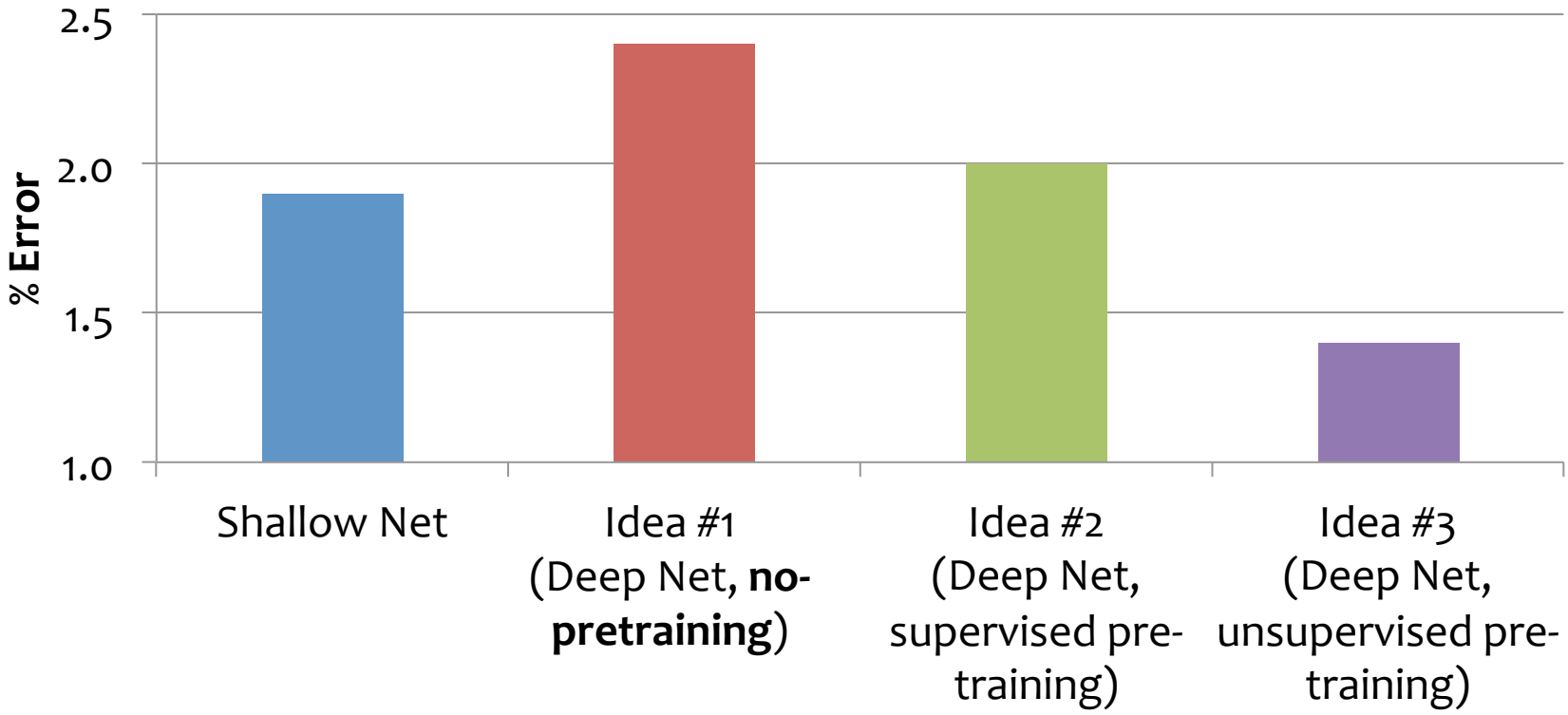
Comparison on MNIST

- 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)



In 2010, a record on handwriting recognition task was set by standard supervised backpropagation (Idea #1).

How? A very fast implementation on GPUs.

