Recall: The Neuron Metaphor

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node

Slide Credit: HKUST
Types of Neurons

Linear Neuron

Logistic Neuron

Perceptron

Potential more. Require a convex loss function for gradient descent training.
Limitation

- A single “neuron” is still a linear decision boundary

- What to do?

- Idea: Stack a bunch of them together!
Multilayer Networks

- Cascade Neurons together
- The output from one layer is the input to the next
- Each Layer has its own sets of weights

\[ f(x, \theta) \]
Fig. 4. (a) Not recommended: the standard logistic function, \( f(x) = 1/(1 + e^{-x}) \). (b) Hyperbolic tangent, \( f(x) = 1.7159 \ \tanh\left(\frac{2}{3}x\right) \).
Rectified Linear Units (ReLU)
Supervised Learning

$$\{ (x^i, y^i), i = 1 \ldots P \}$$ training dataset

$x^i$ i-th input training example

$y^i$ i-th target label

$P$ number of training examples

Goal: predict the target label of unseen inputs.
Supervised Learning: Examples

Classification

Denoising

OCR
Supervised Deep Learning

Classification

Denoising

OCR

“dog”

“2345”
Forward Propagation

\[ x \rightarrow \max(0, W^1 x) \rightarrow \max(0, W^2 h^1) \rightarrow W^3 h^2 \rightarrow o \]

\[ x \in \mathbb{R}^D \quad W^1 \in \mathbb{R}^{N_1 \times D} \quad b^1 \in \mathbb{R}^{N_1} \quad h^1 \in \mathbb{R}^{N_1} \]

\[ h^1 = \max(0, W^1 x + b^1) \]

- \( W^1 \) 1-st layer weight matrix or weights
- \( b^1 \) 1-st layer biases

The non-linearity \( u = \max(0, v) \) is called ReLU in the DL literature. Each output hidden unit takes as input all the units at the previous layer: each such layer is called “fully connected”.
Forward Propagation

\[ x \xrightarrow{\max(0, W^1 x)} h^1 \xrightarrow{\max(0, W^2 h^1)} h^2 \xrightarrow{W^3 h^2} o \]

\[ h^1 \in \mathbb{R}^{N_1}, \quad W^2 \in \mathbb{R}^{N_2 \times N_1}, \quad b^2 \in \mathbb{R}^{N_2}, \quad h^2 \in \mathbb{R}^{N_2} \]

\[ h^2 = \max(0, W^2 h^1 + b^2) \]

\[ W^2 \quad \text{2-nd layer weight matrix or weights} \]

\[ b^2 \quad \text{2-nd layer biases} \]
Forward Propagation

\[ x \rightarrow \max(0, W^1 x) \rightarrow \max(0, W^2 h^1) \rightarrow W^3 h^2 \rightarrow o \]

\[ h^2 \in \mathbb{R}^{N_2}, \quad W^3 \in \mathbb{R}^{N_3 \times N_2}, \quad b^3 \in \mathbb{R}^{N_3}, \quad o \in \mathbb{R}^{N_3} \]

\[ o = \max(0, W^3 h^2 + b^3) \]

- \( W^3 \): 3-rd layer weight matrix or weights
- \( b^3 \): 3-rd layer biases
Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

$$\theta^* = \arg \min_{\theta} \sum_{n=1}^{P} L(x^n, y^n; \theta)$$

**Question:** How to minimize a complicated function of the parameters?

**Answer:** Chain rule, a.k.a. Backpropagation! That is the procedure to compute gradients of the loss w.r.t. parameters in a multi-layer neural network.

Given $\frac{\partial L}{\partial o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial W^3}$$

$$\frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2}$$
Backward Propagation

Given \( \frac{\partial L}{\partial h^2} \) we can compute now:

\[
\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial W^2}
\]

\[
\frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial h^1}
\]
Backward Propagation

Given $\frac{\partial L}{\partial h^1}$ we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial h^1} \frac{\partial h^1}{\partial W^1}$$
Convolutional Neural Networks
Fully Connected Layer

Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

Stationarity? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Ranzato
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 1 & -1 \\
-1 & 1 & -1
\end{bmatrix}
\begin{bmatrix}
\end{bmatrix}
\]
Convolutional Layer

Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
Convolutional Layer

\[ h_j^n = \max(0, \sum_{k=1}^{K} h_{kj}^{n-1} \ast w_{kj}^n) \]

- output feature map
- input feature map
- kernel
Pooling Layer

Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?
Pooling Layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
Pooling Layer: Examples

Max-pooling:

$$h^n_j(x, y) = \max_{\bar{x} \in N(x), \bar{y} \in N(y)} h^{n-1}_j(\bar{x}, \bar{y})$$

Average-pooling:

$$h^n_j(x, y) = \frac{1}{K} \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h^{n-1}_j(\bar{x}, \bar{y})$$

L2-pooling:

$$h^n_j(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h^{n-1}_j(\bar{x}, \bar{y})^2}$$

L2-pooling over features:

$$h^n_j(x, y) = \sqrt{\sum_{k \in N(j)} h^{n-1}_k(x, y)^2}$$
Convolutional Nets

- Example:
Alex Net

Krizhevsky et al. NIPS 2012
Visualizing Learned Filters

Layer 1

Layer 2

Figure Credit: [Zeiler & Fergus ECCV14]
Visualizing Learned Filters

Layer 3

Figure Credit: [Zeiler & Fergus ECCV14]
Visualizing Learned Filters

Layer 4

Layer 5

Figure Credit: [Zeiler & Fergus ECCV14]
Industry Deployment

- Used in Facebook, Google, Microsoft
- Image Recognition, Speech Recognition, ....
- Fast at test time

Taigman et al. DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR’14