Agenda

Last lecture:

- Linear models and the Perceptron
- Perceptron mistake bound
- Online to batch conversion generalization bound
- Started: an artificial neuron
- In tutorial: Gradient-based optimization, linear regression

Today:

- Binary neurons and their expressiveness
- Activation functions
- Gradient descent
- Linear and Logistic regressions
- Stochastic gradient descent
A biological neuron

- Cell body
- Axon
- Telodendria
- Synaptic terminals
- Nucleus
- Axon hillock
- Golgi apparatus
- Endoplasmic reticulum
- Mitochondrion
- Dendrite
- Dendritic branches
Average neuron receives: $10^3 - 10^4$ inputs (spike pulses)
Neuron generates one output, duplicated to $10^3 - 10^4$ synaptic terminals (at the end of its axon)
Electric current speed: 1-100m/sec
Firing rate $\leq 200$ spikes/sec (average 10 spikes/sec)
Size in mice: - $10^5$ neurons in 1mm$^2$ in which there are $7 \times 10^8$ synaptic connections (about 4km wiring)
Many neuron types - no consensus on how to classify
Neuron firing dynamics models: quite complicated (involve several differential equations to model electric potential activities)
Perceptron as artificial neuron approximation

\[ \hat{y} = a \left( \sum_i w_i x_i \right) \]

Activation

Weights \( w_1, w_2, \ldots, w_d \)

Input \( x_1, x_2, \ldots, x_d \)
Perceptron - with bias

\[ \hat{y} = a \left( b + \sum_i w_i x_i \right) \]

1943 McCulloch + Pitts
On the expressiveness of a single (binary) neuron

**Claim:** If $x_1, x_2, \ldots, x_k$ are binary variables, $x_i \in \{\pm 1\}$, we can implement AND, OR and NOT gates, each with a single binary neuron.

**AND gate:**

$$f_{\text{AND}}(x_1, \ldots, x_k) = \bigwedge_{i=1}^{k} x_i$$

$$\leq x_i - k + 1 = k - k + 1 = 1$$

$$\leq x_i - k + 1 \leq 1 + k - 1 - k + 1 = -1 < 0$$

$$\omega_i = 1$$

$$b = 1 - k$$
On the expressiveness of a single (binary) neuron

**OR gate:**

\[
\text{for } = \text{sign} \left( \sum_{i=1}^{k} x_i + k - 1 \right)
\]

\[
\begin{array}{c|c|c|c|c|c}
\text{Input} & \text{w1} & \text{w2} & \text{w3} & \text{w4} & \text{w5} \\
\hline
\text{w1} & -1 & 1 & -1 & 1 & -1 \\
\hline
\text{w2} & 1 & -1 & 1 & -1 & 1 \\
\hline
\text{w3} & -1 & 1 & -1 & 1 & -1 \\
\hline
\text{w4} & 1 & -1 & 1 & -1 & 1 \\
\hline
\text{w5} & -1 & 1 & -1 & 1 & -1 \\
\end{array}
\]

\[
\text{Output} = \text{for}
\]

Ran El-Yaniv  Deep Learning 236606, Winter 2019
A set $S$ (in a vector space) is **convex** if for any two vectors $u, v \in S$, for any scalar $\lambda \in [0, 1],$

$$\lambda u + (1 - \lambda)v \in S$$
Linear separability

- **Input space**: the set of possible inputs
- A (binary labeled) training set $S_m$ is **linearly separable** if there exists a hyperplane $H_w$ that separates the pluses and minuses (assume bias absorbed in $w$)
- **Easy result**: If $S_m$ is linearly separable, the set of $+1$ labeled points is convex (and the set of $-1$ labeled points is convex)

$$w \cdot u + b > 0$$
$$w \cdot v + b > 0$$

$$w \cdot (\lambda \cdot u + (1-\lambda)v) + b$$

$$= \lambda (w \cdot u + b) + (1-\lambda)(w \cdot v + b) > 0$$
- **Model (weight) space**: set of possible weights

For any $\mathbf{x}$ with label $y \in \{\pm 1\}$, let

$$W_x = \{w | yw^T \mathbf{x} \geq 0\}.$$  

Easy: $W_x$ is a half-space and convex.

