NN Lecture notes

Bellamie

May 2015

Contents

1 Introduction slides

1.1 Neural networks in a historical context

- **Associationism**: the idea that mental processes operate by the association of mental state with its successor states.
- **Aristotle** (400 BC): memory consists of simple elements, connected through various mechanisms like temporal succession and spatial proximity
- **Materialism** (18th century, de La Mettrie, Hobbes): Mind as machine
- **Empiricism** (18th century, Berkely, Locke, Hume): Human knowledge is derived from sensory experience and it is the association of these experiences that lead to thought. Therefore, human cognition is governed by physical laws and can be studied empirically (e.g. through observations)
- **Connectionism**: modeling mental or behavioral phenomena as emergent processes of interconnected networks of simple units

1.2 The different network architectures

- Feed-forward neural networks

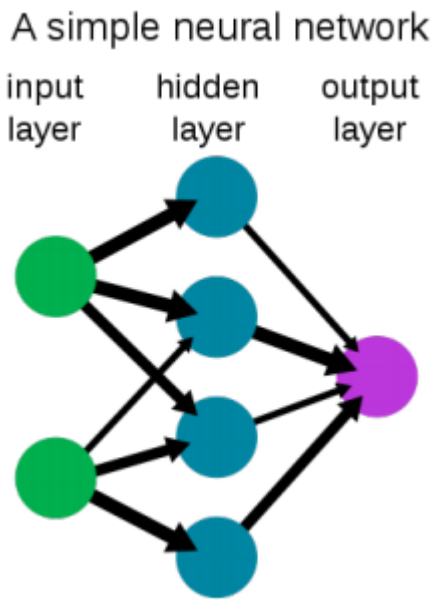


Figure 1: Caption

  - The most common type of neural networks in practical applications

  - First layer is input; last layer is output
  - If there is more than one hidden layer then we call them deep neural networks

- Recurrent neural networks

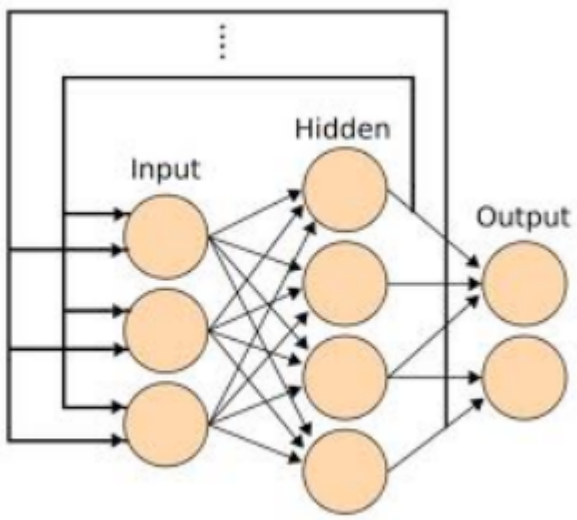


Figure 2: Caption

  - Directed cycles in the connection graph
  - They can have complicated dynamics which makes them difficult to train
  - They are more biologically realistic

- Undirected neural networks

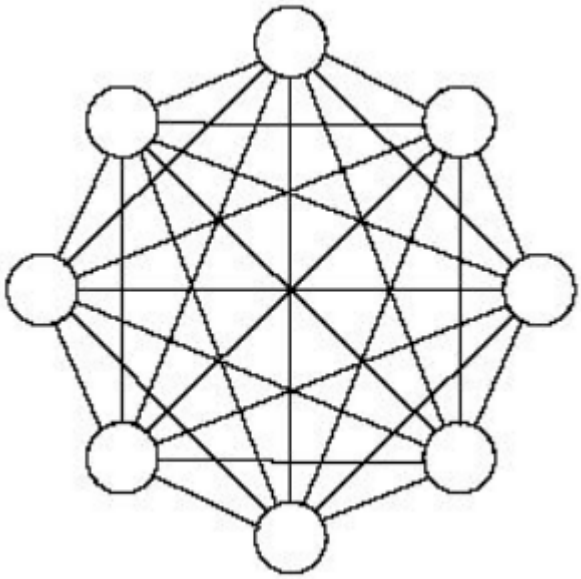


Figure 3: Caption

  - Like recurrent neural networks but undirected (symmetrical) connections between units
  - Much easier to analyze but also more restricted in what they can do
  - Examples: Hopfield nets, Boltzmann machines

