Neural Networks

Philipp Koehn

11 April 2019
Supervised Learning

- Examples described by attribute values (Boolean, discrete, continuous, etc.)

- E.g., situations where I will/won’t wait for a table:

<table>
<thead>
<tr>
<th>Example</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>WillWait</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₂</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
</tr>
<tr>
<td>X₃</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₄</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
</tr>
<tr>
<td>X₅</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X₆</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₇</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₈</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₉</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
<td>F</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
<td>F</td>
</tr>
<tr>
<td>X₁₁</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₁₂</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
</tr>
</tbody>
</table>

- Classification of examples is positive (T) or negative (F)
Naive Bayes Models

- **Bayes rule**

\[ p(C|A) = \frac{1}{Z} p(A|C) p(C) \]

- **Independence assumption**

\[ p(A|C) = p(a_1, a_2, a_3, ..., a_n|C) \]

\[ \approx \prod_i p(a_i|C) \]

- **Weights**

\[ p(A|C) = \prod_i p(a_i|C)^{\lambda_i} \]
Naive Bayes Models

- Linear model
  \[ p(A|C) = \exp \prod_i p(a_i|C)^{\lambda_i} \]

- Probability distribution as features
  \[ h_i(A, C) = \log p(a_i|C) \]
  \[ h_0(A, C) = \log p(C) \]

- Linear model with features
  \[ p(C|A) \propto \sum_i \lambda_i h_i(A, C) \]
Linear Model

- Weighted linear combination of feature values $h_j$ and weights $\lambda_j$ for example $d_i$

$$\text{score}(\lambda, d_i) = \sum_j \lambda_j h_j(d_i)$$

- Such models can be illustrated as a "network"
Limits of Linearity

- We can give each feature a weight

- But not more complex value relationships, e.g,
  - any value in the range [0;5] is equally good
  - values over 8 are bad
  - higher than 10 is not worse
- Linear models cannot model XOR

```plaintext

XOR

<table>
<thead>
<tr>
<th>good</th>
<th>bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad</td>
<td>good</td>
</tr>
</tbody>
</table>
```

Philipp Koehn
Artificial Intelligence: Neural Networks
11 April 2019
Multiple Layers

- Add an intermediate ("hidden") layer of processing (each arrow is a weight)

- Have we gained anything so far?
Non-Linearity

- Instead of computing a linear combination

\[
\text{score}(\lambda, \mathbf{d}_i) = \sum_j \lambda_j h_j(d_i)
\]

- Add a non-linear function

\[
\text{score}(\lambda, \mathbf{d}_i) = f\left( \sum_j \lambda_j h_j(d_i) \right)
\]

- Popular choices

\[
\begin{align*}
tanh(x) & \quad \text{and} \\
\text{sigmoid}(x) & = \frac{1}{1+e^{-x}}
\end{align*}
\]

(sigmoid is also called the ”logistic function”)

Philipp Koehn
Artificial Intelligence: Neural Networks
11 April 2019
• More layers = deep learning
example
Simple Neural Network

One innovation: bias units (no inputs, always value 1)
Sample Input

- Try out two input values
- Hidden unit computation

\[
\text{sigmoid}(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = \text{sigmoid}(2.2) = \frac{1}{1 + e^{-2.2}} = 0.90
\]

\[
\text{sigmoid}(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = \text{sigmoid}(-1.6) = \frac{1}{1 + e^{1.6}} = 0.17
\]
Computed Hidden

- Try out two input values
- Hidden unit computation

\[
sigmoid(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = sigmoid(2.2) = \frac{1}{1 + e^{-2.2}} = 0.90
\]

\[
sigmoid(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = sigmoid(-1.6) = \frac{1}{1 + e^{1.6}} = 0.17
\]
Compute Output

- Output unit computation

\[
\text{sigmoid}(0.90 \times 4.5 + 0.17 \times -5.2 + 1 \times -2.0) = \text{sigmoid}(1.17) = \frac{1}{1 + e^{-1.17}} = 0.76
\]
Computed Output

- Output unit computation

\[
\text{sigmoid}(0.90 \times 4.5 + 0.17 \times -5.2 + 1 \times -2.0) = \text{sigmoid}(1.17) = \frac{1}{1 + e^{-1.17}} = 0.76
\]
why “neural” networks?
Neuron in the Brain

- The human brain is made up of about 100 billion neurons

- Neurons receive electric signals at the dendrites and send them to the axon
Neural Communication

- The axon of the neuron is connected to the dendrites of many other neurons.
The Brain vs. Artificial Neural Networks

