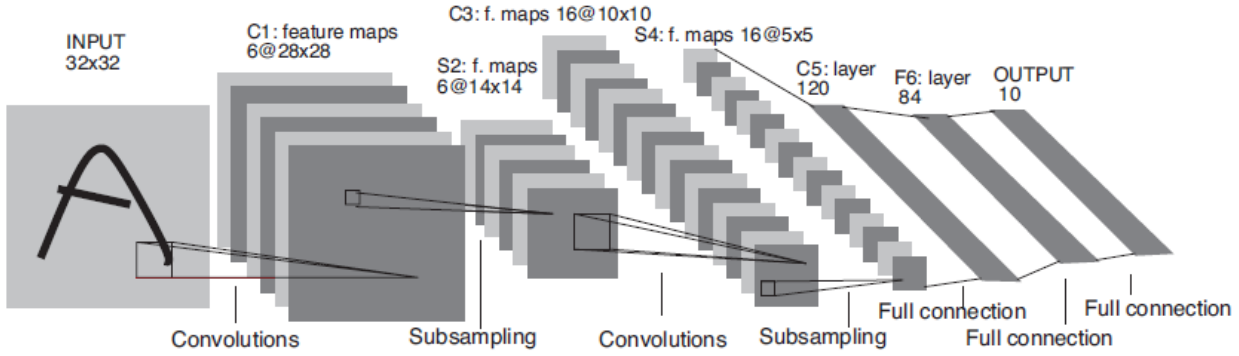
ENEE698A : Machine Learning Seminar

Introduction to Deep Learning



Raviteja Vemulapalli

Image credit: [LeCun 1998]

Resources

- Unsupervised feature learning and deep learning (UFLDL) tutorial (http://ufldl.Stanford.edu/wiki/index.php/UFLDL_Tutorial)

- Y. Bengio, "**Learning Deep Architectures for AI**", Foundations and Trends in Machine Learning, Vol. 2, No. 1, pp. 1–127, 2009, Now publishers.

Overview

- Linear, Logistic and Softmax regression
- Feed forward neural networks
- Why deep models?
- Why is training deep networks (except CNNs) difficult?
- What makes convolutional neural networks (CNNs) relatively easier to train?
- How are deep networks trained today?
- Autoencoders and Denoising Autoencoders

Linear Regression

- Input $x \in R^d$, Output $y \in R$
- Parameters: $\theta = \{w, b\}$
- Model: $\hat{y} = wx + b$
- Squared error cost function : $\ell(\theta) = \sum_{j=1}^{N} \left\| y_j - (wx_j + b) \right\|_2^2$
- Cost function is convex and differentiable.

Logistic Regression

- Used for binary classification.
- Input $x \in R^d$, Output $y \in \{0,1\}$
- Parameters: $\theta = \{w, b\}$
- Model: $P(y = 1 \mid x, \theta) = \frac{1}{1+e^{-(wx+b)}}$
- NLL cost function: $\ell(\theta) = -\log\left[\prod_{j=1}^{N} P\left(y = y_j \mid x_j, \theta\right)\right] = -\sum_{j=1}^{N} \log\left[P\left(y = y_j \mid x_j, \theta\right)\right]$
- Cost function is convex and differentiable.

Softmax Regression

- Used for multi-class classification.
- Input $x \in R^d$, Output $y \in \{1, 2, \ldots, C\}$
- Parameters: $\theta = \{(w_i, b_i)\}_{i=1}^{C}$
- Model: $P(y = i \mid x, \theta) = \frac{e^{(w_i x + b_i)}}{\sum_{k=1}^{C} e^{(w_k x + b_k)}}$
- NLL cost function: $\ell(\theta) = -\log\left[\prod_{j=1}^{N} P\left(y = y_j \mid x_j, \theta\right)\right] = -\sum_{j=1}^{N} \log\left[P\left(y = y_j \mid x_j, \theta\right)\right]$
- Cost function is convex and differentiable.

Regularization & Optimization

- ℓ_2-Regularization (Weight decay)

$$\min_{w,b} \left[\ell(w,b) + \lambda\|w\|_2^2\right]$$

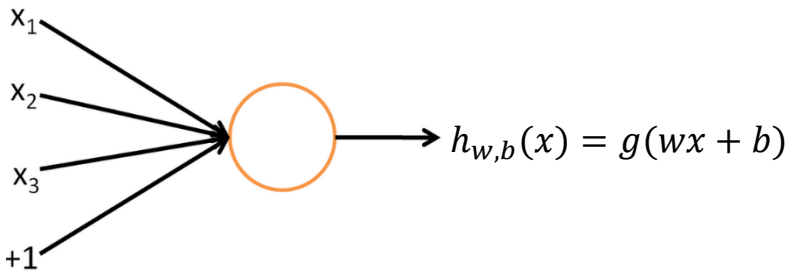
  - Regularization helps to avoid overfitting.
  - ℓ_2-Regularization is equivalent to MAP estimation with zero-mean Gaussian prior on w.

- Optimization
  - $[\ell(w,b) + \lambda\|w\|_2^2]$ is convex and differentiable in the case of linear, logistic and softmax regression.
  - Gradient based optimization techniques can be used to find the global optimum.

Feed Forward Neural Networks

Feed Forward Neural Networks

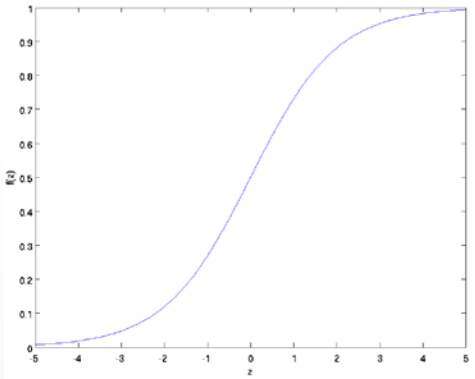
- Artificial neuron



- g is a non-linear function referred to as activation function.

- Logistic and hyperbolic tangent functions are two commonly used activation functions.

Logistic function



Hyperbolic tangent function

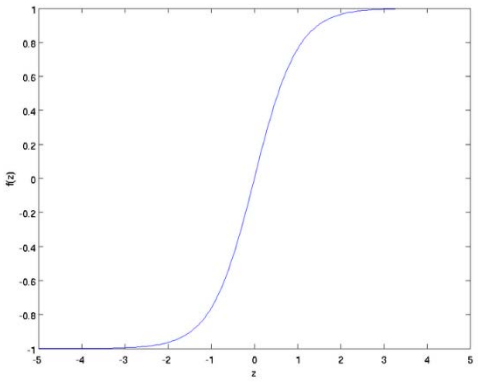
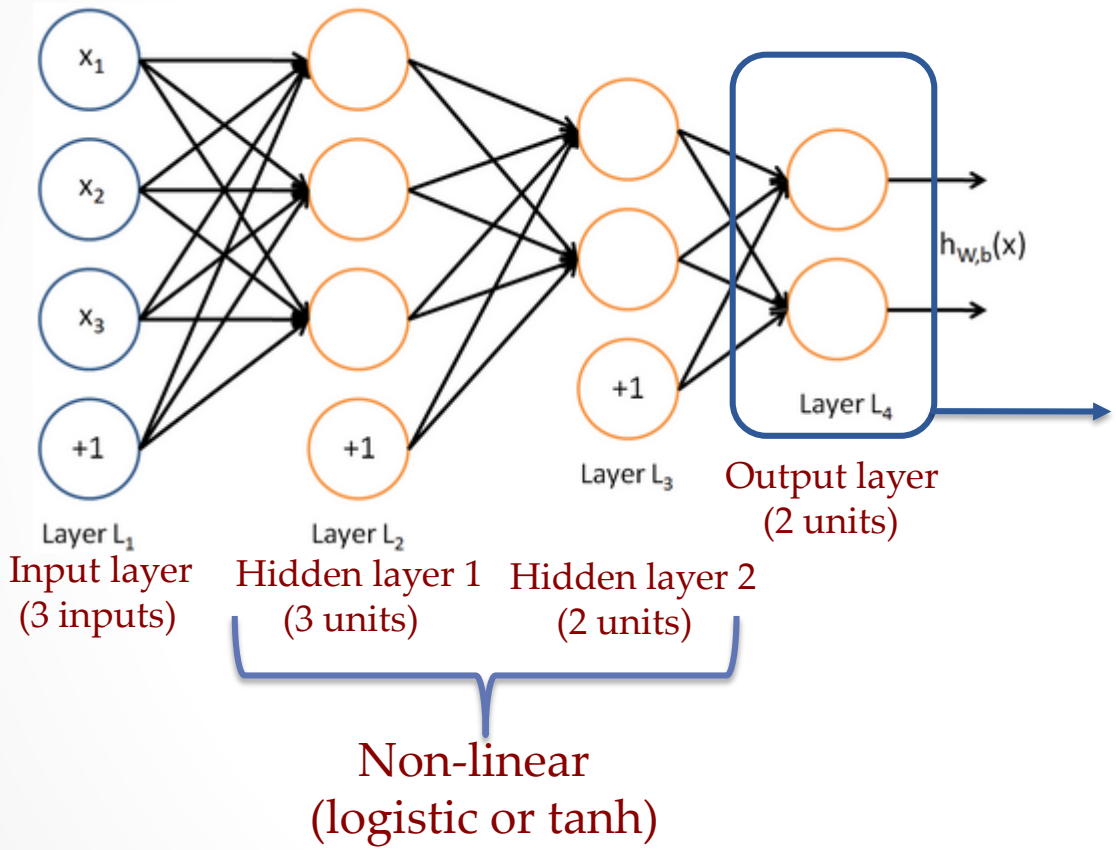


