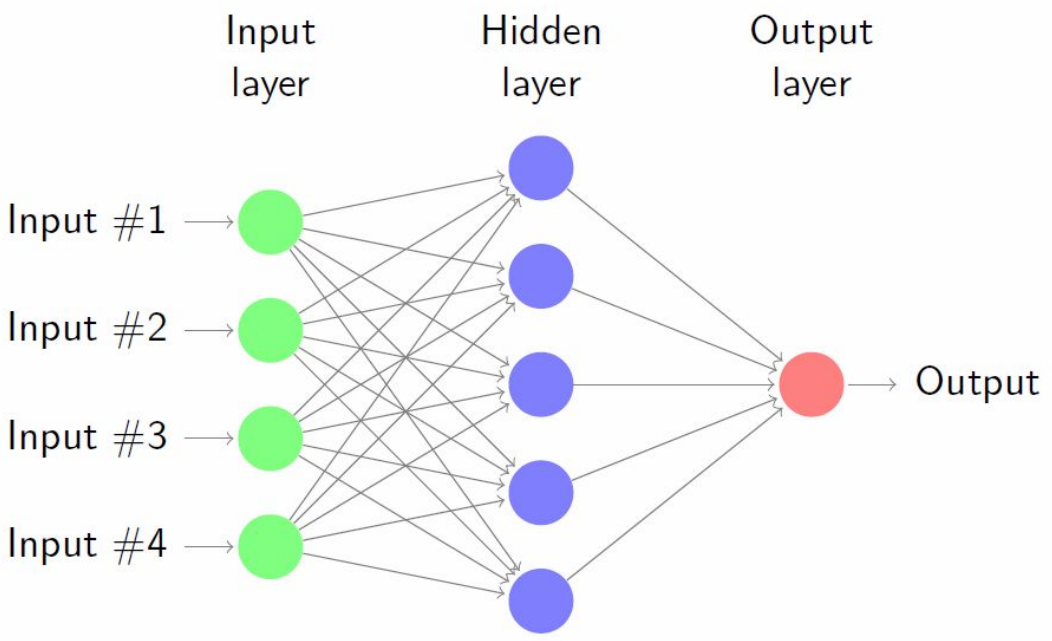
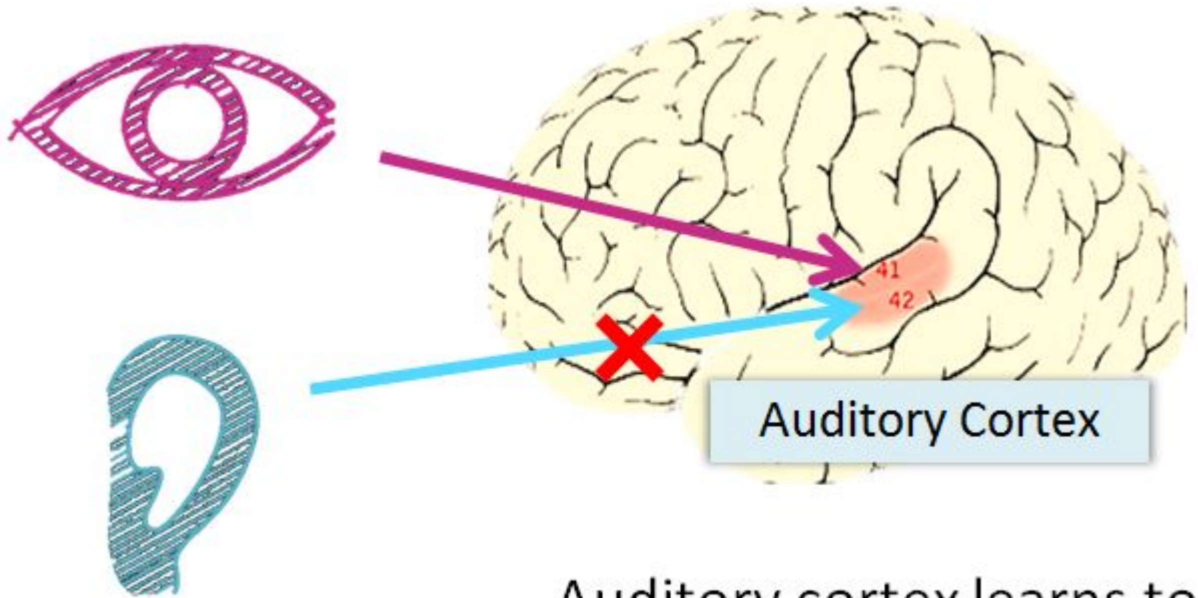
Neural Nets