- Modular neural networks

Figure 4: Caption

- Collections of processing units organized into competitive pools
- Used in classical work by Grossberg, Rumelhart, McClelland
- Example: Interactive activation and competition model

1.3 Definitions

- **Input activation**: $a_i = \sum_j w_{ij} * x_j$ or $a_i = w^T * x$
- **Input vector**: x is the input vector
- **Weight vector**: w is the weight vector
- **Output**: $x_i = f(a_i)$ with f activation function

1.4 Difference between linear threshold and sigmoid activation functions

- **Linear threshold**
- **Sigmoid activation function**

1.5 The three kinds of learning tasks

- **Supervised learning**: learn to predict an output when given an input vector

- **Unsupervised learning**: discover a good internal representation of the input
- **Reinforcement learning**: learn to select an action to maximize payoff

1.6 The Hebbian learning rule

- $\Delta w_{ij} = \eta x_i x_j$
- Cells that fire together, wire together.
- Change in weight is a scaled product of the output x_j in pre synaptic unit j and output x_i in post synaptic unit i.
- Partial explanation for long-lasting changes in synaptic strength in biological neural networks

2 Perceptrons

2.1 Compute the output of a perceptron

- $y = f(w^T x)$
- $f(x) = \begin{cases} 1 & \text{if } error \geq 0 \\ 0 & \text{otherwise} \end{cases}$
  - f: activation function (here threshold function)
  - x: the input vector
  - w: a weight vector
  - y: the binary output

2.2 What is a decision boundary

- Decision boundary: the set of points for which the activation $a = w^T x$ is zero (cross-over point for function f)

2.3 The steps of the perceptron convergence procedure

- The PCP is given by:
  - Assume training data (x^n, t^n) for $n = 1, ..., N$
  - Pick training cases using any policy that ensures that every training case will keep getting picked.
    - If the output unit incorrectly outputs a $y = 0$, add the input vector to the weight vector. $y \neq 0 \rightarrow x + w$

    - If the output unit incorrectly outputs a $y = 1$, subtract the input vector from the weight vector. $y \neq 1 \rightarrow w - x$
    - If the output is correct, leave it weights alone.

- An as equation: $w \leftarrow w + \eta(T^n - y^n)x^n$ with learning rate eta an predicted output y.

2.4 What kind of problems can and cannot be solved by a perceptron

- Any solution to a problem that can be solved by a percetron can be found in a finite number of steps using the perceptron convergence procedure
  - Perceptrons are an example of so-called linear discriminant functions, whose decision boundary forms a hyper-plane in input space.
  - A problem is linearly separable when it can be solved using a hyper-plane in input space
  - thus, perceptrons can only solve linearly separable problems

2.5 How does gradient descent work by using partial derivatives of an error function

- The delta rule is a more general and principled approach to training supervised neural networks.
  - Instead of showing that the weights get closer to a good set of weights show that the actual output values get closer to the target values
  - The delta rule is based on the definition of an objective function and direct optimization of this function using gradient descent methods
- Defining an error function
  - $y = f(w^T x)$
  - in case f is the identity function, we have a linear activation function.
  - we define a sum of squares error function which measures the difference between the predicted output and a target output: $E(w) = \frac{1}{2}\sum_n (y^n - t^n)^2$
  - The goal is to find parameters which minimize E (or maximize -E)
  - This measures locally the change in slope of E(w) as a function of w take a step in the direction of the steepest descent (gradient descent)
- Gradient descent is based on the observation that if a function f(x) is defines and differentiable in the neighbourhood of a point a, then f(x) decreases **fastest** if one goes from a in the direction of the negative gradient of f at a.

