Lecture 2: Neural Networks

April 5th, 2021

Logistics: pls submit HW notebooks with the outputs
Feature transform

Non-linear hypothesis!
Artificial Neural Networks

Vaguely inspired by biological neural networks

Input vector → Hidden layers → Output vector

\[ x_0^0 \rightarrow x_1^1 \rightarrow \ldots \rightarrow x_0^{L-1} \]

\[ x_1^0 \rightarrow x_1^1 \rightarrow \ldots \rightarrow x_1^{L-1} \]

\[ x_2^0 \rightarrow x_2^1 \rightarrow \ldots \rightarrow x_2^{L-1} \]

\[ x_3^0 \rightarrow x_3^1 \rightarrow \ldots \rightarrow x_3^{L-1} \]
\[ x_j^l = \sigma \left( \sum_i w_{j,i}^l \cdot x_{i}^{l-1} + b_j^l \right) \]

\[ x^l = \sigma (W^l x^{l-1} + b^l) \]
Learning by SGD

We need

\[
\frac{\partial \mathcal{L}(\theta; x, y)}{\partial w^l_{ji}}, \quad \frac{\partial \mathcal{L}(\theta; x, y)}{\partial b^l_j}
\]

To all l, l, j

\[
x^l_j = \sum_i W^l_{ji} x^l_{i-1} + b_j^l
\]

\[
\sum_i W^l_{ji} \sigma \left( \sum_k W^l_{ik} x^{l-1}_{k-1} + b_{i-1}^l + b_j^l \right)
\]

\[
\sum_i W^l_{ji} \sigma \left( \sum_k W^l_{ik} x^{l-1}_{k-2} \sum_m W^l_{km} x^{l-3} + b_k^{l-2} + b_i^{l-1} \right) + b_j^l
\]
Back Propagation - preliminaries

\[ x_j^l = \sigma \left( \sum_i w_{ji}^l \cdot x_i^{l-1} + b_j \right) \]

\[ \frac{\partial L}{\partial w_{ji}^l} = \frac{\partial L}{\partial x_j^l} \cdot \frac{\partial x_j^l}{\partial w_{ji}^l} \]

\[ \Delta = g_j^l \]

Obtained by backprop

\[ x_i^{l-1} \cdot \sigma'(z_j^l) \]

Easy!

Derivatives of common activations are easy!

**Sigmoid**

\[ f(x) = \frac{1}{1 + e^{-x}} \]

\[ \sigma'(z) = \sigma(z)(1 - \sigma(z)) \]

**Tanh**

\[ \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \]

\[ \tanh'(z) = 1 - \tanh^2(z) \]

**ReLU**

\[ f(x) = \begin{cases} 
0 & \text{for } x < 0 \\
x & \text{for } x \geq 0 
\end{cases} \]

1 for z>0, 0 otherwise
Back Propagation

\[ g_j^l = \frac{\partial L}{\partial x_j^l} \]

\[ = \sum_k \frac{\partial L}{\partial x_k^{l+1}} \cdot \frac{\partial x_k^{l+1}}{\partial x_j^l} \]

\[ = \sum_k g_k^{l+1} \cdot w_{kj}^{l+1} \cdot \sigma'(z_k^{l+1}) \]

Stopping criterion:

\[ x_j^l = \sigma \left( \sum_i w_{ji}^l \cdot x_i^{l-1} + b_j \right) \]

\[ g_j^l = \frac{\partial L}{\partial x_j^l} \]

\[ z_j^l \]
• Initialize weights
• Repeat until convergence:

1. Sample a batch from the data: \(\{(x_i, y_i) \ldots\}\)
2. Forward pass: \(x^l = \sigma(W^l x^{l-1} + b^l)\), \(L = \text{loss}(x^L, y)\)
3. Backward pass: \(g^L = \frac{\partial L}{\partial x^L}\), \(g^l = \left(W^{l+1T}(\sigma'_l g^{l+1})\right)\)
4. Calculate weights gradient: \(\frac{\partial L}{\partial W^l} = x^l \cdot (\sigma(z^l) \circ g^l)^T\), \(\frac{\partial L}{\partial b^l} = \sigma'(z^l) \circ g^l\)
5. Update weights: \(W^l := W^l - \alpha \frac{\partial L}{\partial W^l}\), \(b^l := b^l - \alpha \frac{\partial L}{\partial b^l}\)

### Back Propagation
Let’s get more generic

OK, Deep Learning has outlived its usefulness as a buzz-phrase. Deep Learning est mort. Vive Differentiable Programming!

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

But the important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization.
Let's get more generic

\[ x^{l+1} = T_{\text{forward}}^{l+1}(x^l; T^{l+1}.\text{params}) \]

\[ x^0 \rightarrow \text{forward } T^1 \rightarrow \text{backward } \rightarrow x^1 \rightarrow \text{forward } T^2 \rightarrow \text{backward } \rightarrow \cdots \rightarrow x^{L-1} \rightarrow \text{forward } T^L \rightarrow \text{backward } \rightarrow x^L \rightarrow \text{forward Loss func} \rightarrow y \]

\[ x^l . \text{grad} = \frac{\partial L}{\partial x^l} \quad \text{grad} = \frac{\partial x^{l+1}}{\partial x^l} \text{params} . \text{grad} \]

\[ x^l . \text{grad} = \frac{\partial L}{\partial x^l} \quad \text{grad} = \frac{\partial x^{l+1}}{\partial x^l} \text{params} . \text{grad} \]

Use to update weights:

\[ W := W - \alpha W . g \]
Example
BTW: You can backprop any DAG!

\[
\begin{align*}
    x^0 &\xrightarrow{\text{forward}} x^1 \\
    x^0 \cdot g &\xrightarrow{\text{backward}} x^1 \\
    W \cdot g &\xrightarrow{\text{backward}} W
\end{align*}
\]
Q: Can we use any function inside a network?

- Has to be differentiable with respect to input and params
- Implications to optimization can be deadly:
  - 0-1 clamp layer in the end of the network.
  - Ideas how to solve this?
  - Square root loss ($L_{1/2}^1$ norm)
http://playground.tensorflow.org
This week’s tutorial:

Intro to PyTorch

Shir Amir

Next week’s lecture:

Convolutional Neural Networks

(Me Again 😞)