Supervised learning

Multilayer Perceptron and Deep Learning

Some slides are adopted from Honglak Lee, Geoffrey Hinton, Yann LeCun and Marc'Aurelio Ranzato
Threshold Logic Unit (TLU)

\[ a = \sum_{i=1}^{n} w_i u_i \]

\[ v = \begin{cases} 
1 & \text{if } a \geq \theta \\
0 & \text{if } a < \theta 
\end{cases} \]
Example: Logical AND and OR

\[ w_1 = 1 \]
\[ w_2 = 1 \]
\[ \theta = 1.5 \]

\[ w_1 = 1 \]
\[ w_2 = 1 \]
\[ \theta = 0.5 \]

<table>
<thead>
<tr>
<th>u_1</th>
<th>u_2</th>
<th>a</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>u_1</th>
<th>u_2</th>
<th>a</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Logical XOR

\[
\begin{array}{c|c|c|c}
\text{u}_1 & \text{u}_2 & a & v \\
\hline
0 & 0 & ? & 0 \\
0 & 1 & ? & 1 \\
1 & 0 & ? & 1 \\
1 & 1 & ? & 0 \\
\end{array}
\]

\[w_1=?\]
\[w_2=?\]
\[\theta=?\]
Logical XOR: Nonlinear Separability
Using Single Units

What does the blue unit ideally compute?

Some sort of AND operation! (You can choose the weights such that it will only fire when ALL input units are active.)

What kind of output space separation do we get from it?

A linear separation!

What does the second group of units produce at the green unit?

A second linear separation!
Using Multiple Units

What does the red unit ideally compute?
Well, an AND operation between the outputs from the blue and the green unit!

For example:

More units (3) for example can yield:
Multi Layer Perceptron

Feed-Forward Network (FFN)
You can get arbitrary separations by adding more hidden layers.

computes an “AND of ANDs“ (inside this convex area AND inside that convex area).
Multi Layer Perceptron

Feed-Forward Network (FFN)
Example: Logical XOR
Example: Logical XOR

\[
\begin{array}{cccc}
  u_1 & u_2 & a_1 & h_1 \\
  0 & 0 & ? & 0 \\
  1 & 0 & ? & 0 \\
  1 & 1 & ? & 0 \\
  0 & 1 & ? & 1 \\
\end{array}
\]
Example: Logical XOR

u₂

[u₁] 

