Artificial Neural Networks

Oliver Schulte
School of Computing Science
Simon Fraser University
Neural Networks

• Neural networks arise from attempts to model human/animal brains
  • Many models, many claims of biological plausibility
• We will focus on multi-layer perceptrons
  • Mathematical properties rather than biological plausibility
Uses of Neural Networks

- **Pros**
  - Good for continuous input variables.
  - General continuous function approximators.
  - Highly non-linear.
  - Learn features.
  - Good to use in continuous domains with little knowledge:
    - When you don’t know good features.
    - You don’t know the form of a good functional model.

- **Cons**
  - Not interpretable, “black box”.
  - Learning is slow.
  - Good generalization can require many datapoints.
Function Approximation Demos

- **Home Value of Hockey State** https://user-images.githubusercontent.com/22108101/28182140-eb64b49a-67bf-11e7-97aa-046298f721e5.jpg
- **Function Learning Examples (open in Safari)**
  http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.html
Applications

There are many, many applications.

- **World-Champion Backgammon Player.**
  
  
  http://en.wikipedia.org/wiki/Backgammon

- **No Hands Across America Tour.**
  

- **Digit Recognition with 99.26% accuracy.**

- **Speech Recognition**
  

- http://deeplearning.net/demos/
Outline

Feed-forward Networks

Network Training

Error Backpropagation

Applications
Outline

Feed-forward Networks

Network Training

Error Backpropagation

Applications
Neurons

• Starting with input \( x = (x_1, \ldots, x_D) \), construct linear combinations:

\[
a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}
\]

These \( a_j \) are known as activations

• Pass through an activation function \( h(\cdot) \) to get output \( z_j = h(a_j) \)
  • Model of an individual neuron

from Russell and Norvig, AIMA2e
Neurons

- Starting with input $x = (x_1, \ldots, x_D)$, construct linear combinations:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

These $a_j$ are known as activations
- Pass through an activation function $h(\cdot)$ to get output $z_j = h(a_j)$
  - Model of an individual neuron

from Russell and Norvig, AIMA2e
Neurons

- Starting with input $x = (x_1, \ldots, x_D)$, construct linear combinations:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

These $a_j$ are known as activations
- Pass through an activation function $h(\cdot)$ to get output $z_j = h(a_j)$
  - Model of an individual neuron

from Russell and Norvig, AIMA2e
Activation Functions

- Can use a variety of activation functions
  - Sigmoidal (S-shaped)
    - Logistic sigmoid $1/(1 + \exp(-a))$ (useful for binary classification)
    - Hyperbolic tangent $\tanh$
  - Radial basis function $z_j = \sum_i (x_i - w_{ji})^2$
  - Softmax
    - Useful for multi-class classification
  - Hard Threshold
  - Rectified Linear Unit (deep learning)
  - ...

- Should be differentiable for gradient-based learning (later)
- Can use different activation functions in each unit
Activation Functions Visualized

Left  Threshold
Middle  Logistic sigmoid $Logistic(x) = \frac{1}{1+\exp(-x)}$ (useful for binary classification)
Right  Logistic regression $Logistic(w \cdot x)$
Feed-forward Networks

- Connect together a number of these units into a feed-forward network (DAG)
- Above shows a network with one layer of hidden units
- Implements function:

\[
y_k(x, w) = h \left( \sum_{j=1}^{M} w_{kj}^{(2)} h \left( \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)
\]

No Hands Across America
A general network

During feedforward operation, a \(d\)-dimensional input pattern \(x\) is presented to the input layer; each input unit then emits its corresponding component \(x_i\). Each of the \(H\) hidden units computes its net activation, \(\text{net}_j\), as the inner product of the input layer signals with weights \(w_{ji}\) at the hidden unit. The hidden unit emits \(y_j = f(\text{net}_j)\), where \(f(\cdot)\) is the nonlinear activation function, shown here as a sigmoid. Each of the \(c\) output units functions in the same manner as the hidden units do, computing \(\text{net}_k\) as the inner product of the hidden unit signals and weights at the output unit. The final signals emitted by the network, \(z_k = f(\text{net}_k)\), are used as discriminant functions for classification.

