What Do Neural Networks Do?
Multi-layer networks

Steve Renals

Machine Learning Practical — MLP Lecture 3
5 October 2016
What Do Single Layer Neural Networks Do?
Single-layer network, 1 output, 2 inputs

\[ x_1 \quad + \quad x_2 \]
Geometric interpretation

Single-layer network, 1 output, 2 inputs

\[ y(w; x) = 0 \]

Bishop, sec 3.1
Example data (three classes)
Single-layer network, 3 outputs, 2 inputs
Classification regions with single-layer network

Single-layer networks are limited to linear classification boundaries
Single layer network trained on MNIST Digits

Output weights define a “template” for each class
Hinton Diagrams

Visualise the weights for class $k$

400 (20x20) inputs
Hinton diagram for single layer network trained on MNIST

- Weights for each class act as a “discriminative template”
- Inner product of class weights and input to measure closeness to each template
- Classify to the closest template (maximum value output)
Multi-Layer Networks
Good classification needs to cope with the variability of real data: scale, skew, rotation, translation, ....

Very difficult to do with a single template per class

Could have multiple templates per task... this will work, but we can do better
From templates to features

Good classification needs to cope with the variability of real data: scale, skew, rotation, translation, ....

Very difficult to do with a single template per class

Could have multiple templates per task... this will work, but we can do better

Use features rather than templates

Incorporating features in neural network architecture

- Layered processing: inputs - features - classification

- How to obtain features - learning!
Incorporating features in neural network architecture

- Layered processing: inputs - features - classification

- How to obtain features - learning!
Incorporating features in neural network architecture

- Layered processing: inputs - features - classification

- How to obtain features - learning!
Multi-layer network

\[ y_k = \text{softmax} \left( \sum_{r=1}^{H} w_{kr}^{(2)} h_r^{(1)} + b_k \right) \]

\[ h_j^{(1)} = \text{sigmoid} \left( \sum_{s=1}^{d} w_{js}^{(1)} x_s + b_j \right) \]
Multi-layer network for MNIST

Hidden units make training the weights more complicated, since the hidden units affect the error function indirectly via all the outputs.

The Credit assignment problem: what is the “error” of a hidden unit? how important is input-hidden weight $w_{ji}^{(1)}$ to output unit $k$?

Solution: Gradient descent – requires derivatives of the error with respect to each weight.

Algorithm: back-propagation of error (backprop)

Backprop gives a way to compute the derivatives. These derivatives are used by an optimisation algorithm (e.g. gradient descent) to train the weights.
Training output weights

\[ y_1 \quad y_\ell \quad y_K \]

\[ h_j \]

\[ x_i \]

\[ w^{(1)}_{ji} \quad w^{(2)}_{1j} \quad w^{(2)}_{\ell j} \quad w^{(2)}_{Kj} \]

Outputs

Hidden units
Training output weights

\[
\begin{align*}
\delta_k^{(2)} &= \frac{\partial E^n}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} h_j^{(1)} \\
\end{align*}
\]
Training MLPs: Error function and required gradients

- Cross-entropy error function:
  \[ E^n = - \sum_{k=1}^{C} t^n_k \ln y^n_k \]

- Required gradients:
  \[ \frac{\partial E^n}{\partial w_{kj}^{(2)}} \quad \frac{\partial E^n}{\partial w_{ji}^{(1)}} \quad \frac{\partial E^n}{\partial b^{(2)}_k} \quad \frac{\partial E^n}{\partial b^{(1)}_j} \]

- Gradient for hidden-to-output weights similar to single-layer network:
  \[ \frac{\partial E^n}{\partial w_{kj}^{(2)}} = \frac{\partial E^n}{\partial a_k^{(2)}} \cdot \frac{\partial a_k^{(2)}}{\partial w_{kj}} = \left( \sum_{c=1}^{C} \frac{\partial E^n}{\partial y_c} \cdot \frac{\partial y_c}{\partial a_k^{(2)}} \right) \cdot \frac{\partial a_k^{(2)}}{\partial w_{kj}} \]

  \[ = (y_k - t_k) h_j^{(1)} \delta_k^{(2)} \]
Back-propagation of error: hidden unit error signal

\[ y_k = \sum_{j=1}^{K} w_{kj}(y_j - K_j) \]

\[ h_j = \sigma(x_i^T w_{ji}) \]

\[ E_n = \frac{1}{2} \sum_{k=1}^{K} (y_k - T_k)^2 \]

\[ \Delta w_{ji} = \eta \frac{\partial E_n}{\partial w_{ji}} \]

\[ \Delta w_{kj} = \eta \frac{\partial E_n}{\partial w_{kj}} \]
Back-propagation of error: hidden unit error signal

\[ \delta_j^{(1)} = \left( \sum_{\ell} \delta_{\ell}^{(2)} w_{\ell j} \right) h_j (1 - h_j) \]

\[ \frac{\partial E^n}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i \]
Training MLPs: Input-to-hidden weights

\[
\frac{\partial E^n}{\partial w^{(1)}_{ji}} = \frac{\partial E^n}{\partial a^{(1)}_j} \cdot \frac{\partial a^{(1)}_j}{\partial w^{(1)}_{ji}} \quad \delta^{(1)}_j \quad x_i
\]