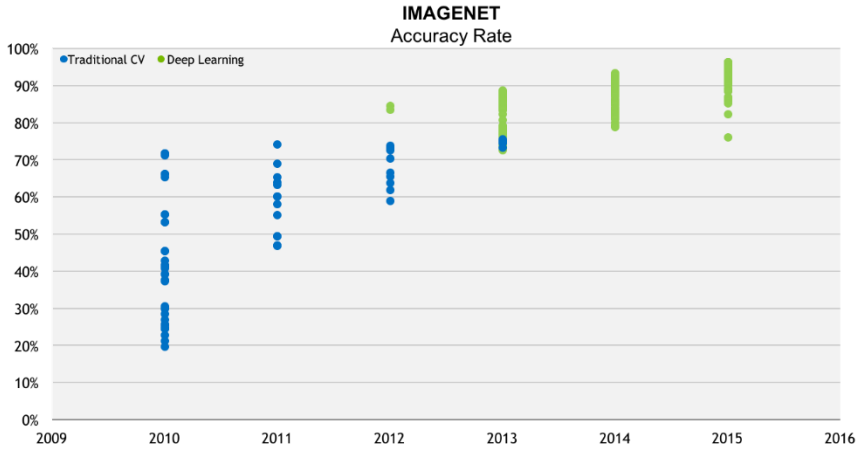
See Ciresen et al. (2010)

Deep Learning

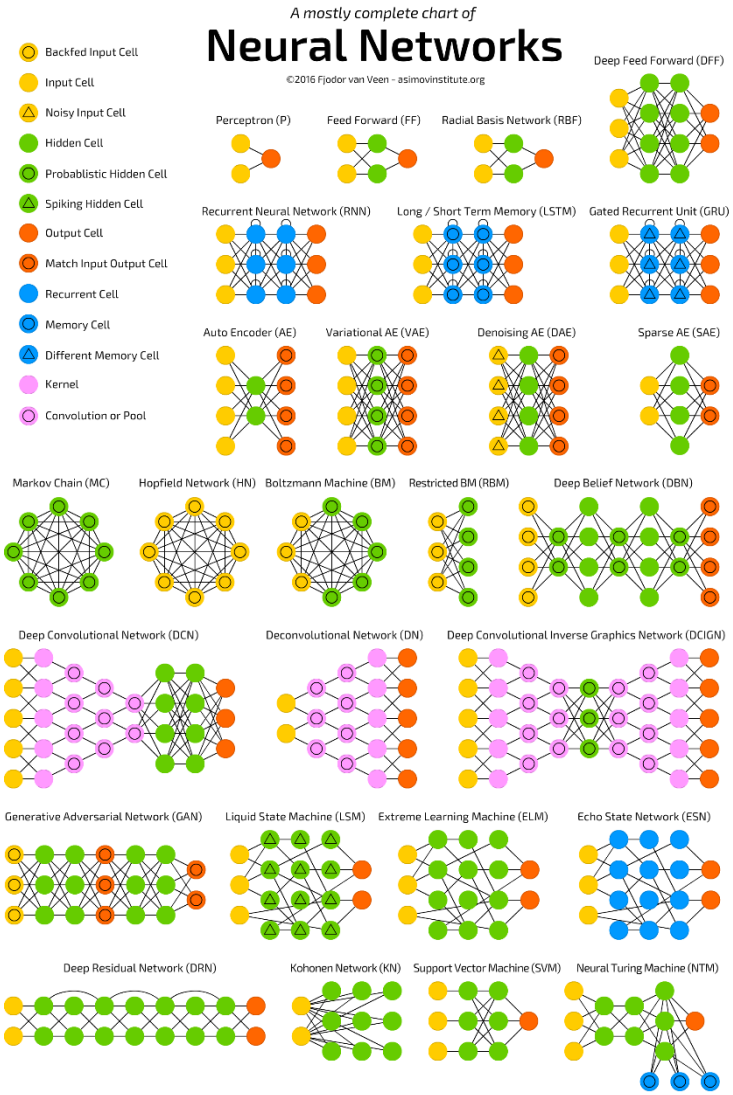
- Goal: learn features at different levels of abstraction
- Training can be tricky due to…
  - Local Optimal
  - Vanishing gradients
- Unsupervised layer-wise pre-training can help with both!

Deep Learning Breakthroughs: What Changed?