Image credit: UFLDL Tutorial

Feed Forward Neural Networks

- Neural network with 2 hidden layers



Output layer

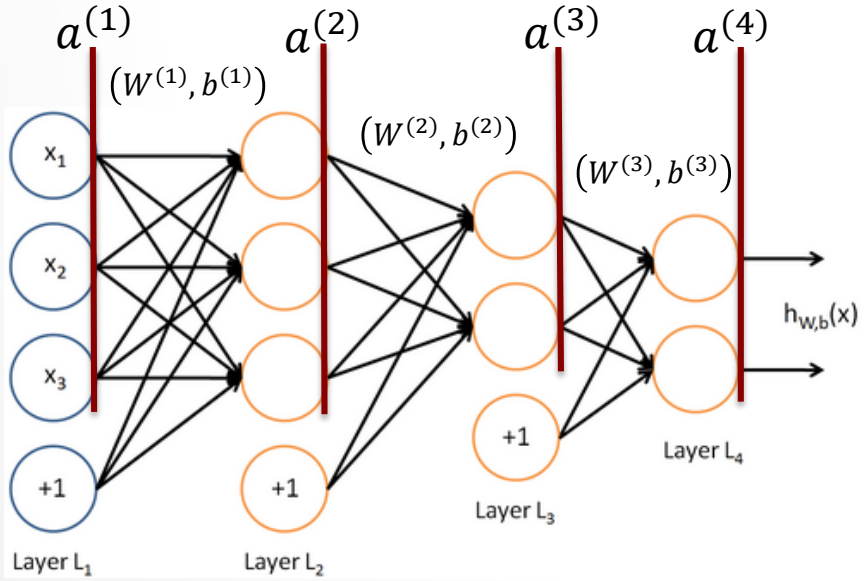
Application	Model
Regression	Linear regression
Binary classification	Logistic regression
Multi-class classification	Softmax regression

- Note that if the activation function is linear, then the entire neural network is effectively equivalent to the output layer.

Image credit: UFLDL Tutorial

Feed Forward Neural Networks - Prediction

- Forward propagation
  - L – Number of layers (including input and output layers)
  - n_i - Number of units in i^{th} layer
  - $a^{(i)} \in R^{n_i}$ - Output of i^{th} layer
  - $W^{(i)} \in R^{n_{i+1} \times n_i}, b^{(i)} \in R^{n_{i+1}}$ - Parameters of the mapping function from layer i to $i+1$.
  - S - Output layer model (linear, logistic or softmax)



$$a^{(1)} = x$$

$$a^{(2)} = g(W^{(1)}a^{(1)} + b^{(1)})$$

$$a^{(L-1)} = g(W^{(L-2)}a^{(L-2)} + b^{(L-2)})$$

$$\vdots$$

$$h_{W,b}(x) = a^{(L)} = S(W^{(L-1)}a^{(L-1)} + b^{(L-1)})$$

Image credit: UFLDL Tutorial

Feed Forward Neural Networks - Learning

- Training samples: $\{(x_i, y_i)\}_{i=1}^{N}$
- Parameters: W - Weights, b - bias

Cost function

- Regression – Linear output layer:

$$J(W, b, x_i, y_i) = \left\| h_{W,b}(x_i) - y_i \right\|_2^2$$

- Binary classification – Logistic output layer

$$J(W, b, x_i, y_i) = -[y_i \log(h_{W,b}(x_i)) + (1 - y_i) \log(1 - h_{W,b}(x_i))]$$

$P(y_i = 1 \mid x_i, W, b)$

- Multi-class classification – Softmax output layer

$$J(W, b, x_i, y_i) = -\sum_{k=1}^{C} \mathbb{I}(y_i = k) \log(h_{W,b}^{k}(x_i))$$

$P(y_i = k \mid x_i, W, b)$

Feed Forward Neural Networks - Learning

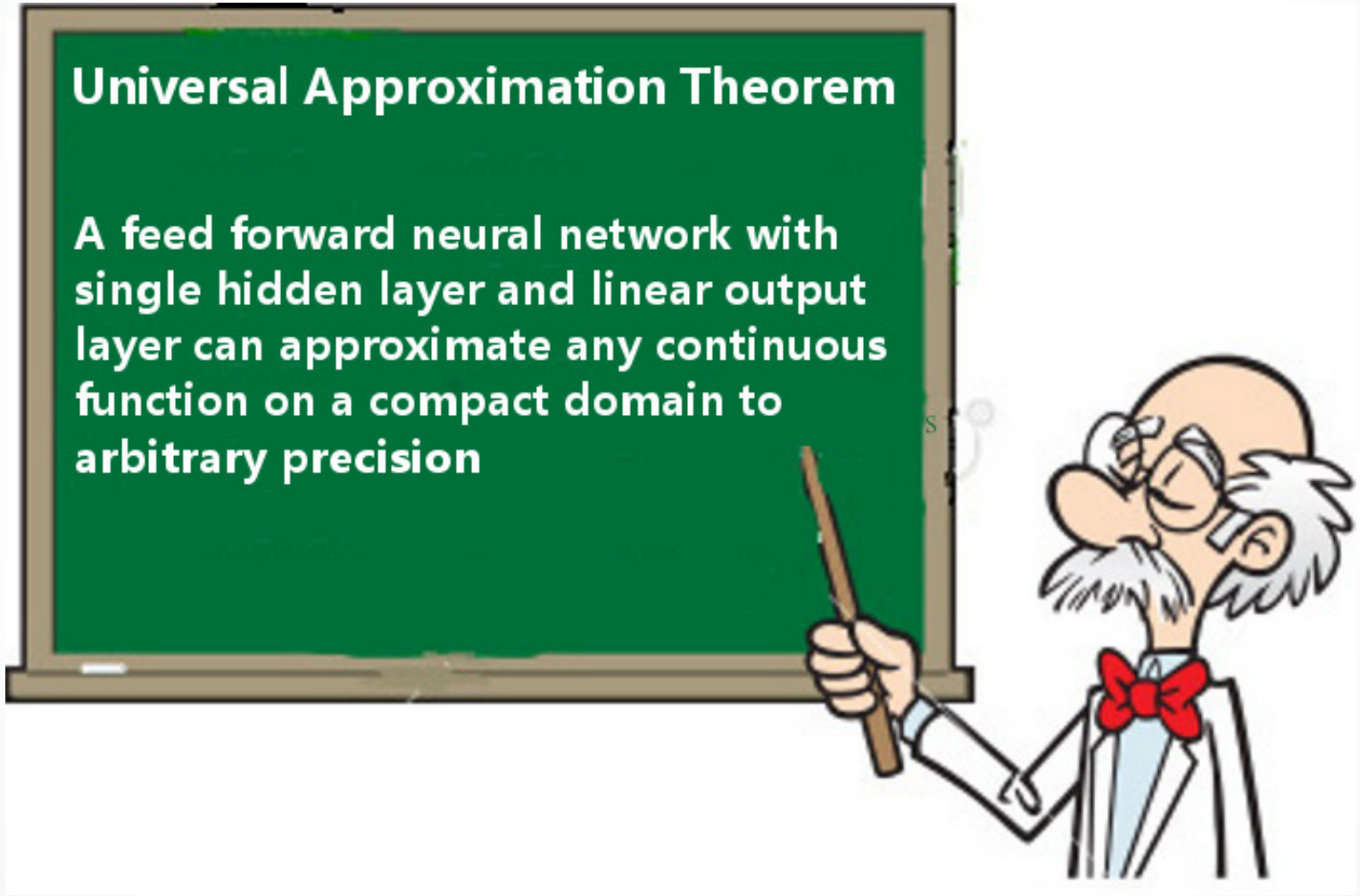
$$\min_{W,b} \sum_{i=1}^{N} J(W, b, x_i, y_i) + \lambda \|W\|_2^2$$