0

1

0

1

0

1

-1 1

0.5

h₁ h₂

0.5

h₂

-1

1

u₁

u₂

h₁

h₂

u₁

u₂

a₁

h₁

0 0 0 0

1 0 -1 0

1 1 0 0

0 1 1 1

<table>
<thead>
<tr>
<th>u₁</th>
<th>u₂</th>
<th>a₁</th>
<th>h₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Logical XOR

```
<table>
<thead>
<tr>
<th>u1</th>
<th>u2</th>
<th>a1</th>
<th>h1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Example: Logical XOR

<table>
<thead>
<tr>
<th></th>
<th>u₁</th>
<th>u₂</th>
<th>a₁</th>
<th>h₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Logical XOR

\[ u_1, u_2 \Rightarrow h_1 / h_2 \]

- \( 0, 0 \Rightarrow 0 / 0 \)
- \( 0, 1 \Rightarrow 1 / 0 \)
- \( 1, 0 \Rightarrow 0 / 1 \)
- \( 1, 1 \Rightarrow 0 / 0 \)
Example: Logical XOR

\[
\begin{array}{c}
u_1, u_2 \rightarrow h_1 / h_2 \\
0, 0 \rightarrow 0 / 0 \\
0, 1 \rightarrow 1 / 0 \\
1, 0 \rightarrow 0 / 1 \\
1, 1 \rightarrow 0 / 0
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{h}_1 & \text{h}_2 & a & v \\
\hline
0 & 0 & ? & 0 \\
1 & 0 & ? & 1 \\
0 & 1 & ? & 1 \\
\hline
\end{array}
\]
Example: Logical XOR

\[ u_1, u_2 \Rightarrow h_1 / h_2 \]

- 0, 0 => 0 / 0
- 0, 1 => 1 / 0
- 1, 0 => 0 / 1
- 1, 1 => 0 / 0

<table>
<thead>
<tr>
<th>h_1</th>
<th>h_2</th>
<th>a</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Logical XOR (Approach 2)
Example: Logical XOR (Approach 2)

Here we transform from 2D to 3D!
Example: Logical XOR (Approach 2)

\[ u_1, u_2 \Rightarrow u_1 / h_1 / u_2 \]

- \(0, 0 \Rightarrow 0 / 0 / 0\)
- \(0, 1 \Rightarrow 0 / 0 / 1\)
- \(1, 0 \Rightarrow 1 / 0 / 0\)
- \(1, 1 \Rightarrow 1 / 1 / 1\)
Example: Logical XOR (Approach 2)

\[ u_1, u_2 \Rightarrow u_1 / h_1 / u_2 \]

- \( 0, 0 \Rightarrow 0 / 0 / 0 \)
- \( 0, 1 \Rightarrow 0 / 0 / 1 \)
- \( 1, 0 \Rightarrow 1 / 0 / 0 \)
- \( 1, 1 \Rightarrow 1 / 1 / 1 \)
Training Multilayer Perceptron

Procedure of backpropagation algorithm:

- Do
  - For each sample in the training set
    - Compute outputs of all neurons
    - Compute errors of output neurons
    - Compute (back-propagate) hidden layer errors
    - Update weights from hidden layer to output layer
    - Update weights from input layer to hidden layer

Until all examples are classified correctly or stopping criterion satisfied
Gradient Descent Rule for Sigmoid Activation Function

Gradient:
\[ \nabla E[w] = [\partial E/\partial w_0, \ldots, \partial E/\partial w_n] \]
\[ \Delta w = -\mu \nabla E[w] \]
\[ E_d[w] = 1/2 (t_d - v_d)^2 \]

\[ -1/\mu \Delta w_i = \partial E/\partial w_i \]
\[ = \partial/\partial w_i 1/2 (t_d - v_d)^2 \]
\[ = \partial/\partial w_i 1/2 (t_d - \sigma(\Sigma_i w_i u_i))^2 \]
\[ = (t_d - v_d) \sigma'(\Sigma_i w_i u_i) (-u_i) \]

\[ v_d = \sigma(a) = 1/(1+e^{-a}) \]
\[ \sigma'(a) = e^{-a}/(1+e^{-a})^2 = \sigma(a) (1-\sigma(a)) \]

We get:
\[ \Delta w_i = \mu v_d (1-v_d) (t_d - v_d) u_i \]
Logsig vs. Tansig transfer function

Logsig: Logarithmic sigmoid transfer function

Tansig: Symmetric sigmoid transfer function

Learning rule for logsig: \[ \Delta w_i = \mu v_d (1-v_d)(t_d-v_d) u_i \]

Learning rule for tansig: \[ \Delta w_i = \mu (1-v_d^2)(t_d-v_d) u_i \]
Forward Propagation

• Calculate outputs of all neurons

\[
\text{out}_A = \Omega \, W_{\Omega A} + \lambda \, W_{\lambda A} \\
\text{out}_B = \Omega \, W_{\Omega B} + \lambda \, W_{\lambda B}
\]

and so on …
Forward Propagation

- Calculate outputs of all neurons

\[
\text{out}_A = \Omega W_{\Omega A} + \lambda W_{\lambda A} \\
\text{out}_B = \Omega W_{\Omega B} + \lambda W_{\lambda B}
\]

and so on …
Backpropagation: Step 1

- Calculate errors of output neurons

\[ \delta_{\alpha} = \text{out}_\alpha (1 - \text{out}_\alpha) (\text{Target}_\alpha - \text{out}_\alpha) \]

\[ \delta_{\beta} = \text{out}_\beta (1 - \text{out}_\beta) (\text{Target}_\beta - \text{out}_\beta) \]
Backpropagation: Step 2

- Calculate (backpropagate) errors of hidden neurons

\[ \delta_A = \text{out}_A \ (1 - \text{out}_A) \ (\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta}) \]
\[ \delta_B = \text{out}_B \ (1 - \text{out}_B) \ (\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta}) \]
\[ \delta_C = \text{out}_C \ (1 - \text{out}_C) \ (\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta}) \]
Backpropagation: Step 3

- Change output layer weights

\[
\begin{align*}
W'_{A\alpha} &= W_{A\alpha} + \eta \delta_{\alpha} \text{out}_A \\
