Deep Learning: Feedforward Neural Nets and Convolutional Neural Nets

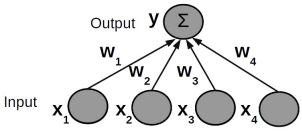
Piyush Rai

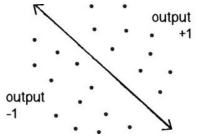
Machine Learning (CS771A)

Nov 2, 2016

A Prelude: Linear Models

- Linear models are nice and simple

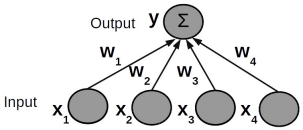


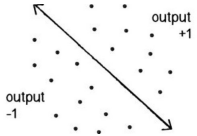


- Were some of the first models for learning from data (e.g., Perceptron, 1958)

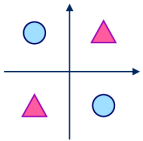
A Prelude: Linear Models

- Linear models are nice and simple



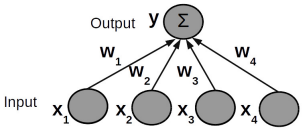


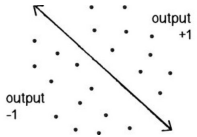
- Were some of the first models for learning from data (e.g., Perceptron, 1958)
- But linear models have limitations: Can't learn nonlinear functions



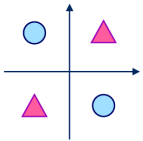
A Prelude: Linear Models

- Linear models are nice and simple





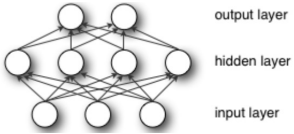
- Were some of the first models for learning from data (e.g., Perceptron, 1958)
- But linear models have limitations: Can't learn nonlinear functions



- Before kernel methods (e.g., SVMs) were invented, people thought about this a lot and tried to come up with ways to address this

Multi-layer Perceptron

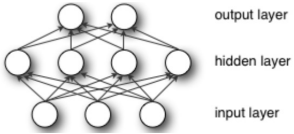
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer

Multi-layer Perceptron

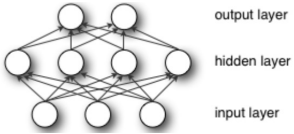
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers compute a nonlinear transform of the inputs

Multi-layer Perceptron

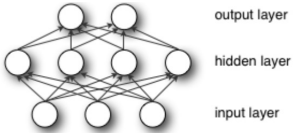
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers compute a nonlinear transform of the inputs
- Also called a Feedforward Neural Network

Multi-layer Perceptron

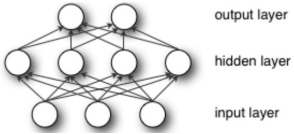
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers compute a nonlinear transform of the inputs
- Also called a Feedforward Neural Network
- "Feedforward": no backward connections between layers (no loops)

Multi-layer Perceptron

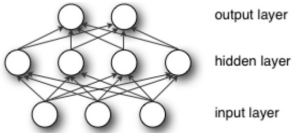
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers compute a nonlinear transform of the inputs
- Also called a Feedforward Neural Network
- "Feedforward": no backward connections between layers (no loops)
- Note: All nodes between layers are assumed connected with each other

Multi-layer Perceptron

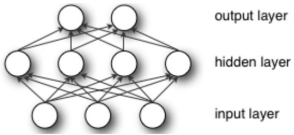
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers compute a nonlinear transform of the inputs
- Also called a Feedforward Neural Network
- "Feedforward": no backward connections between layers (no loops)
- Note: All nodes between layers are assumed connected with each other
- Universal Function Approximator (Hornik, 1991): A one hidden layer FFNN with sufficiently large number of hidden nodes can approximate any function

Multi-layer Perceptron

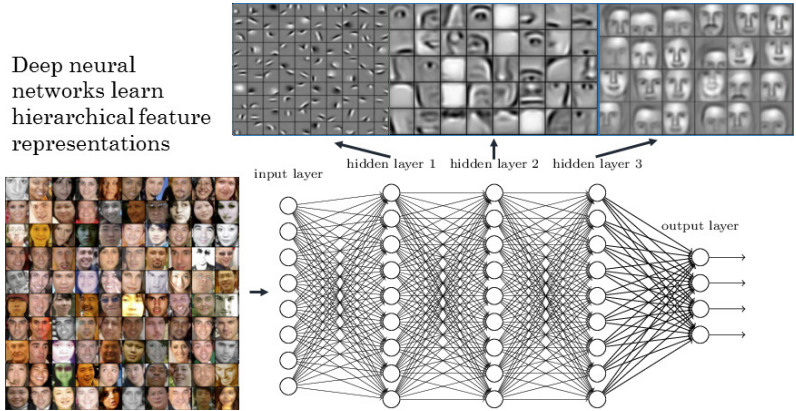
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers compute a nonlinear transform of the inputs
- Also called a Feedforward Neural Network
- "Feedforward": no backward connections between layers (no loops)
- Note: All nodes between layers are assumed connected with each other
- Universal Function Approximator (Hornik, 1991): A one hidden layer FFNN with sufficiently large number of hidden nodes can approximate any function
  - Caveat: This results is only in terms of theoretical feasibility. Learning the model can be very difficult in practice (e.g., due to optimization difficulties)