Regularization

- J is non-convex but differentiable.
- Gradient of the cost function with respect to W and b can be computed using backpropagation algorithm.
- Stochastic gradient descent is commonly used to find a local optimum.
- The weights are randomly initialized to small values around zero.

How powerful are neural networks?

Expressive Power of Neural Networks



Professor Kurt Hornik

Expressive Power of Neural Networks

➢ Universal Approximation Theorem [Hornik (1991)]

Let g be any continuous, bounded, non-constant function. Let $X \subseteq R^m$ be a closed and bounded set. Let $C(X)$ denote the space of continuous functions on X. Then, given any function $f \in C(X)$ and $\epsilon > 0$, there exists an integer N, and real constants $\alpha_i, b_i \in R, w_i \in R^m$ for $i = 1, \ldots, N$ such that

$$F(x) = \sum_{i=1}^{N} \alpha_i \, g(w_i' x + b_i)$$

Hidden layer outputs

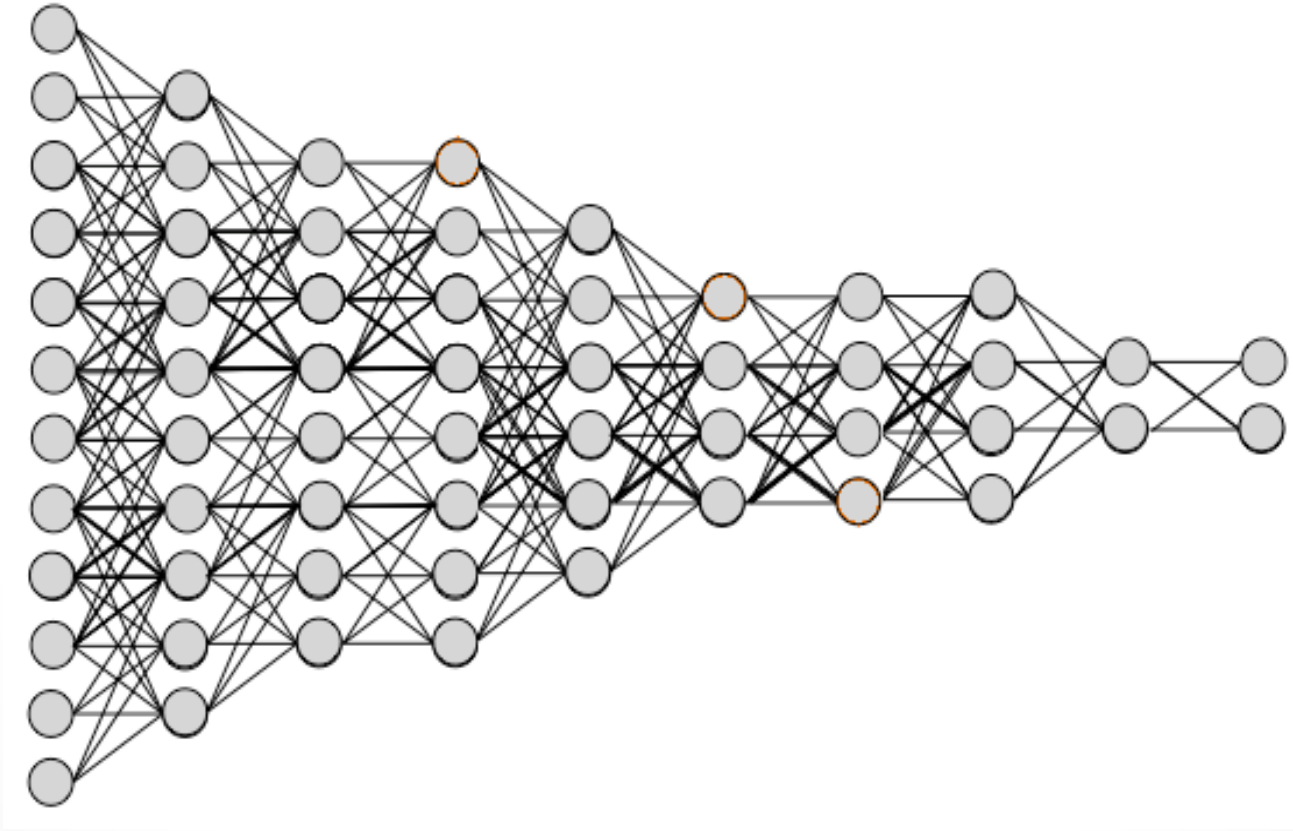
satisfies

$$|F(x) - f(x)| < \epsilon, \forall x \in X.$$

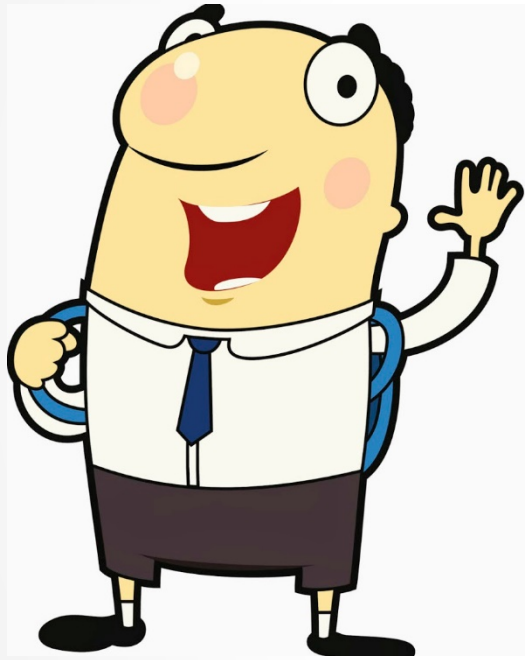
Kurt Hornik, "Approximation capabilities of multilayer feed forward networks", *Neural Networks*, 4:251-257, 1991.

Deep Neural Network

A deep neural network is a neural network with many hidden layers.



Why Deep Networks?



Universal approximation theorem says that one hidden layer is enough to represent any continuous function. Then why deep networks??

Why Deep Networks?

