Lecture 1
Introduction

- Machine leaning can be used to produce software programs from input (and output) patterns.
- Machine learning can be used for:
  - Recognizing patterns
  - Recognizing anomalies
  - Prediction
- Standard example of machine learning -MNIST database of hand-written digits.
- Beyond MNIST – The ImageNet task.

Reasons to study neural computation

- To understand how the brain actually works.
- To understand a style of computation inspired by neurons and their adaptive connections.
  - Should be good for things that brains are good at – vision.
  - Should be bad for things that brains are bad at – Multiplication.
- To solve practical problems by using novel learning algorithms inspired by the brain.

A typical cortical neuron

- There is an axon hillock that generates outgoing spikes whenever enough charge has flowed in at synapses to depolarize the cell membrane.

Synapses

- When a spike of activity travels along an axon and arrives at a synapse it causes vesicles of transmitter chemical to be released.
- The transmitter molecules diffuse across the synaptic cleft and bind to receptor molecules in the membrane of the post-synaptic neuron thus changing their shape.
  - This opens up holes that allow specific ions in or out.
- The effectiveness of the synapse can be changed by:
  - Varying the number of vesicles of transmitter.
  - Varying the number of receptor molecules.
- Synapses are slow, but they have advantages over RAM:
  - They are very small and very low-power.
  - They adapt using locally available signals.

How the brain works

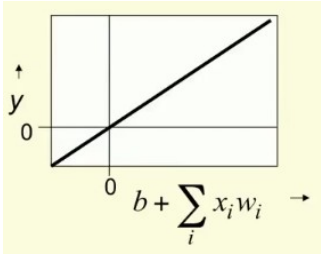
- The effect of each input line on the neuron is controlled by a synaptic weight.
  - The weights can be positive or negative.
- The synaptic weights adapt so the whole network learns to perform useful computations.
- You have about 10^{11} neurons each with about 10^{4} weights.
  - A huge number of weights can affect the computation in a very short time.

Modularity and the brain

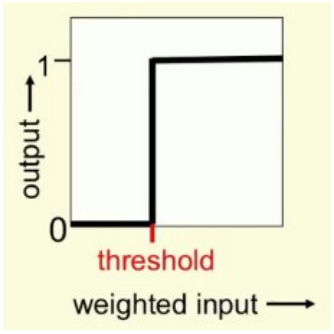
- Different bits of the cortex do different things.
- Cortex looks pretty much the same all over.
  - That strongly suggests that it's got a fairly flexible learning algorithm in it.

Some simple models of neurons

- **Linear neurons:**
  - Computationally limited.
  - y = b + $\sum_i x_i w_i$; b -> bias
  - Plot



- **Binary threshold neuron:**
  - By McCulloch-Pitts
  - First compute a weighted sum of the inputs.
  - Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.
  - Plot



  - Two equivalent ways to write the equations:
    - z = $\sum_i x_i w_i$

      y = $\begin{cases} 1 \text{ if } z \geq \Theta \\ 0 \text{ otherwise} \end{cases}$

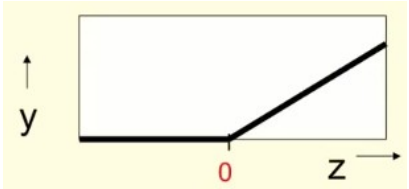
      OR

      z = $b + \sum_i x_i w_i$

      y = $\begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$

      which means, Θ = -b
- Rectified Linear Neurons:
  - Sometimes called linear threshold neurons.
  - They compute a linear weighted sum of their inputs.
  - The output is a non-linear function of the total input.
  - z = $b + \sum_i x_i w_i$

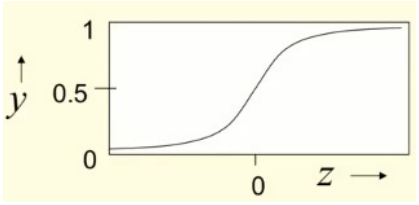
    y = $\begin{cases} z \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$
  - Plot



- Sigmoid neurons:

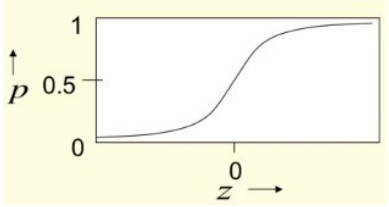
    - These give a real-valued output that is a smooth and bounded function of their total input.
        - Typically they use the logistic function.
        - They have nice derivatives which make learning easy.
    - z = $b+\sum_i x_i w_i$