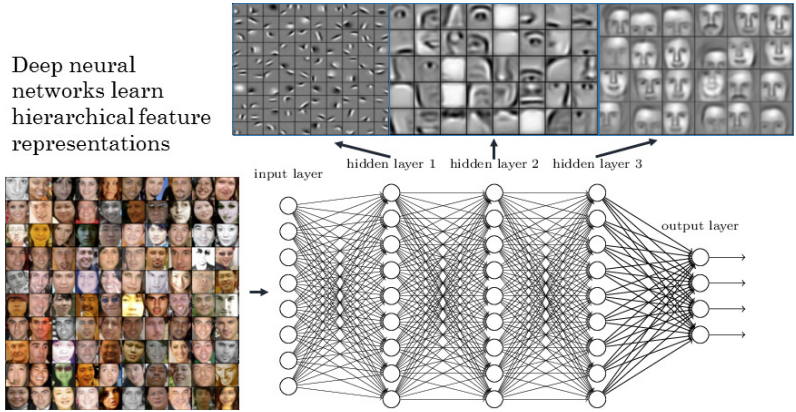
What do Hidden Layers Learn?

- Hidden layers can automatically extract features from data



What do Hidden Layers Learn?

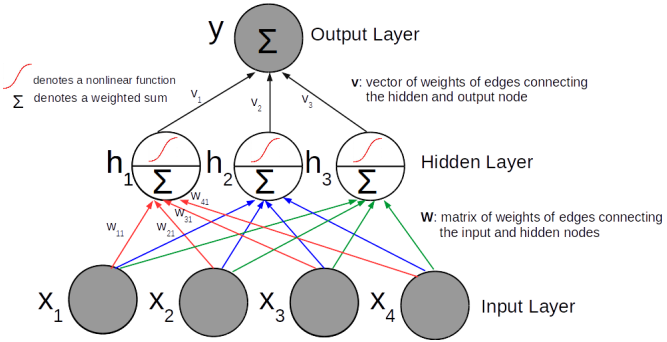
- Hidden layers can automatically extract features from data



- The bottom-most hidden layer captures very low level features (e.g., edges). Subsequent hidden layers learn progressively more high-level features (e.g., parts of objects) that are composed of previous layer's features

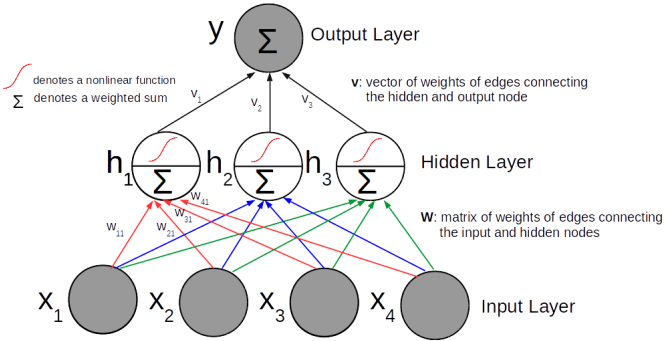
A Simple Feedforward Neural Net

- Below: FFNN with 4 inputs, one hidden layer with 3 nodes, and 1 output



A Simple Feedforward Neural Net

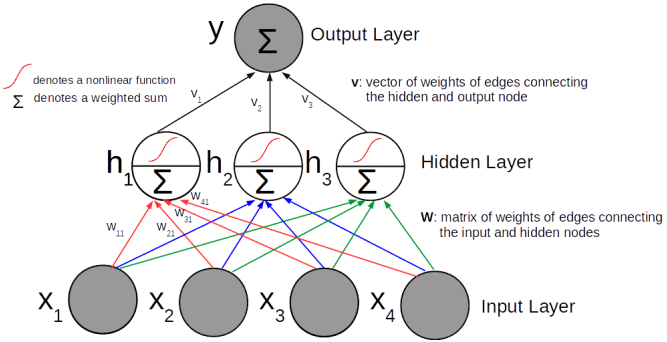
- Below: FFNN with 4 inputs, one hidden layer with 3 nodes, and 1 output



- Each hidden node computes a nonlinear transformation of its incoming inputs

A Simple Feedforward Neural Net

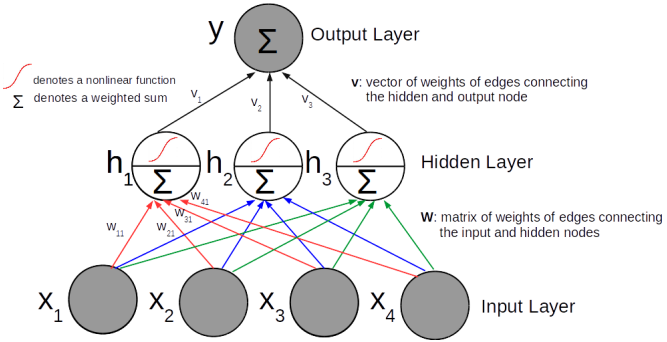
- Below: FFNN with 4 inputs, one hidden layer with 3 nodes, and 1 output



- Each hidden node computes a nonlinear transformation of its incoming inputs
  - Weighted linear combination followed by a nonlinear "activation function"

A Simple Feedforward Neural Net

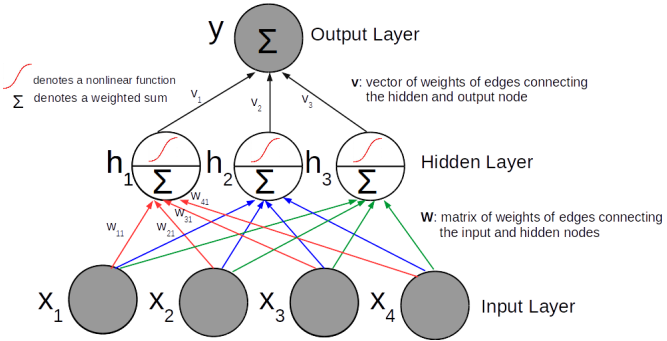
- Below: FFNN with 4 inputs, one hidden layer with 3 nodes, and 1 output