Computational efficiency

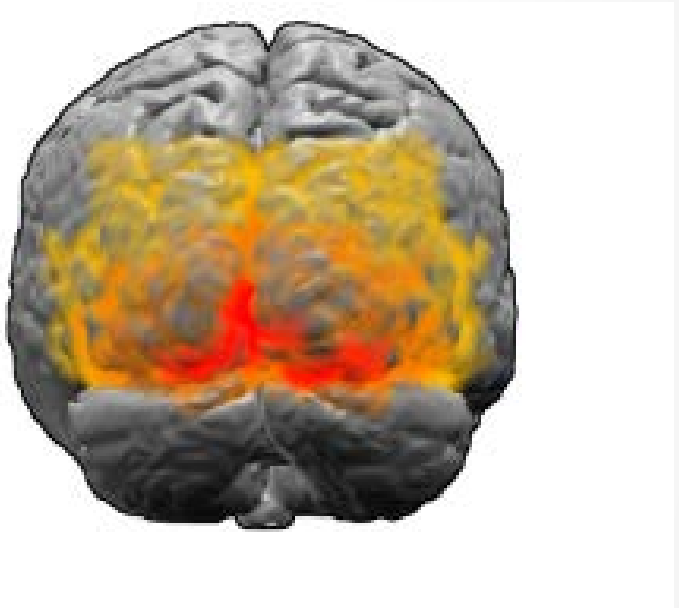
Functions that can be compactly (number of computational units polynomial in the number of inputs) represented by a depth k architecture might require an exponentially large number of computational units in order to be represented by a shallower (depth $< k$) architecture.

In other words, for the same number of computational units, deep architectures have more expressive power than shallow architectures.

Why Deep Networks?

Biological motivation

Visual cortex is organized in a deep architecture (with 5-10 levels) with a given input percept represented at multiple levels of abstraction, each level corresponding to a different area of cortex. [Serre 2007]



T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio, "A quantitative theory of immediate visual recognition," *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function*, vol. 165, pp. 33–56, 2007.

Training Deep Neural Networks

- Prior to 2006, the standard approach to train a neural network was to use gradient based techniques starting from a random initialization.

- Researches have reported positive experimental results with typically one or two hidden layers, but supervised training of deeper networks using random initialization consistently yielded poor results (except for convolutional neural networks).

Why is training deep neural networks difficult?

Training Deep Neural Networks

- Lack of sufficient labeled data
  - Deep neural networks represent very complex data transformations.
  - Supervised training of such complex models requires a large number of labeled training samples.
- Local optima
  - Supervised training of deep networks involves optimizing a cost function that is highly non-convex in its parameters.
  - Gradient based approaches with random initialization usually converge to a very poor local minima.

Training Deep Neural Networks

- Diffusion of gradients
  - The gradients that are propagated backwards rapidly diminish in magnitude as the depth of the network increases.
  - The weights of the earlier layers change very slowly and the earlier layers fail to learn well.
  - When earlier layers do not learn well, a deep network is equivalent to a shallow network (consisting of top layers) on corrupted input (output of the bad earlier layers), and hence may perform badly compared to a shallow network on original input.

Training Deep Neural Networks

What to do??

Training Deep Neural Networks

Lots of free unlabeled data on internet these days!!

Can we use unlabeled data to train deep networks?

Unsupervised Layer-wise Pre-training

- In 2006, Hinton et. al. introduced an unsupervised training approach known as pre-training.
- The main idea of pre-training is to train the layers of the network one at a time, in an unsupervised manner.
- Various unsupervised pre-training approaches have been proposed in the recent past based on restricted Boltzmann machines [Hinton 2006], autoencoders [Bengio 2006], sparse autoencoders [Ranzato 2006], denoising autoencoders [Vincent 2008], etc.
- After unsupervised training of each layer, the learned weights are used for initialization and the entire deep network is fine-tuned in a supervised manner using the labeled training data.

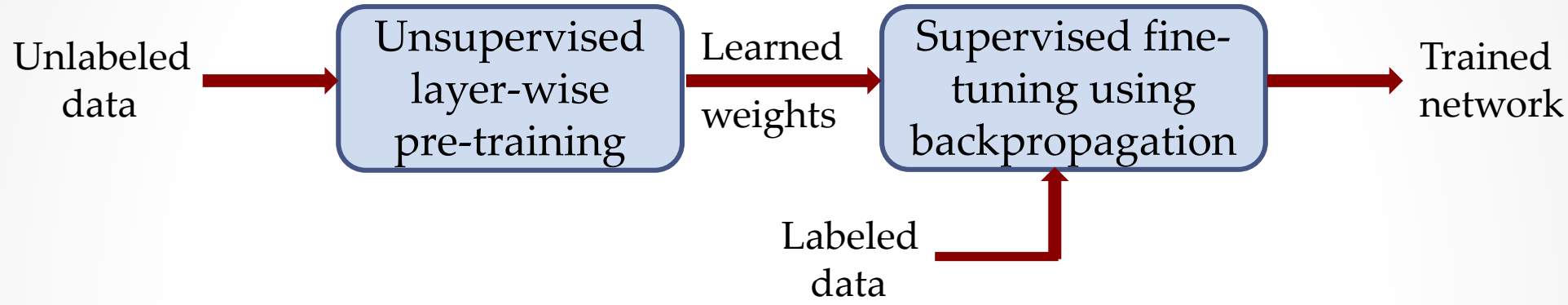
G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," NIPS, 2006.

M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," NIPS, 2006.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," ICML, 2008.

Unsupervised Layer-wise Pre-training

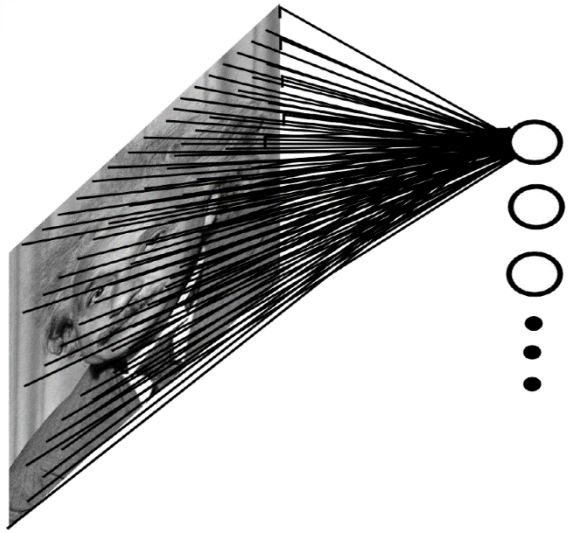


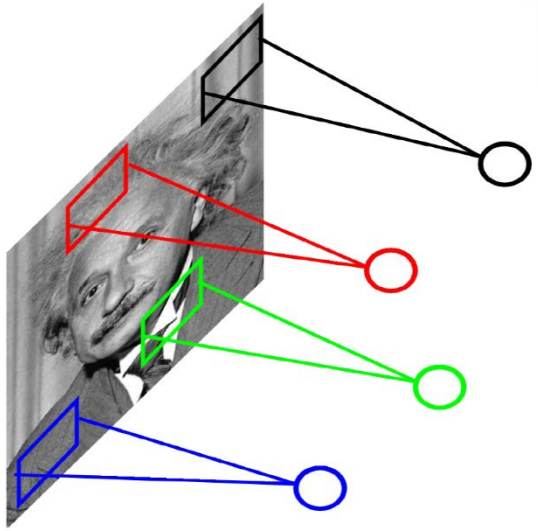
- The weights learned using the layer-wise unsupervised training approach provide a very good initialization compared to random weights.

- Supervised fine-tuning with this initialization usually produces a good local optimum for the entire deep network, because the unlabeled data has already provided a significant amount of prior information.

