Multi-Layer Perceptron (MLP)

Introduction

M. R. Hasan

Deep Learning
Readings

• Bishop: 5.1, 5.3, 5.5
• Murphy: 16.5, 16.5.4
• Alpaydin: 11
• Geron: 10
What We Will Cover

• Linear Neural Network: Linear Decision Boundary
• Nonlinear Neural Network: Multi-layer Perceptron (MLP)
• MLP: Feedforward Neural Network
• MLP: Fully-connected or Dense Network
• MLP: Nonlinear network function
• MLP: Universal Approximator
Machine Learning

- Machine Learning is an **induction based** learning paradigm.
- Different techniques of Machine Learning…
Machine Learning: Domingos’ Categorization

- **Five “tribes”** or approaches in Machine Learning.

<table>
<thead>
<tr>
<th>Tribe</th>
<th>Strength</th>
<th>Tools</th>
<th>Inspiration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbolists</td>
<td>Formal Systems</td>
<td>Decision Tree, Inductive Logic</td>
<td>Logic, Philosophy</td>
</tr>
<tr>
<td>Connectionists</td>
<td>Neural Networks</td>
<td>Backpropagation, Reinforcement Learning</td>
<td>Neuroscience</td>
</tr>
<tr>
<td>Bayesians</td>
<td>Probabilistic Inference</td>
<td>Hidden Markov Model (HMM), Graphical Models, NB Classifier</td>
<td>Statistics</td>
</tr>
<tr>
<td>Evolutionaries</td>
<td>Structure Learning</td>
<td>Genetic Algorithms, Evolutionary Programming</td>
<td>Biology</td>
</tr>
<tr>
<td>Analogizers</td>
<td>Mapping to Novelty</td>
<td>K-NN, Support Vector Machine (SVM)</td>
<td>Psychology</td>
</tr>
</tbody>
</table>
Neural Connection-based Learning
• Binary: cast logistic regression as a LNN
• Multi-class: cast softmax regression as a LNN
Linear Neural Network

• See the following Jupyter notebooks for the Keras (with TensorFlow 2.0 backend) based implementation of the two LNN models (Binary Logistic Regression and Softmax Regression):
• [https://github.com/rhasanbd/Linear-Neural-Networks](https://github.com/rhasanbd/Linear-Neural-Networks)
Linear Neural Network: Limitation
• Using LNN, we can learn only **linear functions** (linear decision boundary).
Classification: LNN

- Can we learn a **nonlinear** function using a LNN?
- For example, the XOR function?
- **No!**
Classification: LNN

- Can we use LNNs to build a **nonlinear** Neural Network?
- Yes!
- We can use **two LNNs** to learn **linear** decision boundaries for partitioning two subspaces of the data distribution.
• Then, we pass the output of the two LNNs to another LNN.
• The final LNN combines two linear boundaries to create a nonlinear decision boundary.
• Thus, by **stacking multiple LNNs** we can create a nonlinear neural network!

It is known as the Multi-Layer Perceptron (MLP)

In MLPs we have intermediate layers (aka **hidden layers**) of computational units or neurons.
Deep Neural Network (DNN)

• We generalize the LNN architecture into Deep Neural Networks (DNNs) by incorporating hidden layers with nonlinear computational units or neurons.
• The simplest DNN architecture is multi-layer perceptron (MLP).

LNN: Softmax Regression

DNN: Multi-Layer Perceptron
Deep Neural Network (DNN)

• We will discuss the **multi-layer perceptron (MLP)** model.
• Based on MLP, two complex DNN models can be built.
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
Multi-Layer Perceptron (MLP)
Two Facts about MLP

The MLP is a **Feedforward** Neural Network model

Signals propagate only from input to output.

The MLP is a **Fully-Connected (FC) or Dense** Neural Network

In a FC/Dense network, each neuron in a layer is connected to **all neurons** in the previous layer.
Multi-Layer Perceptron

• Consider the basic **feedforward & fully-connected** MLP model.

• It can be described as a series of **functional transformations**.

It’s a *3-layer* MLP.

**Layer 1**: input

Each input has **d features** + the bias (= 1)

**Layer 2**: Hidden with **m neurons** and a bias neuron

**Layer 3**: Output
Multi-Layer Perceptron

- Observe that there are \( m \) neurons in the hidden layer.
- So, we construct \( m \) linear combinations \((j = 1, \ldots, m)\) of the input variables \(x_0, x_1, \ldots, x_d\) in the form:

\[
Z_j^{(2)} = \sum_{i=0}^{d} W_{ji}^{(1)} x_i
\]

Superscript of \( W \) indicates the input layer it is associated with
Multi-Layer Perceptron

- Here $z_j^{(2)}$ are the **activation neurons** of layer 2.
- Each of them is then transformed using a **differentiable, nonlinear activation function** $g(\cdot)$ given by:

$$a_j^{(2)} = g(z_j^{(2)})$$

$$z_j^{(2)} = \sum_{i=0}^{d} W_{ji}^{(1)} x_i$$
Multi-Layer Perceptron

- The activation signals $a_j^{(2)}$ of the hidden layer are again **linearly combined** to give output unit activations:

$$z_j^{(3)} = \sum_{i=0}^{m} W_{ji}^{(2)} a_j^{(2)}$$

**Subscript of $z_j^{(3)}$:** There are **K output neurons**: $j = 1, 2, .., K$

$$z_j^{(2)} = \sum_{i=0}^{d} W_{ji}^{(1)} x_i$$

$$a_j^{(2)} = g(z_j^{(2)})$$
Finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs $y_k$.