- Each hidden node computes a nonlinear transformation of its incoming inputs
  - Weighted linear combination followed by a nonlinear "activation function"
  - Nonlinearity required. Otherwise, the model would reduce to a linear model

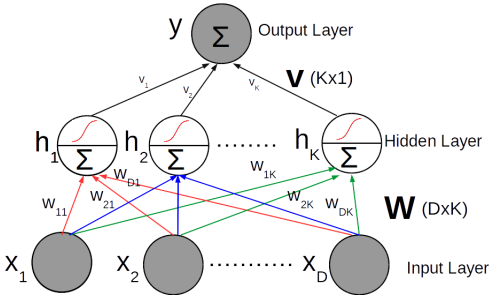
A Simple Feedforward Neural Net

- Below: FFNN with 4 inputs, one hidden layer with 3 nodes, and 1 output



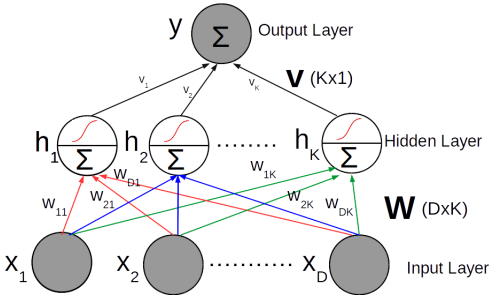
- Each hidden node computes a nonlinear transformation of its incoming inputs
  - Weighted linear combination followed by a nonlinear "activation function"
  - Nonlinearity required. Otherwise, the model would reduce to a linear model
- Output y is a weighted comb. of the preceding layer's hidden nodes (followed by another transform if y isn't real valued, e.g., binary/multiclass label)

Feedforward Neural Net



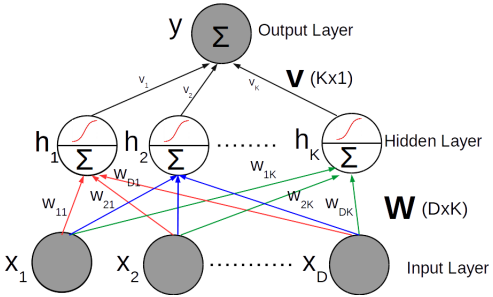
- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$

Feedforward Neural Net



- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$, a single hidden layer with K hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_K]$

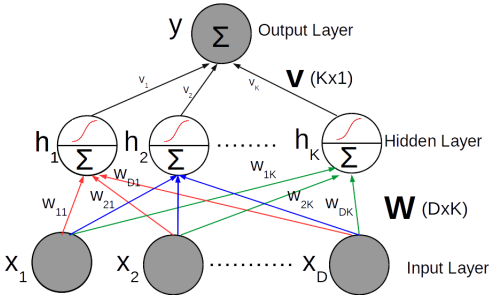
Feedforward Neural Net



- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$, a single hidden layer with K hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node y

$$y = \boldsymbol{v}^\top \boldsymbol{h}$$

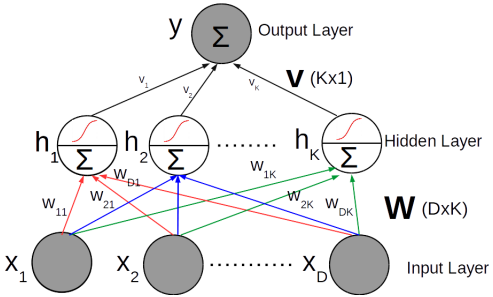
Feedforward Neural Net



- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$, a single hidden layer with K hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node y

$$y = \boldsymbol{v}^\top \boldsymbol{h} = \boldsymbol{v}^\top f(\mathbf{W}^\top \boldsymbol{x})$$

Feedforward Neural Net

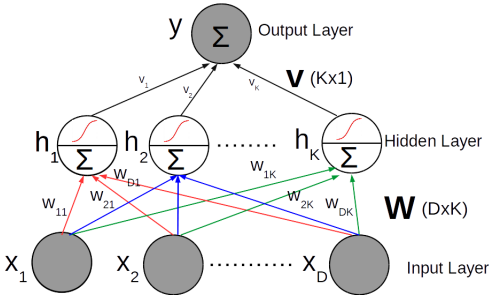


- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$, a single hidden layer with K hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node y

$$y = \boldsymbol{v}^\top \boldsymbol{h} = \boldsymbol{v}^\top f(\mathbf{W}^\top \boldsymbol{x})$$

where $\boldsymbol{v} = [v_1 \ v_2 \ \ldots \ v_K] \in \mathbb{R}^K$

Feedforward Neural Net

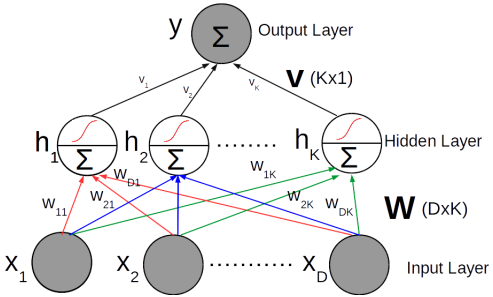


- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$, a single hidden layer with K hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node y

$$y = \boldsymbol{v}^\top \boldsymbol{h} = \boldsymbol{v}^\top f(\mathbf{W}^\top \boldsymbol{x})$$

where $\boldsymbol{v} = [v_1 \; v_2 \; \ldots \; v_K] \in \mathbb{R}^K$, $\mathbf{W} = [\boldsymbol{w}_1 \; \boldsymbol{w}_2 \; \ldots \; \boldsymbol{w}_K] \in \mathbb{R}^{D \times K}$