- The feasible set of models: For a training set $S_m$, the model subspace that classifies all points in $S_m$ correctly is

$$W(S_m) = \bigcap_i W_{x_i}.$$  

$W(S_m)$ is called the **feasible set** (and also the “version space”)

- The feasible set $W(S_m)$ is convex. If $S_m$ is linearly separable, $W(S_m) \neq \emptyset$
Example: the feasible set
Minsky & Papert "Perceptrons"

Perceptron cannot solve XOR

\[\begin{array}{ccc}
X_1 & X_2 & y \\
-1 & -1 & -1 \\
-1 & 1 & 1 \\
1 & -1 & 1 \\
1 & 1 & -1 \\
\end{array}\]

\[\begin{array}{ccc}
X_1 & X_2 & y \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}\]
Perceptron solves XOR: with feature generation

Instead of $x$, let’s feed the perceptron with a vector of generated features

$$\phi(x) = (\phi_1(x), \ldots, \phi_D(x)),$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^D$. Can we invent a useful $\phi$?
Activation Functions
Activation functions: Binary

\[ z = \mathbf{w} \cdot \mathbf{x} + b \]

\[ a(z) = \text{sign}(z) \]
Activation functions: Tanh

\[ a(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]

\[ \tanh(z) \]
Activation functions: Sigmoid

\[ a(z) = \frac{1}{1 + e^{-z}} \]

\[ \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]

\[ \tanh(z) = 2 \tanh(\frac{z}{2}) - 1 \]
Activation functions: **ReLU**

Rectified Linear

\[ a(z) = \max\{0, z\} \]
Activation functions: ReLU variants

**gReLU**: generalized ReLU

\[ gReLU(x) = \max\{x, 0\} + \alpha \min\{x, 0\} \]

- **Leaky-ReLU**:
  \[ \alpha \text{ small } \approx 0.01 \]

- **Parametric-ReLU**:
  \[ \alpha \text{ trained} \]

- **Exponential-Linear (ELU)**:
  \[ \alpha = a(e^x - 1) \]
Activation functions: Linear
Gradient Descent
Gradient descent

For a multivariate function $f(x) = f(x_1, \ldots, x_n)$:

- The gradient of $f$ is the vector of all partial derivatives,

$$\nabla f(x) = \left[ \frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n} \right]^T$$

- Steepest descent given by: $\nabla f(x)$
  
  (gradient minimizes directional derivative $\nabla_{u}f(x)$)
Gradient descent
In general: Life isn’t easy for gradient descenders
Gradient descent (GD) algorithm

Gradient descent algorithm

- Initialize $\theta_1$ (e.g., $\theta_1 = 0$)
- Choose learning rate $\eta$
- Iterate $t$ until convergence ($t = T$)

$$
\theta_{t+1} = \theta_t - \eta \nabla \hat{L}(\theta_t)
$$

- Output final hypothesis $\theta_{GD} = \theta_T$
What can be **achieved** with GD (aka “batch GD”)?

- In general: it will achieve **local minimum** of $\hat{L}$
- But... choice of **learning rate** can be critical:
  - **too small**: painfully **slow** convergence
  - **too large**: fluctuations or even **divergences**
GD is a first order optimization method. There are many variants (we will see some) and many second-order versions.

For convex, Lipschitz-bounded learning problems (like logistic regression!) there are nice guarantees. For example,

$$\hat{L}(\theta_{\text{GD}}) - \hat{L}(\theta^*) \leq \frac{B\alpha}{\sqrt{T}}$$

Faster rates for Lipschitz smooth and strongly convex problems
Convex function

Let $S$ be a convex set. $f : S \to \mathbb{R}$ is convex if for any $u, v \in S$, $\lambda \in [0, 1],$

$$f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda)f(v)$$

**Basic result:** $f : \mathbb{R} \to \mathbb{R}$ is convex (a) iff $f'$ is non-decreasing; (b) iff $f''$ is non-negative

**Example:** $f(x) = x^2$ is convex

**Example:** $f(x) = \log(1 + e^x)$ is convex
Lemma

Let \( f : \mathbb{R}^d \rightarrow \mathbb{R} \), where

\[
f(w) = g(w \cdot x + b),
\]

where \( x \in \mathbb{R}^d \) and \( g : \mathbb{R} \rightarrow \mathbb{R} \).