      $y=\frac{1}{1+e^{-z}}$
    - Plot



- Stochastic binary neurons:
    - These use the same equations as logistic units.
        - But they treat the output of the logistic as the probability of producing a spike in a short time window.
    - z = $b+\sum_i x_i w_i$

      $p(s=1)=\frac{1}{1+e^{-z}}$
    - Plot



- Rectified linear neurons with output treated as the Poisson rate for spikes:
    - We can say that the output, the real value that comes out of a rectified linear unit, if its input is above zero, is the rate of producing spikes. So that's deterministic.
    - But once we've figured out these rate of producing spikes, the actual times at which spikes are produced is a random process. It's a Poisson process.
    - So the rectified linear unit determines the rate, but intrinsic randomness in the unit determines when the spikes are actually produced.

Three types of learning

- Supervised learning
    - Learn to predict an output when given an input vector.
- Reinforcement learning
    - Learn to select an action to maximize payoff.
- Unsupervised learning
    - Discover a good internal representation of the input.

Two types of supervised learning

- Each training case consists of an input vector x and a target output t.
- Regression: The target output is a real number or a whole vector of real numbers.
- Classification: The target output is a class label.

How supervised learning typically works

- We start by choosing a model-class: y = f(x; W)
  - A model-class, f, is a way of using some numerical parameters, W, to map each input vector, x, into a predicted output y.
- Learning usually means adjusting the parameters to reduce the discrepancy between the target output, t, on each training case and the actual output, y, produced by the model.
  - For regression, $\frac{1}{2}(y-t)^2$ is often a sensible measure of the discrepancy.
  - For classification there are other measures that are generally more sensible.

Reinforcement learning

- In reinforcement learning, the output is an action of sequence of actions and the only supervisory signal is an occasional scalar reward.
  - The goal in selecting each action is to maximize the expected sum of the future rewards.
  - We usually use a discount factor for delayed rewards so that we don't have to look too far into the future.
- Reinforcement learning is difficult:
  - The rewards are typically delayed so it's hard to know where we went wrong (or right).
  - A scalar reward does not supply much information.

Unsupervised learning

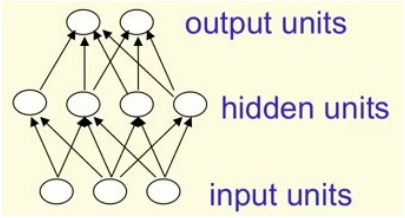
- One major aim is to create an internal representation of the input that is useful for subsequent supervised or reinforcement learning.
- It provides a compact, low-dimensional representation of the input.
  - High-dimensional inputs (like images) typically live on or near a low-dimensional manifold (or several such manifolds).
  - Principal Component Analysis is a widely used linear method for finding a low-dimensional representation.
- It provides an economical high-dimensional

representation of the input in terms of learned features.
    - Binary feature are economical.
    - So are real-valued features that are nearly all zero.
- It finds sensible clusters in the input.
    - This is an example of a very sparse code in which only one of the features in non-zero.

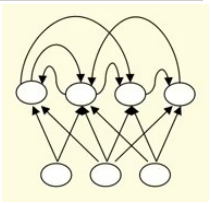
Lecture 2

Types of neural network architectures

- Feed-forward neural networks:

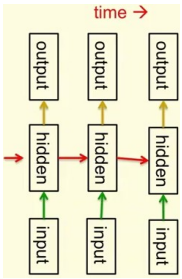


    - These are the commonest type of neural network in practical applications.
        - The first layer is the input and the last layer is the output.
        - If there is more than one hidden layer, we call them “deep” neural networks.
    - They compute a series of transformations that change the similarities between cases.
        - The activities of the neurons in each layer are a non-linear function of the activities in the layer below.
- Recurrent networks:



    - 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 difficult to train.
    - 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.