Feedforward Neural Net

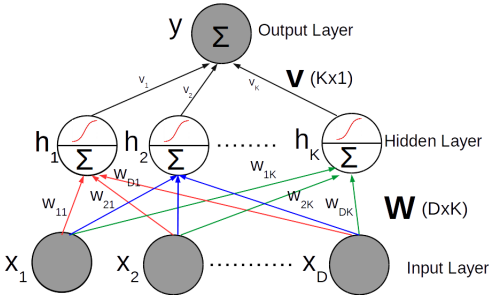


- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$, a single hidden layer with K hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node y

$$y = \boldsymbol{v}^\top \boldsymbol{h} = \boldsymbol{v}^\top f(\mathbf{W}^\top \boldsymbol{x})$$

where $\boldsymbol{v} = [v_1 \ v_2 \ \ldots \ v_K] \in \mathbb{R}^K$, $\mathbf{W} = [\boldsymbol{w}_1 \ \boldsymbol{w}_2 \ \ldots \ \boldsymbol{w}_K] \in \mathbb{R}^{D \times K}$, f is the nonlinear activation function

Feedforward Neural Net



- For an FFNN with D inputs $\boldsymbol{x} = [x_1, \ldots, x_D]$, a single hidden layer with K hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node y

$$y = \boldsymbol{v}^\top \boldsymbol{h} = \boldsymbol{v}^\top f(\mathbf{W}^\top \boldsymbol{x})$$

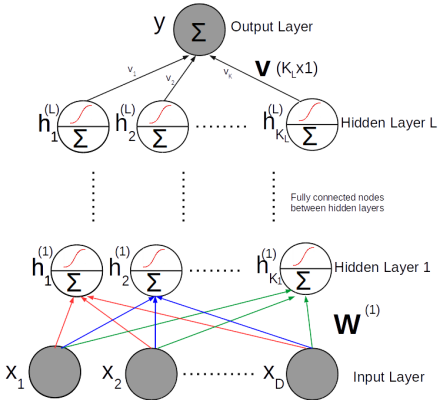
  where $\boldsymbol{v} = [v_1 \; v_2 \; \ldots \; v_K] \in \mathbb{R}^K$, $\mathbf{W} = [\boldsymbol{w}_1 \; \boldsymbol{w}_2 \; \ldots \; \boldsymbol{w}_K] \in \mathbb{R}^{D \times K}$, f is the nonlinear activation function

- Each hidden node's value is computed as: $h_k = f(\boldsymbol{w}_k^\top \boldsymbol{x}) = f(\sum_{d=1}^{D} w_{dk} x_d)$

(Deeper) Feedforward Neural Net

- Feedforward neural net with L hidden layers $\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \ldots, \boldsymbol{h}^{(L)}$ where

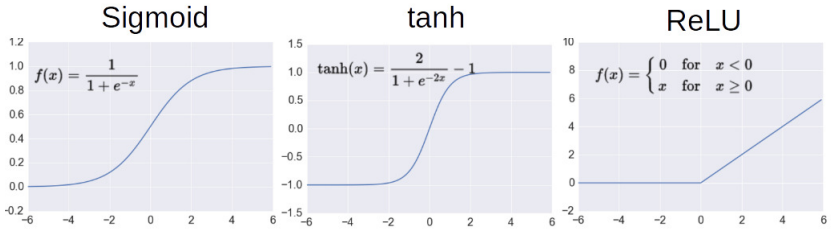
$$\boldsymbol{h}^{(1)} = f(\mathbf{W}^{(1)\top}\boldsymbol{x}) \quad \text{and} \quad \boldsymbol{h}^{(\ell)} = f(\mathbf{W}^{(\ell)\top}\boldsymbol{h}^{(\ell-1)}), \ell \geq 2$$



- Note: The hidden layer ℓ contains K_ℓ hidden nodes, $\mathbf{W}^{(1)}$ is of size $D \times K_1$, $\mathbf{W}^{(\ell)}$ for $\ell \geq 2$ is of size $K_\ell \times K_{\ell+1}$, $\boldsymbol{v}$ is of size $K_L \times 1$

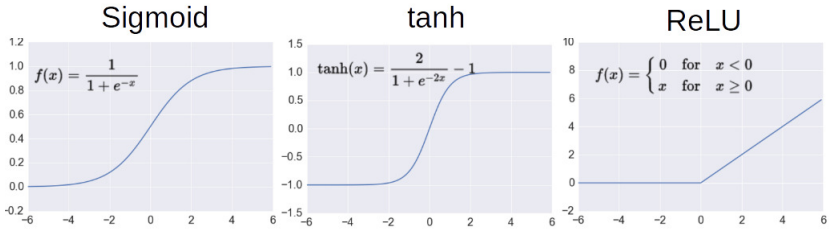
Nonlinear Activation Functions

- Some popular choices for the nonlinear activation function f
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and $+1$)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



