Artificial Intelligence

Albert-Ludwigs-Universität Freiburg

Thorsten Schmidt
Abteilung für Mathematische Stochastik

www.stochastik.uni-freiburg.de
thorsten.schmidt@stochastik.uni-freiburg.de
SS 2017
Literature (incomplete, but growing):

Our goal today

Backpropagation

Regularization
Forward propagation

- In a feedforward neural network to produce an output $\hat{y}$ from an input $x$ information flows forward through the network
- This is called forward propagation
- During training, forward propagation produces a scalar cost $J(\theta)$
Forward propagation algorithm for a typical deep neural net

- Require: Network depth, \( l \)
- Require: \( W^{(i)} \), \( i \in \{1, ..., l\} \), the weight matrices of the model
- Require: \( b^{(i)} \), \( i \in \{1, ..., l\} \), the bias parameters of the model
- Require: \( x \), the input to process
- Require: \( y \), the target output
- set \( h^{(0)} = x \)
- for \( k = 1, \ldots, l \) do:
  - \( a^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)} \)
  - \( h^{(k)} = f(a^{(k)}) \)
- at the end of the loop set:
  - \( \hat{y} = h^{(l)} \)
  - \( J(\theta) = L(\hat{y}, y) + \lambda \Omega(\theta) \), where \( \theta \) is \((W^{(i)}, b^{(i)}) \) \( i \in \{1, ..., l\} \)
The back-propagation algorithm allows the information from the cost to flow backwards through the network, in order to compute the gradient.

The term back-propagation is not the whole learning algorithm.

Back-propagation is only a method to compute the gradient.

Another algorithm, e.g. stochastic gradient descent, is used to perform learning using this gradient.
Computing an analytical expression for the gradient is straightforward.

Numerically evaluating such an expression can be computationally expensive.

The back-propagation algorithm does so using a simple and inexpensive procedure, that relates to the chain rule.

\[
\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} = f'(y) f'(x) f'(w) = f'(f(f(w))) f'(f(w)) f'(w)
\]
Backward propagation algorithm for a typical deep neural net

After the forward computation, compute the gradient on the output layer:
\[ g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y) \]

for \( k = l, l - 1, \ldots, 1 \) do
  Convert the gradient on the layer’s output into a gradient into the pre-
  nonlinearity activation (element-wise multiplication if \( f \) is element-wise):
  \[ g \leftarrow \nabla_{a^{(k)}} J = g \odot f'(a^{(k)}) \]
  Compute gradients on weights and biases (including the regularization term, 
  where needed):
  \[ \nabla_{b^{(k)}} J = g + \lambda \nabla_{b^{(k)}} \Omega(\theta) \]
  \[ \nabla_{W^{(k)}} J = g h^{(k-1)\top} + \lambda \nabla_{W^{(k)}} \Omega(\theta) \]
  Propagate the gradients w.r.t. the next lower-level hidden layer’s activations:
  \[ g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)\top} g \]
end for

this is Algorithm 6.4 in Goodfellow 2016
The gradients on weights and biases can be used for a stochastic gradient update.

Symbol-to-number differentiation (Torch, Caffe): Use a set of numerical values for the inputs and return a set of numerical values describing the gradient at those input values.

Symbol-to-symbol differentiation (Theano, Tensorflow): Add additional nodes to the graph that provide a symbolic description of the desired derivatives.

Because the derivatives are just another computational graph, it is possible to run back-propagation again, to obtain higher derivatives.
Figure from Goodfellow 2016
Regularization in Neural Networks

- Regularization is a way to overcome underfitting, overfitting issues by trading variance of the prediction error against bias.

\[ \mathbb{E}[L(\hat{y}, y)] = \text{Irreducible Error} + \text{Bias}^2 + \text{Variance} \quad \text{(exercise)} \]

- Regularization is a modification to a learning algorithm that is intended to reduce its generalization error but not its training error.

- We have already seen bagging as a regularization method.

- In the context of deep learning, most regularization strategies are based on regularizing estimators, by adding a parameter norm penalty \( \Omega(\theta) \) to \( J \)

\[ J(\theta; X, y) + \lambda \Omega(\theta) \]
weight decay

- weight decay refers to the $L^2$ penalty.
- also known as ridge regression
- if we do not punish the bias $b$ the objective function for weight decay is given by

$$\tilde{J}(w; X, y) = \frac{\lambda}{2} w^T w + J(w; X, y)$$

- this means in a single gradient update step the update changes to

$$w \leftarrow (1 - \varepsilon \lambda) w - \varepsilon \nabla_w J(w; X, y)$$

- the addition of the weight decay term has modified the learning rule to shrink the weight vector on each step
we make a quadratic approximation to the objective function in the neighborhood of the value $w^*$, the optimal weights where unregularized training cost is minimal

$$\hat{J}(w) = J(w^*) + \frac{1}{2}(w - w^*)^T H (w - w^*)$$

where $H$ is the Hessian matrix of $J$ with respect to $w$ evaluated at $w^*$

the minimum of the regularized version of $\tilde{J}$ is at

$$\tilde{w} = (H + \lambda I)^{-1} H w^*$$

If we decompose $H = Q \Lambda Q^T$ into a diagonal matrix $\Lambda$ and an orthonormal basis of eigenvectors $Q$ we get

$$\tilde{w} = Q (\Lambda + \lambda I)^{-1} \Lambda Q^T w^*$$
Figure from Goodfellow 2016
In comparison to $L^2$ regularization, $L^1$ regularization results in a solution that is more sparse.

Sparsity in this context refers to the fact that some weights have an optimal value of zero.
Let us have a look at the learning procedure at playground.tensorflow.org