Lecture 12: Artificial Neural Networks (Mitchell Chapter 4)

CS 167: Machine Learning

Extra Credit Opportunity

Prof. Tianbao Yang, University of Iowa

Deep Learning with Big and Small Data

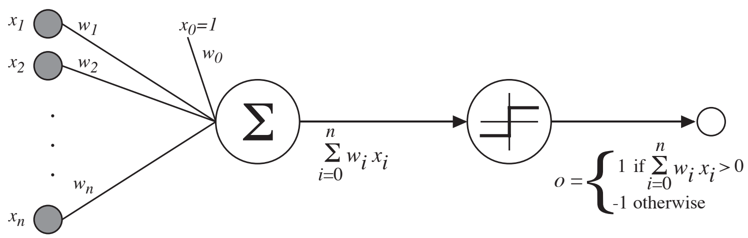
Deep learning has brought tremendous success in many areas with the help of big data and super computing. In this talk, I will present the state of art results of deep learning for image classification. I will also talk about our recent research on how to learn a deep convolutional neural network for fine-grained image classification where big labeled data is difficult to be obtained.

Friday, October 21, in Meredith 106 at 2:00pm.

Earn 3 extra credit (homework) points for

- attend the talk
- write up a paragraph on something you learned

Reminder: Perceptron Unit

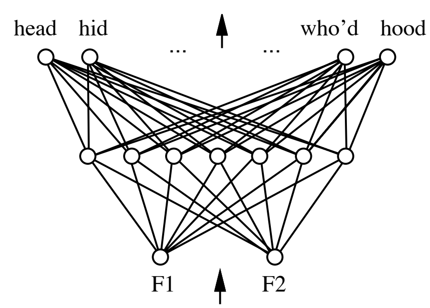


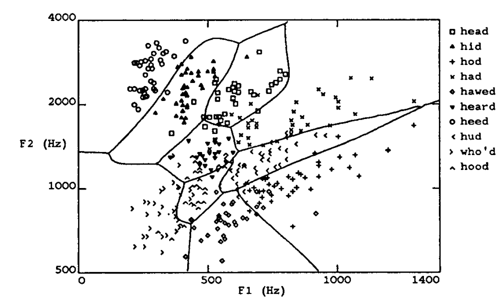
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

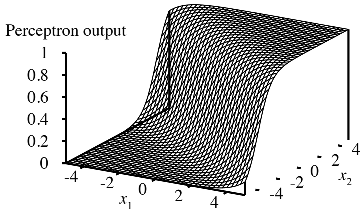
Multilayer Networks



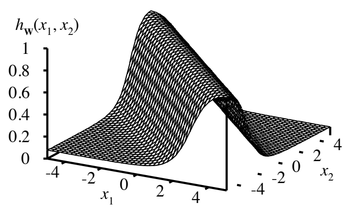


input layer → hidden layer → output layer

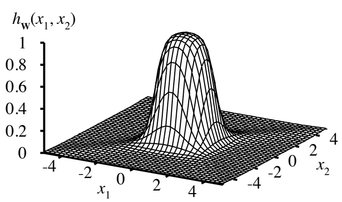
Expressiveness of Multilayer Networks



Single perceptron decision surface



2 opposite thresholds: ridge

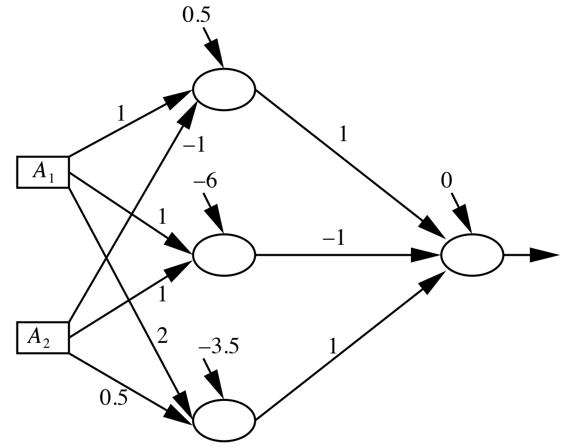


2 opposite ridges: bump

Multilayer Perceptron Classification Example

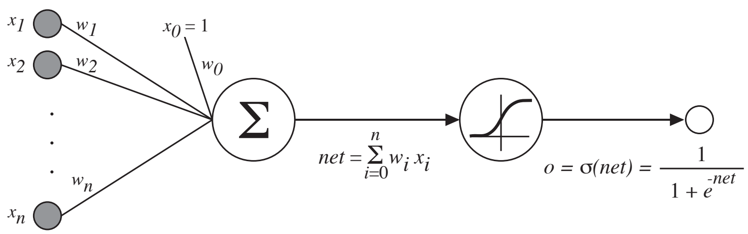
Classify the following four examples (on two attributes A_1 and A_2) using the given multilayer perceptron artificial neural network.

example #	A_1	A_2
1	3	5
2	2	7
3	1	1
4	2	3



Exercise: Do examples 2-4.

