Deep Learning and the Dropout Algorithm

Jacob Goldberger

Engineering Faculty, Bar-Ilan University

Feb 25, 2015
1. Deep Learning
2. Dropout
3. Conclusion
Outline

1. Deep Learning
2. Dropout
3. Conclusion
### Definition (Wikipedia)

Classification is the problem of identifying to which of a set of classes a new observation belongs, on the basis of a training data whose class membership is known.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>features:</td>
<td>$x_1, x_2, \ldots, x_n$</td>
<td>$x$</td>
</tr>
<tr>
<td>labels:</td>
<td>$y_1, y_2, \ldots, y_n$</td>
<td>?</td>
</tr>
</tbody>
</table>
Linear Classifier

\[ w^\top x = \sum_i w_i x_i \]

\[ w^\top x > 0 \quad \rightarrow \quad y = 1 \]
\[ w^\top x < 0 \quad \rightarrow \quad y = 0 \]

The weights \( w \) are learned from the training data.
Logistic Regression

Sigmoid:

\[ p(y = 1| x; w) = Sigmoid(w^T x) = \frac{1}{1 + \exp(-w^T x)} \]

\[ p(y = 1| x; w) > 0.5 \iff w^T x > 0 \]
Logistic Regression - A single neuron

\[ p(y = 1|x; w) = \frac{1}{1 + e^{-w^\top x}} \]

Input layer

Output layer

Linear: \( x \rightarrow w^\top x \)

non-Linear: \( w^\top x \rightarrow Sigmoid(w^\top x) \)
Logistic Regression - training

Likelihood:

\[ S(w) = \sum_i \log p(y_i|x_i; w) \]

ML:

\[ \hat{w} = \arg \max_w S(w) \]

Gradient Ascent:

\[ w_{t+1} \leftarrow w_t + \epsilon \frac{dS}{dw}(w_t) \]
Logistic Regression

\[ p(y = 1 | x; w) = \frac{1}{1 + e^{-w^\top x}} \]

Pros: \( S(w) \) is concave \( \Rightarrow \) no local maxima.

Cons: The classifier is linear.
Neural Networks

\[ h = \text{Sig}(w_x^\top x) \]

\[ p(y = 1|x; w) = \text{Sig}(w_h^\top h) \]
Neural Networks - Deep Architecture

$p(y = 1 | x; w)$
(Soft) Classifier: $x \in R^d \rightarrow p(y = 1|x) \in [0, 1]$

universal approximation theory:

Standard multi-layer neural networks are capable of approximating any function to any desired degree of accuracy.
Neural Network - training

Likelihood:

\[ S(w) = \sum_i \log p(y_i|x_i; w) \]

**ML**:

\[ \hat{w} = \arg \max_w S(w) \]

Gradient Ascent:

\[ w_{t+1} \leftarrow w_t + \epsilon \frac{dS}{dw}(w_t) \]

Back-Propagation: efficient computation of

\[ \frac{dS}{dw}(w_t) = \sum_i \frac{d}{dw} \log p(y_i|x_i; w) \]
Pros: The classifier is non-linear.

Cons: $S(w)$ is not concave

⇒ many local maxima

⇒ difficult to train
1930 - Logistic-Regression (Perceptron)
1986 - Neural Networks, back-propagation
1992 - SVM, Kernels
2006 - Deep Learning
Deep Learning Success

- Speech recognition
- Image classification
- Object detection
- Automatic language translation
- ...

Deep Learning Success

- Much more available data
- Much more processing power
- pre-training (RBM, Autoencoder)
- Dropout
8 layers, 50M parameters
layer 6, 4K features
Outline

1. Deep Learning
2. Dropout
3. Conclusion
Training problem

- Getting stuck in a local optimum.
- Over-fitting to the training data.
Model Averaging

- Training several classifiers and average their decisions.

- Dropout is a clever way to model averaging. [Hinton, 2012]
Dropout at train time

- Set the output of each hidden neuron to zero w.p. 0.5.
- The neuron which is dropped-out in this way do not contribute to the forward pass.
- Every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.
Dropout at train time

$p(y = 1| x; w)$

Input layer → Hidden layer → Output layer

Jacob Goldberger  Deep Learning and the Dropout Algorithm
Dropout at train time

\[ p(y = 1 | x; w) \]
Dropout at train time

\[ p(y = 1 \mid x; w) \]
We could sample many different architectures and take the geometric mean of their output distributions.

It better to use all of the hidden units, but to halve their outgoing weights. This exactly computes the geometric mean of the predictions of all $2^m$ models.
Dropout at test time

\[ p_T(y = 1|x; w) = \text{Sig}(\frac{1}{2} w_h^T h) \]

\[ h = \text{Sig}(w_x^T x) \]
Dropout at test time

\[ p_A(y = 1|x; w) \]

\[
\log \frac{p_T(y = 1|x)}{p_T(y = 0|x)} = \frac{1}{2^m} \sum_A \log \frac{p_A(y = 1|x)}{p_A(y = 0|x)}
\]
Avoiding Co-adaptation

- If a hidden unit knows which other hidden units are present, it can co-adapt to them on the training data.
  - But complex co-adaptations are likely to go wrong on new test data.
  - Big, complex conspiracies are not robust.

- If a hidden unit has to work well with combinatorially many sets of co-workers, it is more likely to do something that is individually useful.
  - But it will also tend to do something that is marginally useful given what its co-workers achieve.
Avoiding Co-adaptation

\[ p(y = 1 \mid x; w) \]
The record breaking object recognition net uses dropout and it helps a lot.

If your deep neural net is significantly over-fitting, dropout will usually reduce the number of errors by a lot.

Any net that uses "early stopping" can do better by using dropout (at the cost of taking quite a lot longer to train).
What about Dropout in the input layer?

- It helps to use dropout there too, but with a higher probability of keeping an input unit.

- This trick is already used by the ‘denoising autoencoder’ pre-training algorithm.
Outline

1. Deep Learning
2. Dropout
3. Conclusion
We described the dropout learning algorithm.
It beats regular back-propagation on many datasets.
It prevents co-adaption of feature detectors.
Dropout roughly doubles the number of iterations required to converge.
Can be combined with other learning methods for improved performance.
Thank You