Input layer
Hidden layer
Output layer
Input #1
Input #2
Input #3
Input #4
Output

Neural Networks

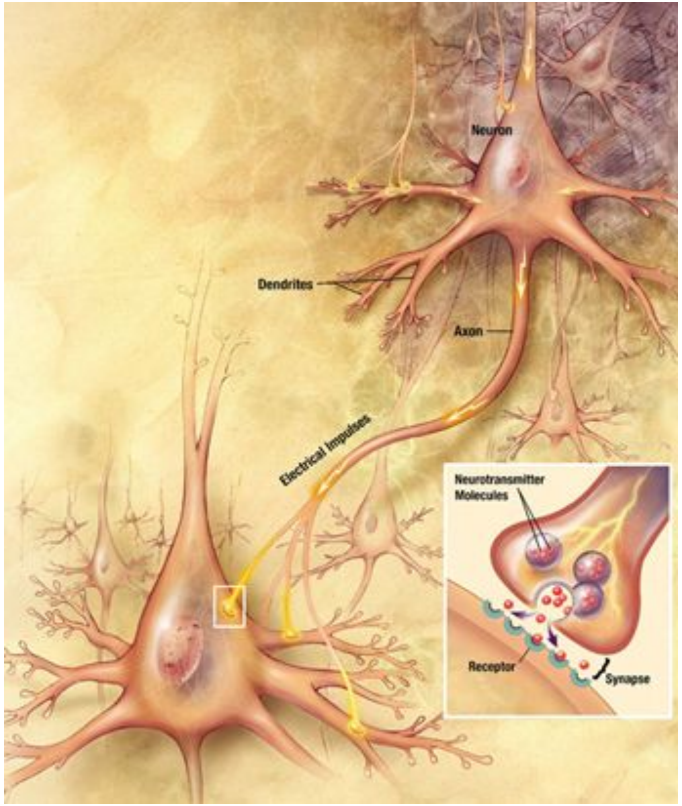
- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Theory is still not fully understood