Then, convexity of \( g \) implies convexity of \( f \)
Examples

1. \( f : \mathbb{R}^d \rightarrow \mathbb{R} \)
   \[ y \in \mathbb{R}^2, \quad w, x \in \mathbb{R}^d \]
   \[ f(w) = (w \cdot x - y)^2 \]

2. \( g(x) = a^2 \)  \( f \)  \( g(x) \)
   \( f \)  \( \log (1 - e^{-y(w \cdot x + b)}) \)
   \( y \in \{0, 1\}, \quad w, x \in \mathbb{R}^d \)
A learning problem \((\mathcal{H}, \mathcal{X}, \mathcal{Y}, \ell)\) is convex if \(\mathcal{H}\) is a convex set, and for all \((x, y) \in \mathcal{X} \times \mathcal{Y}\), the loss function

\[
\ell(h, x, y) = \ell(y, h(x))
\]

is convex in \(h\)

**Examples:**

- If \(\ell\) is convex (in \(h\)), then for any sample \(S_m\) the empirical risk \(\hat{L}(h)\) is convex. The true loss \(L_D(h)\) is also convex
- Linear regression (squared loss) is convex
We saw a neuron with a **linear activation** function, \( h(x) = w^T x + b \). This neuron can solve **linear regression**:

- We are given a training set \( S_m = \{(x_1, y_1), \ldots, (x_m, y_m)\} \), \( y_i \in \mathbb{R} \), \( x_i \in \mathbb{R}^d \)

- To quantify individual errors we use the **squared loss** function

\[
\ell(y, \hat{y}) = (y - \hat{y})^2
\]

- The **empirical error** of a neuron \( w, b \) is

\[
\hat{L}(w, b) = \sum_{i=1}^{m} (y_i - w^T x_i + b)^2
\]
An ERM solution for linear regression given $S_m$ means

$$(\hat{w}, \hat{b}) = \arg \min_{w, b} \hat{L}(w, b)$$

In tutorial we saw:
- analytical solution involving matrix inversion
- a solution using gradient descent
Recall analytical solution

- Arrange training set $S_m$ as a $m \times d$ matrix $X$ (row $i$ is $x_i$) let $y$ (vector) be training labels
- Linear regression predictions: $\hat{y} = Xw + 1b$ (but we absorb bias into $w$ extending $w$ to $w \in \mathbb{R}^{d+1}$)
- Loss function is $\hat{L}(w, b) = \frac{1}{2m}||y - \hat{y}||^2$
- Differentiating:
  \[
  \nabla_w \hat{L} = \frac{1}{m}X^T(Xw - y)
  \]
- And final (global optimum) solution is:
  \[
  w^* = (X^T X)^{-1} Xy
  \]
Recall GD solution

- Gradient:
  \[ \nabla_w \hat{L} = \frac{1}{m} X^T (Xw - y) \]

- Apply gradient descent algorithm

- Linear regression is a convex learning problem and GD can potentially converge (appropriate learning rate)
We saw how to optimize (ERM) a linear neuron (linear regression) using both analytical (closed form) solution and gradient descent.

Which type of solution is better?
Logistic Regression
Recall Sigmoid neuron

The sigmoid activation function:

\[ a(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \]

\[ z = w \cdot x + b \]
The sigmoid neuron $h_{w,b}(x)$ can be used for **binary classification** (where $Y = \{0, 1\}$ or $\{\pm 1\}$). Called **logistic regression**

$$\hat{y} = h_{w,b}(x) = \sigma(w \cdot x + b)$$

- When there are outliers: boundedness of $\sigma$ makes classification with the sigmoid neuron better than classification with a linear neuron
The logistic regression loss function

Consider a classification problem with a binary $Y = \{0, 1\}$.

- **Probabilistic** interpretation ($0 < \sigma < 1$):
  \[
  \sigma(w \cdot x + b) \triangleq \Pr\{y = 1 \mid x\} \quad \Pr\{y = 0 \mid x\} = 1 - \sigma(\cdot)
  \]

- Combined:
  \[
  \Pr\{y \mid x\} = \sigma(\cdot)^y \cdot (1 - \sigma(\cdot))^{1-y}
  \]

- The **negative log-likelihood** of a single point $(x, y)$ is
  \[
  - \log \left( \sigma(\cdot)^y \cdot (1 - \sigma(\cdot))^{1-y} \right) = - \left[ y \log \sigma + (1 - y) \log(1 - \sigma) \right]
  \]

Logistic regression loss function (0/1)

\[
\ell(y, \hat{y}) = - (y \log \hat{y} + (1 - y) \log(1 - \hat{y}))
\]
Remark: the case $Y = \pm 1$