To compute \(\delta^{(1)}_j = \frac{\partial E^n}{\partial a^{(1)}_j}\), the error signal for hidden unit \(j\), we must sum over all the output units’ contributions to \(\delta^{(1)}_j\):

\[
\delta^{(1)}_j = \sum_{c=1}^{K} \frac{\partial E^n}{\partial a^{(2)}_c} \cdot \frac{\partial a^{(2)}_c}{\partial a^{(1)}_j} = \left( \sum_{c=1}^{K} \delta^{(2)}_c \cdot \frac{\partial a^{(2)}_c}{\partial h^{(1)}_j} \right) \cdot \frac{\partial h^{(1)}_j}{\partial a^{(1)}_j} = \left( \sum_{c=1}^{K} \delta^{(2)}_c w^{(2)}_{cj} \right) h^{(1)}_j (1 - h^{(1)}_j)
\]
Training MLPs: Gradients

\[
\frac{\partial E^n}{\partial w_{kj}^{(2)}} = \left( y_k - t_k \right) \cdot h_j^{(1)} \delta_k^{(2)}
\]

\[
\frac{\partial E^n}{\partial w_{ji}^{(1)}} = \left( \sum_{c=1}^{k} \delta_c^{(2)} w_{cj}^{(2)} \right) h_j^{(1)} (1 - h_j^{(1)}) \cdot x_i \delta_j^{(1)}
\]

Exercise: write down expressions for the gradients w.r.t. the biases

\[
\frac{\partial E^n}{\partial b_k^{(2)}} \quad \frac{\partial E^n}{\partial b_j^{(1)}}
\]
Back-propagation of error: hidden unit error signal

\[ y_1 \quad y_\ell \quad y_K \]

\[ h_j \]

\[ x_i \]

\[ w_{1j}^{(2)} \quad w_{\ell j}^{(2)} \quad w_{Kj}^{(2)} \]

\[ w_{ji}^{(1)} \]
Back-propagation of error: hidden unit error signal

\[ \frac{\partial E^n}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} h_j^{(1)} \]

Outputs \( y^K \)

Hidden units

\[ h_j \]

\[ x_i \]

\[ \delta_1^{(2)} \]

\[ \delta_\ell^{(2)} \]

\[ \delta_K^{(2)} \]

\[ w_{1j}^{(2)} \]

\[ w_{\ell j}^{(2)} \]

\[ w_{kj}^{(2)} \]
Back-propagation of error: hidden unit error signal

\[ \delta_j^{(1)} = \sum_\ell \delta_\ell^{(2)} w_{\ell j} h_j (1 - h_j) \]

\[ \frac{\partial E^n}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i \]
The back-propagation of error algorithm is summarised as follows:

1. Apply an input vectors from the training set, \( x \), to the network and forward propagate to obtain the output vector \( y \)
2. Using the target vector \( t \) compute the error \( E^n \)
3. Evaluate the error signals \( \delta_k^{(2)} \) for each output unit
4. Evaluate the error signals \( \delta_j^{(1)} \) for each hidden unit using back-propagation of error
5. Evaluate the derivatives for each training pattern

Back-propagation can be extended to multiple hidden layers, in each case computing the \( \delta^{(\ell)} \)s for the current layer as a weighted sum of the \( \delta^{(\ell+1)} \)s of the next layer
Training with multiple hidden layers

\[ y_1, y_\ell, y_K \]

\[ h_1^{(2)}, h_2^{(2)}, \ldots, h_J^{(2)}, h_H^{(2)}, \ldots \]

\[ \delta_1^{(3)}, \delta_\ell^{(3)}, \delta_K^{(3)} \]

\[ w_{1k}^{(3)}, w_{\ell k}^{(3)}, w_{Kk}^{(3)} \]

\[ h_1^{(1)}, h_2^{(1)}, \ldots, h_j^{(1)}, \ldots \]

\[ \delta_1^{(2)}, \delta_2^{(2)}, \ldots, \delta_j^{(2)} \]

\[ w_{1j}^{(2)}, w_{kj}^{(2)}, w_{Hj}^{(2)}, \ldots \]

\[ x_i \]

\[ \delta_k^{(2)} = \left( \sum_m \delta_m^{(3)} w_{mk} \right) h_k^{(2)} (1 - h_k^{(2)}) \]
Summary

- Understanding what single-layer networks compute
- How multi-layer networks allow feature computation
- Training multi-layer networks using back-propagation of error
- Reading:
  Michael Nielsen, chapters 1 & 2 of *Neural Networks and Deep Learning*
  Chris Bishop, Sections 3.1, 3.2, and Chapter 4 of *Neural Networks for Pattern Recognition*