CogSci 109: Lecture 23

Mon Dec 2, 2007

Multilayer artificial neural networks, examples, and applications (II)

- Announcements
- Homework announcement
- Instead of a threshold, we can consider other activation functions
  - **Threshold, sigmoid, linear, etc**
  - **Fitting arbitrary functions**
- Multilayer neural networks
  - **Some of the typical network topologies**

- Methods of training networks
  - **What is Supervised learning**
  - **What is Unsupervised learning**
  - **What is Reinforcement learning**
- Matlab neural network toolbox demos
- Potential issues with training networks
  - **Overfitting and generalization**
- Methods of dealing with these issues

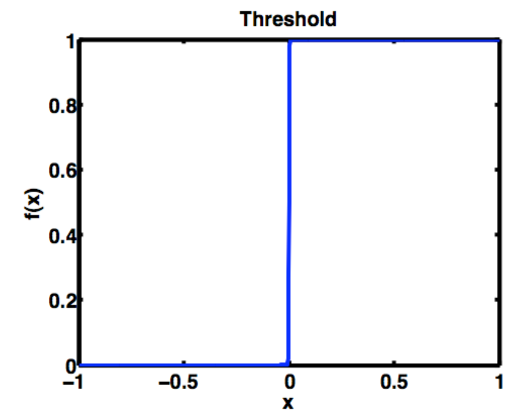
Announcements

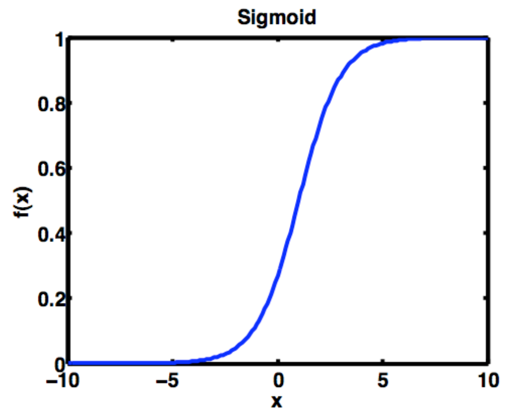
- Homework 6
- Grades online updated - please check again
  - **Blank midterm short answer sections**
    - Two people no names on tests!!!
  - **Student who may not be registered**
  - **About grade changes/late midterms/homeworks 0's**

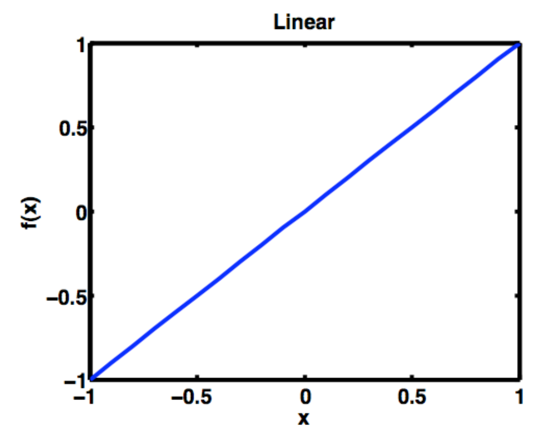
- About the final
  - **Takehome portion like a homework, worth 200 points**
    - Due Saturday at 12 midnight at the end of finals week
  - **In class portion multiple choice, like the midterm mult choice, but more questions**
  - **Practice final posted later this week, probably Wed**
    - With solutions
  - **Cumulative, but will focus on material since midterm**
  - **Bring 3 double sided pages of notes, handwritten**
  - **Bring calculator**
  - **Bring red scantron**
  - **Bring plenty of pencils and erasers**

Other activation function concepts

- Threshold
- Sigmoid
- Gaussian
- Hyperbolic tangent
- Sine
- Unit sum
- Square root
- Logistic
- Softmax
- Linear
- Many others







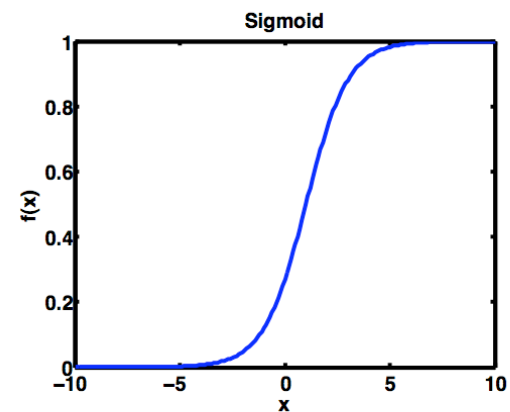
So you can have any shape activation function

- Not just threshold
- Allows you to create real-valued outputs

$$y = f\left(\sum_{i=0}^{n} w_i x_i + b\right)$$

Sigmoid

- Equation
  - **Mentioned last time**
  - **Matlab - SIGMF()**
  - **Vary parameters b and c to control the steepness of the transition from 0-1**
  - **Saturates to 0 as x->-inf, and 1 as x->+inf**

