Neural Networks

Part 1

Jon Dehdari

January 28, 2016

Extending Logistic Regression (=Softmax Regression)

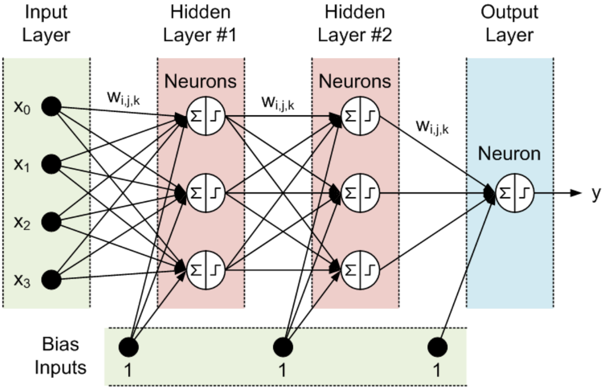
- Recall that **logistic regression** involves the dot product of an input vector and a weight matrix, then a normalized sigmoid function (softmax)

Extending Logistic Regression (=Softmax Regression)

- Recall that **logistic regression** involves the dot product of an input vector and a weight matrix, then a normalized sigmoid function (softmax)
- A **feedforward neural network** just adds one or more layers between the input vector and the softmax output

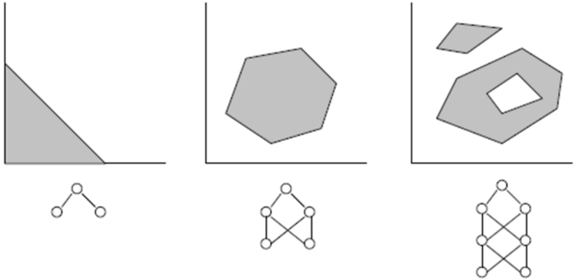
Extending Logistic Regression (=Softmax Regression)

- Recall that **logistic regression** involves the dot product of an input vector and a weight matrix, then a normalized sigmoid function (softmax)
- A **feedforward neural network** just adds one or more layers between the input vector and the softmax output



Why Use Hidden Layers?

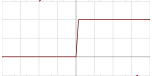
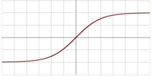
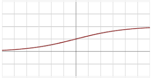
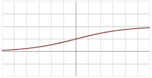
- In contrast to log-linear models, neural networks can have **non-linear** representations of data
- The **universal approximation theorem** (George Cybenko, 1989) found that a neural network with one hidden layer can approximate **any continous function**
- A network with two hidden layers can represent discontinuous functions



Courtesy of mql5.com

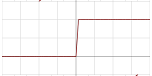
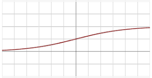
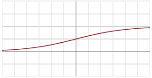
Activation Functions (σ)

In each layer, the output of the dot product goes through an **activation function** (σ). Here are some examples:

Name	**Visualization**	$\mathbf{f(x)} =$	**Notes**
Linear (= Identity)		x	Not useful for hidden layers
Heaviside Step		$\begin{cases} 0 & \text{if} \quad x < 0 \\ 1 & \text{if} \quad x \geq 0 \end{cases}$	Not differentiable
Rectified Linear (ReLU)		$\begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad x \geq 0 \end{cases}$	Surprisingly useful in practice
Tanh		$\frac{2}{1+e^{-2x}} - 1$	A soft step function; ranges from -1 to 1
Logistic ('sigmoid')		$\frac{1}{1+e^{-x}}$	Another soft step function; ranges from 0 to 1
Softmax		$\frac{e^{W_y \cdot \mathbf{x}}}{Z}$	Normalized sigmoidal function. Useful for last layer when training on cross entropy

Activation Functions (σ)

In each layer, the output of the dot product goes through an **activation function** (σ). Here are some examples:

Name	**Visualization**	$\mathbf{f(x)} =$	**Notes**
Linear (= Identity)		x	Not useful for hidden layers
Heaviside Step		$\begin{cases} 0 & \text{if} \quad x < 0 \\ 1 & \text{if} \quad x \geq 0 \end{cases}$	Not differentiable
Rectified Linear (ReLU)		$\begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad x \geq 0 \end{cases}$	Surprisingly useful in practice
Tanh		$\frac{2}{1+e^{-2x}} - 1$	A soft step function; ranges from -1 to 1
Logistic ('sigmoid')		$\frac{1}{1+e^{-x}}$	Another soft step function; ranges from 0 to 1
Softmax		$\frac{e^{\mathbf{w}_y \cdot \mathbf{x}}}{Z}$	Normalized sigmoidal function. Useful for last layer when training on cross entropy