- **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, …



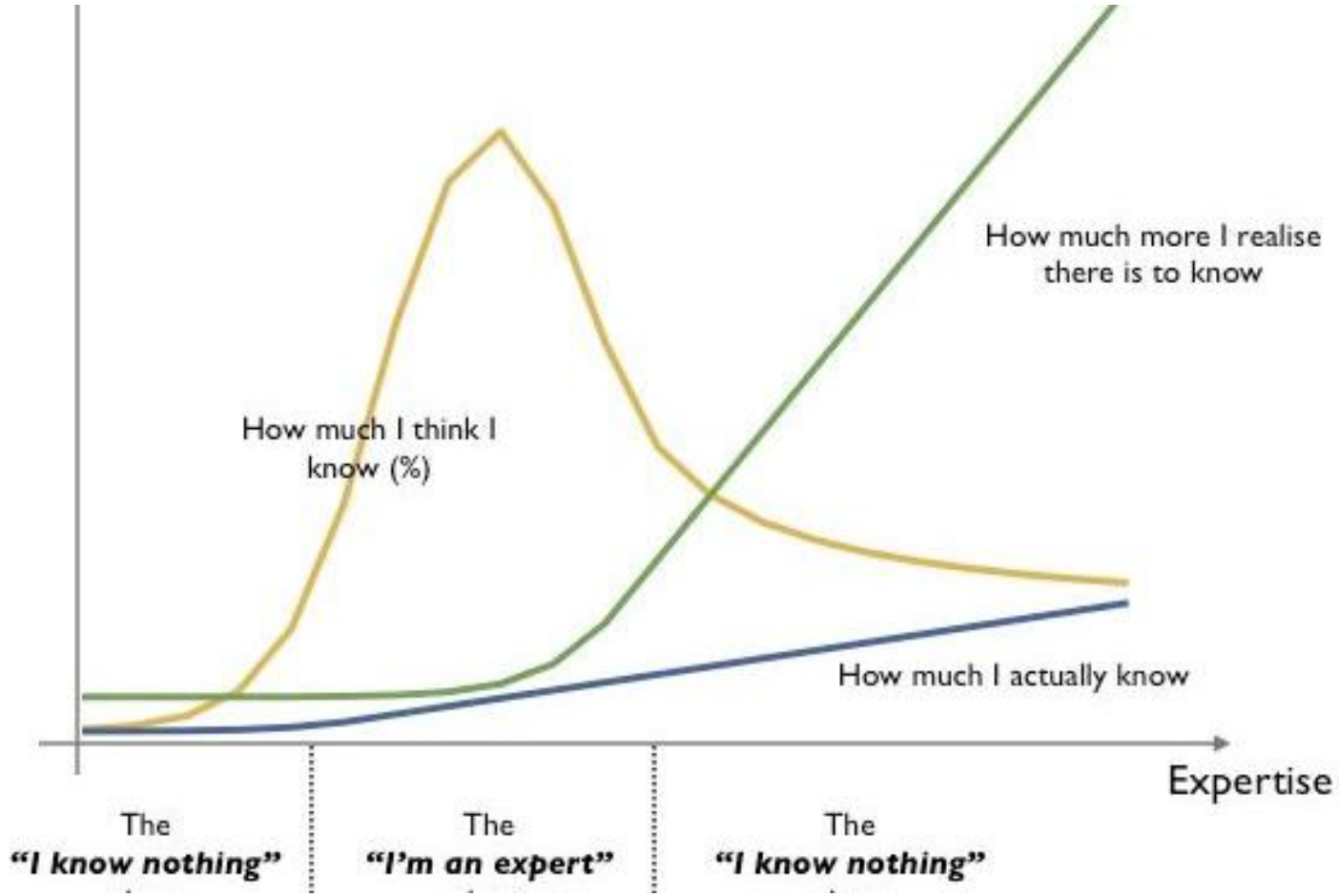
Useful Deep Learning Terms



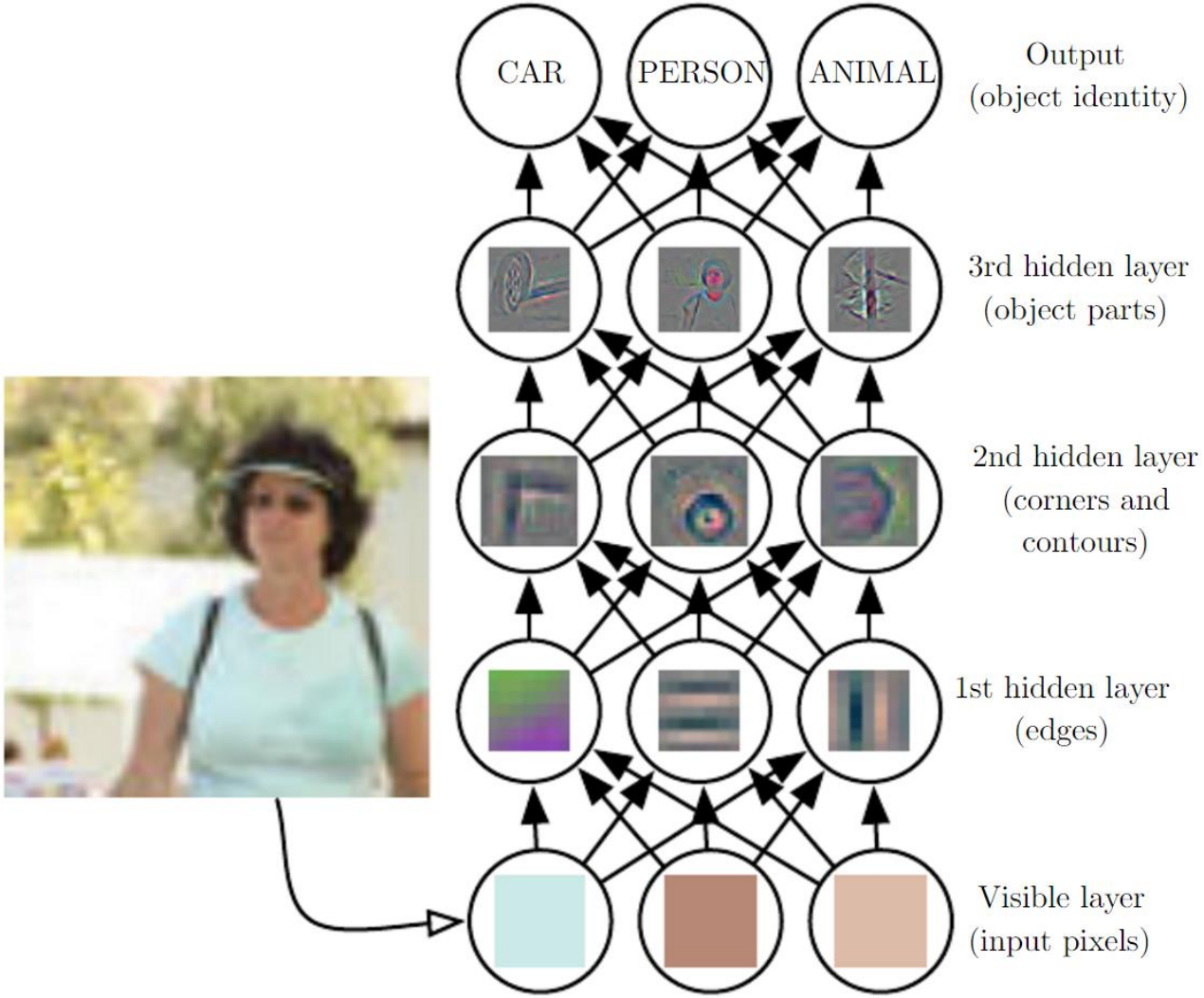
- Basic terms:
  - **Deep Learning = Neural Networks**
  - **Deep Learning** is a subset of **Machine Learning**
- Terms for neural networks:
  - **MLP:** Multilayer Perceptron
  - **DNN:** Deep neural networks
  - **RNN:** Recurrent neural networks
    - **LSTM:** Long Short-Term Memory
  - **CNN or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

