Machine Learning for Intelligent Systems

Lecture 14: Backpropagation in Networks

Reading: UML 20.6

The slides are altered from class to use column matrices throughout.

Instructors: Nika Haghtalab (this time) and Thorsten Joachims
Multi Layer Neural Network

Input layer
\[ V_0 = \mathbf{x} \]

Hidden layers
\[ V_1 = \mathbf{v}_1 \quad V_2 = \mathbf{v}_2 \]
\[ \text{Depth: } d \]

Output
\[ V_{d-1} = \mathbf{v}_{d-1} \]
\[ V_d \]

\[ \Delta \quad \mathbf{W}_{1,2} \]

\[ v_{0,1}, v_{0,2}, \ldots, v_{0,n}, v_{0,n+1} \]

\[ v_{1,1}, v_{1,2}, \ldots, v_{1,k-1}, v_{1,k}, v_{2,1}, v_{2,2}, \ldots, v_{2,k-1}, v_{2,k} \]

\[ v_{d-1,1}, v_{d-1,2}, \ldots, v_{d-1,k-1}, v_{d-1,k} \]

\[ v_{d,1} \]

Vector of weights going from layer \( i \) to the \( j^{th} \) node of layer \( i + 1 \):
\[ \mathbf{W}_{i,j} \]

Vector of values in layer \( i + 1 \),
\[ \mathbf{v}_{i+1} = \begin{pmatrix} \sigma(\mathbf{v}_i \cdot \mathbf{W}_{i,1}) \\ \vdots \\ \sigma(\mathbf{v}_i \cdot \mathbf{W}_{i,k-1}) \\ 1 \end{pmatrix} \]

Output: \( \mathbf{v}_{d,1} \)
# Common Activation Functions

Use a non-linear activation function on nodes of a hidden layer.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Gradient</th>
<th>Graph</th>
</tr>
</thead>
</table>
| Binary step    | \( \text{sign}(x) \) | \[
\begin{cases}
0 & x \neq 0 \\
N/A & x = 0
\end{cases}
\] | ![Graph](image) |
| sigmoid        | \( \sigma(x) = \frac{1}{1 + \exp(-x)} \) | \( \sigma(x)(1 - \sigma(x)) \) | ![Graph](image) |
| Tanh           | \( \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \) | \( (1 - \tanh(x))^2 \) | ![Graph](image) |
| Rectified Linear (ReLu) | \( \text{relu}(x) = \max(x, 0) \) | \[
\begin{cases}
1 & x \geq 0 \\
0 & x < 0
\end{cases}
\] | ![Graph](image) |

Sometime, \( \sigma(x) \) denotes the “generic” notion of activation function, not necessarily sigmoid.
Closer look at the nodes

\[ \vec{w}_{i,j}: \text{vector of weight from layer } i \text{ to } j^{th} \text{ node in the next layer.} \]

\[ s_{i+1,j} = \overrightarrow{w}_{i,j} \cdot \vec{v}_i \]

\[ v_{i+1,j} = \sigma(s_{i+1,j}) \]

Weighted sum of values from the previous layer.  

The value of the node.

Vectorized form:

\[ \vec{v}_{i+1} = \begin{pmatrix} \sigma(\vec{v}_i \cdot \overrightarrow{w}_{i,1}) \\ \vdots \\ \sigma(\vec{v}_i \cdot \overrightarrow{w}_{i,k-1}) \end{pmatrix} \]

Bias term, has no incoming weights, so it's a fixed constant.
Simpler form

Vectorized form:
\[
\tilde{v}_{i+1} = \begin{pmatrix}
\sigma(\tilde{v}_i \cdot \tilde{w}_{i,1}) \\
\vdots \\
\sigma(\tilde{v}_i \cdot \tilde{w}_{i,k-1}) \\
\frac{1}{2}
\end{pmatrix}
\]

Bias term, has no incoming weights.

Presentation Trick:
→ When \( \sigma \) is sigmoid, \( \sigma(0) = \frac{1}{2} \).
→ We can assume that \( \tilde{w}_{i,k} = 0 \).
→ The network is fully connected.

\[ \tilde{v}_{i+1} = \sigma(W_i^T \tilde{v}_i) \]
Prediction with Neural Networks

How do we use Neural Networks for predictions?

\[
\sigma \left( W_{d-1}^T \ldots \sigma \left( W_2^T \sigma \left( W_1^T \sigma \left( W_0^T \hat{v}_0 \right) \right) \right) \ldots \right)
\]

**Input:** Neural Network with weight matrices \( W_0, W_1, \ldots, W_{d-1} \) and instance \((\tilde{x}, y)\)

\( \hat{v}_0 = \tilde{x} \)

**For** \( \ell = 1, \ldots, d \)

- \( \tilde{s}_\ell = W_{\ell-1}^T \hat{v}_{\ell-1} \)
- \( \hat{v}_\ell = \sigma(\tilde{s}_\ell) \)

**End For**

**Output** \( \hat{v}_d \)
Learning the Weights

How can we learn weight matrices $W_0, W_1, ..., W_{d-1}$ given a sample set $S = \{(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)\}$.

