Perceptron

- **2018/11/15**

➢ **Announcement:**

❑ Slides for this lecture are here

http://www.public.asu.edu/~yzhan442/teaching/CSE571/Lectures/perceptron.pdf

Last Time

- Machine Learning
  - Introduction
  - Neural networks
  - Perceptron –

    an earlier learning framework

- **Required reading (red means it will be on your exams):**
  - **R&N: Chapter 18.1-2**

Outline for today

- Perceptron
- Neural network types

Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

Perceptron

- For non-mathematicians, this is going to be tougher than the previous material.
  - You may have to spend a long time studying the next few slides.

- If you are not used to thinking about hyper-planes in high-dimensional spaces, now is the time to learn.

- To deal with hyper-planes in a 14-dimensional space:
  - visualize a 3-D space and say "fourteen" to yourself very loudly.
  - Everyone does it.
  - But remember that going from 13-D to 14-D creates as much extra complexity as going from 2-D to 3-D.

Weight space

- This space has one dimension per weight.

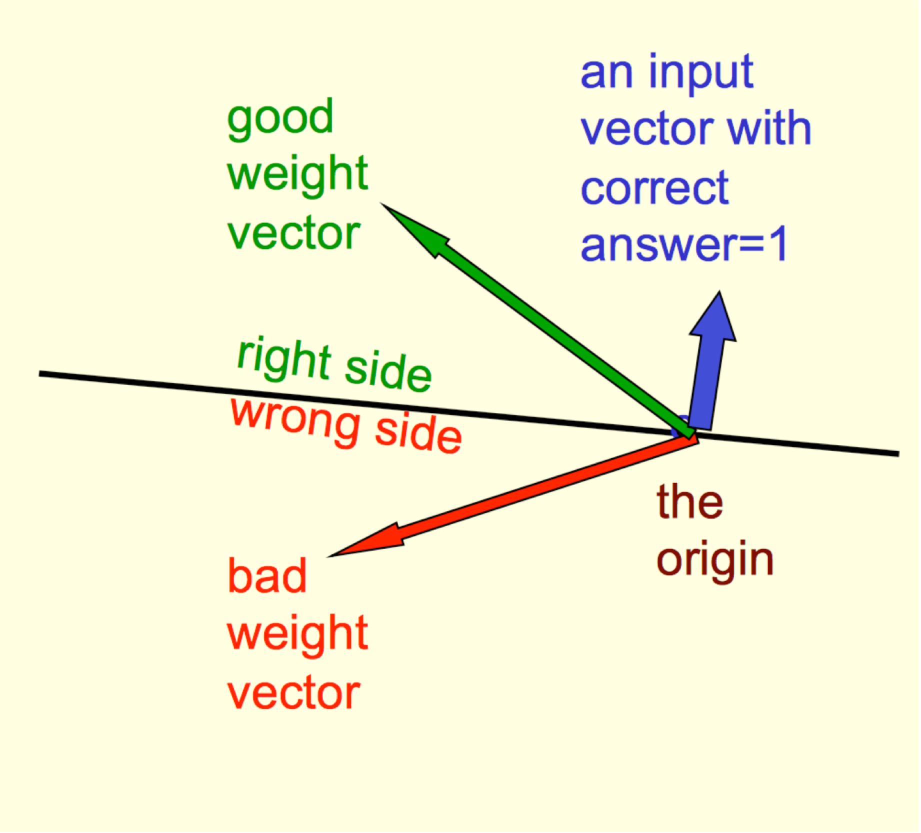
- A point in the space represents a particular setting of all the weights.

- Assuming that we have eliminated the threshold, each training case can be represented as a hyperplane through the origin.
  - The weights must lie on one side of this hyper-plane to get the answer correct.

Weight space

- Consider a binary classification setting (>= 0 class 1; < 0 class 0)

- Each training case defines a plane (shown as a black line)
  - The plane goes through the origin and is perpendicular to the input vector.
  - On one side of the plane the output is wrong because the scalar product of the weight vector with the input vector has the wrong sign.

