Multi-Layer Perceptron (MLP)
Training: Backpropagation Algorithm I

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

- Training MLP
- Backprop algorithm
- Forward propagation
- Backward propagation
- Activation function: hidden layer
- Activation function: final layer
Training MLP
Multi-Layer Perceptron

- Consider a MLP, which is a dense feedforward network.
- The process of evaluating its output $y_k$ can be interpreted as a forward propagation of information through the network.

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

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

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

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

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

• We will discuss how to **train** a MLP for learning its **parameters** (weights of the neurons).
• Assume a **fixed architecture for the MLP** (k-layer)
• There are N training samples: \{ (x_1, y_1), \ldots, (x_N, y_N) \}
• The general **objective/loss function** is defined as (for batch GD):

\[
L(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} [\text{Loss}(y_i, \hat{y}(\vec{w}))] + \text{Regularizer}(\vec{w})
\]
Multi-Layer Perceptron

- Recall the **linear regression loss function** ($l_2$ regularization):

\[
L(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (y_i - \vec{w}^T \vec{x}_i)^2 + \frac{\lambda}{2} \|\vec{w}\|^2
\]

**Logistic regression** (classification) loss function ($l_2$ regularization):

\[
L(\vec{w}) = -\frac{1}{N} \sum_{i=1}^{N} \left[ y_i \log \sigma(\vec{w}^T \vec{x}_i) + (1 - y_i) \log (1 - \sigma(\vec{w}^T \vec{x}_i)) \right] + \frac{\lambda}{2} \|\vec{w}\|^2
\]

\[
L(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} \left[ \text{Loss}(y_i, \hat{y}(\vec{w})) + \text{Regularizer}(\vec{w}) \right]
\]
Multi-Layer Perceptron

- Both in **regression and classification** our goal was to learn weight parameter $w$.

- Because of the **convexity** of the **loss function**, we were able to find the **globally optimum** solution.

$$L(\overline{w}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (y_i - \overline{w}^T \hat{x}_i)^2 + \frac{\lambda}{2} \|\overline{w}\|_2^2$$

$$L(\overline{w}) = -\frac{1}{N} \sum_{i=1}^{N} \left[ y_i \log \sigma(\overline{w}^T \hat{x}_i) + (1 - y_i) \log (1 - \sigma(\overline{w}^T \hat{x}_i)) \right] + \frac{\lambda}{2} \|\overline{w}\|_2^2$$
Multi-Layer Perceptron

• However, in a MLP the output function is a **nonlinear combination of input units** (neurons).

• As a consequence, the MLP loss function is **non-convex**.

\[
L(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (y_i - \vec{w}^T \vec{x}_i)^2 + \frac{\lambda}{N} \|\vec{w}\|^2
\]

\[
L(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} [y_i \log_a(\vec{w}^T \vec{x}_i) + (1 - y_i) \log_2(1 - \sigma(\vec{w}^T \vec{x}_i))] + \frac{\lambda}{2N} \|\vec{w}\|^2
\]
Multi-Layer Perceptron

• Because of the non-convexity of the MLP loss function, we are unable to find any closed form solution.
• Hence, we use gradient descent to find an optimal solution.

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

\[
L(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} \left[ \text{Loss}(y_i, \hat{y}(\vec{w})) \right] + \text{Regularizer}(\vec{w})
\]
Multi-Layer Perceptron

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

- Another consequence of the non-convex loss function is that multiple local minima may exist.
Multi-Layer Perceptron

Finding the **global minimum is infeasible!**
Multi-Layer Perceptron

• How do we **implement** the gradient descent algorithm for a MLP?

• Our goal is to find an **efficient technique** for **evaluating the gradient of a loss function** $L(w)$ for a feedforward neural network.

$$\vec{w}(t+1) = \vec{w}(t) - \eta \frac{\partial L}{\partial \vec{w}}$$
Multi-Layer Perceptron

- This can be achieved using a **local message passing scheme** in which information is sent **alternately forwards and backwards** through the network.
- It is known as **error backpropagation** algorithm, or sometimes simply as **backprop**.

