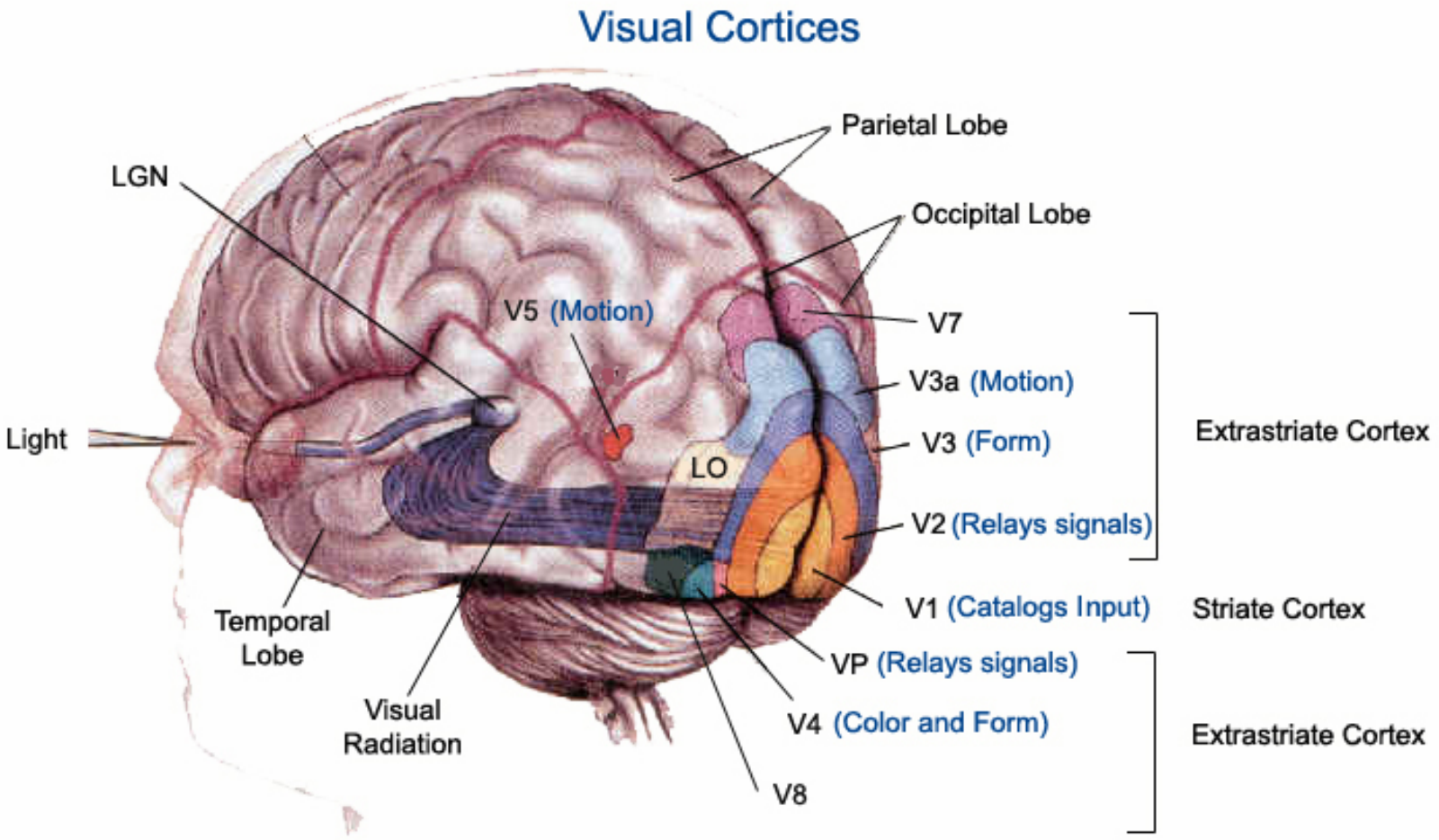
Deep Learning

Online material

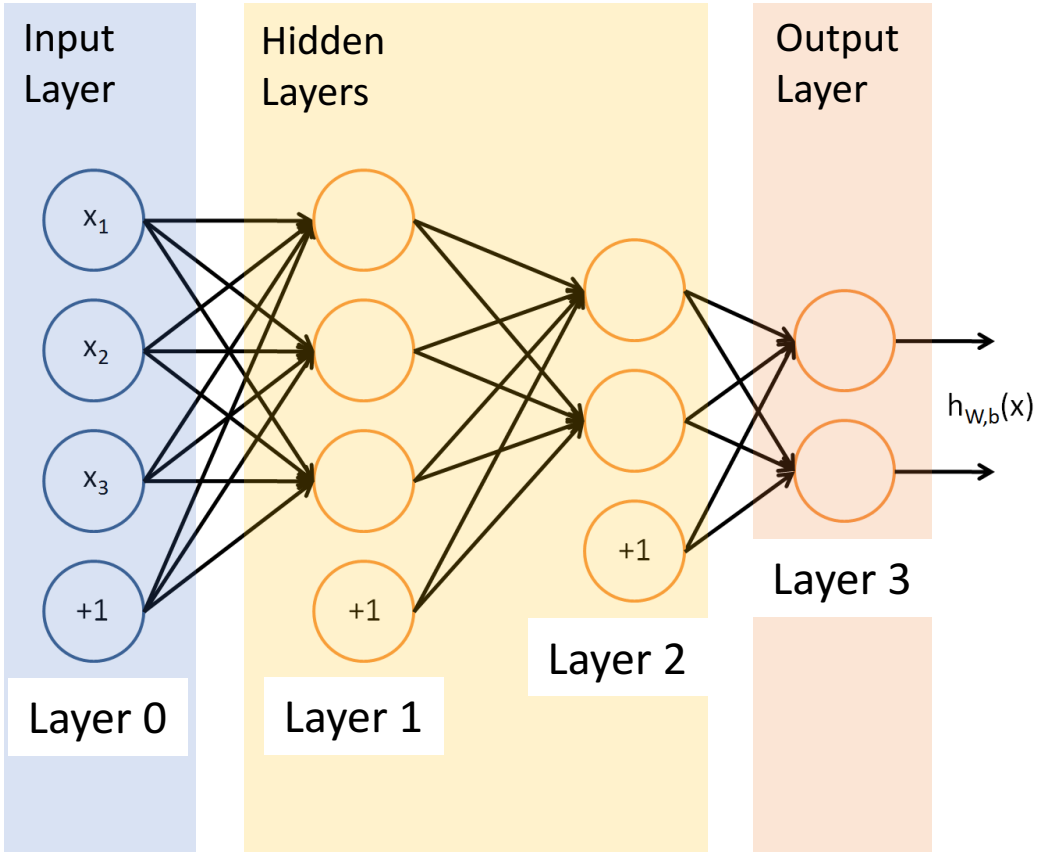
- http://neuralnetworksanddeeplearning.com/chap2.html

What is Deep Learning?