W'_{B\alpha} &= W_{B\alpha} + \eta \delta_{\alpha} \text{out}_B \\
W'_{C\alpha} &= W_{C\alpha} + \eta \delta_{\alpha} \text{out}_C \\
W'_{A\beta} &= W_{A\beta} + \eta \delta_{\beta} \text{out}_A \\
W'_{B\beta} &= W_{B\beta} + \eta \delta_{\beta} \text{out}_B \\
W'_{C\beta} &= W_{C\beta} + \eta \delta_{\beta} \text{out}_C
\end{align*}
\]
Backpropagation: Step 4

- Change hidden layer weights

\[ W^+_{\lambda A} = W_{\lambda A} + \eta \delta_A \, \text{in}_\lambda \]
\[ W^+_{\lambda B} = W_{\lambda B} + \eta \delta_B \, \text{in}_\lambda \]
\[ W^+_{\lambda C} = W_{\lambda C} + \eta \delta_C \, \text{in}_\lambda \]
\[ W^+_{\Omega A} = W_{\Omega A} + \eta \delta_A \, \text{in}_\Omega \]
\[ W^+_{\Omega B} = W_{\Omega B} + \eta \delta_B \, \text{in}_\Omega \]
\[ W^+_{\Omega C} = W_{\Omega C} + \eta \delta_C \, \text{in}_\Omega \]
Demo: Multilayer Perceptron
Regression vs. Classification

- **Regression**
  - Use *linear* transfer function at the output layer

- **Binary classification**
  - Use *logsig* transfer function at the output layer
Regression vs. Classification (cont.)

- Multi-class classification
  - Use **softmax** transfer function at the output layer

- Softmax function normalizes outputs between 0 and 1:

\[
y_i = \frac{e^{x_i}}{\sum_{i} e^{x_i}}
\]
How to Train Neural Networks?

• **Training set**
  – Is used to adjust weights of the neural network (e.g., 70%)

• **Validation set**
  – Is used to minimize overfitting (e.g., 15%)

• **Test set**
  – Is used only for testing the final solution (e.g., 15%)
Bias-variance tradeoff

- A model has "**high bias**" if it is too simple
  - Can not explain the data well enough, thus is **underfitting**
  - Reduces the impact caused by noise
Bias-variance tradeoff (cont.)

- A model has “**high variance**” if it is too complex
  - Instead of modeling the data, it is modeling noise, thus is **overfitting**
  - Will perform well on a training set but badly on a test set
Bias-variance tradeoff (cont.)

• We look for a **balance** between underfitting and overfitting

• We want to have minimal error on the test set
Demo: Training ANNs

Use validation stop to prevent overfitting!
Cross Validation

• **Monte Carlo cross validation**
  – Sub-sample data randomly into training and test sets (e.g., 70% and 30%)

• **K-fold cross validation**
  – Divide data into $k$ subsets
  – Each time (in total $k$ times) one of the subsets is used for testing and the rest $k-1$ subsets are joined and used as a training set

• **Leave-$p$-out cross validation**
  – Use $p$ observations as testing samples and the rest $(n-p)$ as training samples
  – Train and test $n!/(p!(n-p)!)$ times
Application: Autonomous Driving

- Autonomous Land Vehicle In a Neural Network (ALVINN; Pomerleau, 1989)
- Automated driving at 70 mph on a highway
Application: Time Series Prediction

- **Time series prediction**: predicting weather, climate, stocks and share prices, currency exchange rates, airline passengers, etc. (Weigend and Gershenfeld, 1994)
Application: Digit Recognition

http://neuralnetworksanddeeplearning.com
Supervised Learning

Deep Neural Networks
Motivation: Mammalian Visual Cortex

- The recognition pathway in the visual cortex has multiple stages
Learning Hierarchical Representations

• Natural progression from low level to high level structures

• Each module transforms its input representation into a higher-level one

• High-level features are more global and more invariant

• Low-level features are shared among categories

• A good lower level representation can be used for many distinct tasks
Generalisable Learning

Shared low level representations
Examples of Hierarchical Representations

- **Image recognition**
  - Pixel → edge → part → object

- **Text**
  - Character → word → word group → sentence → story

- **Speech**
  - Sample → spectral band → sound → phoneme → word
Learning Hierarchical Representations

• **Purely Supervised**
  - Train in supervised mode using backpropagation
  - Used in most practical systems for speech and image recognition

• **Unsupervised, layerwise + supervised classifier on top**
  - Train each layer unsupervised, one after the other
  - Train a supervised classifier on top, keeping the other layers fixed
  - Good when very few labeled samples are available