Nonlinear Activation Functions

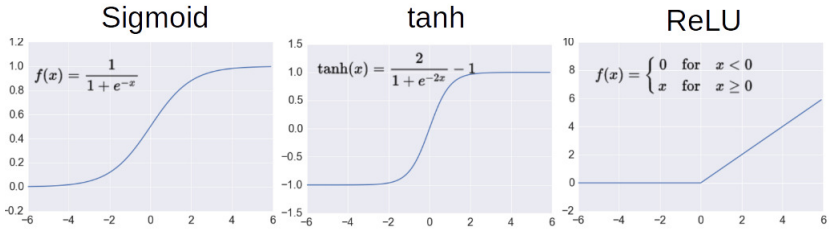
- Some popular choices for the nonlinear activation function f
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and $+1$)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



- Sigmoid saturates and can kill gradients. Also not "zero-centered"

Nonlinear Activation Functions

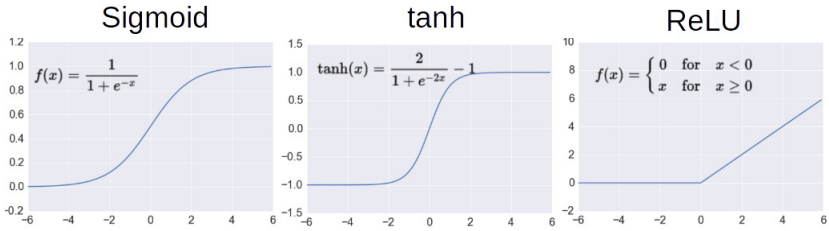
- Some popular choices for the nonlinear activation function f
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and $+1$)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



- Sigmoid saturates and can kill gradients. Also not "zero-centered"
- tanh also saturates but is zero-centered (thus preferred over sigmoid)

Nonlinear Activation Functions

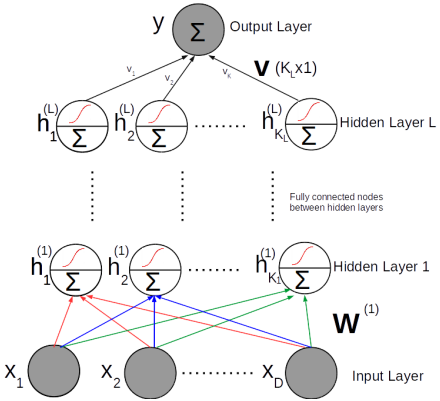
- Some popular choices for the nonlinear activation function f
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and $+1$)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



- Sigmoid saturates and can kill gradients. Also not "zero-centered"
- tanh also saturates but is zero-centered (thus preferred over sigmoid)
- ReLU is currently the most popular (also cheap to compute)

Learning Feedforward Neural Nets

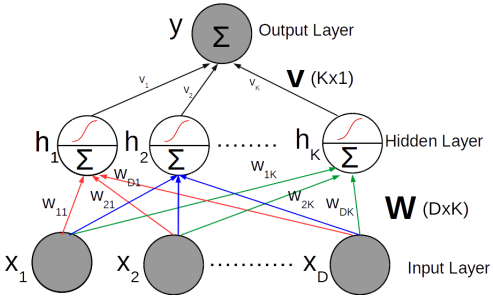
- Want to learn the parameters by minimizing some loss function



- Backpropagation (gradient descent + chain rule for derivatives) is commonly used to do this efficiently

Learning Feedforward Neural Nets

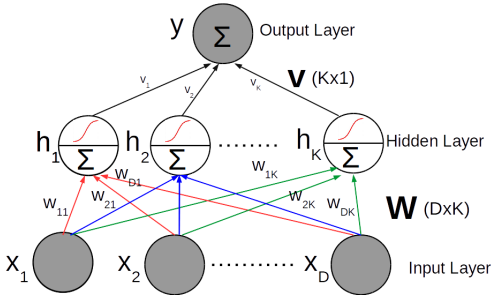
- Consider the feedforward neural net with one hidden layer



- Recall that $\boldsymbol{h} = [h_1 \ h_2 \ \ldots \ h_K] = f(\mathbf{W}^\top \boldsymbol{x})$

Learning Feedforward Neural Nets

- Consider the feedforward neural net with one hidden layer



- Recall that $\boldsymbol{h} = [h_1 \ h_2 \ \ldots \ h_K] = f(\mathbf{W}^\top \boldsymbol{x})$
- Assuming a regression problem, the optimization problem would be

$$\min_{\mathbf{W},\boldsymbol{v}} \frac{1}{2} \sum_{n=1}^{N} \left(y_n - \boldsymbol{v}^\top f(\mathbf{W}^\top \boldsymbol{x}_n)\right)^2 = \min_{\mathbf{W},\boldsymbol{v}} \frac{1}{2} \sum_{n=1}^{N} \left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2$$

where $\boldsymbol{w}_k$ is the k-th column of the $D \times K$ matrix $\mathbf{W}$

Learning Feedforward Neural Nets

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function

$$\mathcal{L} = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{v}^\top \boldsymbol{h}_n)\right)^2$$

Learning Feedforward Neural Nets

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function

$$\mathcal{L} = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{v}^\top \boldsymbol{h}_n\right)^2$$

- Gradient w.r.t. $\boldsymbol{v} = [v_1 \; v_2 \; \ldots \; v_K]$ is straightforward

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} = -\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)\boldsymbol{h}_n = -\sum_{n=1}^{N} \boldsymbol{e}_n \boldsymbol{h}_n$$

Learning Feedforward Neural Nets

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function

$$\mathcal{L} = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{v}^\top \boldsymbol{h}_n)\right)^2$$

