Neural Networks

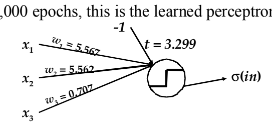
Louis Oliphant
oliphant@cs.wisc.edu
slides borrowed from Burr H. Settles
CS-540-2, UW-Madison
www.cs.wisc.edu/~cs540-2

Announcements

- Midterm
  - average: 74.0
  - std: 10.6
- HW 4
  - get busy, due next Thursday
  - blank lines, multi-character feature values
- Projects
  - proposals in, progress report next week

Training Example Results

- After 1,000 epochs, this is the learned perceptron:



Predictions on the training data:

$x = 001$ $f(x) = 0$ $\sigma(in) = 0.070$	$x = 111$ $f(x) = 1$ $\sigma(in) = 1.000$
$x = 110$ $f(x) = 1$ $\sigma(in) = 1.000$	$x = 101$ $f(x) = 1$ $\sigma(in) = 0.951$
$x = 000$ $f(x) = 0$ $\sigma(in) = 0.030$	$x = 011$ $f(x) = 1$ $\sigma(in) = 0.951$

And on some novel examples:

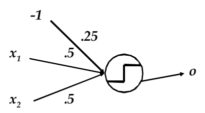
$x = 010$ $f(x) = ?$ $\sigma(in) = 0.906$	$x = 100$ $f(x) = ?$ $\sigma(in) = 0.906$

Any ideas what function this might be?

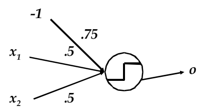
Perceptrons and Logic

- Perceptrons can learn logical functions:

$x_1 \vee x_2$



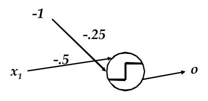
$x_1 \wedge x_2$



Perceptrons and Logic

- Perceptrons can learn logical functions:

$\neg x_1$



$x_1 \otimes x_2$

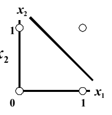
A perceptron cannot represent the XOR function! Why not??

Linear Separability

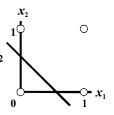
- Consider a perceptron with two inputs and and a threshold (bias):
  - The perceptron fires if $w_1x_1 + w_2x_2 - t \geq 0$
  - Recall the weights for the “and” perceptron:
    - $0.5x_1 + 0.5x_2 - 0.75 \geq 0$
  - This is really the equation for a line!
    - $x_2 \geq - x_1 + 1.5$ (in slope-intercept form)
- The activation threshold for a perceptron is actually a linearly separable “hyperplane” in the space of inputs

Linear Separability

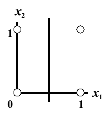
$x_1 \wedge x_2$



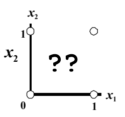
$x_1 \vee x_2$



$\neg x_1$

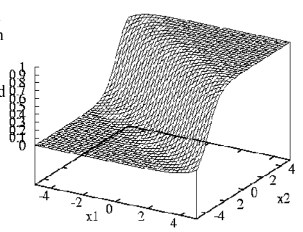


$x_1 \otimes x_2$



Linear Separability

Using the sigmoid activation function achieves the same general effect, sliding the sigmoid surface across the linear hyperplane…



The Need for a Network

- Clearly a single perceptron is limited compared to the expressiveness of a decision tree or *k*-NN
  - They can handle XOR, for example
- But remember: the brain is a network of neurons: the axon (output) is connected to the dendrites (inputs) of others through synapses (weights)
- By this analogy, we can create a multi-layer feed-forward neural network made up of perceptrons, which might learn more expressive functions

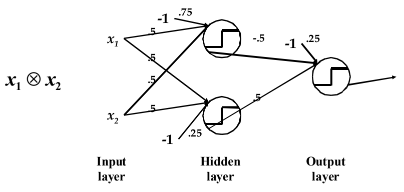
Multi-Layer Networks

- The structure of a multi-layer network is fairly straightforward:
  - The input layer is the set of features (percepts)
  - Next is a hidden layer, which has an arbitrary number of perceptrons called hidden units that take the features (input layer) as inputs
  - The perceptron(s) in the output layer then takes the outputs of the hidden units as its inputs

∗ *This is looking more like a model of the brain!*

Multi-Layer Networks

- Here is a very simple multi-layer network that can handle the XOR function:

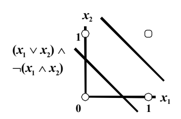


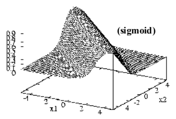
Multi-Layer Networks

- This network is essentially equivalent to a more complex logical function:

$$(x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2)$$

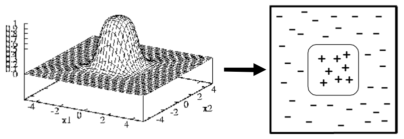
- Which, represented graphically, is:





Multi-Layer Networks

- We aren’t limited to just *one* layer of hidden units, though, we could have even more, which will allow us to learn even more complex functions:

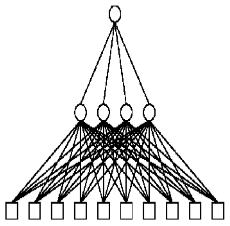


Multi-Layer Networks

Multi-layer networks are called feed forward because the information is fed forward from the input layer (features) toward the output layer

For most problems, one layer of hidden units is sufficient

Such networks are also usually fully connected: every output from one layer is connected to every input of the next (but they don’t need to be)



Training Multi-Layer Networks

- Training multi-layer networks can be a bit complicated (the weight space is larger!)
  - The perceptron rule worked fine for a single unit that mapped input features to the final output value
  - But hidden units don’t produce the final output
  - Output unit(s) take other perceptrons – not known feature values – as inputs
- The solution is to use the back-propagation algorithm, which is an intuitive extension of the perceptron training algorithm

Back-Propagation (BP)

- BP generalizes the perceptron rule:
  - Gradient-descent search to minimize error on the training data (again, usually in iterative mode)
  - In the forward pass, features are fed forward to the output layer where error is calculated
  - Then, in the backward pass:
    - Update weights from hidden layer to output layer as usual: $\Delta w_{ho} = \alpha \times x_h \times ERR_o$; where $ERR_o = g'(in_o) \times (true - g(in_o))$
    - Update weights from input layer to the hidden layer: $\Delta w_{ih} = \alpha \times x_i \times ERR_h$; where $ERR_h = g'(in_h) \times \Sigma_o (w_{ho} \times ERR_o)$
- More complete version of the algorithm on p.746 of *AIMA*

Back-propagation Algorithm

```
Back-propagation(training-examples, $\alpha$, $n_{in}$, $n_{out}$, $n_{hidden}$)
Each training example is a pair of the form <x,t>, where x is the vector of network input values, and t is the vector of target network output values.
$\alpha$ is the learning rate (e.g., 0.05). $n_{in}$ is the number of network inputs, $n_{hidden}$ the number of units in the hidden layer, and $n_{out}$ the number of output units.
The input from unit i into unit j is denoted $x_{ji}$, and the weight from unit i to unit j is denoted $w_{ji}$.
Create a feed-forward network with $n_{in}$ inputs, $n_{hidden}$ hidden units, and $n_{out}$ output units.
Initialize all network weights to small random numbers (e.g., between -0.05 and 0.05).
Until the termination condition is met, do:
    For each <x,t> in training_examples, do:
        Propagate the input forward through the network:
            1. Input the instance x to the network and compute the output $o_u$ of every
               unit u in the network
        Propagate the errors backward through the network:
            2. For each network output unit k, calculate its error term $\delta_k$
                $\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$
            3. For each hidden unit h, calculate its error term $\delta_h$
                $\delta_h \leftarrow o_h(1-o_h) \sum_{k \in outputs} (w_{kh}\delta_k)$
            4. Update each network weight $w_{ji}$
                $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
                where $\Delta w_{ji} = \alpha\delta_{hj}x_{ji}$
```

17

Problems with BP

- Because BP is a gradient descent (hill-climbing) search, it suffers from the same problems:
  - Doesn't necessarily find the globally best weight vector
    - Convergence is determined by the starting point (randomly initialized weights)
    - If α is set too large, can "bounce" right over the global minimum into a local minimum
- To deal with these problems:
  - Usually initialize weights close to 0
  - Can repeat training with multiple random restarts

18

Non-Boolean Features

- So far, the networks we've described only take Boolean features [1,0] as inputs
- To hand discrete-valued features, we can create a unique input for each feature-value pair
  - e.g. Outlook = {Sunny, Overcast, Rainy} would be converted into 3 Boolean inputs
  - For classification purposes, the observed value's input is set to 1, the others are 0
- Continuous features can be left alone and fed through the network as real numbers
  - The weights will figure out what to do with them!

19

Handling Multiple Labels

- Similarly, the networks so far have been for Boolean classification functions
- To handle multi-label classification tasks we can simply create extra output units:
  - Each unit corresponds to one label (e.g. animal/vegetable/mineral)
  - For classification, the unit with the highest output is the "winner," and the network assigns the corresponding label to that example
  - For training purposes, the "true" label's output unit is set to 1, and the others are set to 0

✱ *This variant on networks has been applied with wide success to several multi-class problems*

20

Regression and Neural Nets

- Are neural networks very well suited for regression (i.e. real-valued function) tasks?
- We currently use the σ function to allow the perceptrons to behave like classifiers, but we *could* just output the weighted sum of their inputs directly
  - Since this is a linear function, the *g'(in)* factor in training is 1

∗ *Neural networks can learn regression problems better than decision trees or* k*-nearest neighbors (though training can be slower than a standard network)*

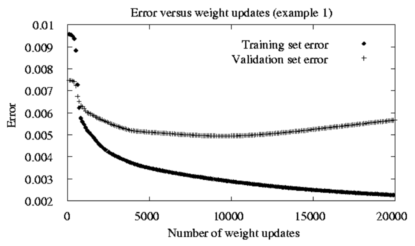
Expressiveness of Neural Nets

- Classification problems
  - Any Boolean function can be represented in a neural network with just 2 layers
  - But might require an exponential number of hidden units (in terms of input features)
- Regression problems
  - Perceptrons can learn any linear function
  - G. Cybenko, "Continuous valued neural networks with two hidden layers are sufficient," Tech. Report, 1988
    - 2-layer networks with enough hidden units can learn any bounded, continuous (e.g. polynomial) function
    - 3-layer networks can learn any function. Period.