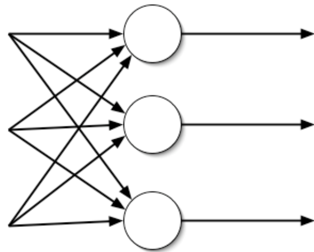


$$y = \frac{1}{1 + e^{-(bx-c)}}$$

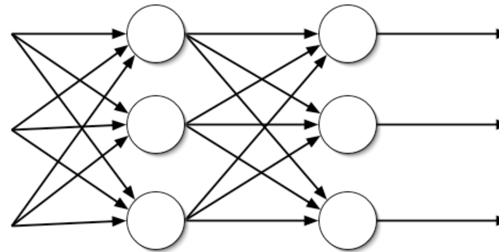
- Networks of neurons with real-valued inputs and sigmoid activation functions can be used to approximate mathematical functions
  - **Any continuous real-valued function can be approximated to arbitrary accuracy with a feedforward network of at least one hidden layer**
  - **Matlab demo -> *nnd11fa***

Some typical network topologies

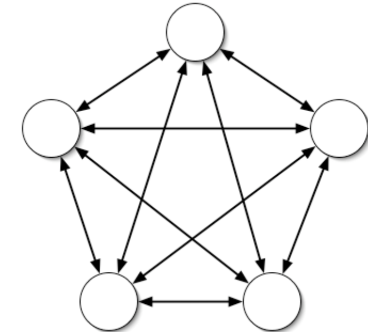
Single layer perceptron



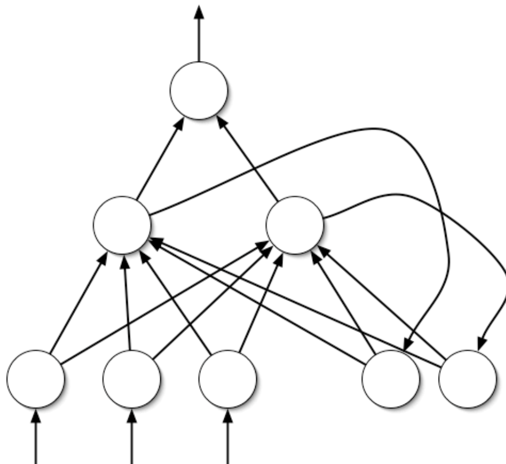
Multi-layer perceptron



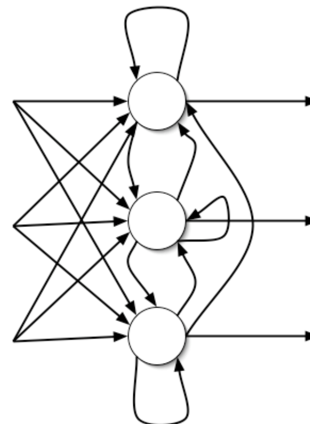
Hopfield network



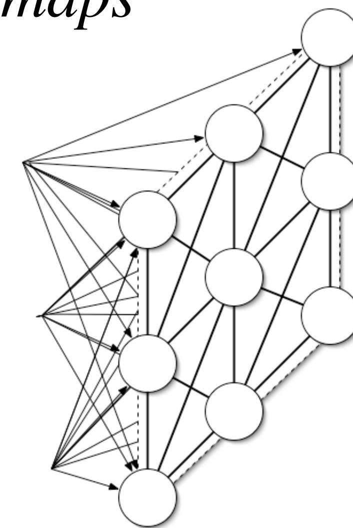
Elman recurrent network

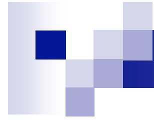


Competitive networks

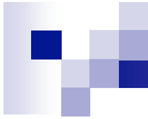


Self-organizing maps





So now we have these fancy networks, how can we get them to ‘learn?’



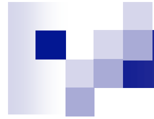
Methods of training networks

- Generally boils down to three learning strategies
  - **Supervised learning**
  - **Unsupervised learning**
  - **Reinforcement learning**
- Many methods with variants, but basically all fall under these above categories

Supervised learning

- *Method of learning whereby an error value is generated from the actual response of the network and the desired response. Following that, the weights are then modified such that the error is gradually reduced*
- **Training set -** A set of known input/output pairs is presented to the network in order to appropriately adjust the weights to produce the desired output given a certain input
- We already saw one example in the perceptron learning algorithm
- We will discuss backpropagation today

- There is still an input/output relationship but no feedback is provided indicating whether network's associations are correct or not
- The network must discover by itself similarities in the patterns of the data
  - **Self-organizing networks** - ***networks that possess the ability to to infer patterns from input-only data***



Reinforcement learning

- Input/output data and a teaching signal
  - **The teaching signal is not a measure of the error, rather an indication of the result as ‘right’ or ‘wrong’ direction**

Neural Network Demos in matlab

- In matlab (you need the Neural Network Toolbox)
  - **nnd2n1 One-input neuron demonstration.**
  - **nnd2n2 Two-input neuron demonstration.**
  - **nnd4db Decision boundaries demonstration.**
  - **nnd4pr Perceptron rule demonstration.**
  - **nnd9sdq Steepest descent for quadratic function demonstration.**
  - **nnd11nf Network function demonstration.**
  - **nnd11bc Backpropagation calculation demonstration**
  - **nnd11fa Function approximation demonstration.**
  - **nnd11gn Generalization demonstration.**