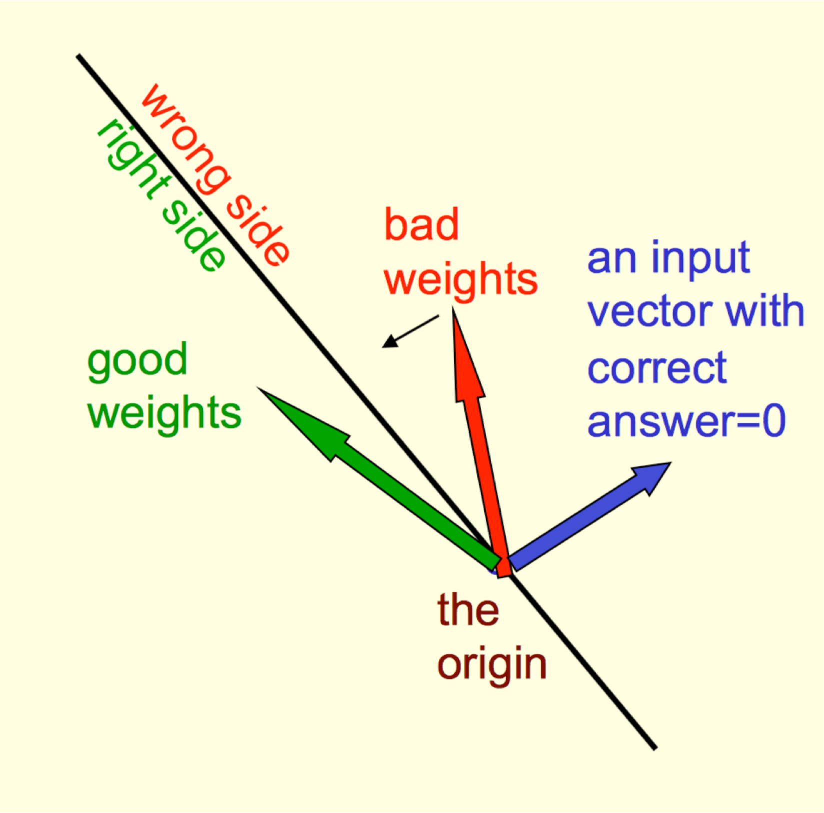


Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

Weight space

- Consider a binary classification setting (>= 0 class 1; < 0 class 0)

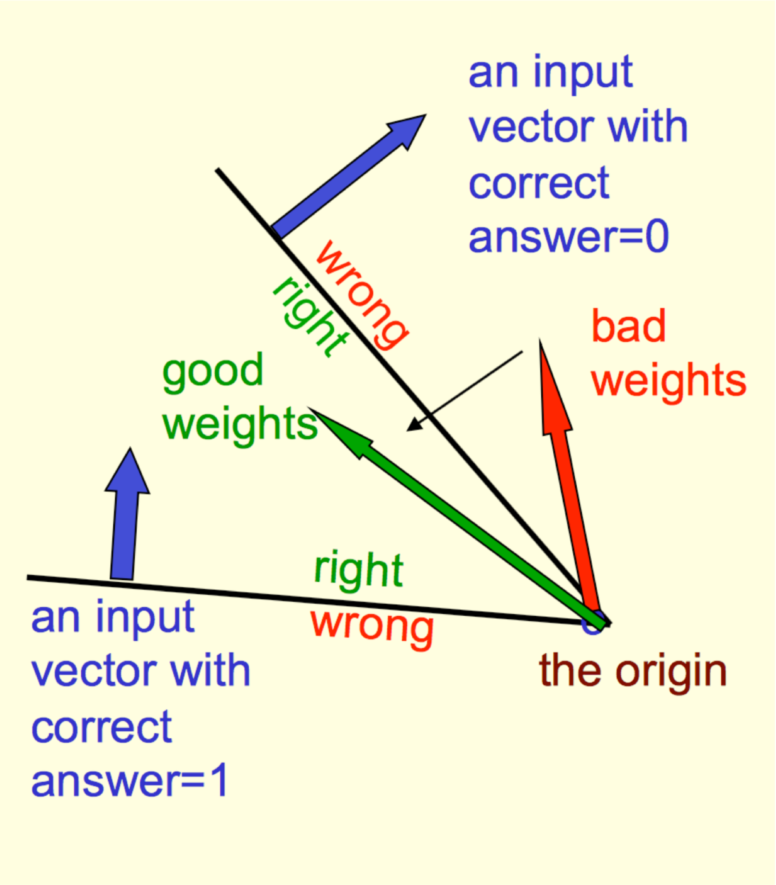
- Each training case defines a plane (shown as a black line)
  - The plane goes through the origin and is perpendicular to the input vector.
  - On one side of the plane the output is wrong because the scalar product of the weight vector with the input vector has the wrong sign.



Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

The cone of feasible solutions

- To get all training cases right we need to find a point on the right side of all the planes.
  - There may not be any such point!
- If there are any weight vectors that get the right answer for all cases:
  - they lie in a **hyper-cone** with its apex at the origin.
  - So the average of two good weight vectors is a good weight vector.
  - The problem is convex.

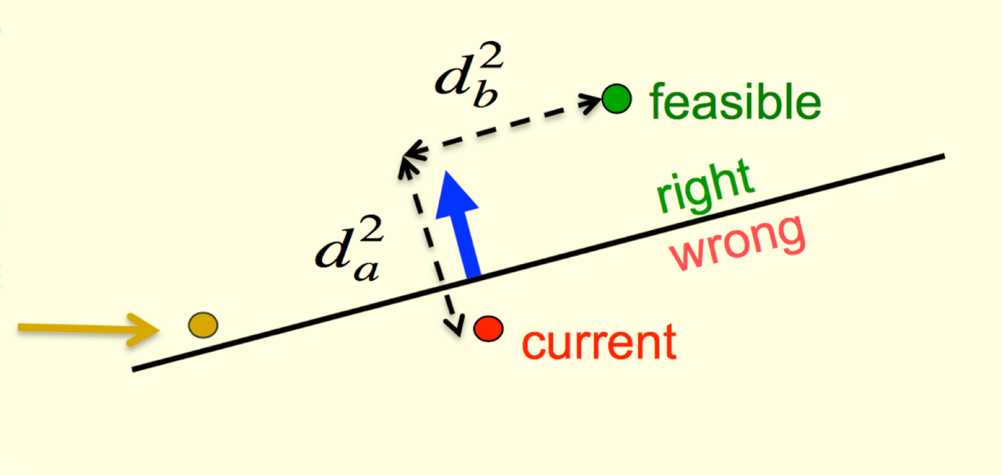


Why does this simple learning method work?

Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

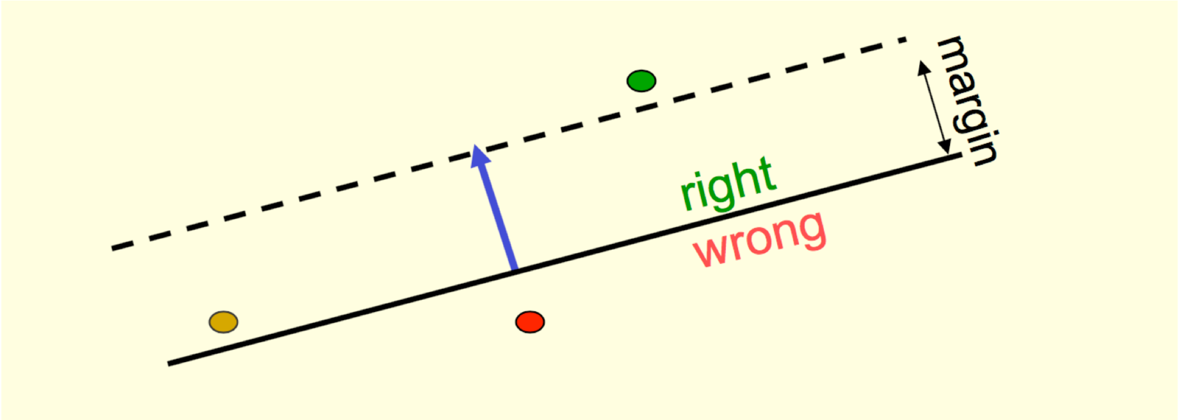
Why the learning procedure works (first attempt)

- Consider the squared distance $d_a^2 + d_b^2$ between any feasible weight vector and the current weight vector.
  - Hopeful claim: Every time the perceptron makes a mistake, the learning algorithm moves the current weight vector closer to all feasible weight vectors
- Problem case: The weight vector may not get closer to this feasible vector!