Sigmoid Unit



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation
- Read textbook for derivation of training rule

Backpropagation Algorithm

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs (o_u for unit u)
  2. For each output unit k

     $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

     $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

     $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

     where

     $$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Backpropagation of the error

The error function in the Perceptron/SGD training rule was $(t_k - o_k)$:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

The Backpropagation training rule is more complex: $o_k(1 - o_k)(t_k - o_k)$

Why? $o_k(1 - o_k)$ is the derivative of the sigmoid squashing function, important for deriving the training rule.

Both rules scale based on the learning rate, η and the input value x_{ij}

For hidden units, the error at the output unit is divided up among the units feeding into it based on their weight - how much each is responsible for the error at the output unit

Discussion Questions

How many times do you do each training example?

What should η be? Can/should it change during training?

How long does training take?

How long does it take to classify a new example?

Can we get unlucky with initial random weights?

Can we do regression?

Expressive Capabilities of ANNs

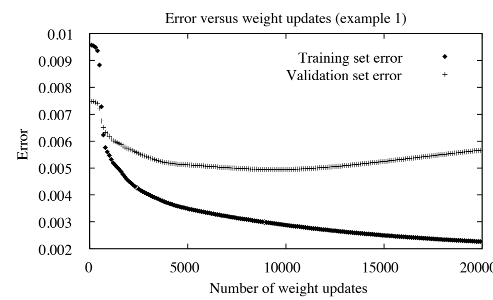
Boolean functions:

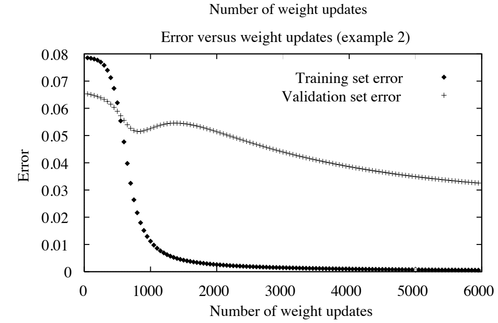
- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

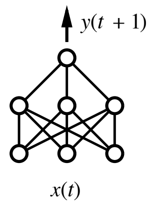
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers

Overfitting in ANNs

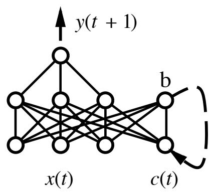




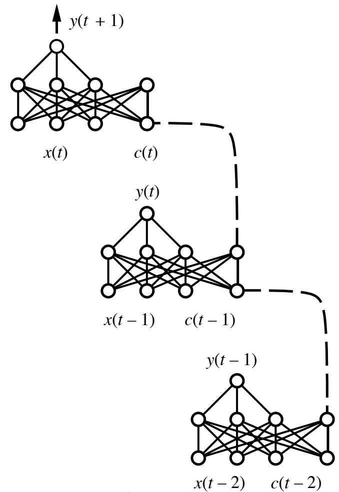
Recurrent Networks



(a) Feedforward network



(b) Recurrent network



(c) Recurrent network unfolded in time

Let's update your version of scikit-learn

It seems that there's a lot of good stuff in scikit-learn 0.18, so let's update it if you haven't already:

1. In Anaconda Navigator, select *Environments* on the left-hand side menu
2. Click the *Update Index* button across the top of the window
3. Scroll to find scikit-learn in the list of packages (or search for it)
4. Look under the version column, hopefully it's blue with an up-right arrow. That mean an upgrade is available.
5. If so, right/crtl-click the package and select *mark for update*.
6. Click the *Apply* button near the bottom right of the window.

Exercise

Determine the answer to these questions:

- Does `scikit-learn` have a neural network for either *classification* or *regression*?
- If so, what is the default network structure it uses (how many layers?, how many nodes in each layer?)? How can I change it?
- If so, does it use the sigmoid squashing (activation) function? If not, can I set it to use it?
- If so, what is the default learning rate and number of epochs? How can you change them?

Exercises:

- What's the smallest network that performs well on the Iris data?
- Can you make an ANN that beats the SGDRegressor performance on the MPG data?