What is Deep Learning?

Biologically-inspired multilayer neural networks



Both supervised and unsupervised

Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." Science 313.5786 (2006): 504-507.

What is Deep Learning?

Example. Face recognition (Facebook)

Edges

Eyes, Noses, Mouths

Faces

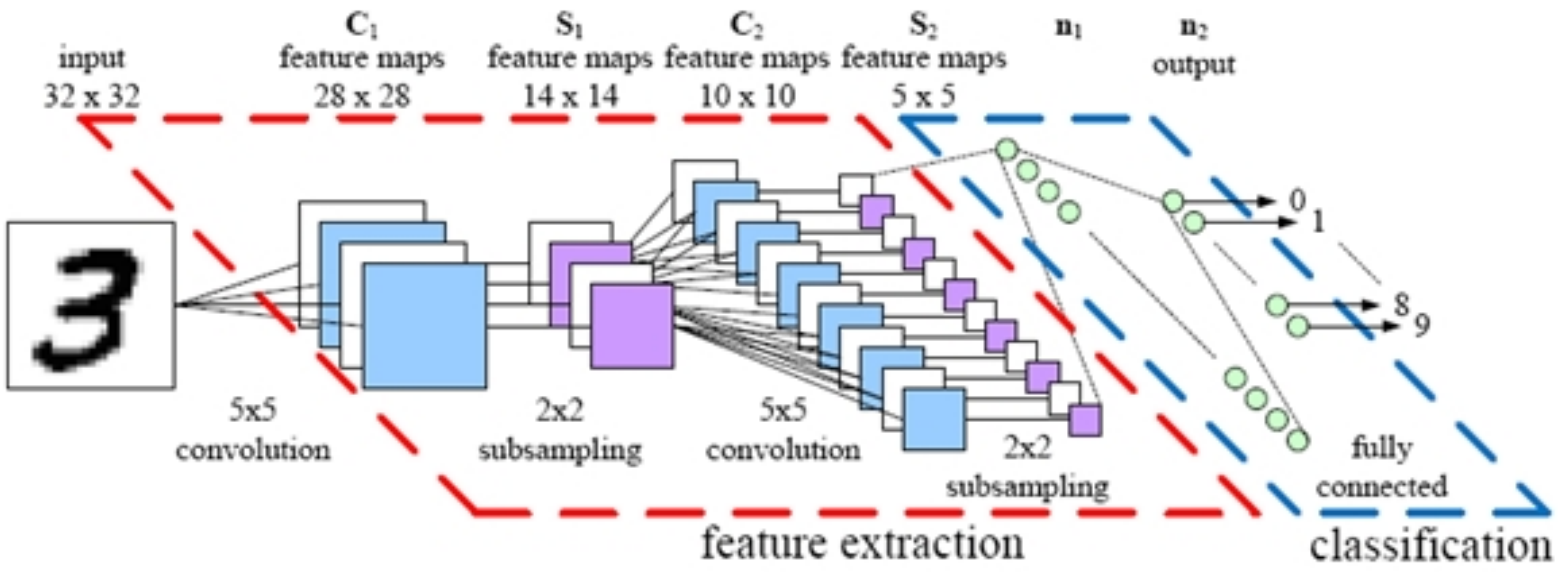
Deeper layers learn higher-order features

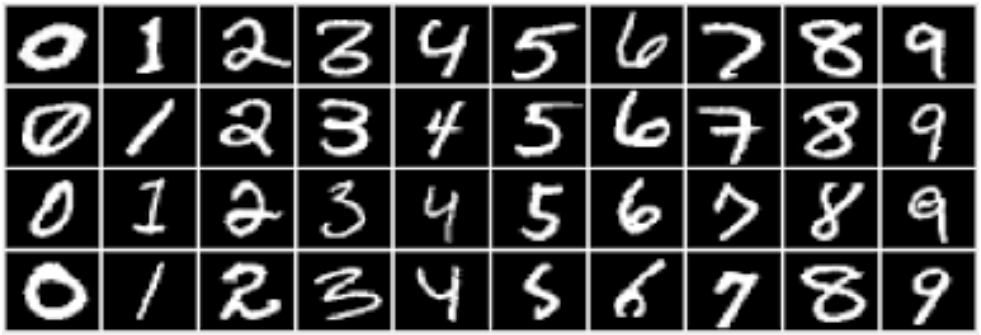
LISA lab, “Deep Learning Tutorials.” (http://www.deeplearning.net/tutorial/) accessed 2014.

Lee, Honglak, et al. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." Proceedings of the 26th Annual International Conference on Machine Learning. ACM, 2009.

