Learning Models

$x$ → $?$ → $y$ → $\hat{y}$ → error → $\hat{y}$

Cat!
Learning Models

Neural Networks (or learned models) are good when:

• Your system needs to learn and adapt
• Original is highly nonlinear / multi-variable
• Physics / model based approaches are not available or are too computationally expensive
What is image recognition?
From Shallow to Deep Learning

Traditional “Shallow” Pipeline

- Feature representation
- Trainable classifier
- Class label

“Deep” Recognition Pipeline

- Layer 1
- Layer 2
- Simple classifier
Classification Improvements
Deep Learning in Computer Vision Conferences

http://jponttuset.cat/are-gans-the-new-deep/
History of Neural Networks

- Created in 1957 and presented as the first learning machine
- In 1969, Marvin Minsky and Seymour Papert showed that a perceptron could not learn an XOR function
  - Three years later, a series of papers introduced networks capable of modeling nonlinear fnns and XOR
- Original work is often-miscited, and caused a significant decline in interest and funding of neural network research
  - It took ten more years for research to recover – resurgence in the 1980s and again in 2010s
- **Key take away:** neural networks may be powerful, but so are words.

“...the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”

Mark I Perceptron by Frank Rosenblatt from Wikipedia
The Perceptron

Activation function

Classification

Output
If the classes are linearly separable, then $\exists$ a weight vector $w$ such that:

$w^T x \geq 0 \quad \forall x \text{ assoc } w/ c_1, \\
w^T x < 0 \quad \forall x \text{ assoc } w/ c_2$

Dataset $w/ K$ examples
while epoch < max or not converged

For $k = 1:K$

$\hat{y}_k = f(w_k^T x_k)$

check if $\hat{y}_k$ is correct

if yes

$w_{k+1} = w_k$

if not

$w_{k+1} = w_k + \eta x_k$, if $\hat{y}_k = 0$, $w_k$ is incorrect

learning rate
Recall the dot product
\[ w^T x = \|w\| \|x\| \cos \theta \]
Training via Gradient Descent

\[ w_{k+1} = w_k + \Delta w_k \]

\[ = w_k + \eta_k (y_k - \hat{y}_k) x_k \]

\( \rightarrow \) learning rate
General Optimization Problems

One a cost function is decided, an optimization problem is formulated

\[
\begin{align*}
\min_{x,u} & \quad J(x,u) \\
\text{subject to} & \quad g(x,u) \geq 0
\end{align*}
\]

• Depending on the cost function, this problem can be nice and easy to solve or quite difficult

• Convex optimization problems have a global minimum, and are readily solved

• What if we have a nonlinear or nonconvex problem?
Extensions to Multi-Class Classification

• one vs one
  • Train $\frac{K(K-1)}{2}$ binary classifiers for a K-way multiclass problem
  • Voting scheme is then applied

• one vs all
  • Train $K$ classifiers where each class is the positive and all others are negatives

• Neural networks with multiple outputs
The Famous XOR Counterexample

Suppose we have a perceptron with a linear activation:

\[ y = f(w^T x) = w^T x \]

\[ y = [w_0 \ w_1 \ w_2][1 \ x_1 \ x_2]^T = w_0 + w_1 x_1 + w_2 x_2 \]

XOR gives the following inequalities:

\[ w_0 + w_1 0 + w_2 0 \leq 0 \rightarrow w_0 \leq 0 \]
\[ w_0 + w_1 0 + w_2 1 > 0 \rightarrow w_0 > -w_2 \]
\[ w_0 + w_1 1 + w_2 0 > 0 \rightarrow w_0 > -w_1 \]
\[ w_0 + w_1 1 + w_2 1 \leq 0 \rightarrow w_0 \leq -w_1 -w_2 \]

\[ w_2 > 0 \]
\[ w_1 > 0 \]
\[ w_1 + w_2 \leq 0 \]
Universal Function Approximators