Convolutional Neural Networks (CNNs)

Fully connected

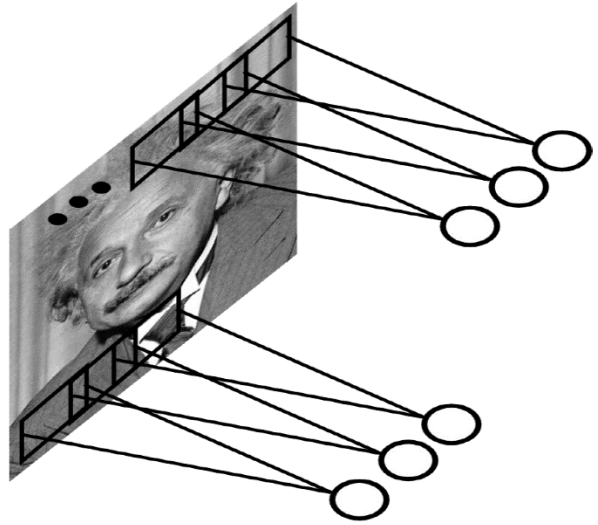


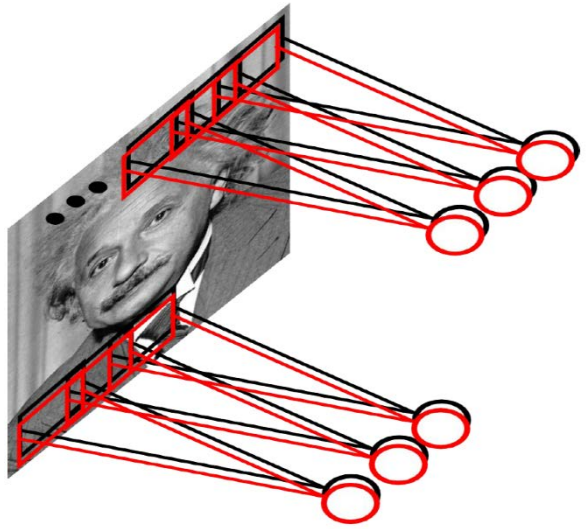


Local connections with different weights

Local connections sharing the weights

Equivalent to convolution by a filter





Convolutional layer
Multiple convolutions using different filters

Image credit: CVPR12 Tutorial on deep learning

Convolutional Neural Networks

- Subsampling or Pooling
  - Reduces computational complexity for upper layers.
  - Pooling layers make the output of convolution networks more robust to local translations.

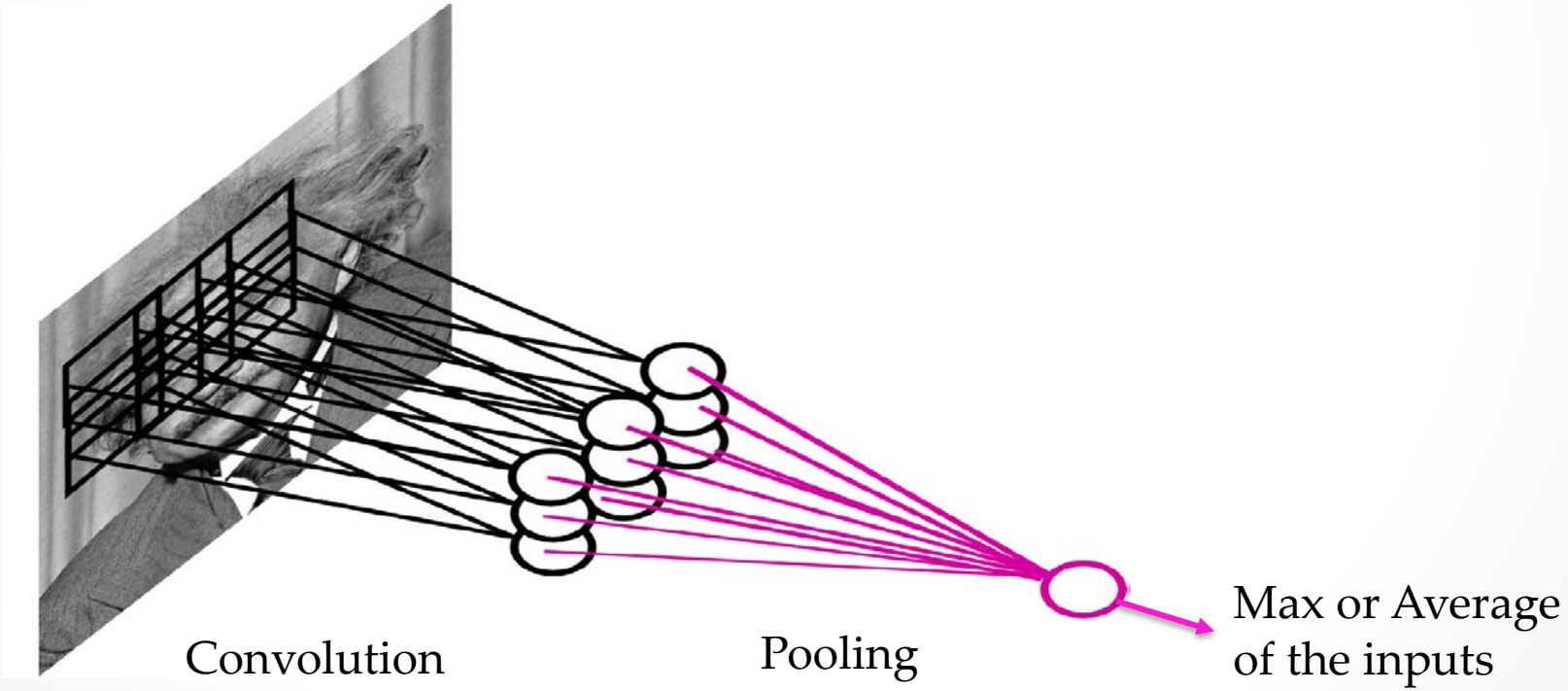
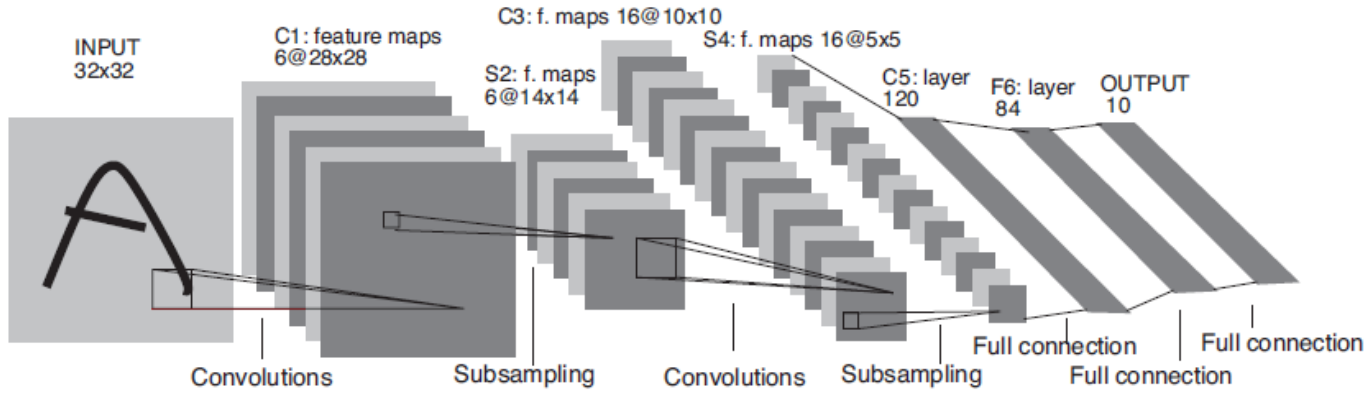


Image credit: CVPR12 Tutorial on deep learning

Convolutional Neural Networks

- CNNs were inspired by the visual system's architecture.
- Modern understanding of the functioning of the visual system is consistent with the processing style found in CNNs [Serre 2007].



LeNet architecture used for character/digit recognition [LeCun 1998].

T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio, "A quantitative theory of immediate visual recognition," *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function*, vol. 165, pp. 33–56, 2007.

Training Deep CNNs

- Supervised training of deep neural networks always yielded poor results except for CNNs.

- Long before 2006, researchers were successful in supervised training of deep CNNs (5-6 hidden layers) without any pre-training.