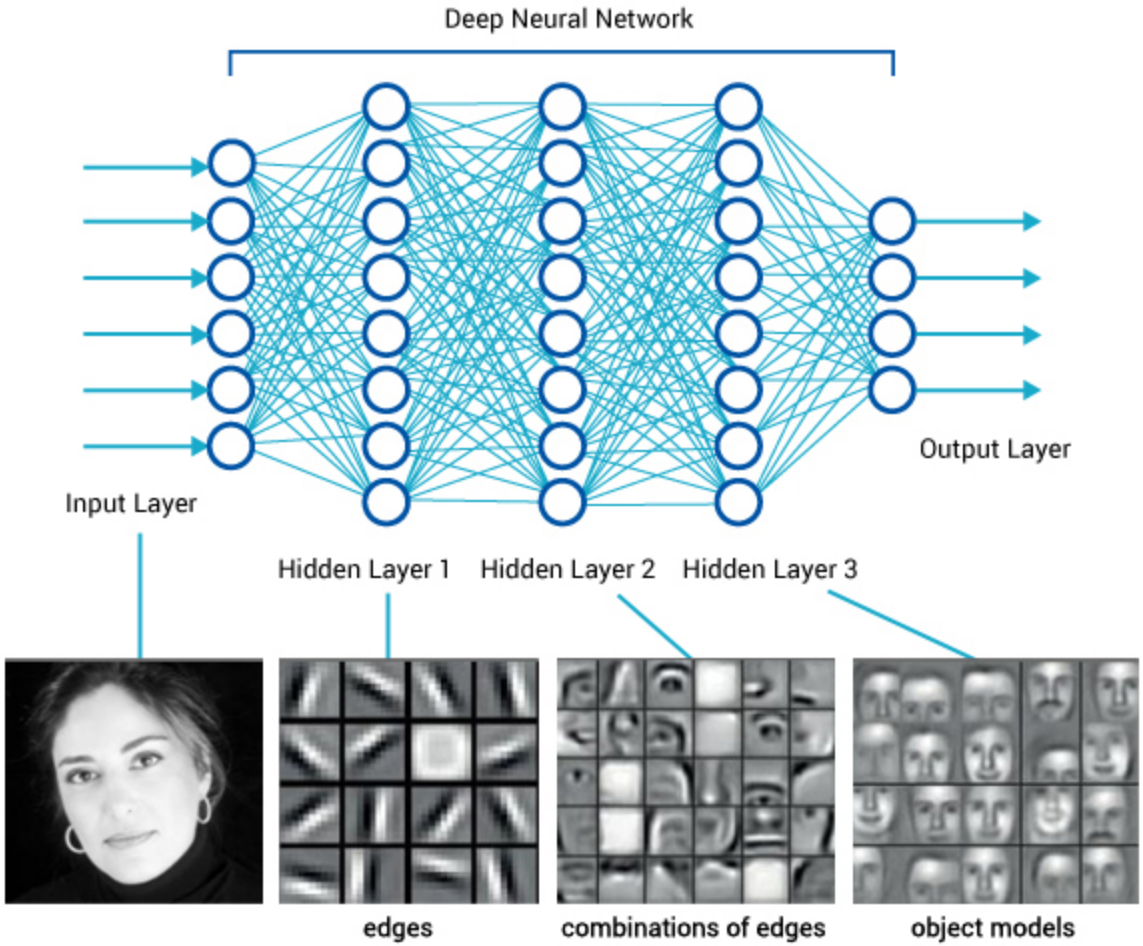
Applications

Handwriting Recognition





Face Recognition

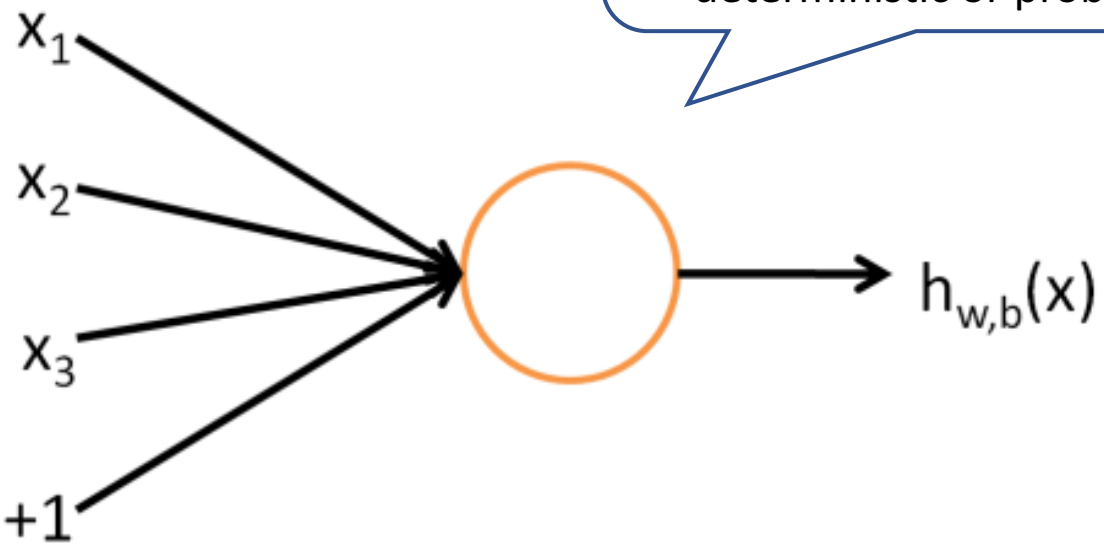


FeedForward Networks

Neuron

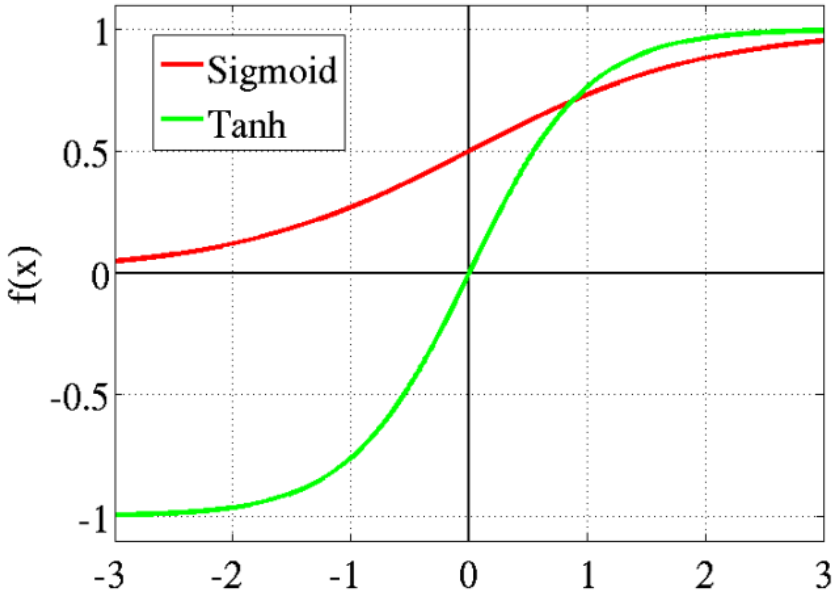
Perceptron.