Let \( \varphi : \mathbb{R} \to \mathbb{R} \) be a nonconstant, bounded, and continuous function. Let \( I_m \) denote the \( m \)-dimensional unit hypercube \([0, 1]^m\). The space of real-valued continuous functions on \( I_m \) is denoted by \( C(I_m) \). Then, given any \( \varepsilon > 0 \) and any function \( f \in C(I_m) \), there exist an integer \( N \), real constants \( v_i, b_i \in \mathbb{R} \) and real vectors \( w_i \in \mathbb{R}^m \) for \( i = 1, \ldots, N \), such that we may define:

\[
F(x) = \sum_{i=1}^{N} v_i \varphi(w_i^T x + b_i)
\]

as an approximate realization of the function \( f \); that is,

\[
|F(x) - f(x)| < \varepsilon
\]

for all \( x \in I_m \). In other words, functions of the form \( F(x) \) are dense in \( C(I_m) \).

A feedforward network with a single hidden layer containing a finite number of units can approximate continuous functions on compact subsets of \( \mathbb{R}^n \), under mild assumptions on the activation function.
Multi-Layer Perceptron (MLP)

Layer 1 $\rightarrow$ Layer 2 $\rightarrow$ Layer 3 $= L$

Weights: $W^l_{jk}$ assoc $l-1$
Bias: $b^l_j$
Activation: $a^l_j = f(\sum_{k} W^l_{jk} a^{l-1}_k + b^l_j)$
Universal Approximators

Visual proof by Michael Nielson.
Neural Networks and Deep Learning / Chap. 4: A visual proof that neural nets can compute any function
Universal Approximator

Visual proof by Michael Nielsen.
Neural Networks and Deep Learning / Chap. 4: A visual proof that neural nets can compute any function
Universal Approximators

Final output is a weighted sum of hidden layer outputs.

Visual proof by Michael Nielson.
Neural Networks and Deep Learning / Chap. 4: A visual proof that neural nets can compute any function
Universal Approximators
Universal Approximators

Given a smooth function, network parameters can be adapted to find a reasonable approximation

- single-layer neural networks essentially build a lookup table
Backpropagation (1)
Backpropagation (2)

Optimize weights w.r.t. errors

\[ E(w) = \frac{1}{2} \sum_{r} e^2 \]

\[ e = y_k - q_k \]

output unit

We propose:

\[ w_{ji} \leftarrow w_{ji} + \eta \frac{\partial E}{\partial w_{ji}} \]

\[ w_{ji} \leftarrow w_{ji} - \Delta w_{ji} \]

Sensitivity of error w.r.t. \( w \)
Backpropagation (3)

\[ \frac{dE}{dw_i} = -2E \frac{\partial E}{\partial e} \cdot \frac{\partial e}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_i} \]

- \( e_i \): error on unit \( i \)
- \( a_i \): output from unit \( i \)
- \( z_i \): weighted input - \( a_i = f(z_i) \)
- \( \Delta E \): change in error with respect to weight \( w_i \)

Output from unit \( i \)
Backpropagation (4)

\[
\begin{align*}
    w_{ij} & \leftarrow w_{ij} - \eta \frac{dE}{dw_{ij}} \\
    z_j & \leftarrow w_{ji} + \sum_{i} \theta_i f_i a_i
\end{align*}
\]

\[
\implies \text{for layer } L, \text{ we know } \varepsilon_L \rightarrow \text{what about hidden layers?}
\]
Backpropagation (5)

for any unit $p$ connected to $q$

$$\Delta w_{pq} = \eta \delta_q a_p$$  \text{local gradient}

if $q$ is output:

$$\delta_q = e_q f_q'(\cdot)$$

if $q$ is hidden

$$\delta_q = \sum_r \delta_r w_{rq} f_q'(\cdot)$$
Backpropagation Review

1. Input: $x$
2. **Feedforward**: For each layer, compute $z^l$ and $a^l$
3. **Compute error**: $E$ or $\delta^l$
4. **Backpropagate** the error: For each inner layer, compute the delta rule
5. **Output**: The gradient of the cost function
Next time...

• Deep architectures
• Tips and tricks
• Common applications and uses
• Risks for autonomy