The “one learning algorithm” hypothesis



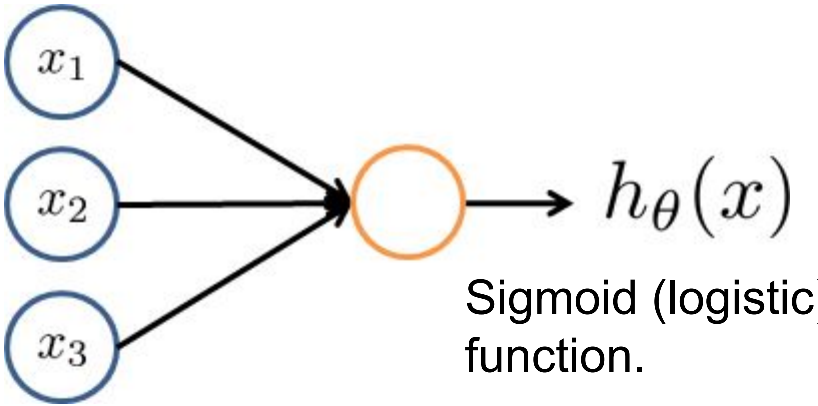
Auditory cortex learns to see

Neurons in the brain



- Axons
- Dendrites
- Synapses
- Receptors

Neuron model: logistic unit



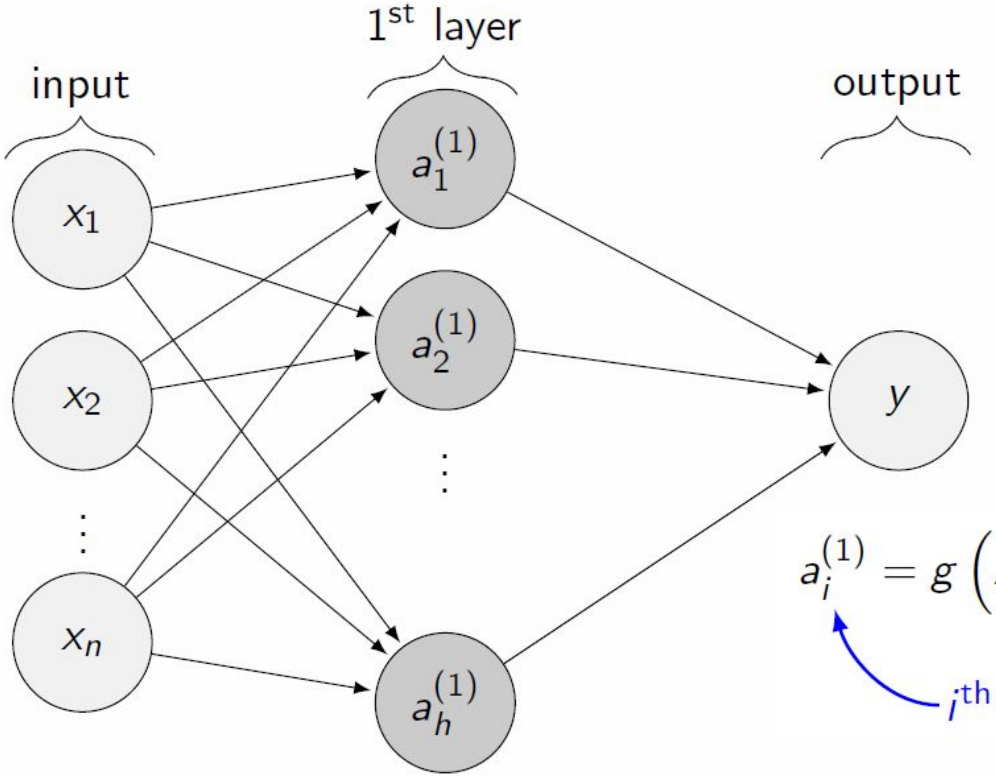
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Sigmoid (logistic) activation function.

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + \exp(-z)}$$

Add a hidden layer of neurons

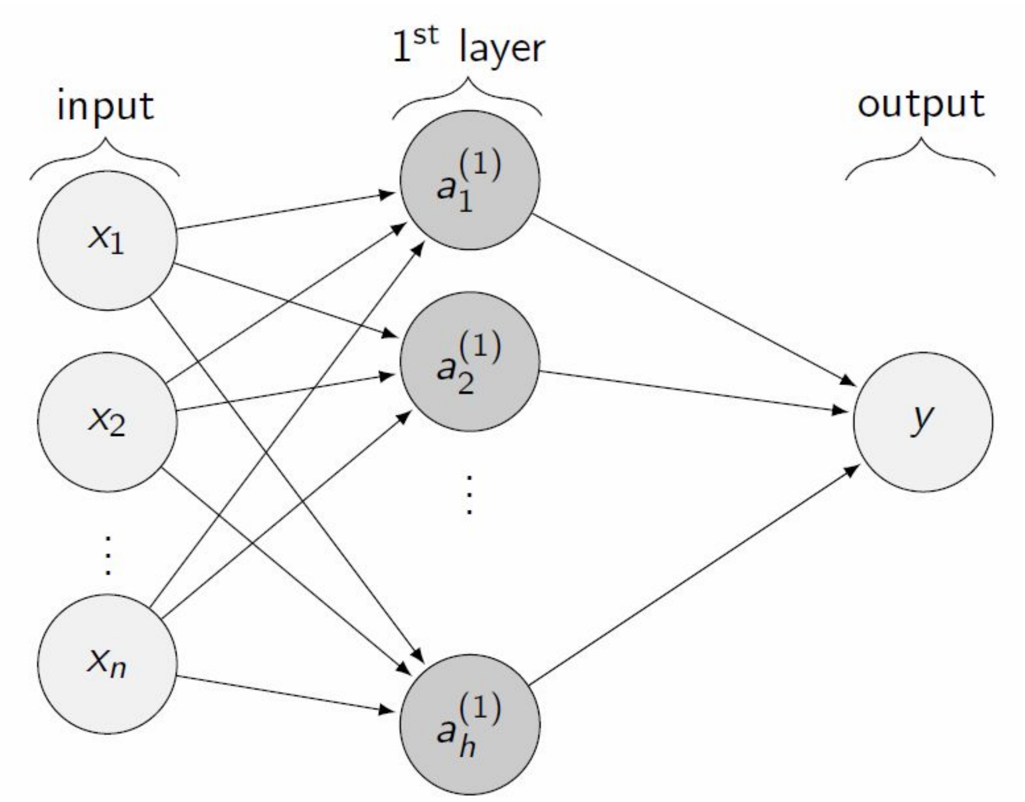


$$a_i^{(1)} = g\left(z_i^{(1)}\right) \qquad \text{with } z_i^{(1)} = \sum_{j=1}^{n} w_{i,j}^{(1)} x_j + b_i^{(1)}$$

i^{th} component is called "unit i"

where g is called activation function and $w_{i,j}^{(1)}$ are adjustable weights.