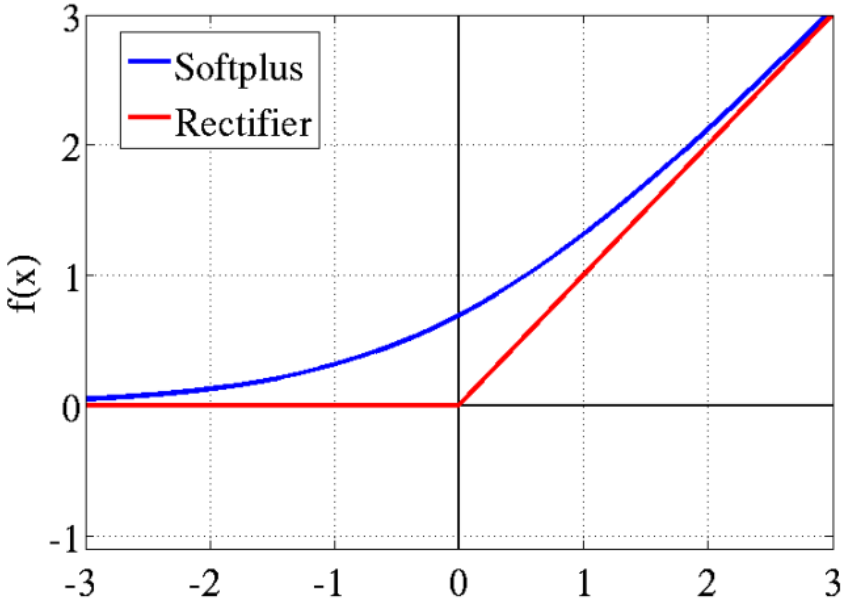
Depending on function f,
neurons can be:
real-valued or binary-valued;
deterministic or probabilistic.