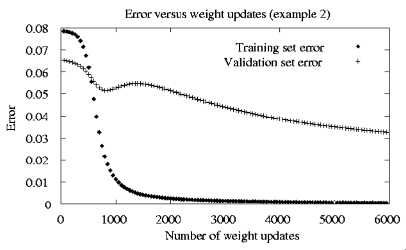
Overfitting in Neural Nets

- Overfitting is when a model that generalizes poorly to new data despite excellent performance on training data.
- As with all machine learning algorithms, there is a risk of overfitting the training data
  - Neural nets with lots of hidden units are particularly prone to overfit, because the model is so expressive!
- Recall that the network ultimately "converges," within some ε of change
  - If we keep cycling through epochs ad infinitum, we end up memorizing the training examples
- But how do we know when to stop?

Overfitting in Neural Nets



Overfitting in Neural Nets



Overfitting in Neural Nets

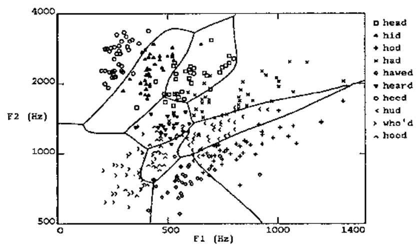
- We can use a tuning set to avoid overfitting in neural nets
  - We can train several candidate structures and use the tuning set to find one that's appropriately expressive
- More common: given a network structure, use early stopping by evaluating the network on the tuning set after each epoch
  - Stop when performance begins to dip on the tuning set
  - Sometimes allow a fixed number of epochs beyond the dip… just in case it goes back up

Neural Network Applications

- W. Huang & R. Lippmann, "Neural net and traditional classifiers," *Neural Information Processing Systems*, 1988

- Used a simple network to disambiguate between vowel phoneme sounds in "h — d" words
  - Only 2 features (percepts) obtained from spectral analysis of recorded data
  - 7 hidden units in 1 layer (fully connected)
  - 10 outputs (words: *head*, *hid*, *who'd*, *hood*, etc…)

Neural Network Applications



Neural Network Applications

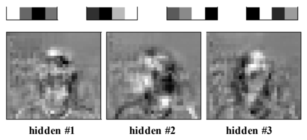
- Facial pose recognition
  - 🗎 Section 4.7 of *Machine Learning* by T. Michell
    - All this image data is available at **www.cs.cmu.edu/~tom/faces.html**
- Used 30×32 pixel images of faces pointing left, straight, right, and up
  - Each of the 30×32 = 960 pixels was a continuous feature (grayscale value)
  - 3 hidden units
  - 4 output units (poses)

Neural Network Applications

Example images:

Network weights (dark–, light+) after 100 epochs of training on 260 examples

Achieved 90% accuracy



Neural Network Applications

🗎 D. Pomerleau, “Knowledge-based training of artificial neural networks for autonomous robot driving,” *Robot Learning*, 1993

- That’s right, folks: ALVINN (Autonomous Land Vehicle In a Neural Network)
  - Takes 30×32 pixel input camera images
  - Only 1 hidden layer: 4 hidden units
  - 30 output units (discrete steering wheel direction)

Issues with Neural Nets

- Neural networks are very powerful and can approximate any function, but they do have drawbacks
  - Many weights to learn: training can take a while
  - Sensitive to structure, initial weights, and learning rate
- Once the network has learned a hypothesis function, *what does it mean?*
  - Neural networks are connectionist models, thus not as comprehensible as decision trees, which are symbolic

Understanding Neural Nets

- M. Craven & J. Shavlik, "Extracting Tree-Structured Representations of Trained Networks," *Advances in Neural Information Processing Systems*, MIT Press, 1996
  - "Trepan" Algorithm: extract decision trees from Neural Nets to better understand what they learned

Problem Dataset	% Accuracy			Fidelity to NN
	Network	ID2/3	Trepan	
Heart	84.5	74.6	81.8	94.1
Promoters	90.6	83.5	87.6	85.7
Protein-coding	94.1	90.9	91.4	92.4
Voting	92.2	87.8	90.8	95.9

Summary

- Perceptrons are mathematical models of neurons (brain cells)
  - Learn linearly separable functions
  - Insufficiently expressive for many problems
- Neural Networks are machine learning models that have multiple layers of perceptrons
  - Trained using back-propagation, a gradient descent search through weight space (NN hypothesis space)
  - Sufficiently expressive for any classification or regression task, also quite robust to noise

Summary

- Many applications:
  - Speech processing, driving, face/handwriting recognition, backgammon, checkers, etc.
- Disadvantages:
  - Overly expressive: prone to overfitting
  - Difficult to design appropriate structure
  - Many parameters to estimate: slow training
  - Hill-climbing can get stuck in local optima
  - Poor comprehensibility