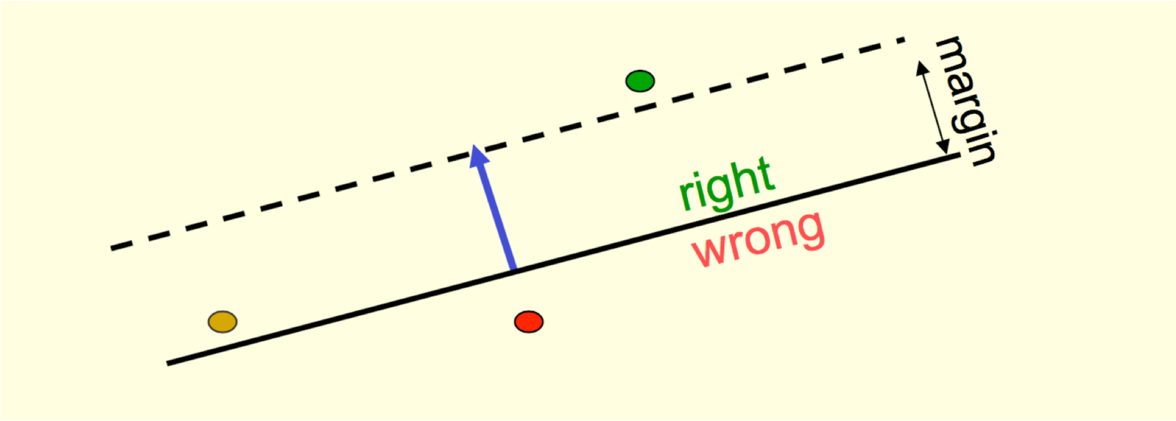
Why the learning procedure works

- So consider "generously feasible" weight vectors that lie within the feasible region by a margin at least as great as the length of the input vector that defines each constraint plane.
  - Every time the perceptron makes a mistake, the squared distance to all of these generously feasible weight vectors is always decreased by at least the squared length of the update vector.



Informal sketch of proof of convergence

- Each time the perceptron makes a mistake, the current weight vector moves to decrease its squared distance from every weight vector in the "generously feasible" region.
- The squared distance decreases by at least the squared length of the input vector.
- So after a finite number of mistakes, the weight vector must lie in the feasible region if this region exists!

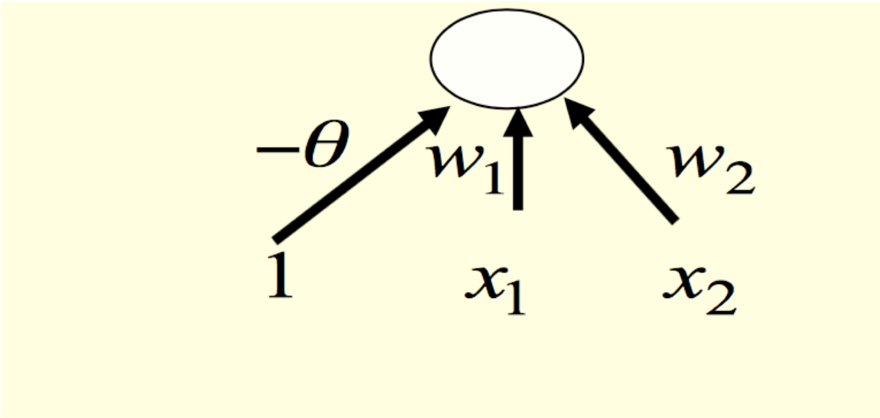


The limitations of perceptron

- If you are allowed to choose the features by hand and if you use enough features, you can do almost anything
  - For binary input vectors, we can have a separate feature unit for each of the exponentially many binary vectors and so we can make any possible discrimination on binary input vectors.
  - This type of table look-up won't generalize.
- But once the hand-coded features have been determined, there are very strong limitations on what a perceptron can learn.