---

**Learning representations by back-propagating errors**

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

*Nature 323*, 533–536 (09 October 1986)  |  Download Citation ↓
Learning representations by back-propagating errors

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

*Nature* **323**, 533–536 (09 October 1986) | Download Citation
Learning representations by backpropagating errors

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

*Nature* **323**, 533–536 (09 October 1986) | Download Citation ↓
Multi-Layer Perceptron

• To present the backprop algorithm, first we will use a simple case.
• Consider a MLP that has just 1 hidden layer.
• In each layer we have only one neuron (excluding bias).
• Note that the neurons represent the features (either sample feature or extracted features in the hidden layer).
Multi-Layer Perceptron

- For simplicity, we will consider a regression problem and ignore regularization.
Multi-Layer Perceptron

- The loss function parameters are (for a 3-layer MLP):
- \( L(w) = L(w_1, w_2) \) where \( w_2 \) is weight of the connection from the hidden layer to the **final layer**.
- In general, for a \( k \)-layer MLP we have:
- \( L(w) = L(w_1, \ldots, w_{(k-1)}) \)

\[
\overrightarrow{w}(t+1) = \overrightarrow{w}(t) - \eta \frac{\partial L}{\partial \overrightarrow{w}}
\]
Multi-Layer Perceptron

- Our goal is to **learn the gradient of loss function** w.r.t. weights from all layers.

\[
\frac{\partial L}{\partial \mathbf{w}_{(k-1)}}, \left\{ \frac{\partial L}{\partial \mathbf{w}_k} \right\}_{k=1}^{(k-2)}
\]

\[
\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \frac{\partial L}{\partial \mathbf{w}}
\]

The **last layer (k-1) weight** is computed *slightly differently* (depending on the implementation).
Multi-Layer Perceptron

• Now let’s work on this **single hidden layer MLP**.

• The **input** variable “x” is denoted by “a₁” (for consistency).
Multi-Layer Perceptron

- We make some changes to **simplify** the calculation.
- **Bias features are consumed** in input vectors, so we don’t show them separately.
- Similarly the bias weights are also consumed in $w$. 

\[
\hat{x} = \{x_0, x_1\} := \hat{a}_1
\]

\[
\hat{z}_2 := \text{Bias} + \hat{z}_2
\]

\[
\hat{w}_1 := \text{Bias} + \hat{w}_1
\]

\[
\hat{w}_2 := \text{Bias} + \hat{w}_2
\]
Forward Propagation

(a) Forward propagation
Multi-Layer Perceptron

- Let’s understand the **forward propagation phase**.
- Input $a_1$ (or $x$) produces the **weighted signal input** for the hidden layer neuron $z_2$.
- The neuron $z_2$ is the **activation neuron**.

$$
\begin{align*}
\tilde{z}_2 &= \bar{w}_1^T \tilde{a}_1 \\
\tilde{a}_2 &= g(\tilde{z}_2) \\
\tilde{z}_3 &= \bar{w}_2^T \tilde{a}_2 \\
\tilde{a}_3 &= g(\tilde{z}_3)
\end{align*}
$$
Multi-Layer Perceptron

• The input to $z_2$ is transformed by a nonlinear activation function $g(\cdot)$ to give the activation to $z_2$.
• Then, the activated signal $a_2$ from the hidden layer produces input for the final layer neuron $z_3$.

\[ \hat{z}_2 = \Hat{\mathbf{w}}_1^T \hat{a}_1 \]
\[ \hat{a}_2 = g(\hat{z}_2) \]
\[ \hat{z}_3 = \Hat{\mathbf{w}}_2^T \hat{a}_2 \]
\[ \hat{a}_3 = g(\hat{z}_3) \]
Multi-Layer Perceptron

• The final layer input is **transformed by a nonlinear activation function** $g(\cdot)$ to give the activation to $z_3$.
• Finally, the activated signal $a_3$ gives the predicted output.
• For simple **regression** problem, $a_3 = z_3$ (g is an **identity function**).
Backward Propagation

(b) Backward propagation
Multi-Layer Perceptron

• During the backward propagation, our goal is to update the weights $w_2$ and $w_1$ by using the Gradient Descent algorithm.