Add a hidden layer of neurons



$$a_1^{(1)} = g(w_1^T x + b_1)$$

$$\vdots$$

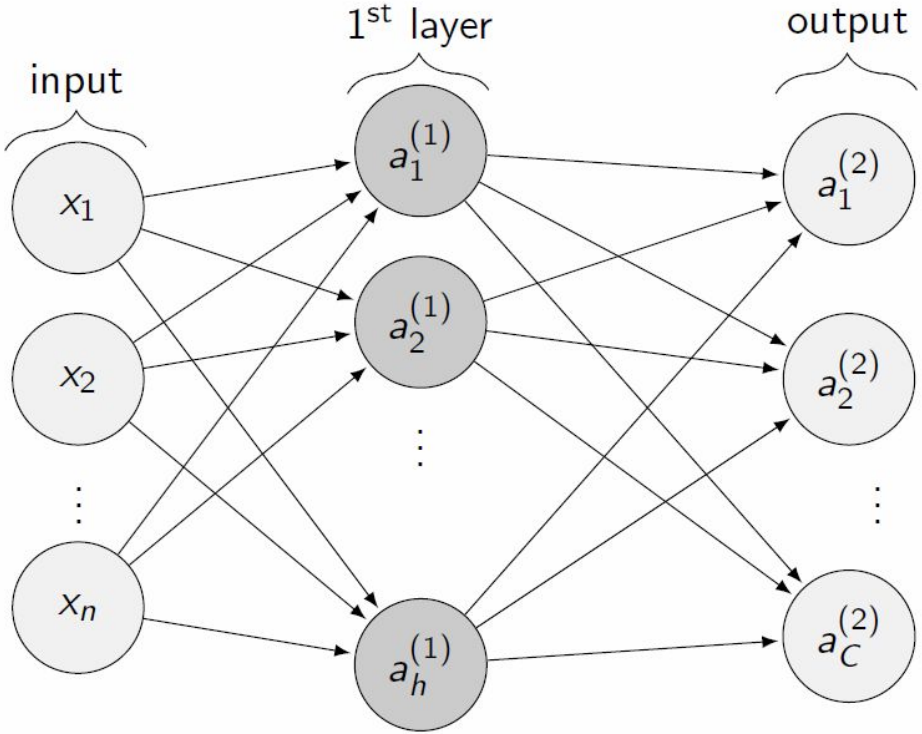
$$a_h^{(1)} = g(w_h^T x + b_h)$$

Or more compactly,

$$\mathbf{a}^{(1)} = g(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

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

For multi-class classification



$$\mathbf{a}^{(1)} = g_1(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{a}^{(2)} = g_2(W^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)})$$

g_2 is the softmax function

Activation Function

How to choose the activation function g in each layer?

- Non-linear activation functions will increase the expressive power: Multilayer perceptrons with $L \geq 2$ are universal approximators!
- Depending on the application: For classification we may want to interpret the output as posterior probabilities:

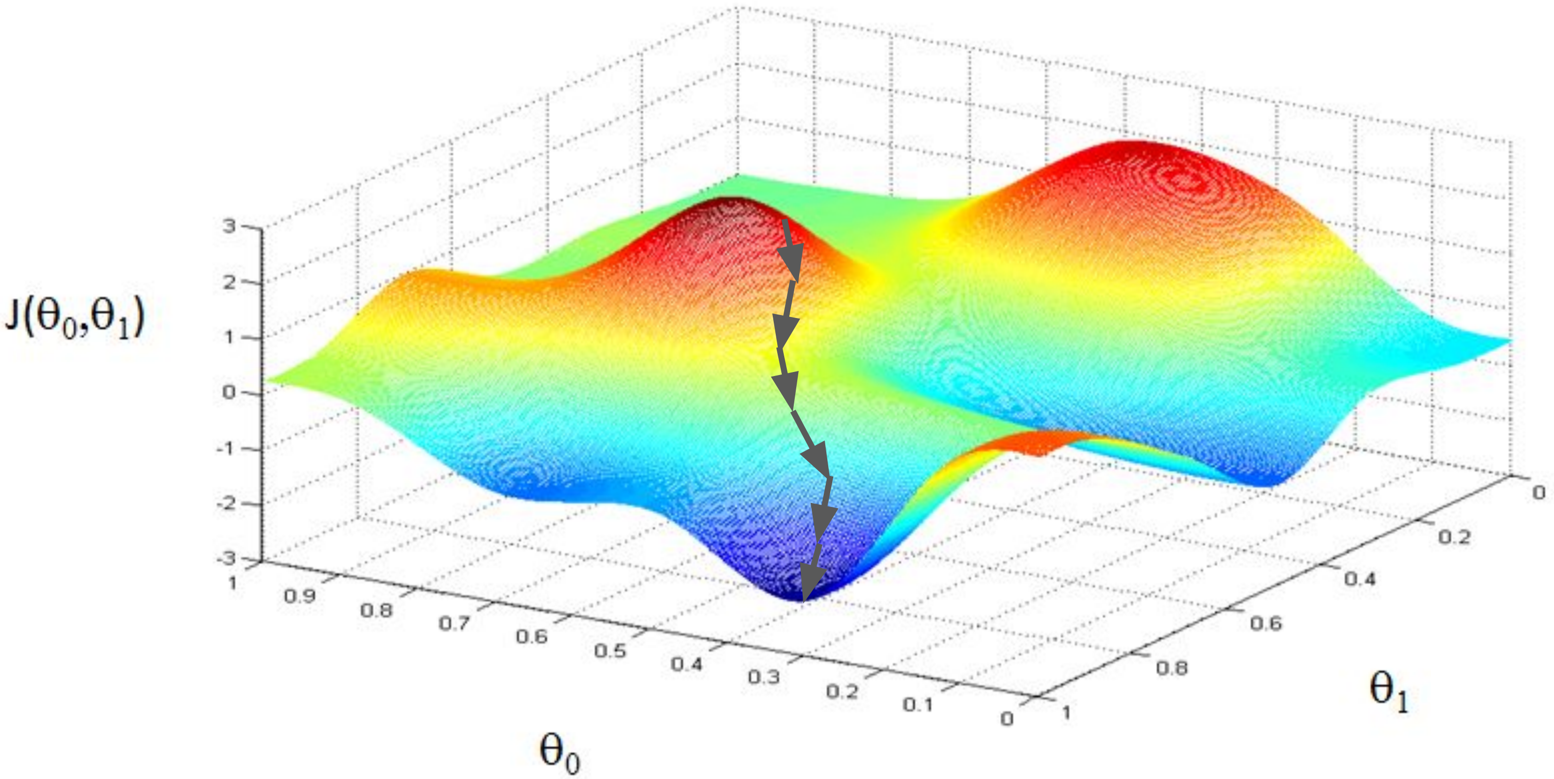
$$y_i(x, w) \stackrel{!}{=} p(c = i|x) \tag{21}$$

where c denotes the random variable for the class.

Common activation functions

- Linear — z
- Logistic function — $\sigma(z) = \dfrac{1}{1 + \exp(-z)}$
- Hyperbolic tangent — $\tanh(z)$
- Softmax — $\hat{y}_i = a_i^{(L)} = \sigma(z^{(L)}, i) = \dfrac{\exp(z_i^{(L)})}{\sum_{k=1}^{C} \exp(z_k^{(L)})}$
- ReLU — max(0, z)

Use gradient descent to learn weights and biases



Gradient descent

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\
&\}
\end{aligned}
$$

- Calculate the gradient of the loss function w.r.t. parameters
- Determine the learning rate