Deep Neural Networks and Backpropagation

- **Simple to construct**
  - Sigmoid nonlinearity for hidden layers
  - Softmax for the output layer

- **But, backpropagation does not work well (if randomly initialized)**
  - Deep networks trained with backpropagation (no pretraining) perform worse than shallow nets

Bengio et al., NIPS 2007
Problems with Backpropagation

- Gradient is progressively getting more dilute
  - Below top few layers, correction signal is minimal
  - Gets stuck in local minima
  - Especially if they start out far from ‘good’ regions (i.e., random initialization)
Deep Neural Networks

- Deep Believe Networks (DBNs)

- Convolutional Neural Networks (CNNs)
CNN Structure

- Stacking multiple stages of convolution and pooling/subsampling
CNN Structure (cont.)

LeCun at al., 1989
Fully Connected vs. Convolutional Net

- **Example: 200x200 image**
  - Fully-connected, 100 hidden units = $4 \times 10^6$ parameters
  - Locally-connected, 100 hidden units with 10x10 fields = $10^4$ parameters
  - Only few inputs per neuron: helps gradients to propagate through so many layers without diffusing so much
Convolution

- Convolution with a kernel (filter):

\[ A_{ij} = \sum_{kl} W_{kl} X_{i+k, j+l} \]

Note: All nodes share the same weight matrix
Multiple Convolutions with Different Kernels

- Detects multiple motifs at each location
- The result is a 3D array, where each slice is a feature map
Example: Convolution

Original image (200x200 px)

Filter 3x3

Filter 3x3

Filter 3x3

Filter 3x3
Pooling/Subsampling

- Small rectangular blocks from the convolutional layer are subsampled to produce single outputs from those blocks.
• Subsampling with 2x2 filter and stride 2

Max pooling

Averaging pooling
Pooling/Subsampling (cont.)

- Subsampling with 4x4 filter and stride 4
Example of Convolutional Net

Multistage Hubel-Wiesel system (LeCun et al., 89; LeCun et al., 98)
Activation functions

• Rectified Linear Unit (ReLU):
  \[ f(x) = \begin{cases} 
  0 & \text{for } x < 0 \\
  x & \text{for } x \geq 0 
  \end{cases} \]

  Most commonly used

• SoftPlus:
  \[ f(x) = \log_e(1 + e^x) \]

• Parametric Rectified Linear Unit (PReLU):
  \[ f(x) = \begin{cases} 
  \alpha x & \text{for } x < 0 \\
  x & \text{for } x \geq 0 
  \end{cases} \]

• Exponential Linear Unit (ELU):
  \[ f(x) = \begin{cases} 
  \alpha(e^x - 1) & \text{for } x < 0 \\
  x & \text{for } x \geq 0 
  \end{cases} \]
Feature visualization of convolutional net trained on ImageNet (Zeiler & Fergus 2013)
### Comparison of Classifiers on MNIST Dataset

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knowledge-free methods</strong></td>
<td></td>
</tr>
<tr>
<td>2-layer NN, 800 HU, CE</td>
<td>1.60</td>
</tr>
<tr>
<td>3-layer NN, 500+300 HU, CE, reg</td>
<td>1.53</td>
</tr>
<tr>
<td>SVM, Gaussian Kernel</td>
<td>1.40</td>
</tr>
<tr>
<td>Unsupervised Stacked RBM + backprop</td>
<td>0.95</td>
</tr>
<tr>
<td><strong>Convolutional nets</strong></td>
<td></td>
</tr>
<tr>
<td>Convolutional net LeNet-5,</td>
<td>0.80</td>
</tr>
<tr>
<td>Convolutional net LeNet-6,</td>
<td>0.70</td>
</tr>
<tr>
<td>Conv. net LeNet-6- + unsup learning</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Yann LeCun
Comparison of Classifiers on Object Recognition

- Linear Classifier on raw stereo images: $30.2\%$ error
- K-Nearest-Neighbors on raw stereo images: $18.4\%$ error
- K-Nearest-Neighbors on PCA-95: $16.6\%$ error
- Pairwise SVM on 96x96 stereo images: $11.6\%$ error
- Pairwise SVM on 95 Principal Components: $13.3\%$ error
- Convolutional Net on 96x96 stereo images: $5.8\%$ error

Training instances

Test instances

Yann LeCun
DNN Tools

• Tensorflow with Keras API
  https://www.tensorflow.org/

• Pytorch
  https://www.pytorch.org/
Summary

• MLP can classify between more complex patterns

• Backpropagation for MLP learning
  – Easy to implement
  – Becomes slow when using many hidden layers due to diluting gradient

• Training ANNs
  – Training set
  – Validation set
  – Test set
Summary (cont.)

• **Deep neural networks**
  - Hierarchical learning

  - **Deep Belief Networks**
    - Unsupervised layerwise learning
    - Supervised learning for final tuning of weights and classification

- **Convolutional Neural Networks**
  - Can be fully trained with backpropagation due to reduced number of parameters