Multilayer neural networks

Administration

**Final exam** is on December 9, 2019 at 2:30pm-3:45pm
- The same classroom as lectures

**Exam is:**
- Cumulative
- Closed book

**Note:** please bring a simple calculator (not a phone) to make numerical calculations (e.g. probabilities)
Classification with the linear model

The majority of the models covered so far are linear
Example: 2 classes (blue and red points)

Modeling nonlinearities

Feature (basis) functions to model nonlinearities

Linear regression
Logistic regression

\[ f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}) \]

\[ f(\mathbf{x}) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})) \]

\( \phi_j(\mathbf{x}) \) - an arbitrary function of \( \mathbf{x} \)
Modeling nonlinearities

Feature (basis) functions model nonlinearities

**Linear regression**
\[ f(x) = w_0 + \sum_{j=1}^{m} w_j \phi_j(x) \]

**Logistic regression**
\[ f(x) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(x)) \]

**Advantage:**
- The same problem as learning of the weights of linear units

**Limitations/problems:**
- How to define the right set of basis functions
- Many basis functions \( \rightarrow \) many weights to learn

---

Multi-layered neural networks

- An alternative way to model **nonlinearities for regression/classification problems**
- **Idea:** Cascade several simple nonlinear models (e.g., logistic units) to **approximate nonlinear functions** for regression/classification. Learn/adapt these simple models.
- **Motivation:** neuron connections
Multilayer neural network

Also called a multilayer perceptron (MLP)
Cascades multiple non-linear (e.g. logistic regression) units

Example: (2 layer) classifier with non-linear decision boundaries

<table>
<thead>
<tr>
<th>Input</th>
<th>Hidden layer</th>
<th>Output layer</th>
</tr>
</thead>
</table>

Multilayer neural network

- Models non-linearity through nonlinear switching units
- Can be applied to both regression and binary classification problems

regression
\[ f(x) = f(x, w) \]

classification
\[ f(x) = p(y = 1 | x, w) \]
Why we need nonlinearities? Why not multiple linear units

Cascading of multiple linear units is equivalent to one linear unit

\[ f(x) = b_{0,1} + b_{1,1}z_1 + b_{2,1}z_2 \]

\[ = b_{0,1} + b_{1,1}(a_{0,1} + a_{1,1}x_1 + a_{2,1}x_2) + b_{2,1}(a_{0,2} + a_{2,1}x_1 + a_{2,2}x_2) \]
**Why we need nonlinearities? Why not multiple linear units**

Cascading of multiple linear units is equivalent to one linear unit

\[
f(x) = b_{0,1} + b_{1,1}z_1 + b_{2,1}z_2 \\
= b_{0,1} + b_{1,1}(a_{0,1} + a_{1,1}x_1 + a_{2,1}x_2) + b_{2,1}(a_{0,2} + a_{2,1}x_1 + a_{2,2}x_2) \\
= b_{0,1} + b_{1,1}a_{0,1} + b_{1,1}a_{1,1}x_1 + b_{1,1}a_{2,1}x_2 + b_{2,1}a_{0,2} + b_{2,1}a_{2,1}x_1 + b_{2,1}a_{2,2}x_2 \\
= b_{0,1} + b_{1,1}a_{0,1} + b_{2,1}a_{0,2} + (b_{1,1}a_{1,1} + b_{2,1}a_{2,1})x_1 + (b_{1,1}a_{2,1} + b_{2,1}a_{2,2})x_2 \\
= c + d_1x_1 + d_2x_2.
\]

**Multilayer neural network**

- **Non-linearities are modeled using multiple hidden nonlinear units (organized in layers)**
- The output layer determines whether it is a **regression** or a **classification problem**

**Input layer** \( x_1, x_2, \ldots, x_d \) \n
**Hidden layers** \( a_{ij}, b_{ij} \)

**Output layer**

- **Classification option** \( \int f(x) = p(y=1|x, w) \)
- **Regression** \( f(x) = f(x, w) \)
Learning with MLP

- How to learn the parameters of the neural network?
- **Gradient descent algorithm**
  - Weight updates based on the error: \( J(D, w) \)
    \[
    w \leftarrow w - \alpha \nabla_w J(D, w)
    \]
- We need to *compute gradients for weights in all units*
- Can be computed in one backward sweep through the net !!!
- The process is called **back-propagation**

---

**Backpropagation: error function**

- **Error function**: \( J(D, w) \) (online) error where \( D \) is a data point
  - **Regression**
    \[
    J(D, w) = (y_u - f(x_u))^2
    \]
  - **Classification**
    \[
    J(D, w) = -\log p(y_u | f(x_u))
    \]

Input layer

Hidden layers

Nonlinearities

regression

\[ f(x) = f(x, w) \]

classification

\[ f(x) = p(y = 1 | x, w) \]

option

regression

\[ f(x) = f(x,w) \]

classification

\[ \int f(x) = p(y = 1 | x, w) \]
Backpropagation

(k-1)-th level  \[ x_j(k-1) \]
\[ w_{i,j}(k) \]
\[ \sum z_i(k) \]
\[ x_i(k) \]
\[ w_{i,j}(k+1) \]
\[ \sum z_i(k+1) \]
\[ x_i(k+1) \]

k-th level

\[ z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1) \]
\[ w_{i,j}(k) \] - weight between units j and i on levels (k-1) and k