Need to write the optimization:

$$\min_{W_0, ..., W_{d-1}} \frac{1}{m} \sum_{i=1}^{m} L(\text{forward pass}_{W_0, ..., W_{d-1}}(\vec{x}_i), y_i)$$

For convenience, we don’t use a regularizer in this lecture.

$L(y', y_i)$: Evaluates how good is $y'$ as a prediction for $y_i$. We use

$$L(y', y_i) = \frac{1}{2} (y' - y_i)^2$$
SGD for Neural Networks

How can we learn weight matrices $W_0, W_1, ..., W_{d-1}$ given a sample set $S = \{(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)\}$.

Stochastic Gradient Descent updates rule:

$$\min_{W_0, ..., W_{d-1}} \frac{1}{m} \sum_{i=1 \ldots m} L(\text{forward pass}_{W_0, ..., W_{d-1}}(x), y_i)$$

Take a random $(\vec{x}, y)$ from the samples, let $\partial L$ denote the partial derivative for this sample with respect to the current weights

For all $i = 0, ..., d - 1$, $W_i \leftarrow W_i - \eta \frac{\partial L}{\partial W_i}$
Derivative of the network

We need to compute the derivates \( \frac{\partial L}{\partial w_i} \)

Chain rule in derivates:

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial h(x)} \cdot \frac{\partial h(x)}{\partial x}
\]

(Element wise) Application of chain rule:

\[
\frac{\partial L}{\partial w_{i,j}(z)} = \frac{\partial L}{\partial v_{i+1,j}} \cdot \frac{\partial v_{i+1,j}}{\partial s_{i+1,j}} \cdot \frac{\partial s_{i+1,j}}{\partial w_{i,j}(z)}
\]

\[
= \frac{\partial L}{\partial v_{i+1,j}} \sigma'(s_{i+1,j}) \cdot v_{i,z}
\]
We need to compute the derivates \( \frac{\partial L}{\partial v_{i,j}} \)

For \( v_d \), it's just the derivate of the loss with respect to the actual label \( y \). For square loss:

\[
\frac{\partial L}{\partial v_d} = v_d - y
\]

Application of chain rule:

\[
\frac{\partial L}{\partial v_{d-1,1}} = \frac{\partial L}{\partial v_d} \cdot \frac{\partial v_d}{\partial s_d} \cdot \frac{\partial s_d}{\partial v_{d-1,1}}
\]

\[
= (v_d - y) \cdot \sigma'(s_d) \cdot w_{d-1,1}(1)
\]
More generally, recursively

\[
\frac{\partial L}{\partial \tilde{v}_i} = \frac{\partial L}{\partial \tilde{v}_{i+1}} \cdot \frac{\partial \tilde{v}_{i+1}}{\partial \tilde{s}_{i+1}} \cdot \frac{\partial \tilde{s}_{i+1}}{\partial \tilde{v}_i}
\]

\[
= W_i \left( \sigma'(\tilde{s}_{i+1}) \odot \frac{\partial L}{\partial \tilde{v}_{i+1}} \right)
\]

Element-wise matrix multiplication: \[
\begin{pmatrix} a \\ b \end{pmatrix} \odot \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} ac \\ bd \end{pmatrix}
\]
All together

We can compute all the derivates \( \frac{\partial L}{\partial \tilde{v}_i} \) before each round of SGD.

\[
\frac{\partial L}{\partial v_d} = v_d - y
\]

\[
\frac{\partial L}{\partial \tilde{v}_i} = W_i \left( \frac{\partial L}{\partial \tilde{v}_{i+1}} \odot \sigma'(\tilde{s}_{i+1}) \right)
\]

When we start, we now have the derivates with respect to weights.

\[
\frac{\partial L}{\partial W_i} = \tilde{v}_i \left( \frac{\partial L}{\partial \tilde{v}_{i+1}} \odot \sigma'(\tilde{s}_{i+1}) \right)^T
\]
Back Propagation

Each Round of SGD:
Forward Pass then Back Propagation

Forward Pass (Prediction)

\[ \hat{\nu}_0 = (\hat{x}, 1) \]
For \( \ell = 1, \ldots, d \)
  - \( \hat{\mu}_\ell = W_{\ell-1}^T \hat{\nu}_{\ell-1} \)
  - \( \hat{\nu}_\ell = \sigma(\hat{\mu}_\ell) \)
End For
Output \( \hat{\mu}_0, \hat{\nu}_0, \ldots, \hat{\mu}_d, \hat{\nu}_d \)

Backward Pass

\[ \hat{\delta}_d = \frac{\partial L}{\partial \nu_d} \sigma'(s_d) \]
For \( \ell = d - 1, \ldots, 0 \)
  - \( W_\ell = W_\ell - \eta \hat{\nu}_\ell \hat{\delta}_{\ell+1}^T \)
  - \( \hat{\delta}_\ell = W_\ell (\sigma'(\hat{\mu}_\ell) \odot \hat{\delta}_{\ell+1}) \)
End For
Output \( W_0, W_1, \ldots W_{d-1} \)
The loss in neural networks is non-convex.

Many of the theoretical guarantees of SGD only hold for convex losses.

1. Initialization.
   - Don’t start $W_0, W_1, ..., W_{d-1} = 0$
   - Randomized your starting weights.

2. Run separate SGDs and take the best.