Machine Learning for Intelligent Systems

Lecture 13: Deep Neural Networks

Reading: UML 20-20.3

Instructors: Nika Haghtalab (this time) and Thorsten Joachims

---

**(Stochastic) Gradient Descent**

Many learning problems can be written as the following optimization on the sample set \( S = \{(x_i, y_i), \ldots, (x_n, y_n)\} \):

\[
\min_{\theta} L_{\mathcal{D}}(\theta) \quad \text{for} \quad L_{\mathcal{D}}(\theta) = R(\theta) + \frac{1}{n} \sum_{i=1}^{n} L(\theta ; x_i, y_i)
\]

**Gradient Descent Update:**

\[
\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L_{\mathcal{D}}(\theta^{(t)})
\]

**Stochastic Gradient Descent Update:** Take a random \((\tilde{x}_i, y_i) \sim S\)

\[
\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L_{\mathcal{D}}(\theta^{(t)}) - \eta \nabla L_{\mathcal{D}}(\tilde{x}_i, y_i)
\]

---

**AdaGrad**

Adaptive Gradient:
- Adapt the learning rate for each parameter based on the previous gradients.

For all coordinates \(i\):  
- Derivative: 
  \[
  \frac{\partial L(\theta^{(t)})}{\partial \theta_i}
  \]
- Update: 
  \[
  \theta_i^{(t+1)} = \theta_i^{(t)} - \eta \frac{\theta_i}{\sqrt{0.01 + \sum_{j=1}^{t} (\theta_j)^2}}
  \]

---

**Momentum Method**

Adaptive Gradient:
- Use previous gradients to encourage movement in important directions.

**Exp-weighted Average Gradient**

\[
\hat{\theta}^{(t)} \leftarrow (1 - \beta) \hat{\theta}^{(t-1)} + \beta \nabla L(\theta^{(t)})
\]

**Update**

\[
\theta^{(t+1)} = \theta^{(t)} - \eta \hat{\theta}^{(t)}
\]

---

**SGD on Non-Convex**

Non convex functions are challenging

Under specific assumptions SGD provably converges to a local minimum

Neural networks are non-convex and SGD is used for training them.

---

**Linear Model**

We can represent a linear function as single layer neural network.
Naïve Hidden Layers

Linear function of linear functions, is linear:
\[ v_1 (x \cdot w + b) + v_2 (x \cdot w' + b') = x \cdot (v_1 w + v_2 w') + (v_1 b + v_2 b'). \]
Beyond linearity: We need each layer to transform a linear function to something else.

Power of Neural Networks

Represent XOR with 1 hidden layer:

\[ \text{sign}(x_1 + x_2 + 0.1) \]
\[ \text{sign}(\text{sign}(x_1 + x_2 + 0.1) + \text{sign}(-x_1 - x_2 + 0.1) = 1) \]

Universal Approximators

If we allow a single hidden layer (depth 2 network) with very large width, we can approximate any continuous function on \( \mathbb{R}^n \).

How large?
- For boolean functions, we need at least \( \exp(n) \) width.
- Restricting ourselves to polynomial size networks
  - Can’t approximate all functions
  - Reduce the chance of overfitting

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>
<tbody>
<tr>
<td>Binary step</td>
<td>( \text{sign}(x) )</td>
<td>[ \begin{cases} 0 &amp; x = 0 \ 1 &amp; x &gt; 0 \ -1 &amp; x &lt; 0 \end{cases} ]</td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Sigmoid</td>
<td>( \sigma(x) = \frac{1}{1 + \exp(-x)} )</td>
<td>( \sigma(x)(1 - \sigma(x)) )</td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Tangent</td>
<td>( \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} )</td>
<td>( (1 - \tanh(x)^2) )</td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Restricted</td>
<td>( \text{relu}(x) = \max(x, 0) )</td>
<td>[ \begin{cases} 1 &amp; x &gt; 0 \ 0 &amp; x &lt; 0 \end{cases} ]</td>
<td><img src="image" alt="Graph" /></td>
</tr>
</tbody>
</table>

Sometimes, \( \sigma(x) \) denotes the “generic” notion of activation function, not necessarily sigmoid.

Multi Layer Neural Network

Vector of weights going from layer \( i \) to the \( j^{th} \) node of layer \( i+1 \): \( \mathbf{w}_{ij} \)

Vector of values in layer \( i+1 \), \( \mathbf{V}_{i+1} = \mathbf{V}_{i+1} \) Output: \( \sigma(\mathbf{V}_{i+1}) \)

Other Types of Neural Networks

- Traditional multi-layer networks:
  - Layers are fully connected
  - Bad for overfitting

Other types
- Convolutional Neural Networks (CNNs)
  - Some structured layers to learn features
  - 1-2 layers of fully connected network at the end
- Recurrent Neural Networks (RNNs)
  - Nodes can feed forward or backward.