Multi-layer neural networks

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Linear units

**Linear regression**

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

On-line gradient update:

\[
 w_0 \leftarrow w_0 + \alpha (y - f(\mathbf{x})) \\
 w_j \leftarrow w_j + \alpha (y - f(\mathbf{x})) x_j
\]

**Logistic regression**

\[
 f(\mathbf{x}) = p(y = 1 | \mathbf{x}, \mathbf{w}) = g(w_0 + \sum_{j=1}^{d} w_j x_j)
\]

On-line gradient update:

\[
 w_0 \leftarrow w_0 + \alpha (y - f(\mathbf{x})) \\
 w_j \leftarrow w_j + \alpha (y - f(\mathbf{x})) x_j
\]

The same
Limitations of basic linear units

Linear regression
\[ f(x) = w_0 + \sum_{j=1}^{d} w_j x_j \]

Logistic regression
\[ f(x) = p(y = 1 | x, w) = g(w_0 + \sum_{j=1}^{d} w_j x_j) \]

Function linear in inputs !!
Linear decision boundary!!

Regression with the linear model.

Limitation: linear hyper-plane only
Regression with the linear model.

**Limitation:** linear hyper-plane only
a non-linear surface can be better

Classification with the linear model.

**Logistic regression model defines a linear decision boundary**
- Example: 2 classes (blue and red points)
Linear decision boundary

- logistic regression model is not optimal, but not that bad

When logistic regression fails?

- Example in which the logistic regression model fails
Limitations of linear units.

- Logistic regression does not work for **parity functions**
  - no linear decision boundary exists

**Solution:** a model of a non-linear decision boundary

---

Extensions of simple linear units

- use **feature (basis) functions** to model **nonlinearities**

**Linear regression**

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

**Logistic regression**

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

\( \phi_j(x) \) - an arbitrary function of \( x \)
Example. Regression with polynomials.

Regression with polynomials of degree $m$
- **Data points**: pairs of $<x, y>$
- **Feature functions**: $m$ feature functions
  \[ \phi_i(x) = x^i \quad i = 1, 2, \ldots, m \]
- **Function to learn**:
  \[ f(x, w) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i \]

Learning with extended linear units

**Feature (basis) functions** model **nonlinearities**

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

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

**Important property**:
- Learning of weights problem is the same as it was for the linear units
- **Trick**: we have changed the inputs – the weights are still linear in the new input
Learning with feature functions.

Function to learn:
\[ f(x, w) = w_0 + \sum_{i=1}^{k} w_i \phi_i(x) \]

On line gradient update for the \(<x,y>\) pair
\[ w_0 = w_0 + \alpha (y - f(x, w)) \]
\[ \vdots \]
\[ w_j = w_j + \alpha (y - f(x, w)) \phi_j(x) \]

Gradient updates are of the same form as in the linear and logistic regression models.

Example. Regression with polynomials.

Example: Regression with polynomials of degree \(m\)
\[ f(x, w) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i \]

- On line update for \(<x,y>\) pair
\[ w_0 = w_0 + \alpha (y - f(x, w)) \]
\[ \vdots \]
\[ w_j = w_j + \alpha (y - f(x, w)) x^i \]
Multi-layered neural networks

- Alternative way to introduce nonlinearities to regression/classification models
- **Idea:** Cascade several simple neural models with logistic units. Much like neuron connections.

![Diagram of a neuron](image)

Multilayer neural network

Also called a **multilayer perceptron (MLP)**
Cascades multiple logistic regression units

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

![Diagram of a neural network](image)
Multilayer neural network

- Models non-linearities through logistic regression units
- Can be applied to both regression and binary classification problems

\[
1 \rightarrow w_{01}(1) \sum z_i(1) \rightarrow w_{02}(2) \rightarrow z(2) \\

\]

\[
1 \rightarrow w_{11}(1) \rightarrow \ldots \rightarrow w_{12}(2) \rightarrow z(2) \\

\]

\[
1 \rightarrow w_{21}(1) \rightarrow \ldots \rightarrow w_{22}(2) \rightarrow z(2) \\

\]

Input layer \( \rightarrow \) Hidden layer \( \rightarrow \) Output layer

- **regression** \( f(x) = f(x, w) \)
- **classification** \( \int f(x) = p(y = 1 | x, w) \)

- **option**
Learning with MLP

- How to learn the parameters of the neural network?
- **Online gradient descent algorithm**
  - Weight updates based on $J_{\text{online}} \left( D_i, w \right)$
  
  $w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J_{\text{online}} \left( D_i, w \right)$

- 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

### (k-1)-th level

- $x_{i}(k-1)$
- $w_{i,j}(k-1)$
- $x_{j}(k)$
- $z_{i}(k)$

### k-th level

- $\sum z_{j}(k)$

### (k+1)-th level

- $x_{i}(k)$
- $w_{i,j}(k+1)$
- $z_{i}(k+1)$
- $x_{j}(k+1)$

- $x_{i}(k)$ - output of the unit $i$ on level $k$
- $z_{i}(k)$ - input to the sigmoid function on level $k$
- $w_{i,j}(k)$ - weight between units $j$ and $i$ on levels (k-1) and k

$$z_{i}(k) = w_{i,o}(k) + \sum_{j} w_{i,j}(k)x_{j}(k-1)$$

$$x_{i}(k) = g(z_{i}(k))$$
Backpropagation

Update weight \( w_{i,j}(k) \) using a data point \( D_u = \langle x, y \rangle \):

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

Let \( \delta_i(k) = \frac{\partial}{\partial z_i(k)} J_{\text{online}}(D_u, w) \)

Then:

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

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

\[
\delta_i(k) = \left[ \sum_j \delta_{i+1}(k+1) w_{i,j+1} \right] x_i(k)(1 - x_i(k))
\]

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

\[
\delta_i(K) = -(y - f(x, w))
\]

It is the same for the classification with the log-likelihood measure of fit and linear regression with least-squares error!!!
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 \(D_u=<x,y>\) from \(D\)

set \(\alpha=1/i\)

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

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

update all weights (in parallel)

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

done

return weights \(w\)

---

Xor Example.

- No linear decision boundary
Xor example. Linear unit

Xor example. Neural network with 2 hidden units
Xor example.
Neural network with 10 hidden units

MLP in practice

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