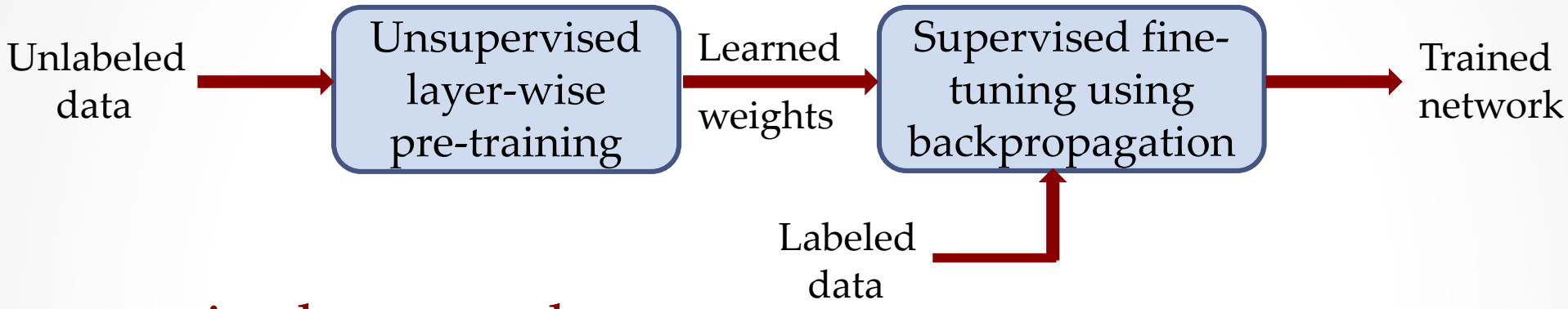
What makes CNNs easier to train?

- The small fan-in of the neurons in a CNN could be helping the gradients to propagate through many layers without diffusing so much as to become useless.

- The local connectivity structure is a very strong prior that is particularly appropriate for vision tasks, and sets the parameters of the whole network in a favorable region (with all non-connections corresponding to zero weight) from which gradient based optimization works well.

Unsupervised Pre-training

Pre-training



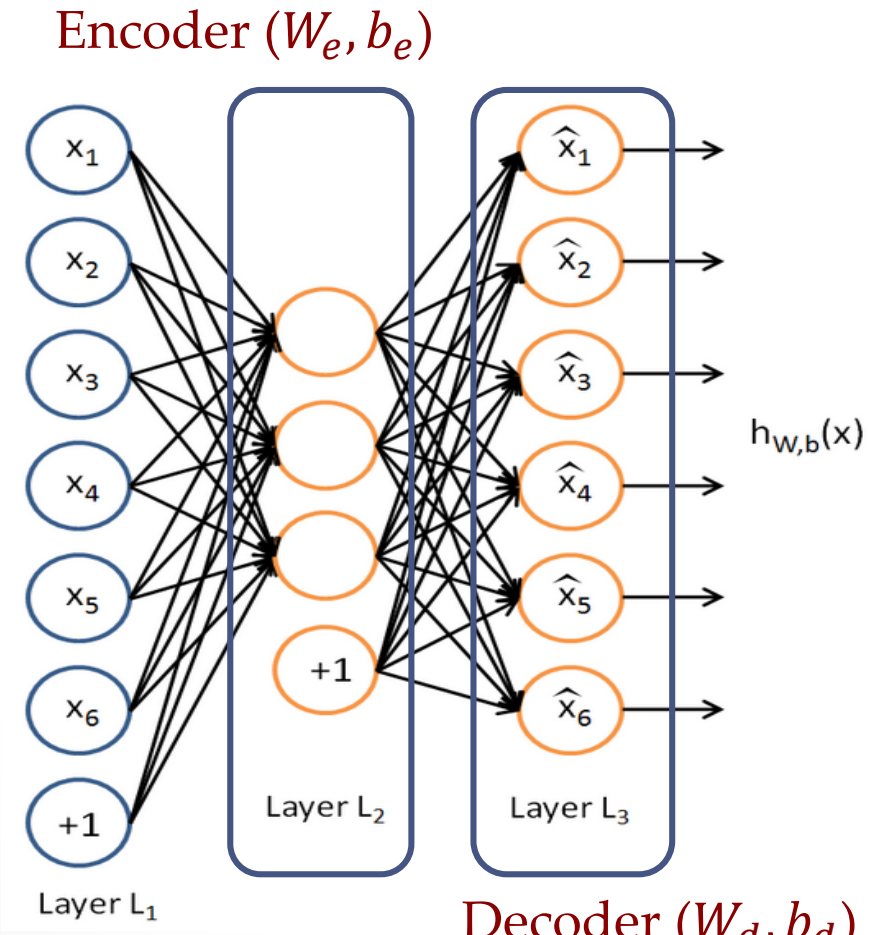
- Unsupervised approaches
  - Restricted Boltzmann machines (RBM)
  - Autoencoders (AE)
  - Sparse autoencoders (SAE)
  - Denoising autoencoders (DAE)

- In this talk we will focus on autoencoders and denoising autoencoders.

Autoencoders

- An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.

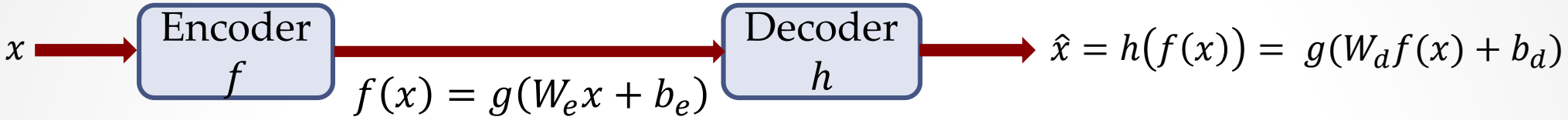
Input $x \in [0,1]^d$



If $W_d = W_e^T$, then the autoencoder is said to have tied weights.

Image credit: UFLDL Tutorial

Autoencoders



- Autoencoders are trained using backpropagation.

- Two common cost functions

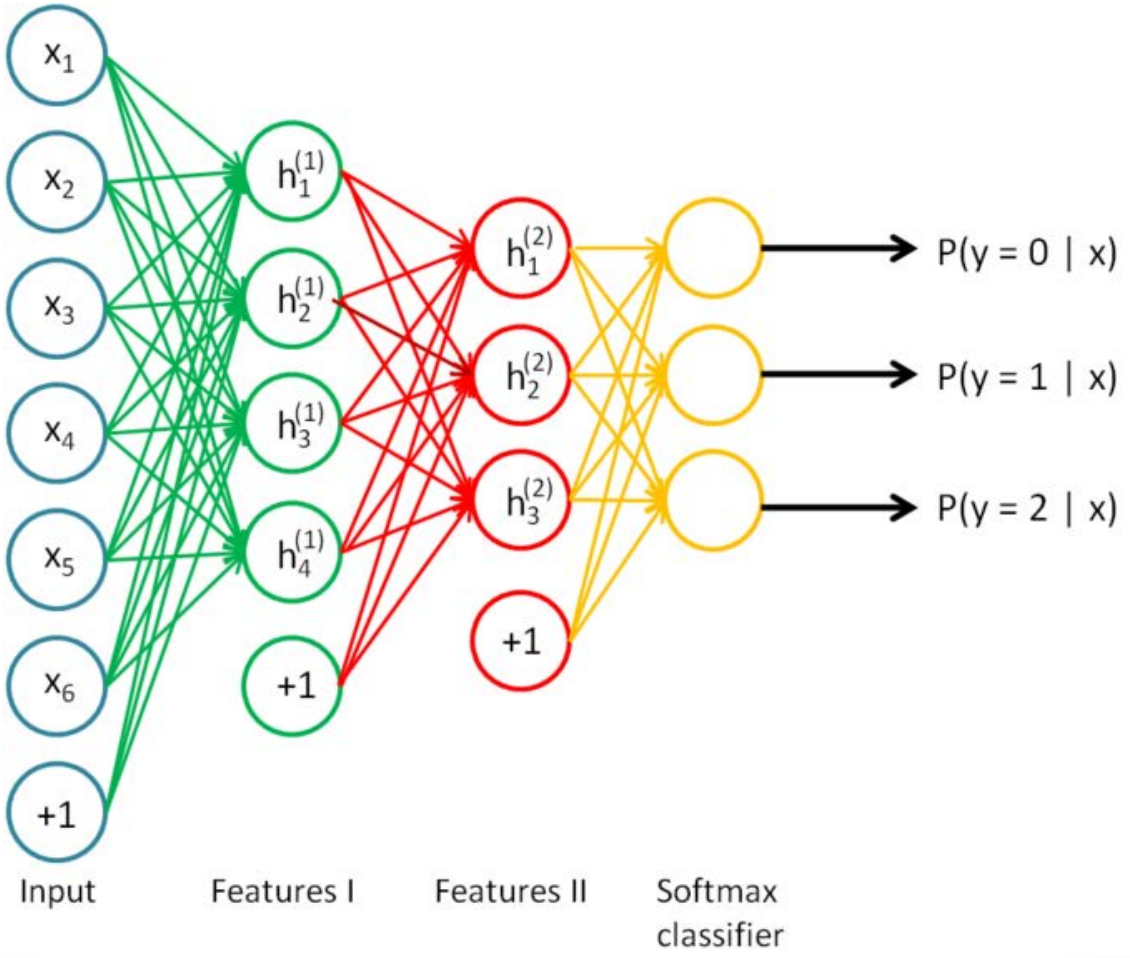
In the case of real-valued inputs: $\sum_{i=1}^{N} \|\hat{x}_i - x_i\|_2^2$