(k+1)-th level

\[ z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1) \]

- Gradient descent:

\[ w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D, w) \]

\[ \frac{\partial}{\partial w_{i,j}(k)} J(D, w) = \frac{\partial J(D, w)}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1) \]

\[ \delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D, w) \]

\[ x_j(k-1) \]

Chain rule

\[ \frac{\partial f(g(u))}{\partial u} = \frac{\partial f(g(u))}{\partial g(u)} \frac{\partial g(u)}{\partial u} \]
Backpropagation

\( x_j(k-1) \)

\( w_{i,j}(k) \)

\( \sum \)

\( \bigcirc \)

\( \bigcirc \)

\( \bigcirc \)

\( x_i(k) = g(z_i(k)) \)

\( w_{i,j}(k+1) \)

\( \sum \)

\( z_i(k+1) \)

\( x_i(k+1) \)

\( z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k)x_j(k-1) \)

**Derivation:**

\( \delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D, w) = \frac{\partial}{\partial x_i(k)} J(D, w) \times \frac{\partial x_i(k)}{\partial z_i(k)} \)

\( \frac{\partial}{\partial x_i(k)} J(D, w) = \sum_j \frac{\partial}{\partial z_i(k+1)} J(D, w) \times \frac{\partial z_i(k+1)}{\partial x_i(k)} \times \frac{\partial x_i(k)}{\partial z_i(k)} \)

\( \delta_i(k+1) \)

\( w_{i,j}(k+1) \)
Backpropagation

(k-1)-th level

\[ x_j(k-1) \]

\[ w_{i,j}(k) \]

\[ z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k)x_j(k-1) \]

k-th level

\[ x_i(k) = g(z_i(k)) \]

\[ w_{i,j}(k+1) \]

\[ z_i(k+1) \]

(k+1)-th level

\[ x_i(k+1) \]

- Derivation:

\[ \delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D, w) = \frac{\partial}{\partial x_i(k)} J(D, w) \frac{\partial z_i(k+1)}{\partial x_i(k)} x_i(k)(1 - x_i(k)) \]

\[ \frac{\partial J(D, w)}{\partial x_i(k)} = \sum_l \frac{\partial z_l(k+1)}{\partial z_i(k+1)} \frac{\partial z_i(k+1)}{\partial x_i(k)} x_i(k)(1 - x_i(k)) \]
**Backpropagation**

(k-1)-th level | k-th level | (k+1)-th level
--- | --- | ---
$x_j(k-1)$ | $x_i(k) = g(z_i(k))$ | $x_i(k+1)$
$w_{i,j}(k)$ | $w_{i,j}(k+1)$ | $\sum_{l} x_l(k+1)$

$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k)x_j(k-1)$

- **Gradient:**

  $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \left[ \delta_i(k)x_j(k-1) \right]$

  $\delta_i(k) = \left[ \sum_l \delta_l(k+1) w_{i,l}(k+1) \right] x_i(k)(1-x_i(k))$

- **Last unit** (is the same as for the regular linear units),

  E.g. for regression:

  $\delta_i(K) = -(y_u - f(x_u,w))$

---

**Backpropagation**

**Update weight** $w_{i,j}(k)$ using data $D$  

$D = \{<x,y>\}$

$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D,w)$

Let $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,w)$

Then: 

$\frac{\partial}{\partial w_{i,j}(k)} J(D,w) = \frac{\partial J(D,w)}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k)x_j(k-1)$

S.t. $\delta_i(k)$ is computed from $x_i(k)$ and the next layer $\delta_i(k+1)$

$\delta_i(k) = \left[ \sum_l \delta_l(k+1) w_{i,l}(k+1) \right] x_i(k)(1-x_i(k))$

**Last unit** (is the same as for the regular linear units):

$\delta_i(K) = -(y_u - f(x_u,w))$

It is the same for the classification with the log-likelihood measure of fit and linear regression with least-squares error!!!
Learning with MLP

- **Online gradient descent algorithm**
  - Weight update:

  \[
  w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J_{\text{online}}(D_u, w)
  \]

  \[
  \frac{\partial}{\partial w_{i,j}(k)} J_{\text{online}}(D_u, w) = \frac{\partial J_{\text{online}}(D_u, w)}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)
  \]

  \[
  w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)
  \]

  \[x_j(k-1)\] - j-th output of the (k-1) layer

  \[\delta_i(k)\] - derivative computed via backpropagation

  \[\alpha\] - a learning rate

---

**Online gradient descent algorithm for MLP**

**Online-gradient-descent** \( (D, \text{number of iterations}) \)

**Initialize** all weights \( w_{i,j}(k) \)

**for** \( i = 1:1: \text{number of iterations} \)

**do**

- **select** a data point \( D_u = \langle x, y \rangle \) from \( D \)

- **set** learning rate \( \alpha \)