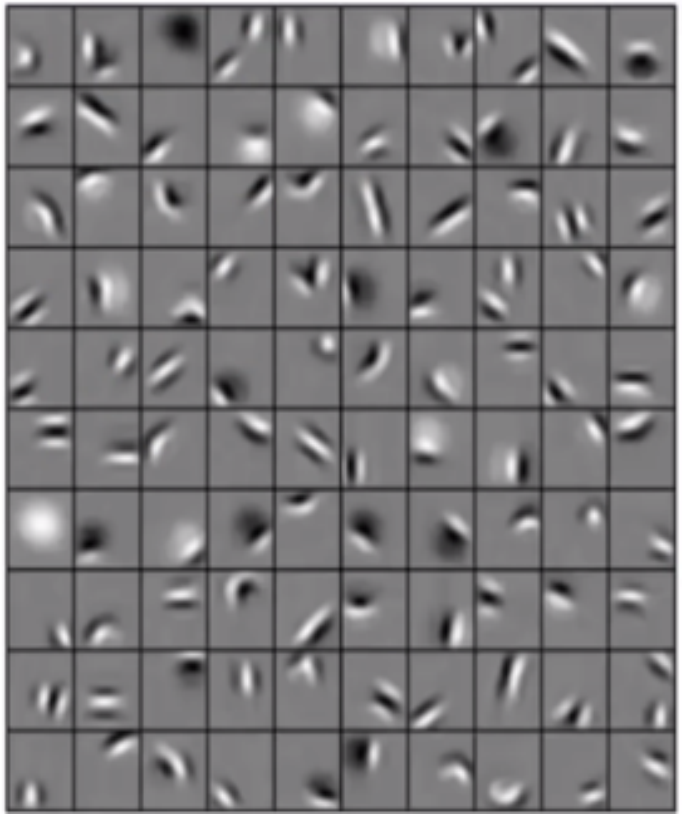
Asimov Institute. “A mostly complete chart of neural networks.”