2.6 Explain the components of function and how is it used to update the weights using the delta rule ($w \leftarrow w = \gamma * \Delta E$

- $\frac{\delta E(w)}{\delta w_i} = \sum_n (y^n - t^n) \frac{\delta f(a^n)}{\delta a^n} x_i^n$
  - with $a^n = w^T x^n$
  - $y^n = f(a^n)$

3 Multilayer perceptrons

Multi-layer neural networks (aka multi-layer perceptrons): Supervised neural networks with hidden layers that can solve more complex problems. These networks can compute any nonlinear function of its input.

3.1 Where comes the sum of squares error function from

$$E(w) = \frac{1}{2} \sum_n (y^n - t^n)^2$$

- Given a training set $(x^n, t^n)_{n=1}^N$
- We can define the **likelihood** as $\mathcal{L} = \prod_n p(x^n, t^n) = \prod_n p(t^n|x^n)p(x^n)$
- Maximizing the likelihood is equivalent to minimizing the **negative log likelihood:** $E = -\sum_n logp(t^n|x^n) - \sum logp(x^n)$
- A feedforward neural network models the conditional density only so we consider $E = -\sum_n logp(t^n|x^n)$
- Now assume that the target variable is a deterministic function of the inputs plus Gaussian noise: $t = f(x; w) + \epsilon$ with $p(\epsilon) = \mathcal{N}(\epsilon|0, \sigma) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} exp(-\frac{\epsilon^2}{2\sigma^2})$
- Since the deterministic function gives the expected value for t and since sigma determines the variance, it follows that $p(t|x) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} exp(-\frac{(f(x;w)-t)^2}{2\sigma^2})$
- This leads to $E = -\sum_n log \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} exp(-\frac{f(x^n;w)-t^n)^2}{2\sigma^2}) = \frac{1}{2\sigma^2} \sum_n (f(x^n; w^n) - t^n)^2 + Nlog\sigma + \frac{N}{2} log 2\pi$
- We have: $E = \frac{1}{2\sigma^2} \sum_n (f(x^n; w^n) - t^n)^2 + \cancel{N} log\sigma + \cancel{\frac{N}{2}} log 2\pi$
- Dropping the scaling factor and terms not depending on **w**, we retrieve the sum of squares error function: $E = \frac{1}{2} \sum_n (y^n - t^n)^2$ with $y^n = f(x^n; w)$
  - We leave the 1/2 factor for convenience
  - The sum of squares error is strongly penalizes outliers
  - Other assumptions lead to different error functions

3.2 How to compute the output of a multi-layer perceptron

3.3 how to compute the derivative of the sigmoid activation function

$$\frac{df}{da} = \frac{d}{da} * \frac{1}{1 + exp(-a)}$$

$$= \frac{\frac{d}{da}1 * (1 + exp(-a)) - 1 * \frac{d}{da}(1 + exp(-a)}{(1 + exp(-a))^2} (quotientrule)$$

$$= -\frac{\frac{d}{da}(1 + exp(-a))}{(1 + exp(-a))^2}$$

$$= \frac{exp(-a)}{(1 + exp(-a))^2}$$

3.4 What are the delta terms and how are they important in backprop learning

Activation of output unit j times output of output unit j

$$\delta_j = \frac{\delta E^n}{\delta a_j} * \frac{\delta y_j}{\delta a_j}$$

$$= \frac{\delta E^n}{\delta y_j} \frac{\delta f(a_j)}{\delta a_j}$$

since $f(a_j) = y_j$

3.5 Explain why backprop learning was a major breakthrough in NN research

- Definition: Backpropagation is a procedure for efficiently calculating the derivatives of some output quantity of a nonlinear system, with respect to all inputs and parameters of that system, through calculations proceeding backwards from outputs to inputs. It permits "local" implementation on parallel hardware (or wetware).
- Outline of backprop
  - To minimize E by gradient descent, the partial derivative of E^n with respect to each weight within the network needs to be computed

- For a given data point, n, this partial derivative is computed in two passes:
  1. a forward pass through the network
  2. a backward pass which propagates derivatives back through the layers.
  3. Hence, backpropagation (of error).