- **compute** outputs \( x_j(k) \) for each unit

- **compute** derivatives \( \delta_i(k) \) via backpropagation

- **update** all weights (in parallel)

  \[
  w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)
  \]

**end for**

**return** weights \( w \)
Xor Example.

- linear decision boundary does not exist
Xor example.
Neural network with 2 hidden units

Xor example.
Neural network with 10 hidden units
Neural networks

Activation (transfer) functions
• Determine how inputs are transformed to output

Possible choices of nonlinear transfer functions:
• Logistic function
\[ f(z) = \frac{1}{1 + e^{-z}} \quad f'(z) = f(z)(1 - f(z)) \]

• Hyperbolic tangent
\[ f(z) = \tanh(z) = \frac{2}{1 + e^{-2z}} \quad 1 \quad f'(z) = 1 - f(z)^2 \]

• Rectified linear function (Relu)
\[ f(z) = \begin{cases} 
0 & z < 0 \\
z & z \geq 0
\end{cases} \]

Limitation of standard NNs

Standard NN:
• do not scale well to high dimensional data (e.g. images)
  – 100x100 image + 100 hidden units = 1 million parameters.
  – Overfitting;
  – Tremendous requirements of computation and storage.
• Sensitive to small translation of inputs
  – Images: objects can have size, slant or position variations
  – Speech: varying speed, pitch or intonation.
• Ignores the topology of the input
  – i.e. the input variables can be presented in any order without affecting the outcome of training.
  – However, images or speech have a strong local structure
    • E.g. pixels nearby are highly correlated.
Deep learning

- **Deep learning**. Machine learning algorithms based on learning multiple levels of representation / abstraction. More than one layer of non-linear feature transformation.

![Deep learning diagram](image)

Deep neural networks

**Early efforts**

- **Optical character recognition** – digits 20x20
  - Automatic sorting of mails
  - 5 layer network with multiple output functions and somewhat restricted topology

<table>
<thead>
<tr>
<th>Layer</th>
<th>Neurons</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>3000</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>1200</td>
</tr>
<tr>
<td>3</td>
<td>1200</td>
<td>50000</td>
</tr>
<tr>
<td>2</td>
<td>784</td>
<td>3136</td>
</tr>
<tr>
<td>1</td>
<td>3136</td>
<td>78400</td>
</tr>
</tbody>
</table>

![Network diagram](image)
Convolutional NN

Take advantage of the local structure of the data (image, speech)

Convolution in Machine Learning
- the **input** array
  - e.g. image pixels.
- a **filter or kernel**
  - a smaller (local) matrix of parameters
- Output: a **feature map**
  - Filter applied to the image

Feature Extraction using Convolution

- The statistics of one part of the image are the same as any other part.
- Meaning that different parts of an image can share the same feature parameters (**kernel**).
- Use this kernel to **convolve** a set of features.
- This is called one feature mapping.
Feature Extraction using Convolution

4 features on full data (image)  4 features on the local data

- Fully connected layer
  - 9 weights per hidden unit
  - $9 \times 4 = 36$ weights
- Locally connected layer
  - 5 weights per hidden unit
  - $5 \times 4 = 20$ weights

Increased #input, #hidden unit, but fewer weights

---

Pooling (Subsampling, Down-sampling)

- **Assumption**: Features useful in one region are likely to be useful for other regions.
- To describe a large image, statistics can be **aggregated**.
- For example, one can calculate mean or max of a particular feature over a region.
  - Called **mean pooling**, **max pooling** respectively.
- These summary statistics are much lower in dimension.
- Also can improve results (less-overfitting).
**Convolution and Pooling**

**Convolution**

- Image:
  - 1 1 1 0 0
  - 0 1 1 1 0
  - 0 0 1 1 1
  - 0 0 1 1 0
  - 0 1 1 0 0

- Convolved Feature:
  - 4

**Pooling**

- Convolved feature
- Pooled feature

---

**Convolutional NN**

- CNN = \( \geq 1 \) convolution layer(s) + standard NN
- **One convolution layer is:**
  - Convolution operation + activation function + pooling
- You can view the convolution layer(s) as a feature extractor.
  - Input: raw image pixels, raw time series
  - Output: summarized features.
CNN vs. NN

- NN is sensitive to local distortions of unstructured data.
  - NN can theoretically be trained to be invariant to these distortions, probably resulting in multiple units with identical weights.
  - But such a training task requires a large number of training instances.
- CNN with pooling can be invariant to small translations:
  - Shifts (automatically)
  - Rotation (with extra mechanism)

Object Recognition Task

- ImageNet Data (2009 - 2016)
**ImageNet 2012**

**Data**
- **Size:**
  - Number of images
    - 1.2 million training images
    - 50K validation images
    - 150K testing images
  - Variable image size
- Supervised task
  - Labeled using Amazon’s Mechanical Turk
- Categories:
  - 1000 categories (objects)
    - Approximately 1000 in each category
  - RGB pictures

**Goal**
Provide a probability for different categories that an image can belong to

---

**Object Recognition**

- **ImageNet**
  - Achieves state-of-the-art on many object recognition tasks.