During network training, these output signals are compared with a teaching or target vector \(t\), and any difference is used in training the weights throughout the network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, Pattern Classification. Copyright © 2001 by John Wiley & Sons, Inc.
Function Composition

Think logic circuits

Two opposite-facing sigmoids = ridge. Two ridges = bump.
The XOR Problem

The two-bit parity or exclusive-OR problem can be solved by a three-layer network. At the bottom is the two-dimensional feature space, along with the four patterns to be classified. The three-layer network is shown in the middle. The input units are linear and merely distribute their feature values through multiplicative weights to the hidden units. The hidden and output units here are linear threshold units, each of which forms the linear sum of its inputs times their associated weight to yield net, and emits a +1 if his net is greater than or equal to 0, and −1 otherwise, as shown by the graphs. Positive or “excitatory” weights are denoted by solid lines, negative or “inhibitory” weights by dashed lines; each weight magnitude is indicated by the line’s thickness, and is labeled. The single output unit sums the weighted signals from the hidden units and bias to form its net, and emits a +1 if its net is greater than or equal to 0 and emits a −1 otherwise. Within each unit we show a graph of its input-output or activation function—$f(\text{net})$ versus net. This function is linear for the inputs, a constant for the bias, and a step or sign function elsewhere. We say that this network has a 2-2-1 fully connected topology, describing the number of units (other than the bias) in successive layers. From: Richard O. Duda, Peter E. Hart, and David G. Stork, Pattern Classification. Copyright © 2001 by John Wiley & Sons, Inc.
The XOR Problem Solved

FIGURE 6.1. The two-bit parity or exclusive-OR problem can be solved by a three-layer network. At the bottom is the two-dimensional feature \( x_1 \) \( x_2 \)-space, along with the four patterns to be classified. The three-layer network is shown in the middle. The input units are linear and merely distribute their feature values through multiplicative weights to the hidden units. The hidden and output units here are linear threshold units, each of which forms the linear sum of its inputs times their associated weight to yield \( \text{net} \), and emits a \(+1\) if his \( \text{net} \) is greater than or equal to 0, and \(-1\) otherwise, as shown by the graphs. Positive or “excitatory” weights are denoted by solid lines, negative or “inhibitory” weights by dashed lines; each weight magnitude is indicated by the line’s thickness, and is labeled. The single output unit sums the weighted signals from the hidden units and bias to form its \( \text{net} \), and emits a \(+1\) if its \( \text{net} \) is greater than or equal to 0 and emits a \(-1\) otherwise. Within each unit we show a graph of its input-output or activation function—\( f(\text{net}) \) versus \( \text{net} \). This function is linear for the input units, a constant for the bias, and a step or sign function elsewhere. We say that this network has a 2-2-1 fully connected topology, describing the number of units (other than the bias) in successive layers. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.
Hidden Units Compute Basis Functions

- red dots = network function
- 3 dashed lines = 3 hidden unit activation functions.
- blue dots = data points + target values

Network function is roughly the sum of activation functions.
Hidden Units As Feature Extractors

sample training patterns

E E E E E E ...
L L L L L L L ...
F F F F F F ...

learned input-to-hidden weights

- 64 input nodes
- 2 hidden units
- 2x learned weight vector at hidden unit
Image Analysis Tasks

Classification

Retrieval

[Krizhevsky 2012]
Neural Net Learned Features

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Outline

Feed-forward Networks

Network Training

Error Backpropagation

Applications
Network Training

- Given a specified network structure, how do we set its parameters (weights)?
  - Define a criterion to measure how well our network performs, optimize against it
- For regression, training data are \((x_n, t), t_n \in \mathbb{R}\)
- Squared error naturally arises:

\[
E(w) = \sum_{n=1}^{N} \left( y(x_n, w) - t_n \right)^2
\]
Network Training

- Given a specified network structure, how do we set its parameters (weights)?
  - Define a criterion to measure how well our network performs, optimize against it
- For regression, training data are \((x_n, t), \ t_n \in \mathbb{R}\)
- Squared error naturally arises:

  \[
  E(w) = \sum_{n=1}^{N} \left\{ y(x_n, w) - t_n \right\}^2
  \]
For either of these problems, the error function $E(w)$ is nasty

- Nasty = non-convex
- Non-convex = has local minima
Descent Methods

- The typical strategy for optimization problems of this sort is a descent method:

\[ w^{(\tau+1)} = w^{(\tau)} + \eta w'(\tau) \]

- These come in many flavours
  - Gradient descent \( \nabla E(w^{(\tau)}) \)
  - Stochastic gradient descent \( \nabla E_n(w^{(\tau)}) \)
  - Newton-Raphson (second order) \( \nabla^2 \)

- All of these can be used here, stochastic gradient descent is particularly effective
  - Redundancy in training data, escaping local minima
The typical strategy for optimization problems of this sort is a descent method:

\[ w^{(\tau+1)} = w^{(\tau)} + \eta w'(\tau) \]

These come in many flavours
- Gradient descent \( \nabla E(w^{(\tau)}) \)
- Stochastic gradient descent \( \nabla E_n(w^{(\tau)}) \)
- Newton-Raphson (second order) \( \nabla^2 \)