In the case of binary inputs: $\sum_{i=1}^{N} \sum_{j=1}^{d} [x_i^j \log(\hat{x}_i^j) - (1 - x_i^j) \log(1 - \hat{x}_i^j)]$

Autoencoders

- An autoencoder encodes the input x into some representation $f(x)$ so that the input can be reconstructed from that representation.
- If we use neurons with linear activation function and train the network using mean squared error criterion, then autoencoder is similar to PCA.
- When the number of hidden units is smaller than the number of inputs, an autoencoder learns a compressed representation of the data by exploiting the correlations among the input variables.
- An autoencoder can be interpreted as a non-linear dimensionality reduction algorithm.
- An autoencoder can be used for denoising by training it with noisy input.

Unsupervised Pre-training with AEs



Neural network with 2 hidden layers

Image credit: UFLDL Tutorial

Unsupervised Pre-training with AEs

- Step 1
  - Train an autoencoder on the raw input to learn $(W_e^{(1)}, b_e^{(1)})$ and $(W_d^{(1)}, b_d^{(1)})$.
  - Compute the primary features $h^{(1)}(x)$ for each input x using the learned parameters $(W_e^{(1)}, b_e^{(1)})$.

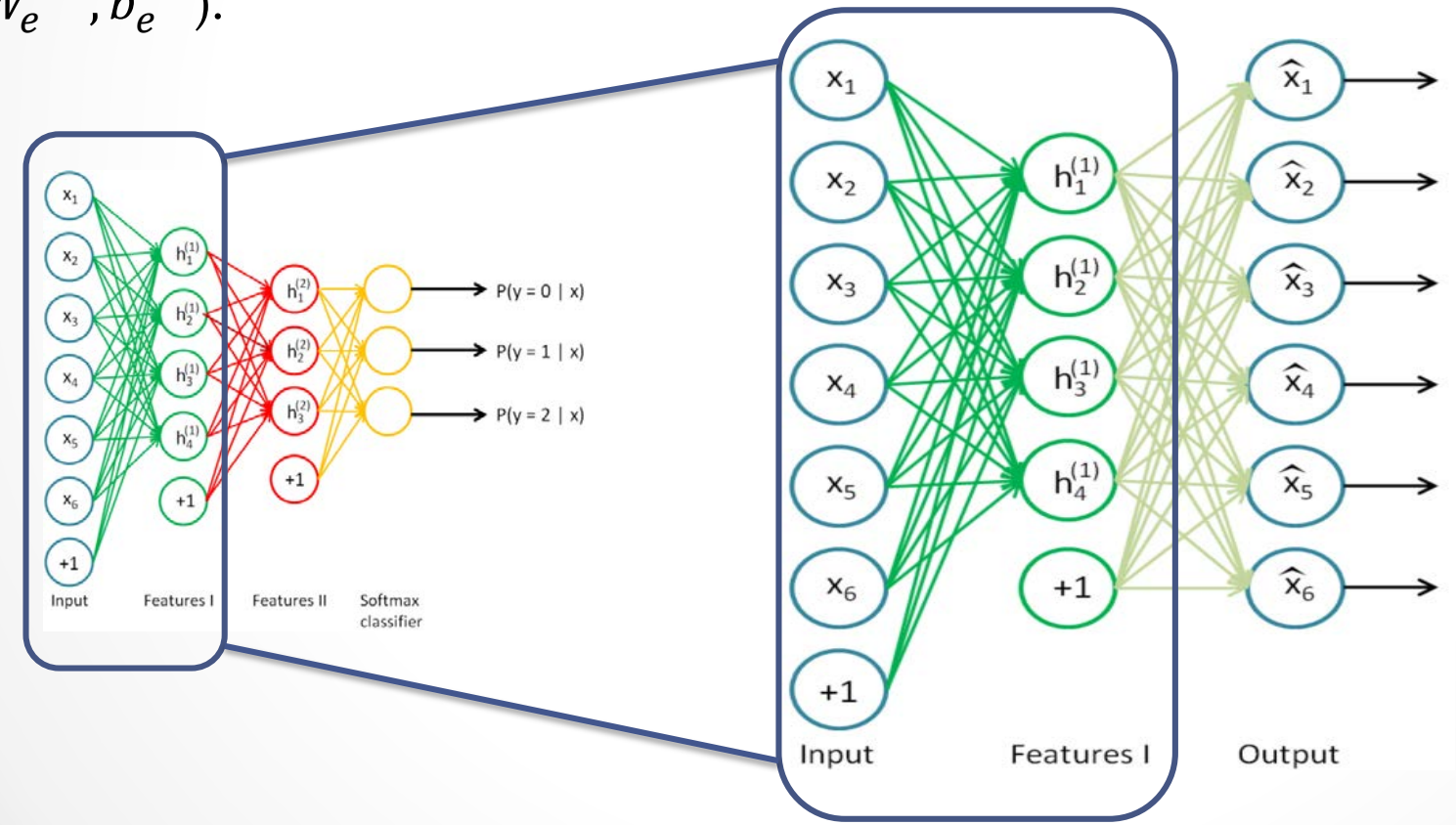


Image credit: UFLDL Tutorial

Unsupervised Pre-training with AEs

- Step 2
  - Use the primary features $h^{(1)}(x)$ as input and train another autoencoder to learn $(W_e^{(2)}, b_e^{(2)})$ and $(W_d^{(2)}, b_d^{(2)})$.
  - Compute the secondary features $h^{(2)}(x)$ for each input $h^{(1)}(x)$ using the learned parameters $(W_e^{(2)}, b_e^{(2)})$.