Neural Networks: Proceed with Caution



Deep Learning is **Representation Learning**



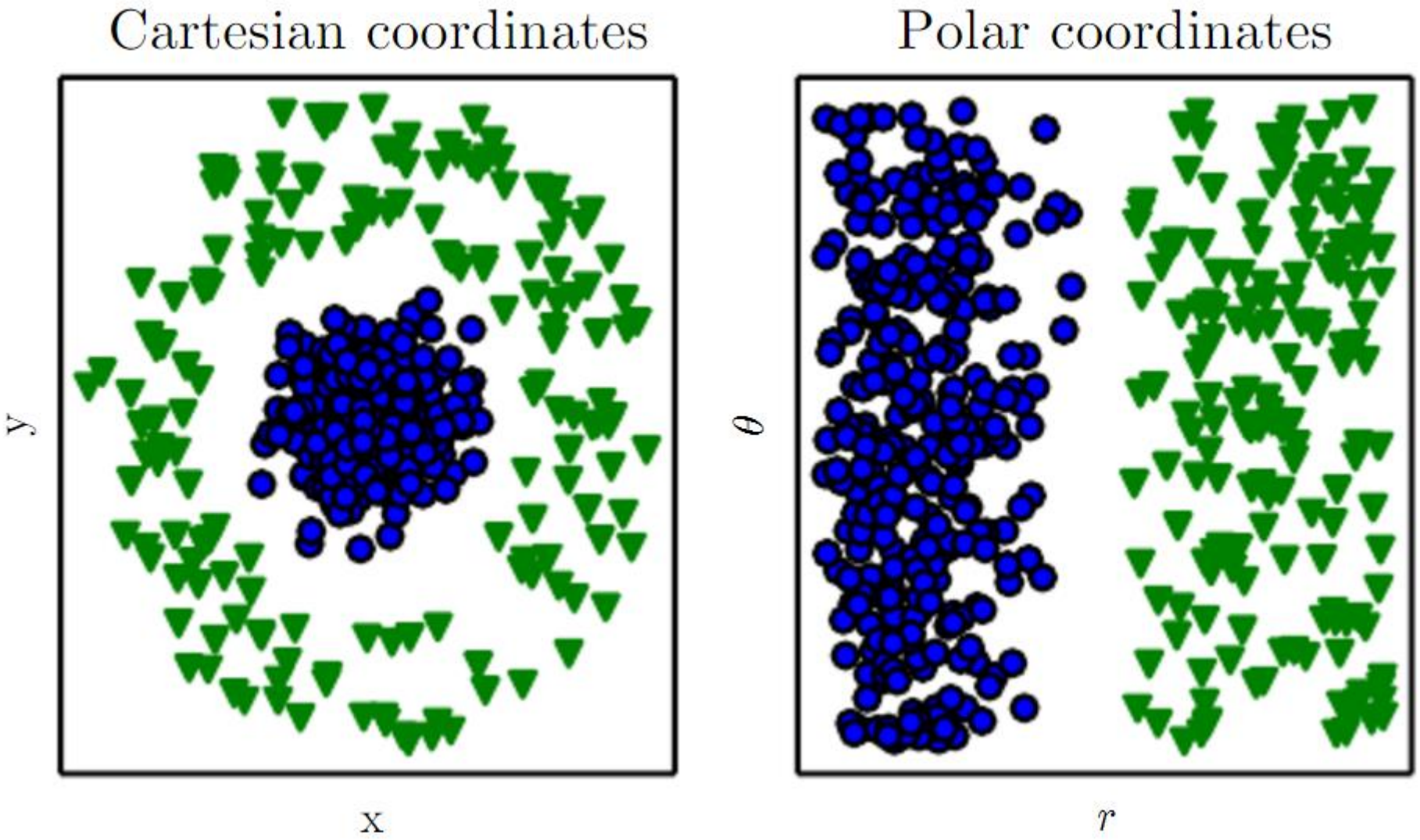


Faces

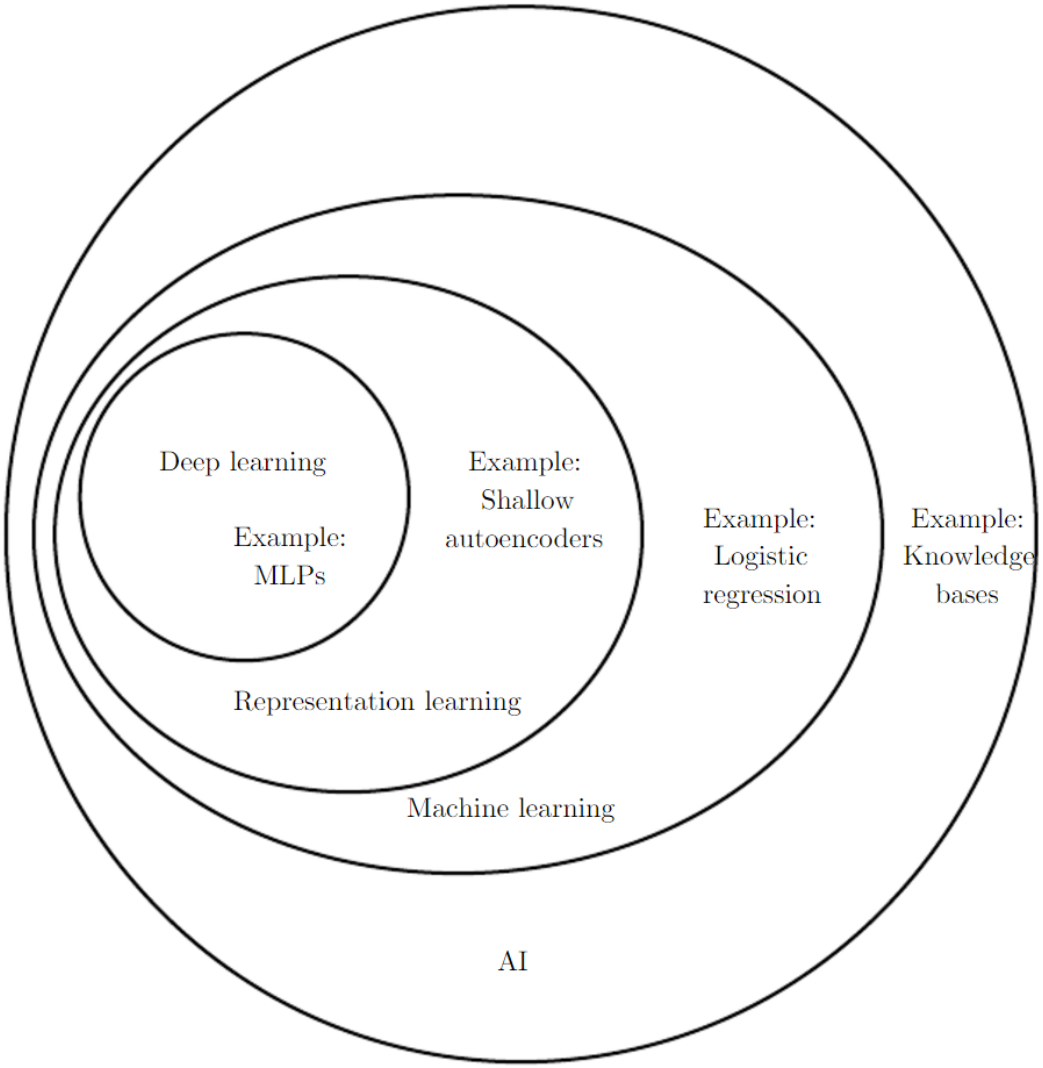
Cars