All of these can be used here, stochastic gradient descent is particularly effective
- Redundancy in training data, escaping local minima
Descent Methods

• The typical strategy for optimization problems of this sort is a descent method:

\[ w^{(\tau+1)} = w^{(\tau)} + \eta w'(\tau) \]

• These come in many flavours
  • Gradient descent \( \nabla E(w^{(\tau)}) \)
  • Stochastic gradient descent \( \nabla E_n(w^{(\tau)}) \)
  • Newton-Raphson (second order) \( \nabla^2 \)

• All of these can be used here, stochastic gradient descent is particularly effective
  • Redundancy in training data, escaping local minima
Computing Gradients

- The function $y(x_n, w)$ implemented by a network is complicated
- It isn’t obvious how to compute error function derivatives with respect to hidden weights.
- The credit assignment problem.
Outline

Feed-forward Networks

Network Training

Error Backpropagation

Applications
Error Backpropagation

- Backprop is an efficient method for computing error derivatives $\frac{\partial E_n}{\partial w_{ji}}$ for all nodes in the network. Intuition:
  1. Calculating derivatives for weights connected to output nodes is easy.
  2. Treat the derivatives as virtual error—how far is each node activation “off”. Compute derivative of error for nodes in previous layer.
  3. Repeat until you reach input nodes.
- Propagates backwards the output error signal through the network.
Functional Picture for NN

Keep in mind this picture.
Error at the output nodes

- First, feed training example \( x_n \) forward through the network, storing all activations \( a_j \).
- For output node with activation \( y_k = g(a_k) \) and target \( t_n \), define
  \[
  \delta_k \equiv (t_n - y_k) g'(a_k).
  \]
- 0 if no error or if input \( z_j \) from node \( j \) is 0.
- error signal at output node = output error \( \times \) how much node output depends on node input.
Error at the output nodes

- First, feed training example $x_n$ forward through the network, storing all activations $a_j$.
- For output node with activation $y_k = g(a_k)$ and target $t_n$, define
  \[ \delta_k \equiv (t_n - y_k)g'(a_k). \]
- 0 if no error or if input $z_j$ from node $j$ is 0.
- error signal at output node = output error $\times$ how much node output depends on node input.
Error at the hidden nodes

- Consider a hidden node $j$ connected to other nodes $k$ in the next layer.
- Virtual error signal

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k.$$ 

- Read: error signal at node $j$ = weighted errors at following nodes $\times$ how much output of node $j$ depends on the input to node $j$
The error signal at a hidden unit is proportional to the error signals at the units it influences:

\[
\delta_j = g'(a_j) \times \sum_k w_{kj} \delta_k
\]
The Backpropagation Algorithm

1. Apply input vector \( x_n \) and forward propagate to find all activation levels \( a_i \) and output levels \( z_i \).

2. Evaluate the error signals \( \delta_k \) for all output nodes.

3. Backpropagate the \( \delta_k \) to obtain error signals \( \delta_j \) for each hidden node.

4. Perform the gradient descent updates for each weight \( w_{ji} \):

\[
w_{ji} \leftarrow w_{ji} + \eta \delta_j z_i.
\]

5. Read: weight increment =
output of source node \( \times \) error at end node

Other Learning Topics

- Regularization: L2-regularizer (weight decay).
- Experimenting with Network Architectures is often key.
- Learn Architecture
  - Prune Weights: the Optimal Brain Damage Method.
  - Grow Network: Tiling, Cascade-Correlation Algorithm.
Outline

Feed-forward Networks

Network Training

Error Backpropagation

Applications
Applications of Neural Networks

- Many success stories for neural networks
  - Credit card fraud detection
  - Hand-written digit recognition
  - Face detection
  - Autonomous driving (CMU ALVINN)
Hand-written Digit Recognition

- MNIST - standard dataset for hand-written digit recognition
  - 60000 training, 10000 test images
LeNet-5

- LeNet developed by Yann LeCun et al.
  - Convolutional neural network
    - Local receptive fields (5x5 connectivity)
    - Subsampling (2x2)
    - Shared weights (reuse same 5x5 “filter”)
    - Breaking symmetry

- See

- Also deep learning tutorial.
• The 82 errors made by LeNet5 (0.82% test error rate)
Conclusion

- Feed-forward networks can be used for predicting discrete or continuous target variables.
- Very expressive, can approximate arbitrary continuous functions.
- Different activation functions possible.
- Learning is more difficult, error function not convex.
  - Use stochastic gradient descent, obtain (good?) local minimum.
- Backpropagation for efficient gradient computation.