When $Y = \{\pm 1\}$:

- Here again, the probabilistic interpretation:
  \[
  \Pr\{1 \mid x\} = \sigma(w \cdot x + b)
  \]

- For $y = -1$:
  \[
  \Pr\{-1 \mid x\} = 1 - \sigma(\cdot) = \frac{1}{1 + e^{-(w \cdot x + b)}}
  \]

Combined:
\[
\Pr\{y \mid x\} = \frac{1}{1 + e^{y(w \cdot x + b)}}
\]
Logistic loss et al.
Logistic loss function is convex

- **Our loss function:** 
  \[ \ell(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \]

- Recalling \( \hat{y} = \frac{1}{1 + e^{-w \cdot x}} \) we write \( \ell(y, \hat{y}) = \ell(w, x, y) \) and prove it is convex in \( w \)

\[
\begin{align*}
\hat{y} &= \frac{1}{1 + e^{-w \cdot x}} \\
-\log \hat{y} &= -\log \left( \frac{1}{1 + e^{-z}} \right) = \log (1 + e^{-z}) \\
-\log (1 - \hat{y}) &= -\log \left( 1 - \frac{1}{1 + e^{-z}} \right) = -\log \left( \frac{e^{-z}}{1 + e^{-z}} \right)
\end{align*}
\]
Logistic loss function is convex -cntd.
ERM for logistic regression

- We are given a training set \( S_m = \{(x_1, y_1), \ldots, (x_m, y_m)\} \), \( y_i \in \mathbb{R}, \quad x_i \in \mathbb{R}^d \)

- The empirical error of our neuron \((w, b)\) is

\[
\hat{L}(w, b) = \frac{1}{m} \sum_{i=1}^{m} \ell(y_i, \hat{y}_i)
\]

- Denoting \( \theta = (w, b) \), in ERM we would like to solve

\[
\hat{\theta} = \arg \min_{\theta} \hat{L}(\theta)
\]

- Empirical loss function is convex - GD potentially achieves global optimum
SGD
**Stochastic gradient descent (SGD)**

- **Idea:** instead of optimizing based on the entire training set, optimize on each step based on one random example

<table>
<thead>
<tr>
<th>Stochastic gradient descent (SGD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize $\theta_1$, choose learning rate $\eta$</td>
</tr>
<tr>
<td>Iterate $t$ until convergence ($t = T$)</td>
</tr>
<tr>
<td>Choose at random a labeled example $(x, y) \sim D$</td>
</tr>
<tr>
<td>$\theta_{t+1} = \theta_t - \eta \nabla \ell(\theta_t, x, y)$</td>
</tr>
<tr>
<td>Output final hypothesis $\theta_T$</td>
</tr>
</tbody>
</table>
**Computational** advantage over GD per iteration: $O(d)$ instead of $O(dm)$

For large NNs, GD is prohibitively expensive

Is SGD any good?

Surprisingly: in many cases it is outstandingly good

Currently some SGD variants dominate NN training (there are other approaches, such as evolutionary optimizations)
SGD and GD in action
SGD example: linear regression

- For labeled example $t$, $(x_t, y_t)$:
  \[
  \ell(w, x_t, y_t) = \frac{1}{2} (w \cdot x_t - y_t)^2
  \]

- SGD step:
  \[
  w_{t+1} = w_t - \eta \nabla \ell(w, x_t, y_t) = w_t - \eta (w_t \cdot x - y_t)x_t
  \]
SGD example: perceptron

Recall perceptron decision: \( \hat{y}_t = \text{sign}(w_t \cdot x_t) \)

- Loss function:
  \[
  \ell(w, x_t, y_t) = -y_t w \cdot x_t \mathbb{I}(\hat{y}_t \neq y_t)
  \]

- \( \nabla \ell = -y_t x_t \mathbb{I}(\hat{y}_t \neq y_t) \)

- SGD step (set \( \eta = 1 \)):
  \[
  w_{t+1} = w_t - \eta \nabla \ell = w_t + y_t x_t \mathbb{I}(\hat{y}_t \neq y_t)
  \]
Mini-batch SGD and other variants

- Instead of one training example, work with a small batch of examples (e.g. 64)
- Many SGD **variants**: learning rate schedules, memory (momentum), hypothesis averaging, variance reduction
- We will see some of these later on