$$h_{w,b}(x) = f(w^{\top}x) = f\left(\sum_{i=1}^{d} w_i x_i + b\right)$$

Activation Functions





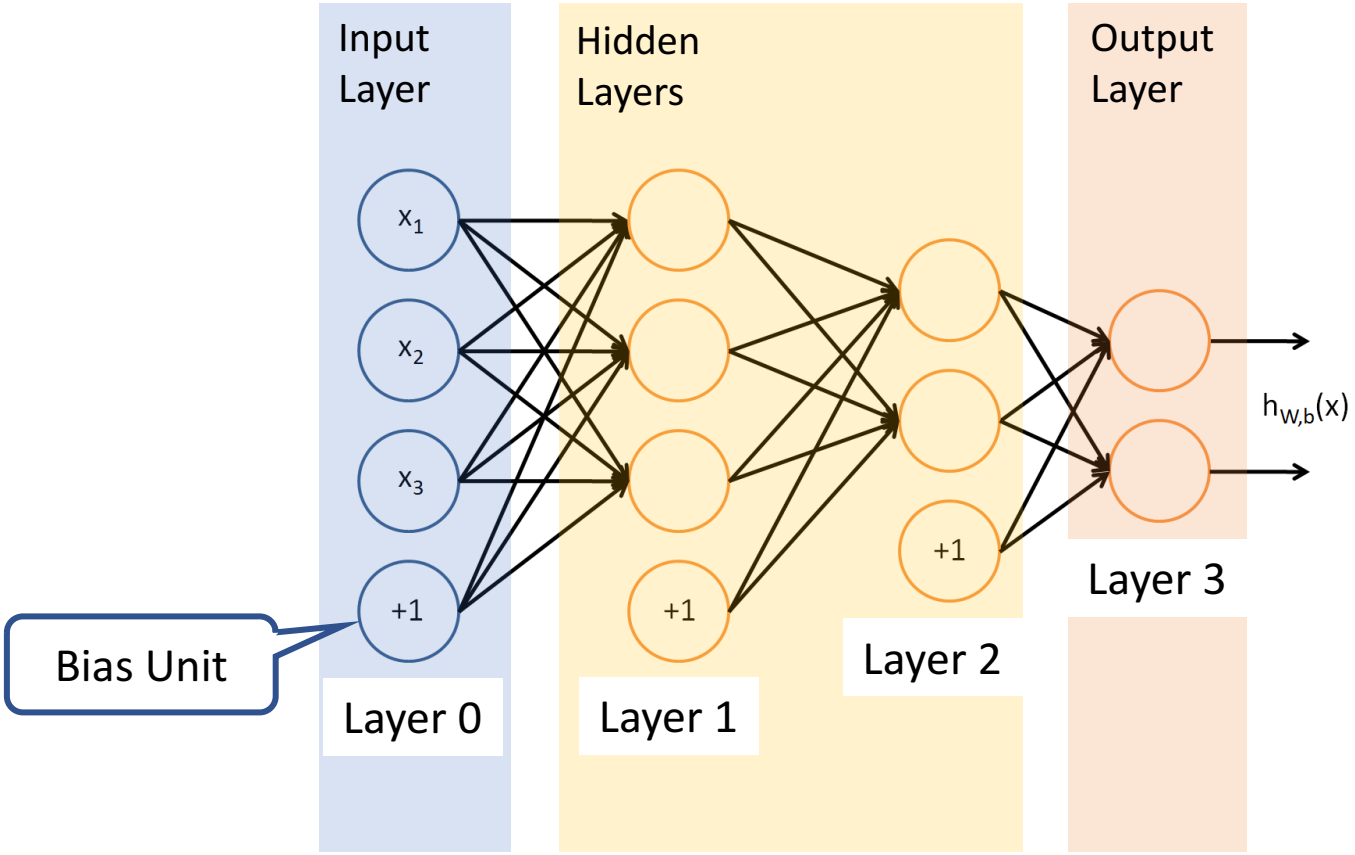
sigmoid $f(z) = \frac{1}{1+e^{-z}}$

softplus $f(z) = \ln(1 + e^{-z})$

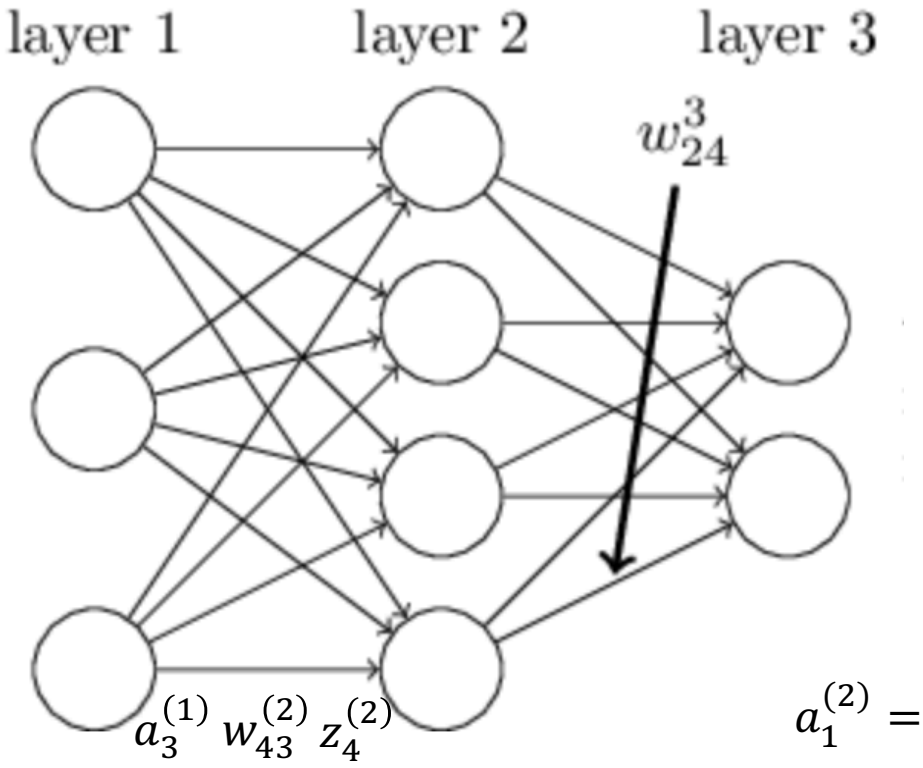
tanh $f(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

rectified linear unit (ReLU) $f(z) = \max(0, z)$

Multi-Layer Neural Network



Multi-Layer Neural Network

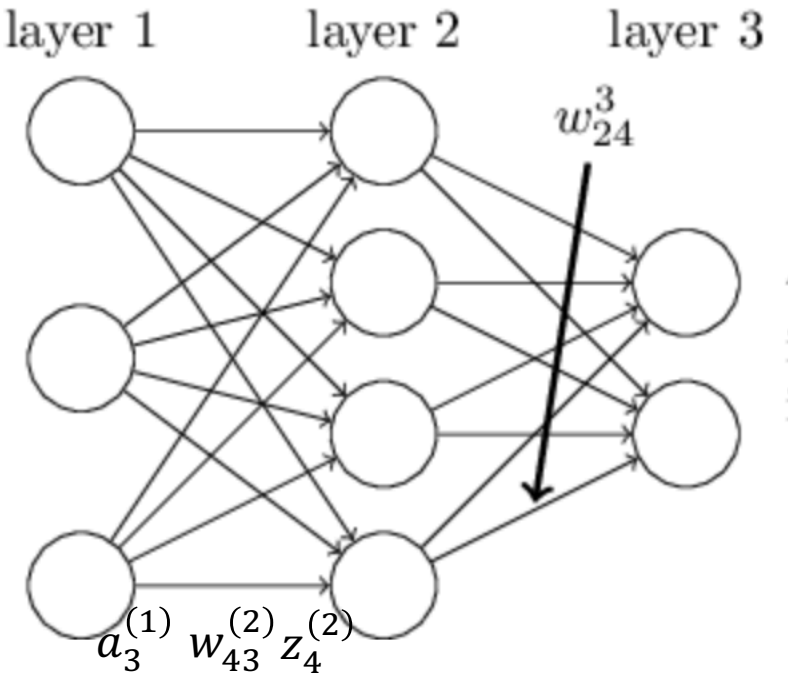


$w_{ji}^{(l)}$ is the weight from the ith neuron in the $(l-1)$th layer to the jth neuron in the lth layer

$$a_1^{(2)} = f\left(w_{11}^{(2)}a_1^{(1)} + w_{12}^{(2)}a_2^{(1)} + w_{13}^{(2)}a_3^{(1)} + b_1^{(2)}\right)$$

$$a_j^{(l)} = f\left(\sum_i w_{ji}^{(l)}a_i^{(l-1)} + b_j^{(l)}\right)$$

Multi-Layer Neural Network



Forward Propagation.

$$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$$
$$a^{(l)} = f\left(z^{(l)}\right)$$

Activation

$z^{(l)}$ is called the *weighted input* to neurons in layer l

Neural Network Architecture.
Arrangement of neurons, e.g. number of neurons in each layer.

Backpropagation

Training Loss

$w_{ji}^{(l)}$ is the weight from the ith neuron in the $(l-1)$th layer to the jth neuron in the lth layer

Point Loss

$$J(W,b;x,y) = \frac{1}{2}\|h_{W,b}(x) - y\|^2 + \frac{\lambda}{2}\Sigma_{l=1}^{L}\Sigma_{i=1}^{s_{l-1}}\Sigma_{j=1}^{s_l}\left(w_{ji}^{(l)}\right)^2$$

weight decay regularization

Training Loss

$$J(W,b)$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\Sigma_{l=1}^{L}\Sigma_{i=1}^{s_{l-1}}\Sigma_{j=1}^{s_l}\left(w_{ji}^{(l)}\right)^2$$