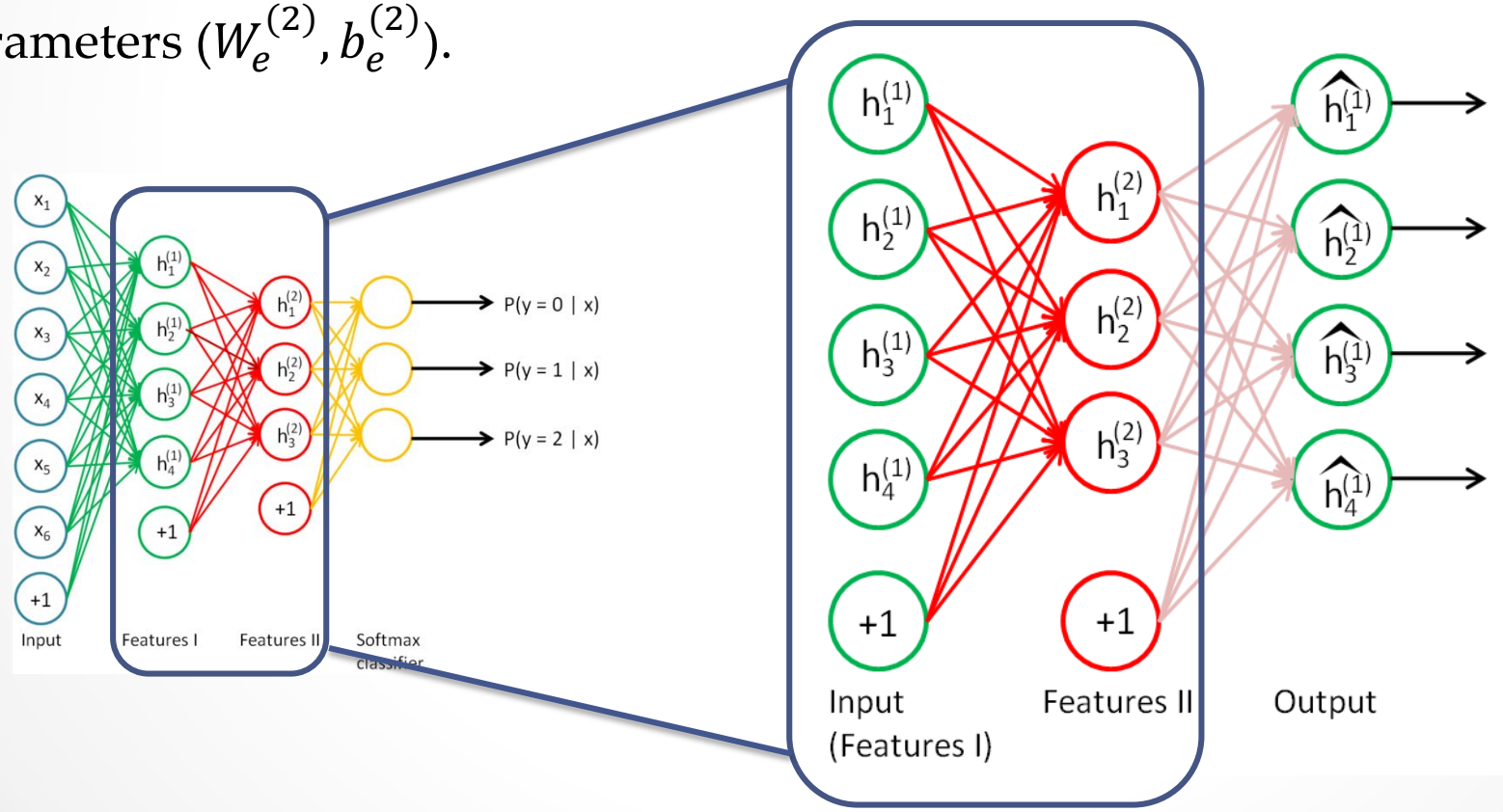


Image credit: UFLDL Tutorial

Supervised Fine-tuning

- Step 3
  - Use $(W_e^{(1)}, b_e^{(1)})$ to initialize the parameters between input and first hidden layer.
  - Use $(W_e^{(2)}, b_e^{(2)})$ to initialize the parameters between first and second hidden layers.
  - Randomly initialize the parameters of final classification layer.
  - Train the entire network in a supervised manner using labeled training data.

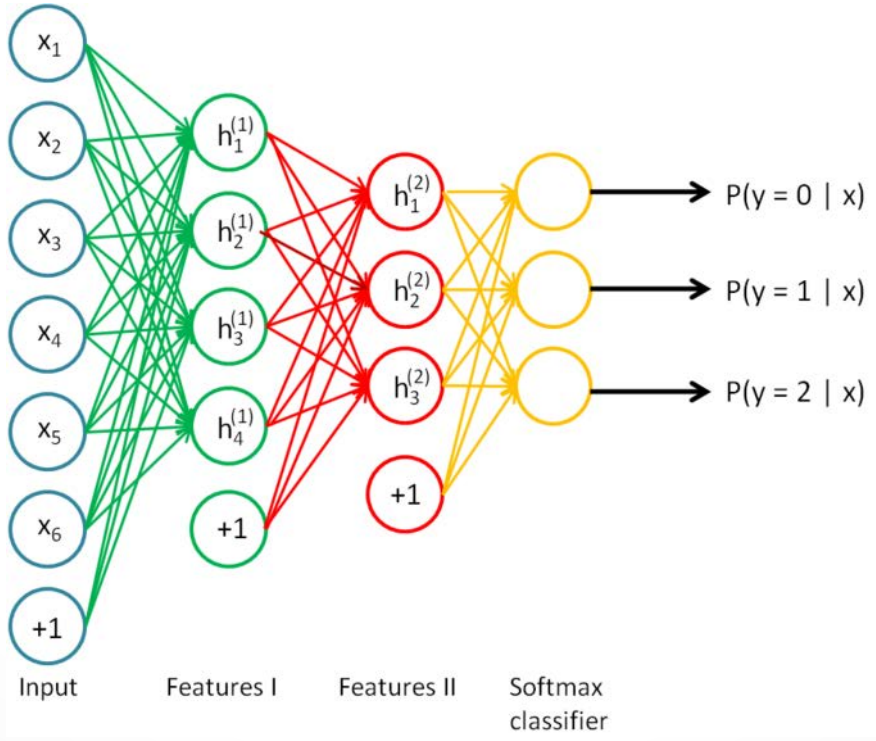


Image credit: UFLDL Tutorial

Denoising Autoencoders

- A denoising autoencoder is nothing but an autoencoder trained to reconstruct a clean input from a corrupted one.

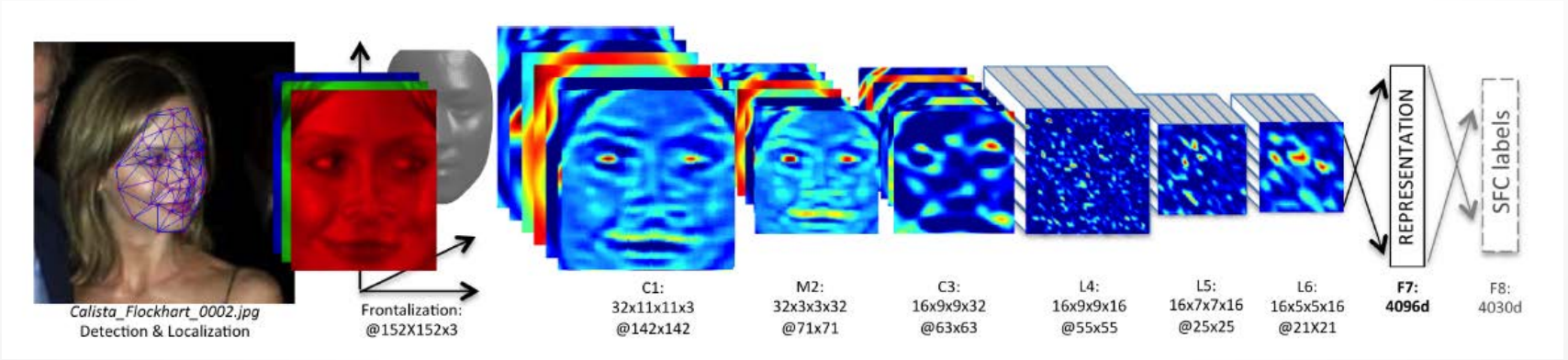
- Motivation
  - Humans can recognize patterns from partially occluded or corrupted data.
  - A good representation should be robust to partial destruction of the data.

- Corrupt the input x to get a partially destroyed version $\tilde{x}$ by means of some stochastic mapping $\tilde{x} \sim q(\tilde{x}|x)$ and train an autoencoder to recover x from $\tilde{x}$.

- Pre-training deep networks using denoising autoencoders was shown to work better than pre-training using ordinary autoencoders on MNIST digit dataset when images were corrupted by randomly choosing some of the pixels and setting their values to zero.

Thank You

. . .



Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato and Lior Wolf, "DeepFace: Closing the Gap to Human-Level performance in Face Verification", CVPR 2014.