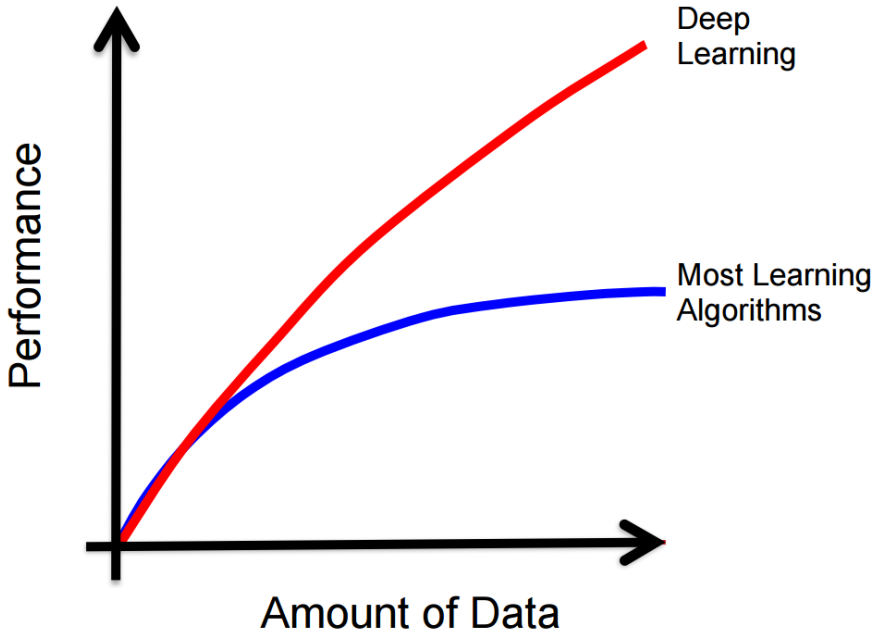
Convolutional Deep Belief Networks
for Scalable Unsupervised Learning of Hierarchical Representations;
Honglak Lee, Roger Grosse
Rajesh Ranganath, Andrew Ng