• For this, we need to propagate the prediction error/loss backward for computing the loss gradients.

$$ w_2^{(t+1)} = w_2^{(t)} - \eta \frac{\partial L}{\partial w_2} $$

$$ w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial L}{\partial w_1} $$

\[ \hat{z}_2 = w_1^T \hat{a}_1 \]
\[ \hat{a}_2 = g(\hat{z}_2) \]
\[ \hat{z}_3 = w_2^T \hat{a}_2 \]
\[ \hat{a}_3 = g(\hat{z}_3) \]
Multi-Layer Perceptron

- In other words, for updating weights, we want to **find the variation in the loss caused by** $w_2$ **and** $w_1$.
- Then, this **variation is used** in the update rule.

\[ \vec{w}_2^{(t+1)} = \vec{w}_2^{(t)} - \eta \frac{\partial L}{\partial \vec{w}_2} \]
\[ \vec{w}_1^{(t+1)} = \vec{w}_1^{(t)} - \eta \frac{\partial L}{\partial \vec{w}_1} \]

- $x = a_1$
- $z_2 \sim a_2$
- $z_3 \sim a_3$
- Layer: $k = 1$
- Layer: $k = 2$
- Layer: $k = 3$

- $\hat{z}_2 = \vec{w}_1^T \hat{a}_1$
- $\hat{a}_2 = g(\hat{z}_2)$
- $\hat{z}_3 = \vec{w}_2^T \hat{a}_2$
- $\hat{a}_3 = g(\hat{z}_3)$

- $\eta$ is the learning rate.
Multi-Layer Perceptron

- First, we will calculate the variation in the **loss** caused by the **last layer weight** \( w_2 \): \( \frac{\partial L}{\partial w_2} \)
- Observe that the variation in \( L \) that is caused by \( w_2 \) is **propagated via the activation neuron** \( z_3 \).

In other words, \( w_2 \) influences \( z_3 \), and consequently \( z_3 \) influences \( L \).
Multi-Layer Perceptron

- To capture this variation of $w_2$ via $z_3$, we use the chain rule of partial derivative.

$$\frac{\partial L}{\partial \vec{w}_2} = \frac{\partial L}{\partial \vec{z}_3} \cdot \frac{\partial \vec{z}_3}{\partial \vec{w}_2}$$

It is the product of two variations: variation in $L$ caused by final layer input $z_3$ and the variation in $z_3$ due to $w_2$
Multi-Layer Perceptron

- The variation in $L$ caused by $z_3$ is denoted as the error ($\delta$) of the final layer neuron.

$$\frac{\partial L}{\partial \vec{w}_2} = \frac{\partial L}{\partial \hat{z}_3} \cdot \frac{\partial \hat{z}_3}{\partial \vec{w}_2}$$

$$\frac{\partial L}{\partial \hat{z}_3} \equiv \delta_3$$

The $2^{nd}$ term:

$$\frac{\partial \hat{z}_3}{\partial \vec{w}_2} = \hat{a}_2$$

- $\hat{z}_2 = \vec{w}_1^T \hat{a}_1$
- $\hat{a}_2 = g(\hat{z}_2)$
- $\hat{z}_3 = \vec{w}_2^T \hat{a}_2$
- $\hat{a}_3 = g(\hat{z}_3)$

It’s the error of the final layer caused by incorrect input $z_3$. 
Multi-Layer Perceptron

• We express $\frac{\partial L}{\partial w_2}$ as follows:

$$
\frac{\partial L}{\partial \vec{w}_2} = \frac{\partial L}{\partial \vec{z}_3} \cdot \frac{\partial \vec{z}_3}{\partial \vec{w}_2}$$

$$
\frac{\partial L}{\partial \vec{z}_3} \equiv \delta_3
$$

$$
\frac{\partial \vec{z}_3}{\partial \vec{w}_2} = \vec{a}_2
$$

We need to calculate the error term $\delta_3$.

$$
\hat{z}_2 = \overrightarrow{w}_1^T \hat{a}_1
$$

$$
\hat{a}_2 = g(\hat{z}_2)
$$

$$
\hat{z}_3 = \overrightarrow{w}_2^T \hat{a}_2
$$

$$
\hat{a}_3 = g(\hat{z}_3)
$$
Multi-Layer Perceptron

- Again observe the error $\delta_3$ at the final layer neuron is caused by $z_3$ via the activation function $a_3$.
- We capture this cascaded influence by the chain rule.

$$\frac{\partial L}{\partial \vec{w}_2} = \delta_3 \cdot \hat{a}_2$$

$$\delta_3 = \frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial \hat{a}_3} \cdot \frac{\partial \hat{a}_3}{\partial z_3}$$

Using the squared error loss function

$$L(\vec{w}_2) = \frac{1}{2} (y - \hat{a}_3)^2$$

$$\frac{\partial L}{\partial \hat{a}_3} = 2 \cdot \frac{1}{2} \cdot (y - \hat{a}_3)(-1) = -(y - \hat{a}_3)$$

$$\frac{\partial L}{\partial \hat{a}_3} = (\hat{a}_3 - y)$$
Multi-Layer Perceptron