$$y_k = g(z_j^{(3)})$$

The choice of $g(.)$ could vary between the output and the hidden layers.

$$z_j^{(3)} = \sum_{i=0}^{m} W_{ji}^{(2)} a_j^{(2)}$$

$$z_j^{(2)} = \sum_{i=0}^{d} W_{ji}^{(1)} x_i$$

$$a_j^{(2)} = g(z_j^{(2)})$$
Multi-Layer Perceptron

- We combine these various stages to give the overall network function that takes the following form:

\[
y_k(\vec{x}, \vec{w}) = g \left( \sum_{i=0}^{m} W_{ji}^{(2)} g \left( \sum_{i=0}^{d} W_{ji}^{(1)} x_i \right) \right)
\]

\[
y_k = g(z_j^{(3)})
\]

\[
z_j^{(3)} = \sum_{i=0}^{m} W_{ji}^{(2)} a_j^{(2)}
\]

\[
a_j^{(2)} = g(z_j^{(2)})
\]

\[
z_j^{(2)} = \sum_{i=0}^{d} W_{ji}^{(1)} x_i
\]
Multi-Layer Perceptron

• What kind of function is this?

• The output function is a **nonlinear** combination of **input units** (neurons).

\[ y_k(\vec{x}, \vec{w}) = g \left( \sum_{i=0}^{m} W_{ji}^{(2)} g \left( \sum_{i=0}^{d} W_{ji}^{(1)} x_i \right) \right) \]

Thus, the output function is **nonlinear in** \( W \) (**&** \( x \))
Multi-Layer Perceptron

- Unlike Linear & Logistic regression (i.e., the LNNs), the MLP output function is **nonlinear in W (& x)**.

\[
y_k(\vec{x}, \vec{w}) = g \left( \sum_{i=0}^{m} W_{ji}^{(2)} g \left( \sum_{i=0}^{d} W_{ji}^{(1)} x_i \right) \right)
\]

**Output is a linear combination of the features (i.e., linear in w & x).**

\[
y(x) = \sum_{i=0}^{d} w_i x_i = \vec{w}^T \vec{x}
\]

- **Linear Regression**
- **Logistic Regression**

\[
y(\vec{x}) = \sigma(\vec{w}^T \vec{x}) = \frac{1}{1 + e^{-\vec{w}^T \vec{x}}}
\]
Multi-Layer Perceptron

- Thus the neural network model is simply a **nonlinear function**.
- It’s a function from a set of input variables \( \{x_i\} \) to a set of output variables \( \{y_k\} \) **controlled by a matrix** \( W \) **of adjustable parameters**.

\[
y_k(x, \bar{w}) = g \left( \sum_{i=0}^{m} W_{ji}^{(2)} g \left( \sum_{i=0}^{d} W_{ji}^{(1)} x_i \right) \right)
\]

\[
z_j^{(3)} = \sum_{i=0}^{m} W_{ji}^{(2)} a_j^{(2)}
\]

\[
y_k = g(z_j^{(3)})
\]

\[
z_j^{(2)} = \sum_{i=0}^{d} W_{ji}^{(1)} x_i
\]

\[
a_j^{(2)} = g(z_j^{(2)})
\]
Multi-Layer Perceptron

• How does the nonlinear MLP model benefit us?

• It enables the NN to learn any *arbitrary functions*.

• We can show that a MLP model is a *universal function approximator*.

\[
y_k(x, w) = g \left( \sum_{i=0}^{m} W_{ji}^{(2)} g \left( \sum_{i=0}^{d} W_{ji}^{(1)} x_i \right) \right)
\]
MLP: Universal Function Approximator
Multi-Layer Perceptron

- MLPs can capture **complex interactions** among the inputs via their hidden neurons.
- In contrast, in a polynomial basis function (Linear & Logistic Regression), the type of feature interactions were limited.
- In a MLP, we can easily **design hidden neurons (in different layers)** to perform arbitrary computations.
Multi-Layer Perceptron

• Using multiple neurons, we can learn any **nonlinear function**.
• Example: learn a function to identify the yellow area (the polygon).
• We can **build multiple neurons each** with a single output that fires if the input is in the colored area.
• **Build a neuron** with a single output that fires if the input is in the colored area.

Stacking these neurons we can build a MLP to learn any arbitrary function!
Multi-Layer Perceptron

- An MLP can learn more complex decision boundaries.
Multi-Layer Perceptron

- An MLP with many neurons can model any arbitrary functions over an input to an arbitrary precision.
- Simply make the individual pulses narrower.
MLP: Optimal Architecture for the Universal Function Approximator
Multi-Layer Perceptron

- To model any function we only need to create a **single-hidden-layer** MLP.
- But a single-hidden-layer network may require an **exponentially large** number of neurons.
- Thus, it’s not an optimal architecture.
Multi-Layer Perceptron

• On the other hand, the same function can be expressed by using far fewer neurons organized in a deep network.
• The number of neurons could be exponentially smaller in deeper networks.

This observation motivates the design of deep MLPs.
Multi-Layer Perceptron

- Now we know the rationale for creating multiple layer-based neural networks (e.g., a MLP).
- Next question is: how do we **learn the weights** (parameters) of the neurons such that a MLP can model a function?
- In other words how do we **train** a MLP?

The Backpropagation Algorithm!
Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

*Nature* 323, 533–536 (09 October 1986)  |  Download Citation ▼