3.6 What do the weights look like when you include momentum

$$\Delta^n w(t) = -\gamma \frac{\delta E^n(w)}{\delta w} + \alpha \Delta^n w(t-1)$$

- At the beginning of learning there may be very large gradients
- So it pays to use a small momentum (e.g. 0.5).
- Once the large gradients have disappeared and the weights are stuck in a ravine the momentum can be smoothly raised to its final value (e.g. 0.9 or even 0.99).
- This allows us to learn at a rate that would cause divergent oscillations without the momentum.

4 Supervised deep learning

4.1 Motivate deep learning from a biological and theoretical point of view

- Biological point of view
  - Visual features (Thorpe) though a bit naive from the point of view of connectivity
  - The perception-action cycle (Fuster) nested feedback loops rather than a hierarchical system
- Theoretical point of view
  - Consider building a neural network to compute the parity (or XOR) of n input bits: $parity : (b_1, ..., b_d) \subset \{0,1\}^d \rightarrow \begin{cases} 1 & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\ -1 & \text{otherwise} \end{cases}$
  - Suppose each node in the network can compute either the logical OR of its inputs (or the OR of the negation of the inputs), or compute the logical AND
  - If we have a network with only one input, one hidden, and one output layer, the parity function would require a number of nodes that is exponential in the input size n.

  - If however we are allowed a deeper network, then the network/circuit size can be only polynomial in n.

4.2 Reproduce an example of a classical deep neural network

- Selfridge's pandemonium (1958)
  - NN consists of four layers of specialized "shouting demons".
    - Layer 4: decision demon (chooses loudest scream)
    - Layer 3: cognitive demons (weigh evidence from feature demons)
    - Layer 2: feature demons (analyzers)
    - Layer 1: data/image demons (sensors)
- Selfridge's pandemonium (1959)
  - initial structure chosen depending on task
  - simple learning mechanisms:
    - hill climbing for top weights
    - pruning of useless feature demons
  - applications: morse and digit recognition
  - used by psychologists as model of human pattern recognition
- Fukushima's neocognitron (1980)
  - Hierarchical multilayer neural network
  - Consists of S-layers which model local features and C-layers which tolerate deformations of the features
  - Modelled after Hubel and Wiesel's simple cells and complex cells
- Serre's HMAX model (2007)
  - Gabor filters as models of simple cells
  - Complex cell layer performs a pooling operation using MAX operator
- LeCun's LeNet (1998)
  - Deep convolutional neural network

4.3 Why is supervised deep learning hard (underfitting vs overfitting

- First problem: Underfitting
  - Optimization becomes much harder (vanishing gradient problem)

- Overfitting
  - Deep nets usually have lots of parameteres
  - These are estimated by finite data
  - Can result in high variance/ low bias situation
- Related Issue: the curse of dimensionality
  - The size of a dataset required to specify a mapping grows exponentially with the size of the input space

4.4 Provide solutions for underfitting and overfitting

- Solutions for underfitting
  - use better/more efficient optimization techniques and train for a long time using GPU architectures
  - Use activation functions with more desirable properties
- Solutions for overfitting
  - Use much larger datasets (data augmentation)
  - Reduce size of the input (dimensionality reduction)
  - Reduce the number of free parameters (weight sharing)
  - Enforce good generalisation (dropout training)

4.5 Give an example of data augmentation

Reduce overfitting on image data by artificially enlarging the data set using label-preserving transformations.

- Generate image translations and horizontal reflections.
- Alter the intensities of the RGB channels in training images. This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination.

Without this scheme the DNN suffers from substantial overfitting, which would force us to use much smaller networks.