Recurrent neural networks for modeling sequences



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

- What we do is we have connections between hidden units and the hidden units act like a network that's very deep in time.
  - They are equivalent to very deep nets with one hidden layer per time slice.
- So at each time step the states of the hidden units determines the states of the hidden units of the next time step.
- One way in which they differ from feed-forward nets is that we use the same weights at every time step.
  - So if you look at the red arrows where the hidden units are determining the next state of the hidden units, the weight matrix depicted by each red arrow is the same at each time step.
  - They also get inputs at every time step and often give outputs at every time step, and those'll use the same weight matrices too.
- They have the ability to remember information in their hidden state for a long time.
  - But it's very hard to train them to use this potential.

- <u>Symmetrically connected networks:</u>
  - These are like recurrent networks, but the connections between units are symmetrical (they have the same weight in both directions).
    - They 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".

Perceptrons

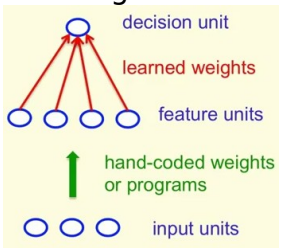
- They are the first generation of neural networks.

The standard paradigm for statistical pattern recognition

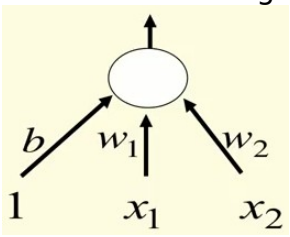
- Convert the raw input vector into a vector of feature activations.
    - Use hand written programs based on common-sense to define the features.
- Learn how to weight each of the feature activations to get a single scalar quantity.
- If this quantity is above some threshold, decide that the input vector is a positive example of the target class.

The standard Perceptron architecture

- Analogous to the standard paradigm for statistical pattern recognition.



- The decision unit in a perceptron is the binary threshold neuron.
- How to learn biases using the same rule as we use for learning weights:
    - A threshold is equivalent to having a negative bias.
    - We can avoid having to figure out a separate learning rule for the bias by using a trick:
        - A bias is exactly equivalently to a weight on an extra input line that always has an activity of 1.
        - We can now learn a bias as if it were a weight.



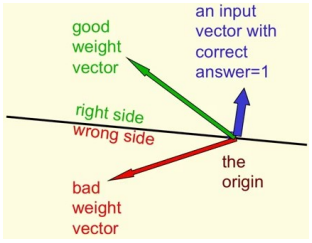
The perceptron convergence procedure: Training binary output neurons as classifiers

- Add an extra component with value 1 to each input vector. The “bias” weight on this component is minus the threshold. Now we can forget the threshold.
- Pick training cases using any policy that ensures that every training case will keep getting picked.
    - If the output unit is correct, leave its weights alone.
    - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
    - If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector.
- This is guaranteed to find a set of weights that gets the right answer for all the

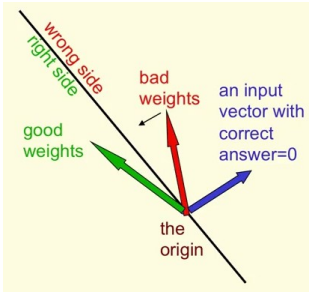
training cases if any such set exists.

A geometrical view of perceptrons 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 hyperplane to get the answer correct.
- CASE 1: Each training case defines a plane (shown as a black line)

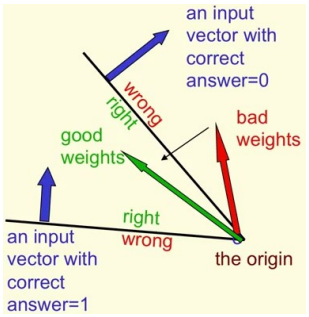


    - The plane goes through the origin and is perpendicular to the input vector.
    - On one side of the plane the output is wrong because the scalar product of the weight vector with the input vector has the wrong sign.
- CASE 2: Each training case defines a plane (shown as a black line)



    - 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.

The cone of feasible solutions



- To get all the 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 the 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.
            - The average of two solutions is itself a solution.
            - In general, in machine learning, if you can get a convex learning problem that makes life easy.

Limitations of perceptrons

- 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 discriminated, there are very strong limitations on what a perceptron can learn.
- They cannot discriminate between the outputs of the XOR/XNOR gates.
- They fail to discriminate simple patterns under translation with wrap-around.

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. It's still linear.
    - Fixed output non-linearities are not enough.
- We need multiple layers of adaptive, non-linear hidden units.
    - 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.