Backpropagation on neural net with one hidden layer

- Given an input $\mathbf{x} \in \mathbb{R}^n$, the network outputs $\hat{\mathbf{y}} = \mathbf{a}^{(2)}$.
- Cross-entropy loss function $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y}^T \log \hat{\mathbf{y}}$, where $\mathbf{y}$ is the ground-truth label in one-hot representation.
- Define "error" signal at the output layer to be

$$\delta^{(2)} = \hat{\mathbf{y}} - \mathbf{y}$$

- Backpropagate the error to the hidden layer

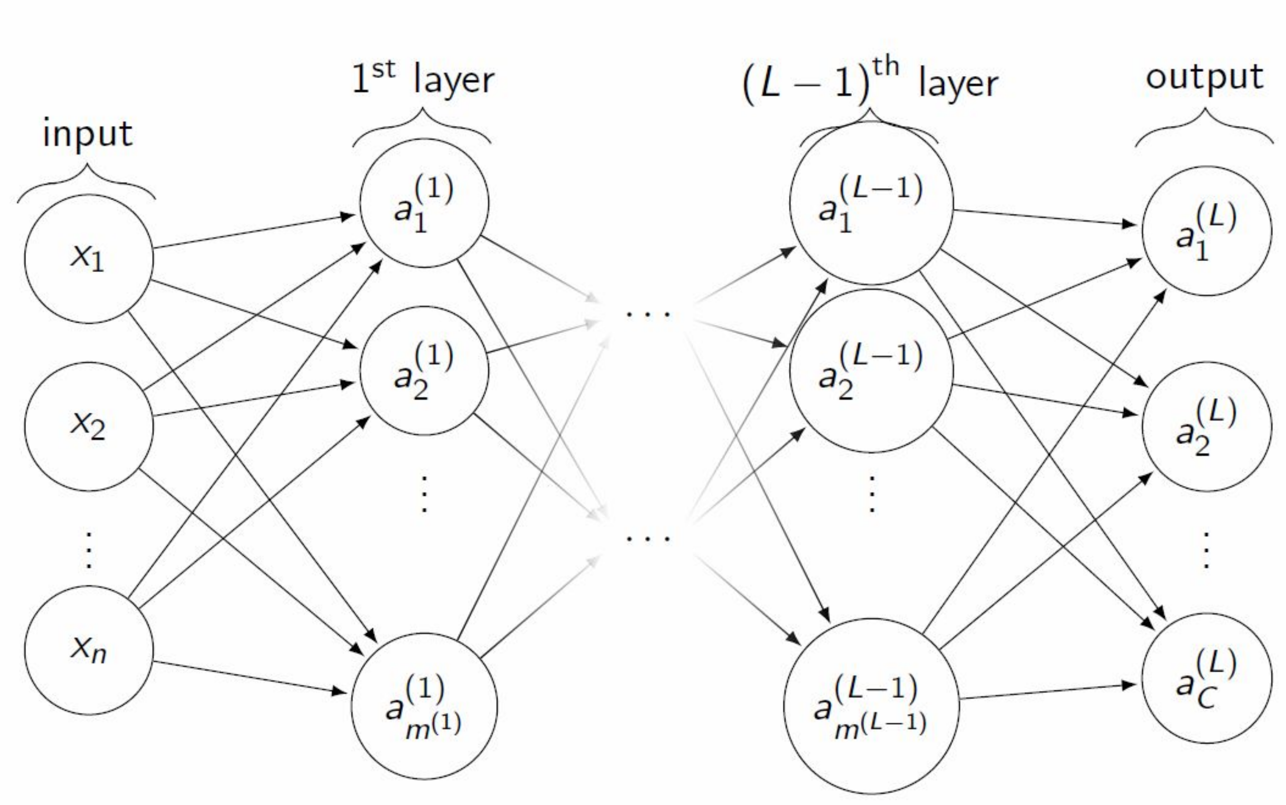
$$\delta^{(1)} = (W^{(2)})^T \delta^{(2)} .* g_1'(\mathbf{z}^{(1)})$$

  where $\mathbf{z}^{(1)} = W^{(1)} x + \mathbf{b}^{(1)}$ is the total input to neurons in the hidden layer.
- If g_1 is the logistic function, then $g_1'(\mathbf{z}^{(1)}) = \mathbf{a}^{(1)} .* \mathbf{a}^{(1)}$
- Gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^T$$