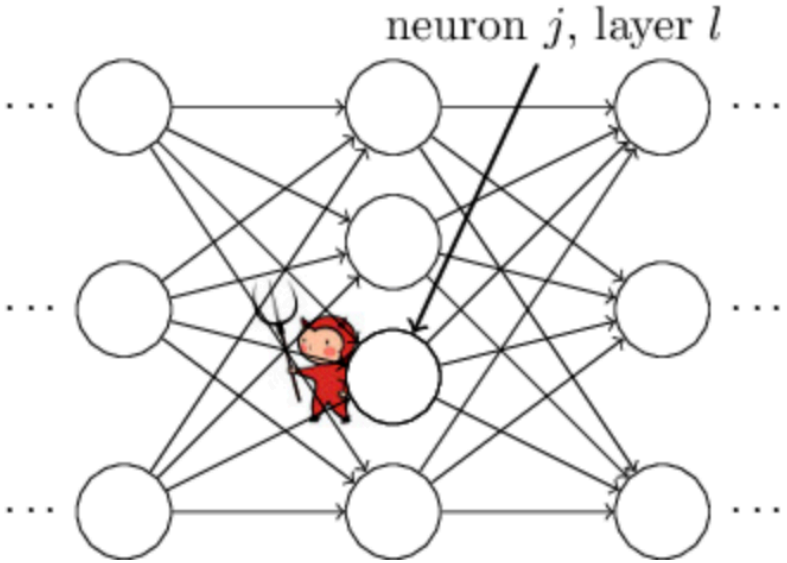
λ weight decay parameter

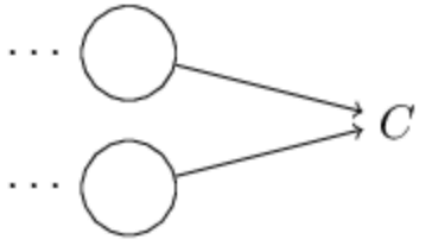
Core of Backpropagation

- The partial derivative $\frac{\partial J}{\partial w}$ of the cost function J with respect to any weight w (or bias b) in the network.
- The expression tells us how quickly the loss changes when we change the weights and biases
- Backpropagation is about understanding how changing the weights and biases in a network changes the cost function

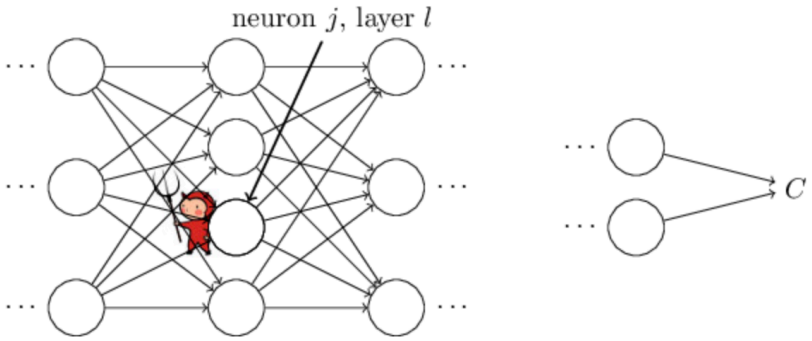
Before that

- We define an intermediate quantity $\delta_j^{(l)}$. We call it the error of in the jth neuron in the lth layer.
- We first compute $\delta_j^{(l)}$ before $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$





$\delta_j^{(l)}$



- The demon sits at jth neuron in the lth layer
- As the input to the neuron comes in, the demon messes with the neuron's operation
- It adds a little change $\Delta z_j^{(l)}$ to the neuron‘s weighted input, so that instead of outputting $f\left(z_j^{(l)}\right)$, the neuron outputs $f\left(z_j^{(l)} + \Delta z_j^{(l)}\right)$.
- This change propagates through later layers in the network, finally causing the overall loss to change by an amount $\frac{\partial J}{\partial z_j^{(l)}} \Delta z_j^{(l)}$
- Now, the demon tries to find $\Delta z_j^{(l)}$ to make the loss smaller.
- Suppose $\frac{\partial J}{\partial z_j^{(l)}}$ has a large value (either positive or negative)
- Then the demon can lower the cost quite a bit by choosing $\Delta z_j^{(l)}$ to have the opposite sign to $\frac{\partial J}{\partial z_j^{(l)}}$
- What if $\frac{\partial J}{\partial z_j^{(l)}}$ is small?

$\delta_j^{(l)}$

- Motivated by this story, we define the error $\delta_j^{(l)}$ of in the jth neuron in the lth layer by

$$\delta_j^{(l)} \equiv \frac{\partial J}{\partial z_j^{(l)}}$$

- We use $\delta^{(l)}$ to denote the vector of errors associated with layer l
- Backpropagation will give us a way of computing $\delta^{(l)}$ for every layer, and then relating those errors to the quantities of real interest, $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$

Error in the output layer $\delta_j^{(L)}$

- Because $a^{(L)} = f(z^{(L)})$

$$\delta_j^{(L)} = \frac{\partial J}{\partial a_j^{(L)}} f'(z_j^{(L)})$$

- The first term on the right measures how fast the cost is changing as a function of the jth output activation
- The second term on the right measures how fast the activation function f is changing at $z_j^{(L)}$

An equation for the Error in the output layer $\delta_j^{(n_l)}$

- Because $a^{(L)} = f(z^{(L)})$

$$\delta_j^{(L)} = \frac{\partial J}{\partial a_j^{(L)}} f'(z_j^{(L)})$$

- The first term on the right measures how fast the cost is changing as a function of the jth output activation
- The second term on the right measures how fast the activation function f is changing at $z_j^{(L)}$
- Recall $J = \frac{1}{2}\sum_j \left(y_j - a_j^{(L)}\right)^2$
- Then we have $\frac{\partial J}{\partial a_j^{(L)}} = \left(a_j^{(L)} - y_j\right)$
- So we have, $\delta^{(L)} = \left(a^{(L)} - y\right) \bullet f'\left(z^{(L)}\right)$

• denotes element-wise multiplication

$$\begin{bmatrix}1\\2\end{bmatrix} \bullet \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}3\\8\end{bmatrix}$$

An equation for the error $\delta^{(l)}$ in terms of the error in the next layer $\delta^{(l+1)}$

- $\delta^{(l)} = \left(\left(W^{(l+1)} \right)^T \delta^{(l+1)} \right) \bullet f'\left(\mathrm{z}^{(l)} \right)$
- $W^{(l+1)}$ is the weight matrix for the $(l+1)$th layer
- When we apply $\left(W^{(l+1)} \right)^T$, we can think intuitively of this as moving the error *backward* through the network, giving us some sort of measure of the error at the output of the lth layer
- With $f'\left(\mathrm{z}^{(l)} \right)$, this moves the error backward through the activation function in layer l, giving us the error $\delta^{(l)}$ in the weighted input to layer l.