- Now the last part of $\delta_3$.
- It is the derivative of the activation function.

\[
\frac{\partial L}{\partial \tilde{a}_3} = (\tilde{a}_3 - y)
\]
\[
\frac{\partial L}{\partial \tilde{w}_2} = \delta_3 \cdot \tilde{a}_2
\]

\[
\delta_3 = \frac{\partial L}{\partial \tilde{z}_3} = \frac{\partial L}{\partial \tilde{a}_3} \cdot \frac{\partial \tilde{a}_3}{\partial \tilde{z}_3}
\]

\[
\frac{\partial \tilde{a}_3}{\partial \tilde{z}_3} = g'(\tilde{z}_3)
\]

\[
\tilde{a}_3 = g(\tilde{z}_3)
\]

\[
\delta_3 = \frac{\partial L}{\partial \tilde{z}_3} = (\tilde{a}_3 - y)g'(\tilde{z}_3)
\]
Multi-Layer Perceptron

- Finally, we derived an expression for the variation in the loss caused by $w_2$: $\frac{\partial L}{\partial w_2}$

$$\frac{\partial L}{\partial w_2} = \delta_3 \cdot \tilde{a}_2 = (\tilde{a}_3 - y)g'(\tilde{z}_3)\tilde{a}_2$$

\[\delta_3 = \frac{\partial L}{\partial \tilde{z}_3} = (\tilde{a}_3 - y)g'(\tilde{z}_3)\]

\[\frac{\partial L}{\partial \tilde{w}_2} = \delta_3 \cdot \tilde{a}_2\]
Now, we **derive the expression** for the variation in the loss caused by $w_1$: $\frac{\partial L}{\partial w_1}$

It is the product of: variation in $L$ caused by hidden layer input $z_2$ and the variation in $z_2$ due to $w_1$.

\[
\frac{\partial L}{\partial \hat{w}_1} = \frac{\partial L}{\partial \hat{z}_2} \cdot \frac{\partial \hat{z}_2}{\partial \hat{w}_1}
\]
Multi-Layer Perceptron

- Let’s calculate the error term $\delta_2$.
- Again we use the chain rule (blue).

\[
\delta_2 \equiv \frac{\partial L}{\partial \hat{z}_2} = \frac{\partial L}{\partial \hat{z}_3} \cdot \frac{\partial \hat{z}_3}{\partial \hat{a}_2} \cdot \frac{\partial \hat{a}_2}{\partial \hat{z}_2}
\]

We calculate these 3 terms separately

\[
\delta_3 = \frac{\partial L}{\partial \hat{z}_3} = (\hat{a}_3 - y)g'(\hat{z}_3)
\]

\[
\begin{align*}
\hat{z}_2 &= \vec{w}_1^T \hat{a}_1 \\
\hat{a}_2 &= g(\hat{z}_2) \\
\hat{z}_3 &= \vec{w}_2^T \hat{a}_2 \\
\hat{a}_3 &= g(\hat{z}_3)
\end{align*}
\]
Finally, we have the expression for $\delta_2$.

\[
\delta_2 = \frac{\partial L}{\partial \hat{z}_2} = \delta_3 \vec{w}_2 g' (\hat{z}_2)
\]

Observe that the 2\textsuperscript{nd} layer error is given by the 3\textsuperscript{rd} layer neuron.

There could be multiple neurons in the 3\textsuperscript{rd} layer that contribute to the error of each neuron in the 2\textsuperscript{nd} layer.
Error of each neuron in the 2\textsuperscript{nd} layer is $\delta_2$:

$$\delta_2 = \frac{\partial L}{\partial \hat{z}_2} = \delta_3 \hat{w}_2 g'(\hat{z}_2)$$

Note that the 3\textsuperscript{rd} layer errors from multiple neurons contribute to the single 2\textsuperscript{nd} layer neuron.
Multi-Layer Perceptron

• The error of the single neuron $z_2$ ($\delta_2$) is expressed as the weighted sum of the 3$^{rd}$ layer errors propagated backwards.