- **Similarities**
  - Neurons, connections between neurons
  - Learning = change of connections, not change of neurons
  - Massive parallel processing

- **But artificial neural networks are much simpler**
  - computation within neuron vastly simplified
  - discrete time steps
  - typically some form of supervised learning with massive number of stimuli
back-propagation training
Computed output: $y = .76$

Correct output: $t = 1.0$

How do we adjust the weights?
Key Concepts

- Gradient descent
  - error is a function of the weights
  - we want to reduce the error
  - gradient descent: move towards the error minimum
  - compute gradient $\rightarrow$ get direction to the error minimum
  - adjust weights towards direction of lower error

- Back-propagation
  - first adjust last set of weights
  - propagate error back to each previous layer
  - adjust their weights
Derivative of Sigmoid

• Sigmoid

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

• Reminder: quotient rule

\[
\left(\frac{f(x)}{g(x)}\right)' = \frac{g(x)f'(x) - f(x)g'(x)}{g(x)^2}
\]

• Derivative

\[
\frac{d}{dx} \text{sigmoid}(x) = \frac{d}{dx} \frac{1}{1 + e^{-x}}
\]

\[
= \frac{0 \times (1 - e^{-x}) - (-e^{-x})}{(1 + e^{-x})^2}
\]

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

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

\[
= \text{sigmoid}(x)(1 - \text{sigmoid}(x))
\]
Final Layer Update

- Linear combination of weights $s = \sum_k w_k h_k$

- Activation function $y = \text{sigmoid}(s)$

- Error (L2 norm) $E = \frac{1}{2}(t - y)^2$

- Derivative of error with regard to one weight $w_k$

$$\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}$$
Final Layer Update (1)

- Linear combination of weights $s = \sum_k w_k h_k$
- Activation function $y = \text{sigmoid}(s)$
- Error (L2 norm) $E = \frac{1}{2} (t - y)^2$
- Derivative of error with regard to one weight $w_k$
  \[
  \frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}
  \]
- Error $E$ is defined with respect to $y$
  \[
  \frac{dE}{dy} = \frac{d}{dy} \frac{1}{2} (t - y)^2 = -(t - y)
  \]
Final Layer Update (2)

- Linear combination of weights \( s = \sum_k w_k h_k \)
- Activation function \( y = \text{sigmoid}(s) \)
- Error (L2 norm) \( E = \frac{1}{2}(t - y)^2 \)
- Derivative of error with regard to one weight \( w_k \)

\[
\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}
\]

- \( y \) with respect to \( x \) is \( \text{sigmoid}(s) \)

\[
\frac{dy}{ds} = \frac{d \text{sigmoid}(s)}{ds} = \text{sigmoid}(s)(1 - \text{sigmoid}(s)) = y(1 - y)
\]
Final Layer Update (3)

- Linear combination of weights $s = \sum_k w_k h_k$

- Activation function $y = \text{sigmoid}(s)$

- Error (L2 norm) $E = \frac{1}{2}(t - y)^2$

- Derivative of error with regard to one weight $w_k$

$$\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k}$$

- $x$ is weighted linear combination of hidden node values $h_k$

$$\frac{ds}{dw_k} = \frac{d}{dw_k} \sum_k w_k h_k = h_k$$
Putting it All Together

- Derivative of error with regard to one weight $w_k$

$$\frac{dE}{dw_k} = \frac{dE}{dy} \frac{dy}{ds} \frac{ds}{dw_k} = -(t - y) \ y(1 - y) \ h_k$$

- Error
- Derivative of sigmoid: $y'$

- Weight adjustment will be scaled by a fixed learning rate $\mu$

$$\Delta w_k = \mu \ (t - y) \ y' \ h_k$$
Multiple Output Nodes

- Our example only had one output node
- Typically neural networks have multiple output nodes
- Error is computed over all $j$ output nodes

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2$$

- Weights $k \to j$ are adjusted according to the node they point to

$$\Delta w_{j \leftarrow k} = \mu (t_j - y_j) y'_j h_k$$
Hidden Layer Update

- In a hidden layer, we do not have a target output value
- But we can compute how much each node contributed to downstream error
- Definition of error term of each node
  \[ \delta_j = (t_j - y_j) y_j' \]
- Back-propagate the error term
  (why this way? there is math to back it up...)
  \[ \delta_i = \left( \sum_j w_{j\leftarrow i} \delta_j \right) y_i' \]
- Universal update formula
  \[ \Delta w_{j\leftarrow k} = \mu \delta_j h_k \]
Our Example