List of activation functions in Keras: `keras.io/activations`

Images courtesy of Wikipedia

Training Neural Networks

- At a high level, the weights in a neural net are set by means of the blame game – whenever it guesses incorrectly, change the weights that were the most responsible for making that guess

Training Neural Networks

- At a high level, the weights in a neural net are set by means of the blame game – whenever it guesses incorrectly, change the weights that were the most responsible for making that guess
- Whenever the network guesses a training instance correctly, don't change anything

Training Neural Networks

- At a high level, the weights in a neural net are set by means of the blame game – whenever it guesses incorrectly, change the weights that were the most responsible for making that guess
- Whenever the network guesses a training instance correctly, don't change anything
- The weights are usually trained by a form of the gradient descent optimization algorithm
- The gradients are calculated by error **backpropagation**
- First, do a normal forward pass through the network, to determine the **error/loss** (how different the output was from the 'correct' answer)
- Then, do a backwards pass (end to start), changing the weights to minimize errors

Loss / Objective Functions

- **Discrete Outputs**:
  - Binary Cross-Entropy (0-1 loss): 0 if correct, 1 if incorrect
  - Categorical Cross-Entropy: good old cross-entropy. Eg.
    0 if $p(y) = 1.0$,
    1 if $p(y) = 0.5$,
    2 if $p(y) = 0.25$,
    3 if $p(y) = 0.125$,
    ...
- **Continuous Outputs**:
  - Mean Squared Error (MSE): $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
  - Root Mean Squared Error (RMSE): $\sqrt{MSE}$
  - Mean Absolute Error (MAE): $\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$

Loss / Objective Functions

- **Discrete Outputs**:
  - Binary Cross-Entropy (0-1 loss): 0 if correct, 1 if incorrect
  - Categorical Cross-Entropy: good old cross-entropy. Eg.
    0 if $p(y) = 1.0$,
    1 if $p(y) = 0.5$,
    2 if $p(y) = 0.25$,
    3 if $p(y) = 0.125$,
    ...
- **Continuous Outputs**:
  - Mean Squared Error (MSE): $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
  - Root Mean Squared Error (RMSE): $\sqrt{MSE}$
  - Mean Absolute Error (MAE): $\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$

List of loss functions in Keras: `keras.io/objectives`

Autoencoders

- An **autoencoder** is a neural network where the size of the output layer is the same size as the input layer
- The hidden layers are usually smaller
- The goal is to generalize the training data
- Since no labeled data is necessary, autoencoders are an unsupervised learning technique
- Autoencoders trained on language data are neural language models

Autoencoders

- An **autoencoder** is a neural network where the size of the output layer is the same size as the input layer
- The hidden layers are usually smaller
- The goal is to generalize the training data
- Since no labeled data is necessary, autoencoders are an unsupervised learning technique
- Autoencoders trained on language data are neural language models
- Autoencoders are occasionally called diabolo networks

Image courtesy of Wikimedia

Tips & Tricks (discussed in class)

- Network depth
- Layer size
- Dropout
- Early stopping
- Optimizers
- Learning rate

Software

- Most popular neural net software are based on the following:

Name	Lang Support	GPU Support	Who
Theano	Python	Yes	Uni Montreal
TensorFlow	Python, C++	Yes	Google
Torch	Lua	Yes	FB, Twitter, etc.
DL4J	Java, Scala	Yes	Skymind.io
CNTK	C++	Yes	Microsoft

Software

- Most popular neural net software are based on the following:

Name	Lang Support	GPU Support	Who
Theano	Python	Yes	Uni Montreal
TensorFlow	Python, C++	Yes	Google
Torch	Lua	Yes	FB, Twitter, etc.
DL4J	Java, Scala	Yes	Skymind.io
CNTK	C++	Yes	Microsoft

- Many others: Caffe, MXNet, CNN

Software

- Most popular neural net software are based on the following:

Name	Lang Support	GPU Support	Who
Theano	Python	Yes	Uni Montreal
TensorFlow	Python, C++	Yes	Google
Torch	Lua	Yes	FB, Twitter, etc.
DL4J	Java, Scala	Yes	Skymind.io
CNTK	C++	Yes	Microsoft

- Many others: Caffe, MXNet, CNN
- We'll use Keras (keras.io), which is really easy and intuitive. It can use either Theano or TensorFlow as a backend.