4.6 Know the mathematical form of 2-D convolution

Definition of convolution for a 2D signal f with filter g:

$$(f * g)[x, y] = \sum_{u=-M}^{M} \sum_{v=-M}^{M} f[x-u, y-v]g[u, v]$$

4.7 Know what is meant by pooling and why it is useful

- Each feature map is sub-sampled by taking the mean or maximum value over p x p contiguous regions.
- This gives translation invariance (tolerance to small displacements of the input)

4.8 be able to reproduce the mathematical form for the ReLU and explain why it is beneficial

4.9 be able to explain local response normalisation in words

- LRN futher aids generalization.
- Denote by $a^i_{x,y}$ the input activity of a neuron i at position (x,y) of its corresponding feature map.
- The response normalized activity $b^i_{x,y}$ is given by:

$$b^i_{x,y} = a^i_{x,y}/(k + \alpha \sum_{j=max(0,i-n/2}^{min(N-1,i+n/2} (a^i_{x,y})^2)^\beta$$

- Normalises the output wrt the output of 'neighbouring' feature maps at the same location (x,y)

4.10 be able to explain how dropout works

- Combining the prediction of many different models is a very successful way to reduce test errors
- Way too expensive for big neural networks that already take several days to train
- Dropout is a very efficient version of model combination that only costs about a factor of two during training.
- Consists of setting to zero the output of each hidden neuron with probability 0.5.
- Neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in back-propagation.
- Every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.
- A neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

- At test time, we use the neurons but multiply their outputs by 0.5
- Dropout was used in the first two fully-connected layers of the ImageNet model.
- Without dropout, our network exhibits substantial overfitting

4.11 understand what is meant by deconvolution

- Pass each image through the network
- Fix the feature map of interest and set the rest to zero
- Propagate backwards through the layers using the DeconvNet
- Show the reconstructions of the features and the corresponding images that gave largest activations of that unit

5 Hopfield Networks

5.1 What is meant by a Hopfield network

- A (discrete) Hopfield network is composed of **binary threshold units**
- each unit has its own bias term θ
- They have **symmetric *recurrent*** connections between them: $\forall i, j : w_{ij} = w_{ji}$ $\forall i : w_{ii} = 0$

5.2 How to update the states of a simple Hopfield network

- Network states are updated via: $x_i = \begin{cases} 1 & \text{if } \sum_j w_{ij} x_i + \theta_i > 0 \\ 0 & \text{otherwise} \end{cases}$
- Updates in the Hopfield network are performed as follows:
  - **Asynchronous**: Only one unit is updated at a time.
  - Unit can be picked at random, or a pre-defined order can be imposed from the very beginning
- Each state conditional on the other states is equivalent to a perceptron
- Hopfield nets are guaranteed to converge to a stable state. This relies on:
  - No self-connection ($w_{ii} = 0$)
  - Symmetric weights ($w_{ij} = w_{ji}$)
  - Asymmetric updates

- EXAMPLE:

$$E(x) = -\frac{1}{2}\sum_{i,j} w_{i,j}x_i x_j - \sum_i \theta_i x_i$$

  - For 0,0 $\rightarrow E(x) = -\frac{1}{2} * (-1*0*0 - 1*0*0) - (0.5*0 + 0.5*0) = 0$
  - For 0,1 $\rightarrow E(x) = -\frac{1}{2} * (-1*0*1 - 1*0*1) - (0.5*0 + 0.5*1) = -0.5$
  - For 1,0 $\rightarrow E(x) = -\frac{1}{2} * (-1*1*0 - 1*1*0) - (0.5*1 + 0.5*0) = -0.5$
  - For 1,1 $\rightarrow E(x) = -\frac{1}{2} * (-1*1*1 - 1*1*1) - (0.5*1 + 0.5*1) = 0$

  The local minimums are:

  - $x_1 = 0$ & $x_2 = 1$
  - $x_1 = 1$ & $x_2 = 0$

5.3 How to write down the energy function of a Hopfield network

$$E(x) = -\frac{1}{2}\sum_{i,j} w_{ij}x_i x_j - \sum_i \theta_i x_i$$

$$= -\frac{1}{2}x^T W x - x^T \theta$$

5.4 What is meant by combinatorial optimization and associative memories

- combinatorial optimization problem
  - An optimization problem is the **problem** of finding the best solution from all **feasible solutions**.
  - Optimization problems can be divided into two categories depending on whether the **variables** are continuous or discrete
  - COP: An optimization problem with discrete variables
- Associative memories:
  To work as an associative memory, a network has to solve the following problem: "Store M patterns S such that when presented with a new pattern Y, the network returns the stored pattern S that is closest in some sense to Y".
  - Hopfield proposed that memories could be energy minima of a neural net.
  - The binary threshold decision rule can then be used to "clean up" incomplete or corrupted memories.