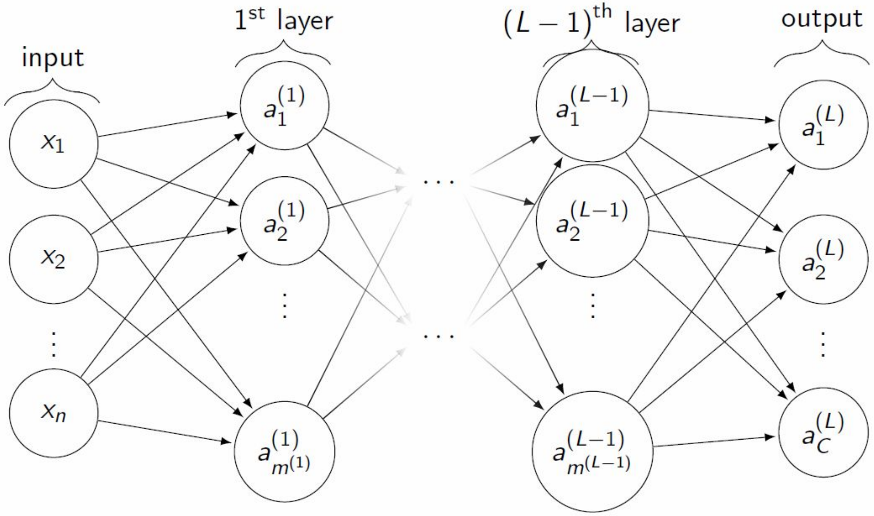
Deep neural nets



Neural nets with L layers

- Given an input $\mathbf{x} \in \mathbb{R}^n$, the network outputs

$$\begin{aligned}
\mathbf{z}^{(1)} &= W^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{a}^{(1)} &= g_1(\mathbf{z}^{(1)}) \\
\mathbf{z}^{(2)} &= W^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)} \\
&\vdots \\
\mathbf{a}^{(L)} &= g_L(\mathbf{z}^{(L)})
\end{aligned}$$



with the final output $\mathbf{y} = \mathbf{a}^{(L)} \in \mathbb{R}^C$.

- Notations:
  - $\mathbf{z}^{(l)}$ - total input to neurons in layer l.
  - $\mathbf{a}^{(l)}$ - activation of neurons in layer l.
  - $W^{(l)}$ - connectivity matrix from neurons in layer $l-1$ to neurons in layers l.
  - $\mathbf{b}^{(l)}$ - bias added to neurons in layer l

Backpropagation in general neural nets

- Given an input $\mathbf{x} \in \mathbb{R}^n$, the network outputs $\hat{\mathbf{y}} = \mathbf{a}^{(L)}$.
- Loss function $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ - the inconsistency between the prediction and ground truth.
- Define "error" signal to be

$$\delta^{(l)} \equiv \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}}$$

- Because $\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} = W^{(l)} g_{l-1}(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)}$,

$$\delta^{(l-1)} = (W^{(l)})^T \delta^{(l)} .* g'_{l-1}(\mathbf{z}^{(l-1)})$$

- Gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$
$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^T$$

Backpropagation algorithm

Input: Training set $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{m}$

1. Set $\Delta^{(l)} = 0$ for all l
2. **for** $i = 1$ **to** m
3. Set $\mathbf{a}^{(0)} = \mathbf{x}^{(i)}$
4. Perform forward propagation to calculate $\mathbf{a}^{(l)}$ for $l = 1, \cdots, L$
5. Using $y^{(i)}$, compute $\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}^{(i)}$
6. Compute $\delta^{(L-1)}, \cdots, \delta^{(1)}$
7. $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l)}(\mathbf{a}^{(l-1)})^T$ for all l
8. $\Delta^{(l)} := \frac{1}{m}\Delta^{(l)}$ for all l
9. $W^{(l)} := W^{(l)} - \eta\Delta^{(l)}$ for all l

Deep Learning

- Multilayer perceptrons are called deep if they have more than three layers: $L > 3$.
- Motivation: Lower layers can automatically learn a hierarchy of features or a suitable dimensionality reduction.
  - No hand-crafted features necessary anymore!
- Training deep neural networks was considered difficult
  - Error measure represents a highly non-convex, “potentially intractable" optimization problem.

Approaches to deep learning

- Carefully designed network architecture
- Unsupervised pre-training can be done layer-wise
- Better training algorithms and heuristics such as good initialization, batch normalization, weight clip, early stop etc
- Drop-out, Early Stop to control complexity, reduce overfitting
- Transfer learning, meta learning, etc
- See “Deep Learning," by Goodfellow, Bengio & Courville for a detailed discussion of state-of-the-art approaches to deep learning.

Summary of multilayer perceptron

- The multilayer perceptron represents a standard model of neural networks. They …
    - allow to taylor the architecture (layers, activation functions) to the problem;
    - can be trained using gradient descent and error backpropagation;
    - can be used for learning feature hierarchies (deep learning).