A proof

Chain Rule for Neural Networks.

$$\frac{\partial}{\partial W_{ij}^{(l)}}\left(\frac{1}{2}\left\|a^{(L)}-y\right\|^2\right)=\left(a^{(L)}-y\right)\frac{\partial}{\partial W_{ij}^{(l)}}f\left(z^{(L)}\right)$$

$$=\left(a^{(L)}-y\right)f'\left(z^{(L)}\right)\frac{\partial}{\partial W_{ij}^{(l)}}\left(W^{(L-1)}a^{(L-1)}+b^{(L-1)}\right)$$

$$=\left(a^{(L)}-y\right)f'\left(z^{(L)}\right)W^{(L-1)}\frac{\partial}{\partial W_{ij}^{(l)}}a^{(L-1)}$$

$$=\underbrace{\left(a^{(L)}-y\right)f'\left(z^{(L)}\right)}_{\delta^{(L)}}\underbrace{W^{(L-1)}f'\left(z^{(L-1)}\right)}_{\delta^{(L-1)}}\frac{\partial}{\partial W_{ij}^{(l)}}z^{(L-1)}$$

A combination of two equations

- $\delta_j^{(L)} = \frac{\partial J}{\partial a_j^{(L)}} f'\left(z_j^{(L)}\right)$ (eq. 1)
- $\delta^{(l)} = \left(\left(W^{(l+1)}\right)^T \delta^{(l+1)}\right) \bullet f'\left(\mathrm{z}^{(l)}\right)$ (eq. 2)
- We can compute the error $\delta^{(l)}$ for any layer in the network. We start by using (eq. 1) to compute $\delta_j^{(L)}$, then apply Equation (eq. 2) to compute $\delta^{(L-1)}$, then Equation (eq. 2) again to compute $\delta^{(L-2)}$, and so on, all the way back through the network.

An equation for the rate of change of the loss with respect to any bias in the network

- $\frac{\partial J}{\partial b_j^{(l)}} = \delta_j^{(l)}$
- This is great news
- δ is being evaluated at the same neuron as the bias b

- Can you prove it?
- Hint: $\delta_j^{(l)} \equiv \frac{\partial J}{\partial z_j^{(l)}}$

$$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$$

$$\text{or } z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

An equation for the rate of change of the cost with respect to any weight in the network

- $\frac{\partial J}{\partial w_{ji}^{(l)}} = a_i^{(l-1)} \delta_j^{(l)}$ or $\frac{\partial J}{\partial w^l} = a^{(l-1)} \delta^{(l)}$
- Can you prove it?

The Backpropagation algorithm

1. **Input x:** Set the corresponding activation $a^{(0)}$ for the input layer.
2. **Feedforward:** For each $l = 1, 2, 3, \ldots, L$ compute $z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$ and $a^{(l)} = f\left(z^{(l)}\right)$
3. **Output error δ^L:** Compute the vector $\delta_j^{(L)} = \frac{\partial J}{\partial a_j^{(L)}} f'(z_j^{(L)})$
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 1$, compute $\delta^{(l)} = \left(\left(W^{(l+1)}\right)^T \delta^{(l+1)}\right) \bullet f'\left(\mathrm{z}^{(l)}\right)$
5. **Output:** The gradient of the cost function is given by $\frac{\partial J}{\partial b_j^{(l)}} = \delta_j^{(l)}$ and $\frac{\partial J}{\partial w_{ji}^{(l)}} = a_i^{(l-1)}\delta_j^{(l)}$