Back propagation algorithms

- General algorithm
  - **Present inputs**
  - **Propagate network responses forward**
  - **Compute the error between output and desired output**
  - **Back-propagate deltas**
  - **Update weights**
  - **Repeat for next pattern**
- Matlab demo **- nnd11bc**

Specifically

1. Initialize weights randomly
2. Present an input vector pattern to the network
3. Evaluate the outputs of the network by propagating signals forwards
4. For all output neurons, calculate $\delta_j = (y_j - d_j)$
   1. **d_j is desired output of neuron j and y_j is current output**

$$y = f\Big(\sum_{i=0}^{n} w_i x_i + b\Big) = \Big(1 + exp(-(\sum_{i=0}^{n} w_i x_i + b))\Big)^{-1}$$

5. For all other neurons compute delta

$$\delta_j = \sum_k w_{jk} g'(x) \delta_k$$

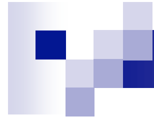
   1. **Where delta_k is the delta_j of succeeding layer, and**

$$g'(x) = y_k(1 - y_k)$$

6. Update weights according to

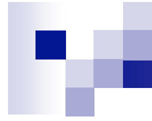
$$w_{ij}(t+1) = w_j(t) - \eta y_i y_j (1 - y_j) \delta_j$$

7. Goto 2 until iterationmax or minimal error



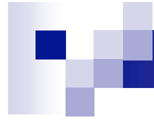
NN matlab demos and commands

- Simple function fit
- Classification



Potential issues to deal with when training neural networks

- Over-fitting
- Generalization
- We want to reduce over-fitting and increase generalization of our fits



Techniques to Prevent Overfitting

- Regularization
  - **Reduction of hidden units**
    - Only fit simpler functions
  - **Weight decay**
- Early stopping
  - **Using validation sets**
- Bayesian regularization
  - **(see the MacKay Book)**

Technique 1: Reduce number of layers to prevent overfitting

- *Note: Remember that overfitting is a problem when fitting many parameters to small amounts of data*
  - ***Infinite data would be then no problem***
- Simplify the function you are fitting by reducing the number of network hidden layers - similar to using a lower degree polynomial to fit data
  - **Limits the capability of your network**
- But ahead of time we may not know the complexity of the function we want to fit, so how do we deal with this?

Technique 2: Regularization to prevent overfitting

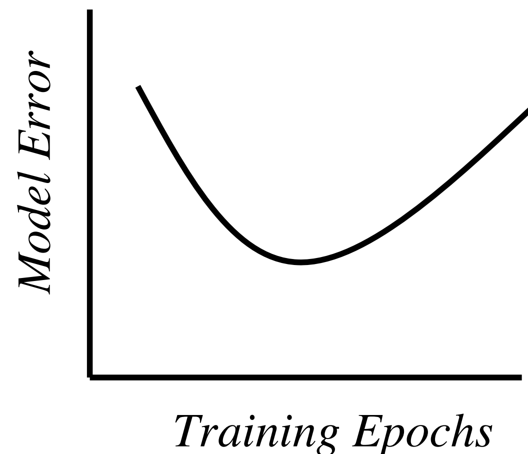
- ***Regularization*** - adding a penalty to the usual error function to encourage smoothness

$$E_{new} = E + \nu * \omega$$

- Here ν is the regularization parameter and ω is the smoothness penalty
- Weight decay sets $\omega = \frac{1}{2}\sum_i w_i^2$
  - **Note that when you then take the partial derivative of E_{new} with respect to a weight the update rule will now include a term that is -w_i.**
  - **This will encourage the weights to decay to zero (hence the name)**

Technique 3: Early stopping to prevent overfitting

- Start the weights very small
  - **Then the neural network starts by behaving fairly linearly**
  - **The weights gradually increase to handle nonlinearities**
- Split the data into a **validation set** and a **training set**
  - **Use the *training set* to adjust the weights**
  - **Use the *validation set* to compute model error**
  - **As the fit improves the error will decrease, when the error starts to increase again, you are fitting the noise in the training set**



- The Bayesian neural network formalism of David MacKay and Radford Neal, considers neural networks not as single networks but as distributions over weights (and biases)
- The output of a trained network is thus not the result of applying one set of weights but an average over the outputs from the distribution.
- This can be computationally expensive but MacKay and Neal have developed approximations and the approach leads to automatic regularization that is very effective.

More training issues

- Improvements on gradient descent
  - **Gradient descent with momentum**
  - ***Conjugate gradient***
  - **Variable learning rate**
  - **For nonquadratic functions, minimization (ie Nelder Mead, golden section line search, Brent's method, etc - See numerical methods book)**
    - Demos:
      - **nnd12sd1**
      - **nnd12sd2**
      - **nnd12mo**
      - **nnd12vl**
      - **nnd12ls**
      - **nnd12cg**