- Gradient w.r.t. $\boldsymbol{v} = [v_1 \; v_2 \; \ldots \; v_K]$ is straightforward

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} = -\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)\boldsymbol{h}_n = -\sum_{n=1}^{N} \boldsymbol{e}_n \boldsymbol{h}_n$$

- Gradient w.r.t. the weights $\mathbf{W} = [\boldsymbol{w}_1 \; \boldsymbol{w}_2 \; \ldots \; \boldsymbol{w}_K]$ is a bit more involved due to the presence of f but can be computed using chain rule

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_k} = \frac{\partial \mathcal{L}}{\partial f_k}\frac{\partial f_k}{\partial \boldsymbol{w}_k} \qquad \text{(note: } f_k = f(\boldsymbol{w}_k^\top \boldsymbol{x})\text{)}$$

Learning Feedforward Neural Nets

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function

$$\mathcal{L} = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{v}^\top \boldsymbol{h}_n)\right)^2$$

- Gradient w.r.t. $\boldsymbol{v} = [v_1 \; v_2 \; \ldots \; v_K]$ is straightforward

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} = -\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)\boldsymbol{h}_n = -\sum_{n=1}^{N} \boldsymbol{e}_n \boldsymbol{h}_n$$

- Gradient w.r.t. the weights $\mathbf{W} = [\boldsymbol{w}_1 \; \boldsymbol{w}_2 \; \ldots \; \boldsymbol{w}_K]$ is a bit more involved due to the presence of f but can be computed using chain rule

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_k} = \frac{\partial \mathcal{L}}{\partial f_k}\frac{\partial f_k}{\partial \boldsymbol{w}_k} \qquad \text{(note: } f_k = f(\boldsymbol{w}_k^\top \boldsymbol{x})\text{)}$$

- We have: $\frac{\partial \mathcal{L}}{\partial f_k} = -\sum_{n=1}^{N}(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n))v_k = -\sum_{n=1}^{N} \boldsymbol{e}_n v_k$

Learning Feedforward Neural Nets

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function

$$\mathcal{L} = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{v}^\top \boldsymbol{h}_n)\right)^2$$

- Gradient w.r.t. $\boldsymbol{v} = [v_1 \; v_2 \; \ldots \; v_K]$ is straightforward

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} = -\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)\boldsymbol{h}_n = -\sum_{n=1}^{N} \boldsymbol{e}_n \boldsymbol{h}_n$$

- Gradient w.r.t. the weights $\mathbf{W} = [\boldsymbol{w}_1 \; \boldsymbol{w}_2 \; \ldots \; \boldsymbol{w}_K]$ is a bit more involved due to the presence of f but can be computed using chain rule

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_k} = \frac{\partial \mathcal{L}}{\partial f_k}\frac{\partial f_k}{\partial \boldsymbol{w}_k} \qquad \text{(note: } f_k = f(\boldsymbol{w}_k^\top \boldsymbol{x})\text{)}$$

- We have: $\frac{\partial \mathcal{L}}{\partial f_k} = -\sum_{n=1}^{N}(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n))v_k = -\sum_{n=1}^{N} \boldsymbol{e}_n v_k$

- We have: $\frac{\partial f_k}{\partial \boldsymbol{w}_k} = \sum_{n=1}^{N} f'(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\boldsymbol{x}_n$, where $f'(\boldsymbol{w}_k^\top \boldsymbol{x}_n)$ is f's derivative at $\boldsymbol{w}_k^\top \boldsymbol{x}_n$

Learning Feedforward Neural Nets

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function

$$\mathcal{L} = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{v}^\top \boldsymbol{h}_n)\right)^2$$

- Gradient w.r.t. $\boldsymbol{v} = [v_1 \; v_2 \; \ldots \; v_K]$ is straightforward

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} = -\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)\boldsymbol{h}_n = -\sum_{n=1}^{N} \boldsymbol{e}_n \boldsymbol{h}_n$$

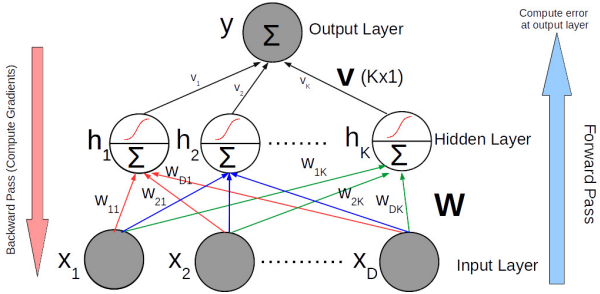
- Gradient w.r.t. the weights $\mathbf{W} = [\boldsymbol{w}_1 \; \boldsymbol{w}_2 \; \ldots \; \boldsymbol{w}_K]$ is a bit more involved due to the presence of f but can be computed using chain rule

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_k} = \frac{\partial \mathcal{L}}{\partial f_k}\frac{\partial f_k}{\partial \boldsymbol{w}_k} \qquad \text{(note: } f_k = f(\boldsymbol{w}_k^\top \boldsymbol{x})\text{)}$$

- We have: $\frac{\partial \mathcal{L}}{\partial f_k} = -\sum_{n=1}^{N}(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n))v_k = -\sum_{n=1}^{N} \boldsymbol{e}_n v_k$
- We have: $\frac{\partial f_k}{\partial \boldsymbol{w}_k} = \sum_{n=1}^{N} f'(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\boldsymbol{x}_n$, where $f'(\boldsymbol{w}_k^\top \boldsymbol{x}_n)$ is f's derivative at $\boldsymbol{w}_k^\top \boldsymbol{x}_n$
- These calculations can be done efficiently using backpropagation