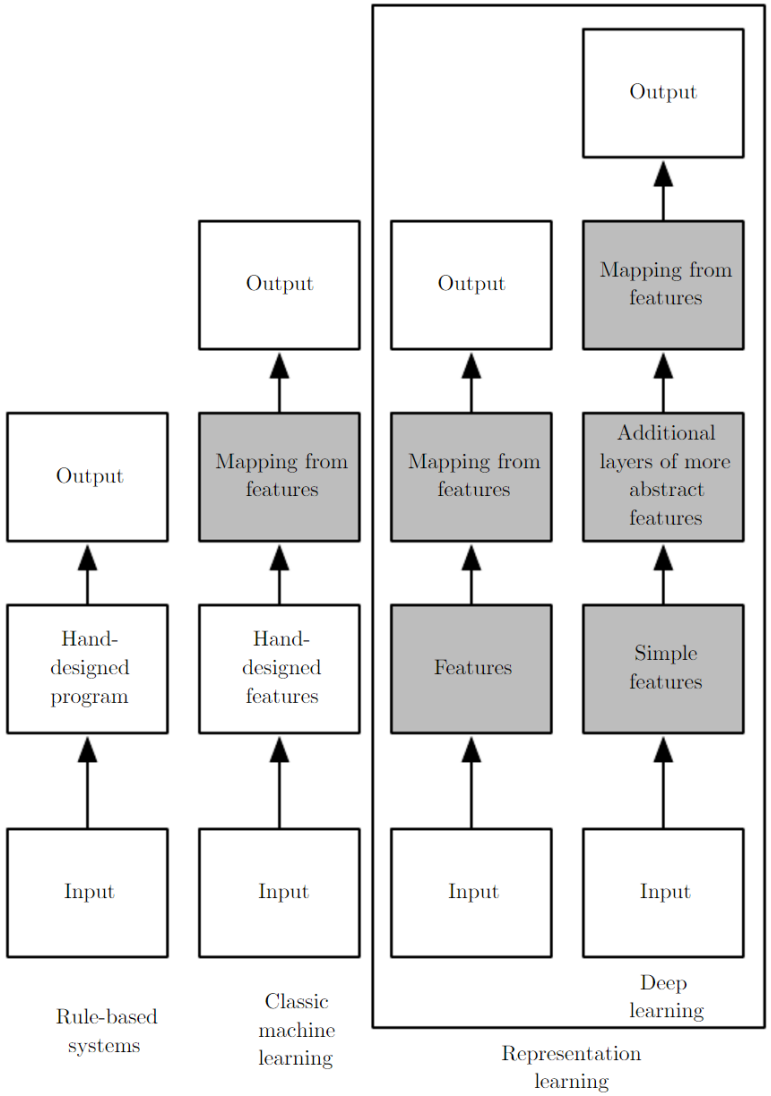
Representation Matters



Deep Learning is **Representation Learning**

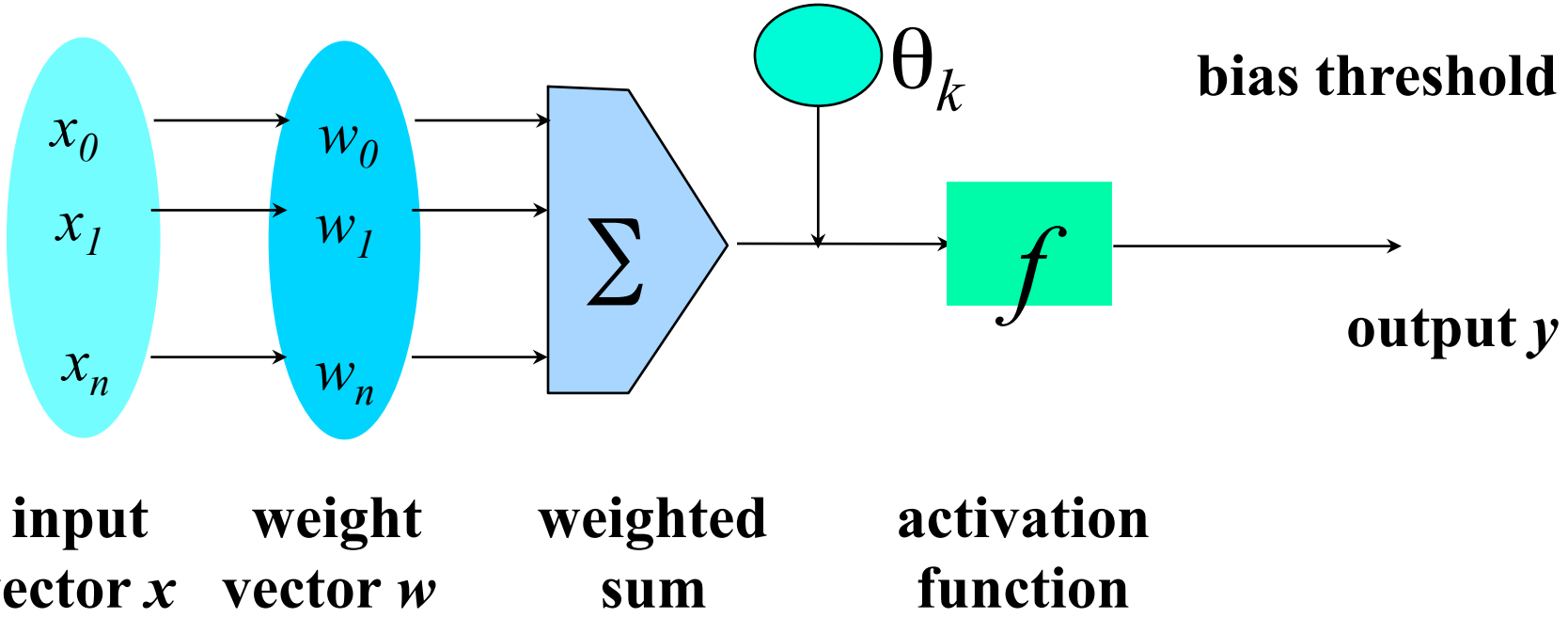



Performance
Amount of Data
Deep Learning
Most Learning Algorithms


Output
Output
Output
Output
Mapping from features
Mapping from features
Mapping from features
Additional layers of more abstract features
Hand-designed program
Hand-designed features
Features
Simple features
Input
Input
Input
Input
Rule-based systems
Classic machine learning
Representation learning
Deep 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