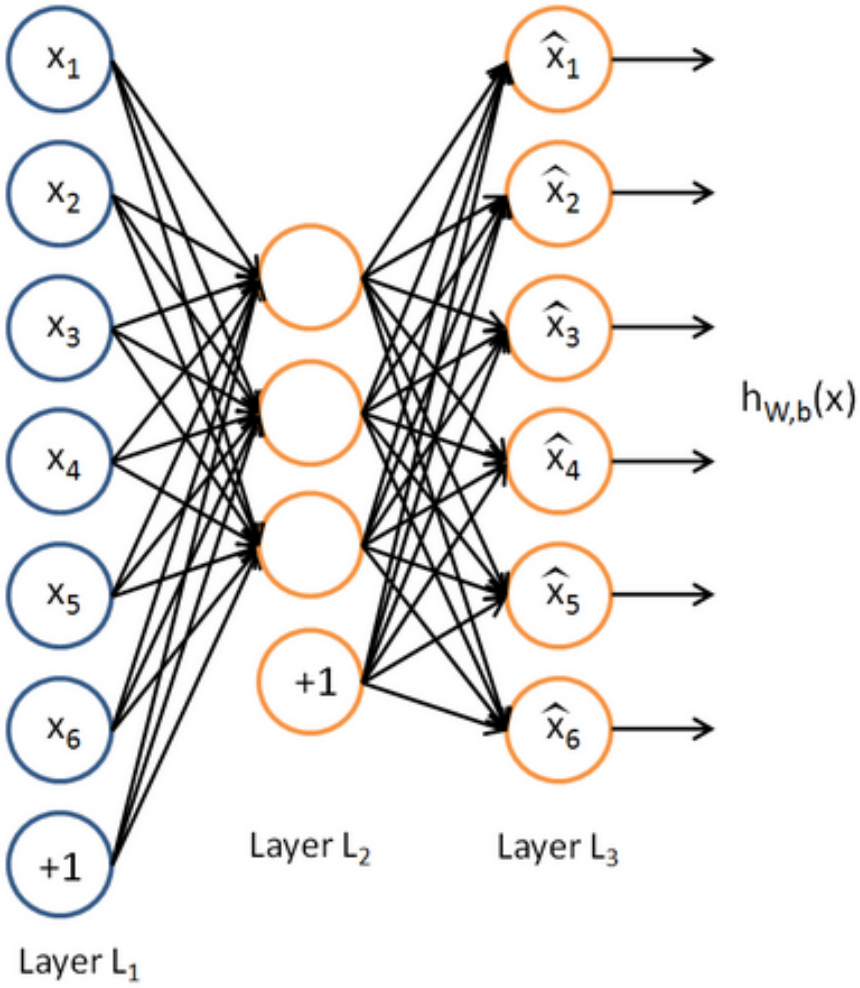
Autoencoder

Autoencoder

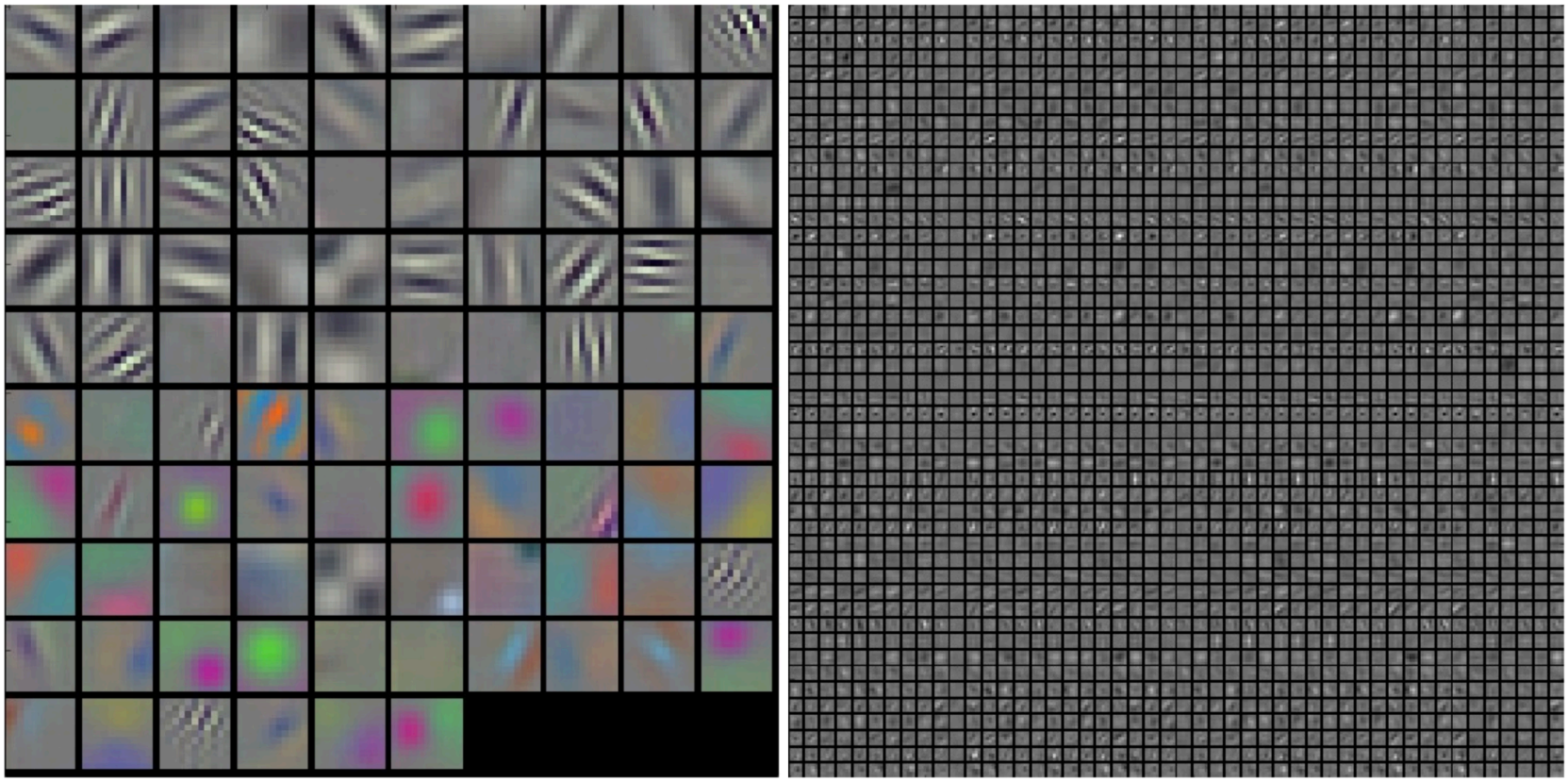
Training a multilayer neural network to reconstruct the input from a dimension reduced representation.

Strategies for Dimensionality Reduction

- Few hidden neurons
- Sparse activations



Learned filters



Typical-looking filters on the first CONV layer (left), and the 2nd CONV layer (right) of a trained AlexNet. Notice that the first-layer weights are very nice and smooth, indicating nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features. The 2nd CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

History

* not in syllabus

Ridiculously Simplified History

1943 Artificial Neuron (McCullough, Pitts)

1957 Perceptrons (Rosenblatt)

1969 Problem with XOR (Minsky, Papert)

FIRST AI WINTER

1986 Backpropagation (Rumelhart, Hinton, Williams)

1989 Convolutional Neural Nets (LeCun)
Autoencoders, Belief Nets, Recurrent NN, Reinforcement

1995 Problems with Backprop.
Rise of SVMs and Random Forests.

SECOND AI WINTER

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

* not in syllabus

Deep Learning Conspiracy

Yann LeCun,
Geoffrey Hinton,
Yoshua Bengio,
Andrew Ng

2006 Greedy Initialization of Layers

2009 Graphics Processing Units

2012 Dropout (ImageNet)

What was wrong with BackPropagation in 1986?

1. Our labeled datasets were thousands of times too small.
2. Our computers were millions of times too slow.
3. We initialized the weights in a stupid way.
4. We used the wrong type of non-linearity.

Hinton, “Deep Learning,” https://royalsociety.org/events/2015/05/breakthrough-science-technologies-machine-learning/

Summary

- Multilayer Neural Networks
  - Neuron
  - Activation Function
  - Forward Propagation
- Learning Algorithm
  - Cost Function
  - Backpropagation
  - Autoencoders

Intended Learning Outcomes

Deep Learning

- Describe how the output of an artificial neuron relates to its inputs. Give examples of activation functions which are commonly used.
- Describe how the output of a multilayer neural network relates to its inputs. In particular, write down formulas for forward propagation. Given a network, identify the neural network architecture.
- Write down the training loss of a feedforward network. Derive the gradient formulas in backpropagation from the training loss.
- Give a definition of an autoencoder. Explain how they are used for dimensionality reduction. Describe two strategies for reduction.
- List some successful applications of deep learning. Give four reasons for the recent success of deep learning.