- Computed output: \( y = .76 \)
- Correct output: \( t = 1.0 \)
- Final layer weight updates (learning rate \( \mu = 10 \))
  - \( \delta_G = (t - y) \ y' = (1 - .76) \ 0.181 = .0434 \)
  - \( \Delta w_{GD} = \mu \ \delta_G \ h_D = 10 \times .0434 \times .90 = .391 \)
  - \( \Delta w_{GE} = \mu \ \delta_G \ h_E = 10 \times .0434 \times .17 = .074 \)
  - \( \Delta w_{GF} = \mu \ \delta_G \ h_F = 10 \times .0434 \times 1 = .434 \)
Our Example

- Computed output: $y = .76$
- Correct output: $t = 1.0$
- Final layer weight updates (learning rate $\mu = 10$)
  - $\delta_G = (t - y) \ y' = (1 - .76) \ 0.181 = .0434$
  - $\Delta w_{GD} = \mu \ \delta_G \ h_D = 10 \times .0434 \times .90 = .391$
  - $\Delta w_{GE} = \mu \ \delta_G \ h_E = 10 \times .0434 \times .17 = .074$
  - $\Delta w_{GF} = \mu \ \delta_G \ h_F = 10 \times .0434 \times 1 = .434$
Hidden Layer Updates

- **Hidden node D**
  - \( \delta_D = \left( \sum_j w_j \cdot \delta_j \right) y_D' = w_{GD} \delta_G y_D' = 4.5 \times 0.0434 \times 0.0898 = 0.0175 \)
  - \( \Delta w_{DA} = \mu \delta_D h_A = 10 \times 0.0175 \times 1.0 = 0.175 \)
  - \( \Delta w_{DB} = \mu \delta_D h_B = 10 \times 0.0175 \times 0.0 = 0 \)
  - \( \Delta w_{DC} = \mu \delta_D h_C = 10 \times 0.0175 \times 1 = 0.175 \)

- **Hidden node E**
  - \( \delta_E = \left( \sum_j w_j \cdot \delta_j \right) y_E' = w_{GE} \delta_G y_E' = -5.2 \times 0.0434 \times 0.1411 = -0.0318 \)
  - \( \Delta w_{EA} = \mu \delta_E h_A = 10 \times -0.0318 \times 1.0 = -0.318 \)
  - etc.
Connectionist Semantic Cognition

- Hidden layer representations for concepts and concept relationships
some additional aspects
Initialization of Weights

- Weights are initialized randomly
e.g., uniformly from interval $[-0.01, 0.01]$

- Glorot and Bengio (2010) suggest
  - for shallow neural networks
    \[ [-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}] \]
    
    $n$ is the size of the previous layer
  
  - for deep neural networks
    \[ [-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}] \]
    
    $n_j$ is the size of the previous layer, $n_j$ size of next layer
Neural Networks for Classification

- Predict class: one output node per class
- Training data output: "One-hot vector", e.g., $\vec{y} = (0, 0, 1)^T$
- Prediction
  - predicted class is output node $y_i$ with highest value
  - obtain posterior probability distribution by soft-max

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$
Speedup: Momentum Term

- Updates may move a weight slowly in one direction
- To speed this up, we can keep a memory of prior updates

\[ \Delta w_{j \leftarrow k}(n - 1) \]

- ... and add these to any new updates (with decay factor \( \rho \))

\[ \Delta w_{j \leftarrow k}(n) = \mu \delta_j h_k + \rho \Delta w_{j \leftarrow k}(n - 1) \]
computational aspects
Vector and Matrix Multiplications

- Forward computation: \( \tilde{s} = W \tilde{h} \)
- Activation function: \( \tilde{y} = \text{sigmoid}(\tilde{h}) \)
- Error term: \( \tilde{\delta} = (\tilde{t} - \tilde{y}) \text{sigmoid}'(\tilde{s}) \)
- Propagation of error term: \( \tilde{\delta}_i = W \tilde{\delta}_{i+1} \cdot \text{sigmoid}'(\tilde{s}) \)
- Weight updates: \( \Delta W = \mu \tilde{\delta} \tilde{h}^T \)
• Neural network layers may have, say, 200 nodes

• Computations such as $W\tilde{h}$ require $200 \times 200 = 40,000$ multiplications

• Graphics Processing Units (GPU) are designed for such computations
  – image rendering requires such vector and matrix operations
  – massively mulit-core but lean processing units
  – example: NVIDIA Tesla K20c GPU provides 2496 thread processors

• Extensions to C to support programming of GPUs, such as CUDA
Toolkits

- Tensorflow (Google)
- PyTorch (Facebook)
- MXNet (Amazon)