Backpropagation

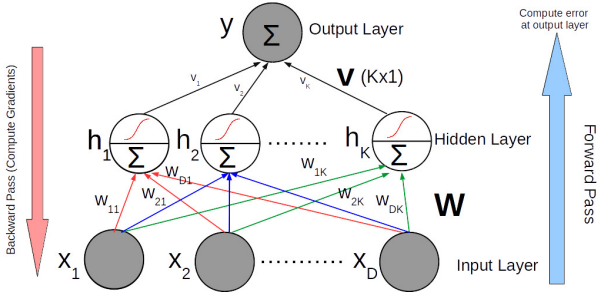
- Basically consists of a forward pass and a backward pass



- Forward pass computes the errors $\boldsymbol{e}_n$ using the current parameters
- Backward pass computes the gradients and updates the parameters, starting from the parameters at the top layer and then moving backwards

Backpropagation

- Basically consists of a forward pass and a backward pass



- Forward pass computes the errors $\boldsymbol{e}_n$ using the current parameters
- Backward pass computes the gradients and updates the parameters, starting from the parameters at the top layer and then moving backwards
- Also good at reusing previous computations (updates of parameters at any layer depends on parameters at the layer above)

Kernel Methods vs Deep Neural Nets

- Recall the prediction rule for a kernel method (e.g., kernel SVM)

$$y = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- This is analogous to a single hidden layer NN with fixed/pre-defined hidden nodes $\{k(\boldsymbol{x}_n, \boldsymbol{x})\}_{n=1}^{N}$ and output layer weights $\{\alpha_n\}_{n=1}^{N}$
- The prediction rule for a deep neural network

$$y = \sum_{k=1}^{K} v_k h_k$$

- In this case, the h_k's are learned from data (possibly after multiple layers of nonlinear transformations)
- Both kernel methods and deep NNs be seen as using nonlinear basis functions for making predictions. Kernel methods use fixed basis functions (defined by the kernel) whereas NN learns the basis functions adaptively from data

Wide vs Deep?

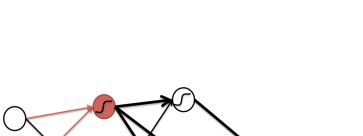
Why might we prefer a deep model over a wide and shallow model?

Wide vs Deep?

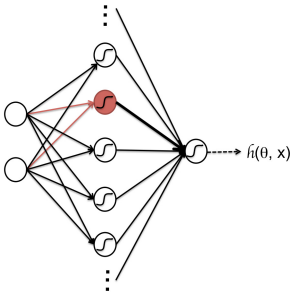
Why might we prefer a deep model over a wide and shallow model?

An informal justification:

- Deep "programs" can reuse computational subroutines (and are more compact)



Deep Network



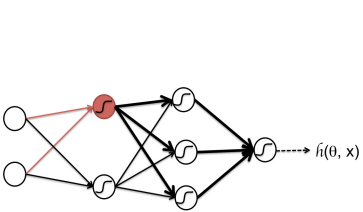
Shallow Network

Wide vs Deep?

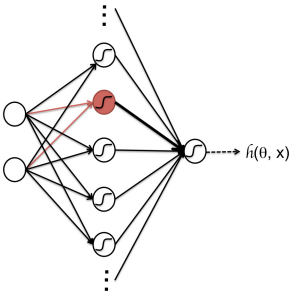
Why might we prefer a deep model over a wide and shallow model?

An informal justification:

- Deep "programs" can reuse computational subroutines (and are more compact)



Deep Network



Shallow Network

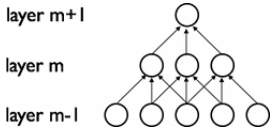
Learning Certain functions may require a huge number of units in a shallow model

Convolutional Neural Network (CNN)

- A feedforward neural network with a special structure

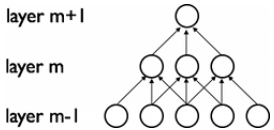
Convolutional Neural Network (CNN)

- A feedforward neural network with a special structure
- Sparse "local" connectivity between layers (except the last output layer). Reduces the number of parameters to be learned

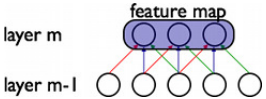


Convolutional Neural Network (CNN)

- A feedforward neural network with a special structure
- Sparse "local" connectivity between layers (except the last output layer). Reduces the number of parameters to be learned

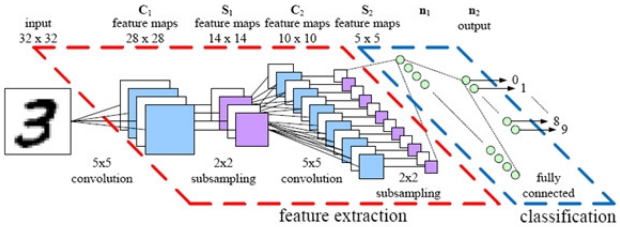


- Shared weights (like a "global" filter). Helps capture the local properties of the signal (useful for data such as images or time-series)



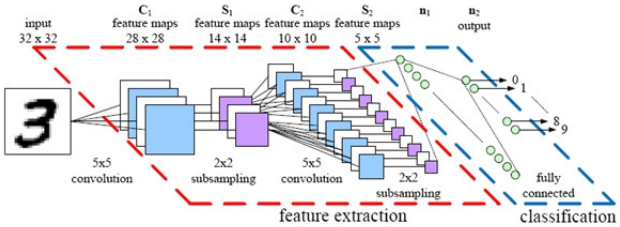
Convolutional Neural Network (CNN)