What binary threshold neuron cannot do

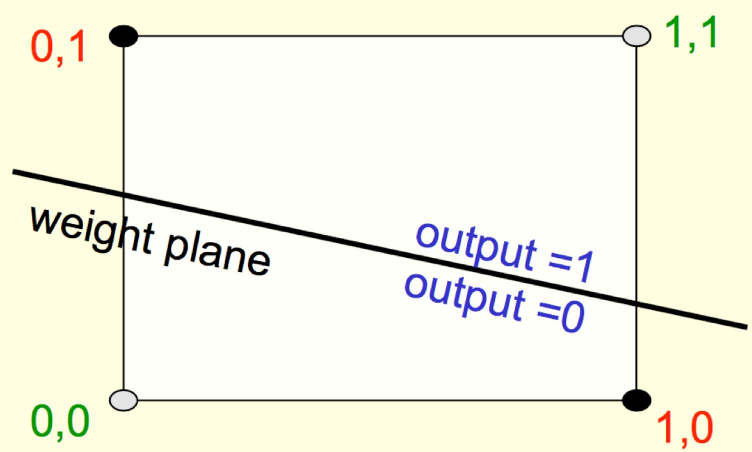
- A binary threshold output unit cannot even tell if two single bit features are the same!
  - Positive cases (same value): (1,1) 1; (0,0) 1
  - Negative cases (different value): (1,0) 0; (0,1) 0
- The four input-output pairs give four inequalities that are impossible to satisfy:



A Geometric view

Imagine “data-space” in which the axes correspond to components of an input vector.

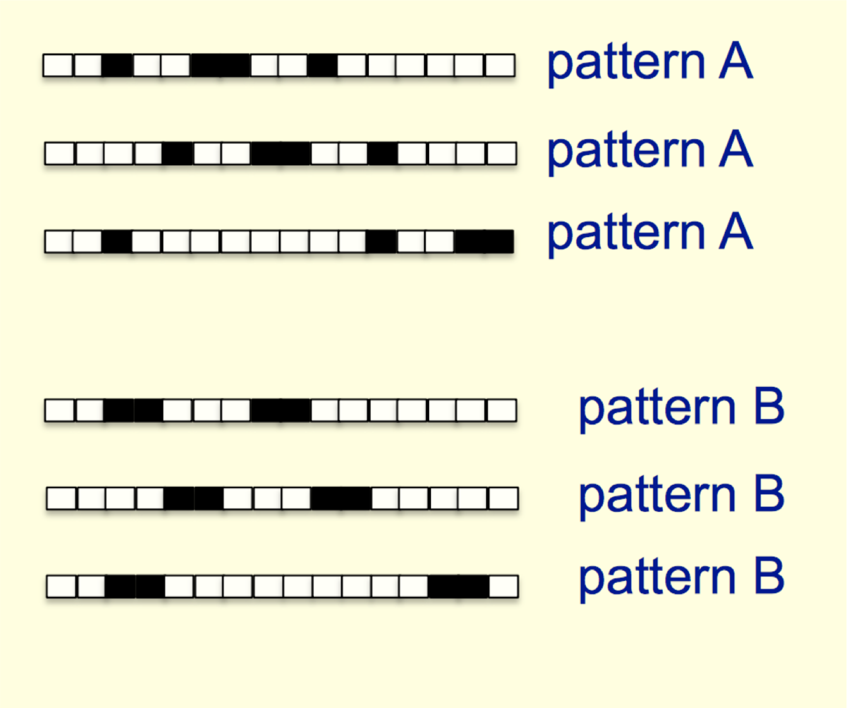
- Each input vector is a point in this space.
- A weight vector defines a plane in data-space.
- The weight plane is perpendicular to the weight vector and misses the origin by a distance equal to the threshold.



The positive and negative cases cannot be separated by a plane

Discriminate simple patterns under translation with wraparound

- Suppose we just use pixels as the features.
- Can a binary threshold unit discriminate between different patterns that have the same number of on pixels?
  - Not if the patterns can translate with wrap-around!



Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

Sketch of a proof that a binary decision unit cannot discriminate patterns with the same number of on pixels

- For pattern A, use training cases in all possible translations.
  - Each pixel will be activated by 4 different translations of pattern A.
  - So the total input received by the decision unit over all these patterns will be four times the sum of all the weights.
- For pattern B, use training cases in all possible translations.
  - Each pixel will be activated by 4 different translations of pattern B.
  - So the total input received by the decision unit over all these patterns will be four times the sum of all the weights.
- But to discriminate correctly, every single case of pattern A must provide more input to the decision unit than every single case of pattern B.
  - This is impossible if the sums over cases are the same!

Why this result is devastating for Perceptrons

- The whole point of pattern recognition is to recognize patterns despite transformations like translation.
- Minsky and Papert's "Group Invariance Theorem" says that the part of a Perceptron that learns cannot learn to do this if the transformations form a group.
  - Translations with wrap-around form a group.
- To deal with such transformations, a Perceptron needs to use multiple feature units to recognize transformations of informative sub-patterns
  - So the tricky part of pattern recognition must be solved by the hand-coded feature detectors, not the learning procedure!

Learning with Hidden units

- Networks without hidden units are very limited in the input-output mappings they can learn to model.
  - More layers of linear units do not help. Its still linear.
  - Fixed output non-linearities are not enough.
- We need multiple layers of adaptive, non-linear hidden units. But how can we train such nets?
  - We need an efficient way of adapting all the weights, not just the last layer. This is hard.
  - Learning the weights going into hidden units is equivalent to learning features.
  - This is difficult because nobody is telling us directly what the hidden units should do.

This is neural networks! (sometimes referred to as multi-layer perceptrons)

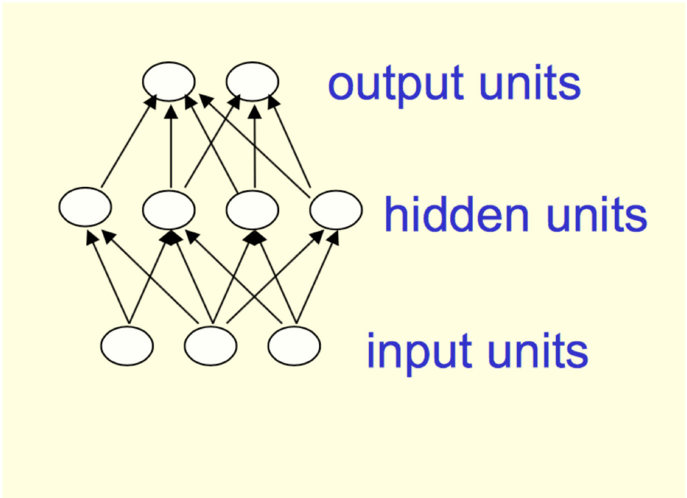
Feed-forward neural network

These are the commonest type of neural network in practical applications.

- The first layer is the input and the last layer is the output.
- If there is more than one hidden layer, we call them "deep" neural networks.

They compute a series of transformations

- The activities of the neurons in each layer are a non-linear function of the activities in the layer below.



Recurrent neural network

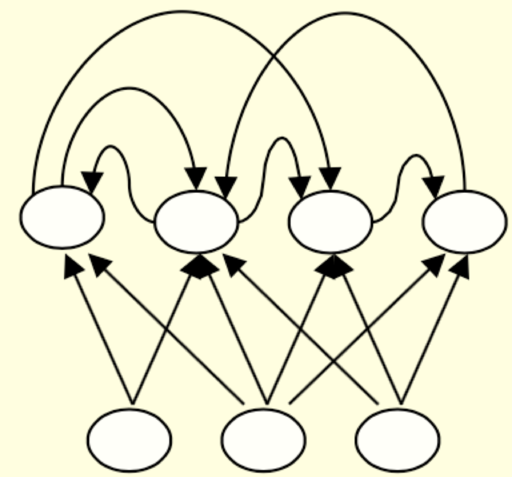
These have directed cycles in their connection graph.

- That means you can sometimes get back to where you started by following the arrows.

They can have complicated dynamics and this can make them very difficult to train.

- There is a lot of interest at present in finding efficient ways of training recurrent nets.

They are more biologically realistic.



Recurrent nets with multiple hidden layers are just a special case that has some of the hidden→hidden connections missing.

Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

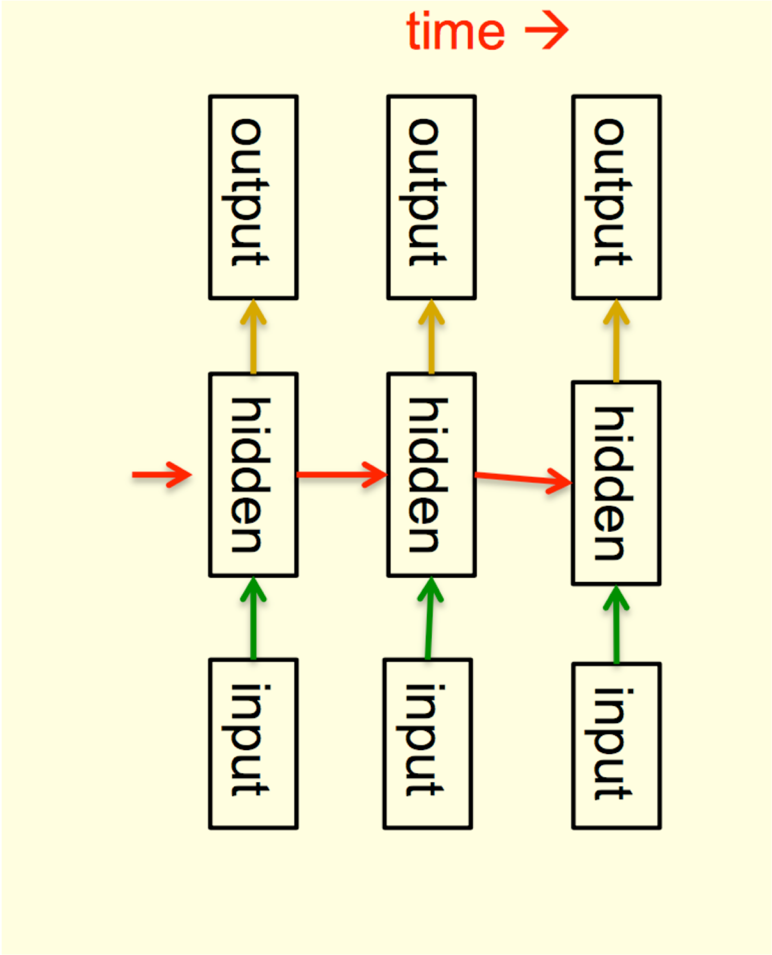
Recurrent neural network for modeling sequences

Recurrent neural networks are a very natural way to model sequential data:

- They are equivalent to very deep nets with one hidden layer per time slice.
- Except that they use the same weights at every time slice and they get input at every time slice.

They have the ability to remember information in their hidden state for a long time.

- But its very hard to train them to use this potential.



An example of what recurrent neural network can do

Ilya Sutskever (2011) trained a special type of recurrent neural net to predict the next character in a sequence.

After training for a long time on a string of half a billion characters from English Wikipedia, he got it to generate new text.

- It generates by predicting the probability distribution for the next character and then sampling a character from this distribution.
- Below shows an example of the kind of text it generates. Notice how much it knows!

In 1974 Northern Denver had been overshadowed by CNL, and several Irish intelligence agencies in the Mediterranean region. However, on the Victoria, Kings Hebrew stated that Charles decided to escape during an alliance. The mansion house was completed in 1882, the second in its bridge are omitted, while closing is the proton reticulum composed below it aims, such that it is the blurring of appearing on any well-paid type of box printer.

Symmetrically connected networks

These are like recurrent networks, but the connections between units are symmetrical (they have the same weight in both directions).

- John Hopfield (and others) realized that symmetric networks are much easier to analyze than recurrent networks.
- They are also more restricted in what they can do because they obey an energy function.
- For example, they cannot model cycles.

Symmetrically connected nets without hidden units are called “Hopfield nets”.

Symmetrically connected networks with hidden units

These are called "Boltzmann machines"

- They are much more powerful models than Hopfield nets.
- They are less powerful than recurrent neural networks
- They have a beautifully simple learning algorithm.

Outline for today

- Perceptron
- Neural network types