• Observe that the errors propagate backwards.

\[ \delta_2 = \frac{\partial L}{\partial \hat{z}_2} = g'(\hat{z}_2) \sum_{\text{units}} \vec{w}_2 \delta_3 \]

Hence, the name backpropagation!
Finally, we derived an expression for the variation in the loss caused by $w_1$: $\partial L / (\partial w_1)$

$$\frac{\partial L}{\partial \tilde{w}_1} = \delta_2 \cdot \tilde{a}_1 = \left[ g' (\tilde{z}_2) \sum_{\text{layer 3 units}} \tilde{w}_2 \delta_3 \right] \cdot \tilde{a}_1$$

$$\delta_2 = \frac{\partial L}{\partial \tilde{z}_2} = g' (\tilde{z}_2) \sum_{\text{layer 3 units}} \tilde{w}_2 \delta_3$$
Now we are ready to **state the update rules** for the gradient descent algorithm.

\[
\frac{\partial L}{\partial \vec{w}_2} = \delta_3. \hat{a}_2 = (\hat{a}_3 - y) g'(\hat{z}_3) \hat{a}_2
\]

\[
\frac{\partial L}{\partial \vec{w}_1} = \delta_2. \hat{a}_1 = \left[ g'(\hat{z}_2) \sum_{\text{layer 3 units}} \vec{w}_2 \delta_3 \right] \hat{a}_1
\]

\[
\vec{w}_2^{(t+1)} = \vec{w}_2^{(t)} - \eta (\hat{a}_3 - y) g'(\hat{z}_3) \hat{a}_2
\]

\[
\vec{w}_1^{(t+1)} = \vec{w}_1^{(t)} - \eta \left[ g'(\hat{z}_2) \sum_{\text{layer 3 units}} \vec{w}_2 \delta_3 \right] \hat{a}_1
\]
Multi-Layer Perceptron

• Carefully note the following **general rule** for computing the loss derivative.

**General rule:** Loss caused by $W^{(p-1)}$ of the layer $(p)$:

- **Error** $\delta^{(p)}$ of the layer $(p)$ *output* $a^{(p-1)}$ of layer $(p - 1)$

$$\frac{\partial L}{\partial \vec{w}_1} = \delta_2 \cdot \hat{a}_1$$

$$\frac{\partial L}{\partial \vec{w}_2} = \delta_3 \cdot \hat{a}_2$$

Layer k = 1
Layer k = 2
Layer k = 3

$$\vec{w}_2^{(t+1)} = \vec{w}_2^{(t)} - \eta \frac{\partial L}{\partial \vec{w}_2}$$

$$\vec{w}_1^{(t+1)} = \vec{w}_1^{(t)} - \eta \frac{\partial L}{\partial \vec{w}_1}$$
**General rule:** Loss caused by $W^{(p-1)}$ of the layer $(p)$: Error $\delta^{(p)}$ of the layer $(p)$ * output $a^{(p-1)}$ of layer $(p - 1)$

First, we calculate the error $\delta_{j}^{(K)}$ due to the last/output layer weight $w_2$, using the error to compute the loss gradient.

$$\frac{\partial L}{\partial w_2} = \delta_3 \cdot \tilde{a}_2$$

Then, we calculate the error $\delta_{j}^{(K-1)}$ due to the the hidden layer weights (e.g., $w_1$) recursively using $\delta_{j}^{(K)}$. Finally, compute loss gradient.

$$\delta_2 = \frac{\partial L}{\partial \tilde{z}_2} = g'(\tilde{z}_2) \sum_{l=3}^{layer 3 \ units} \overrightarrow{w}_2 \delta_3$$

$$\frac{\partial L}{\partial w_1} = \delta_2 \cdot \tilde{a}_1$$
Multi-Layer Perceptron

- The activation function $g(.)$ in the hidden layers must be differentiable nonlinear.
- E.g., sigmoid/tanh/ReLU.

Why $g(.)$ needs to be differentiable?

Limitation of the Perceptron step activation function!

$$
\overrightarrow{w}_2^{(t+1)} = \overrightarrow{w}_2^{(t)} - \eta (\overrightarrow{a}_3 - y) g' (\overrightarrow{z}_3) \overrightarrow{a}_2
$$

$$
\overrightarrow{w}_1^{(t+1)} = \overrightarrow{w}_1^{(t)} - \eta \left[ g' (\overrightarrow{z}_2) \sum_{\text{layer 3 units}} \overrightarrow{w}_2 \delta_3 \right] \cdot \overrightarrow{a}_1
$$
Now let’s see how to **generalize** the Gradient Descent algorithm for an arbitrary size (k-layer) MLP.