- Uses a sequence of 2 operations, convolution and pooling (subsampling), applied repeatedly on the input data



Convolutional Neural Network (CNN)

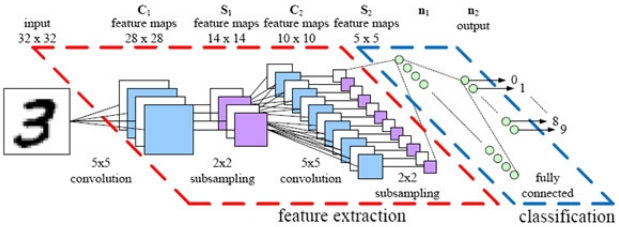
- Uses a sequence of 2 operations, convolution and pooling (subsampling), applied repeatedly on the input data



- Convolution: Extract "local" properties of the signal. Uses a set of "filters" that have to be learned (these are the "weighted" **W** between layers)

Convolutional Neural Network (CNN)

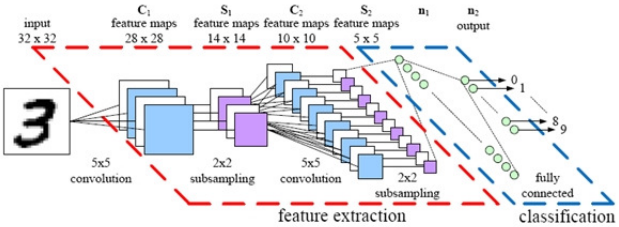
- Uses a sequence of 2 operations, convolution and pooling (subsampling), applied repeatedly on the input data



- Convolution: Extract "local" properties of the signal. Uses a set of "filters" that have to be learned (these are the "weighted" **W** between layers)
- Pooling: Downsamples the outputs to reduce the size of representation

Convolutional Neural Network (CNN)

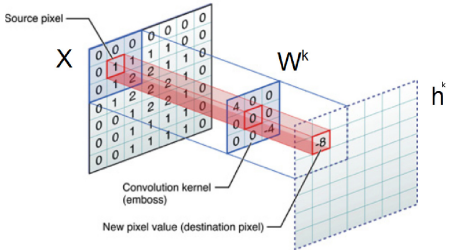
- Uses a sequence of 2 operations, convolution and pooling (subsampling), applied repeatedly on the input data



- Convolution: Extract "local" properties of the signal. Uses a set of "filters" that have to be learned (these are the "weighted" **W** between layers)
- Pooling: Downsamples the outputs to reduce the size of representation
- Note: A nonlinearity is also introduced after the convolution layer

Convolution

- An operation that captures local (e.g., spatial) properties of a signal



- Mathematically, the operation is defined as

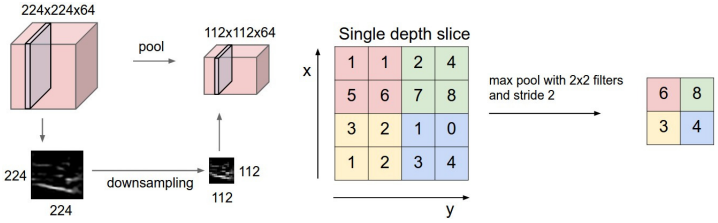
$$h_{ij}^k = f((W^k * \mathbf{X})_{ij} + b_k)$$

where W^k is a filter, $*$ is the convolution operator, and f is a nonlinearity

- Usually a number of filters $\{W^k\}_{k=1}^K$ are applied (each will produce a separate "feature map"). These filters have to be learned
- Size of these filters have to be specified

Pooling/Subsampling

This operation is used to reduce the size of the representation



Deep Neural Nets: Some Comments

- Highly effective in learning good feature representations from data in an "end-to-end" manner

Deep Neural Nets: Some Comments

- Highly effective in learning good feature representations from data in an "end-to-end" manner
- The objective functions of these models are highly non-convex
  - But lots of recent work on non-convex optimization, so non-convexity doesn't scare us (that much) anymore

Deep Neural Nets: Some Comments

- Highly effective in learning good feature representations from data in an "end-to-end" manner
- The objective functions of these models are highly non-convex
  - But lots of recent work on non-convex optimization, so non-convexity doesn't scare us (that much) anymore
- Training these models is computationally very expensive
  - But GPUs can help to speed up many of the computations

Deep Neural Nets: Some Comments

- Highly effective in learning good feature representations from data in an "end-to-end" manner
- The objective functions of these models are highly non-convex
  - But lots of recent work on non-convex optimization, so non-convexity doesn't scare us (that much) anymore
- Training these models is computationally very expensive
  - But GPUs can help to speed up many of the computations
- Training these models can be tricky, especially a proper initialization
  - But now we have several ways to intelligently initialize these models (e.g., unsupervised layer-wise pre-training)

Deep Neural Nets: Some Comments

- Highly effective in learning good feature representations from data in an "end-to-end" manner
- The objective functions of these models are highly non-convex
  - But lots of recent work on non-convex optimization, so non-convexity doesn't scare us (that much) anymore
- Training these models is computationally very expensive
  - But GPUs can help to speed up many of the computations
- Training these models can be tricky, especially a proper initialization
  - But now we have several ways to intelligently initialize these models (e.g., unsupervised layer-wise pre-training)
- Deep learning models can also be probabilistic and generative, e.g., deep belief networks (we did not consider these here)