- This gives a content-addressable memory in which an item can be accessed by just knowing part of its content
- Applications: models of biological memory, hand-written digit recognition, face recognition, information retrieval in database systems,..

5.5 How to derive what Hopfield network can solve the multiflop problem

- By assuming we want to obtain vectors with all zeros except for a single one. ?????
- The hopfield network needs weights to -1 and all thresholds 2.

5.6 How to write down the weight update for a Hopfield network with bipolar units

$$w_{ij} \leftarrow w_{ij} + x_i^{(k)} x_j^{(k)}$$

5.7 Understand the proof that Hebbian learning works in case of training on one input pattern

- Recall that $E(x) = -\frac{1}{2} x^t W x - x^T \theta$
- We can ignore the bias term if it is represented as a constant in $\mathbf{x}$
- For $W_1 = x_1 x_1^T - I$ it holds that $E(x) = -\frac{1}{2} x^T W_1 x = -\frac{1}{2}(x^T x_1 x_1^T x - x^T x$
- This is equivalent to $E(x) = -\frac{1}{2}(x^T x_1)(x_1^T x)^T + \frac{n}{2}$
- since $x^T x = n$ for bipolar units and $(ab)^T = b^T a^t$
- For $E(x) = -\frac{1}{2}(x^T x_1)(x_1^T x)^T + \frac{n}{2}$ the minimum does not depent on the second term.
- Hence $E(x) = -\frac{1}{2}(x^T x_1)(x_1^T x)^T$ is minimized when the scalar $x^T x_1 = x_1^T x$ is maximized
- Since $x^T x_1$ can be at most n for bipolar units and $x^T x_1 = n$ it holds that Hebbian learning locates the minimum of the energy function at x_1

5.8 Know what is meant by spurious patterns

- Retrieval states: patterns that the network uses for training. These become attractors of the system
- Repeated updates would eventually lead to convergence to one of the retrieval states.

- Sometimes the network will converge to **spurious patterns**, that are different from the training patterns.
- Energy of these spurious patterns is also a local minimum.
- For each stored pattern x, the **reversed state** -x is a spurious pattern
- A spurious pattern can also be a **mixture state**; a linear combination of an odd number of retrieval states
- A spurious state can also be unrelated to the original patterns; these are called **spin glass states**.

5.9 Know what is meant by network capacity

- **Network capacity** of a Hopfield network = number of memories that can reliably stored.
- Number of memories that can be stored depends on of nodes and connections
- Each time we memorize a pattern we hope to create a new energy minimum.
- However, nearby minima may merge to create one minimum at an intermediate location (spurious pattern).
- This limits storage capacity of the Hopfield net.

5.10 Explain in words what is meant by simulated annealing

- It is used to find better minima in Hopfield networks.
  - Annealing is a metallurgical process in which a material is heated an then slowly brought to a lower temperature
  - The crystalline structure of the material can reach a global minimum in this way.

6 Boltzmann machines

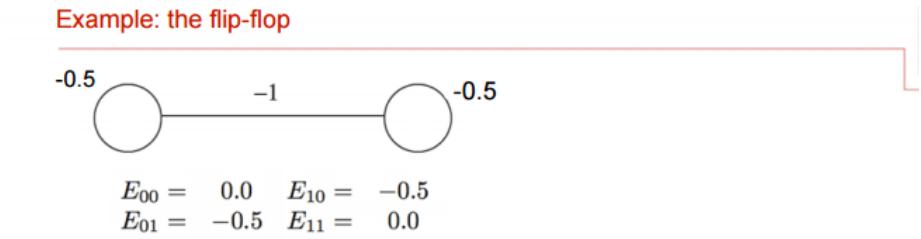
6.1 What does the update rule for a Boltzmann machine looks like

$$P(x_i = 1) = \frac{1}{1 + exp(-\frac{1}{T}(\sum_j w_{ij}x_j + \theta_i}$$

Where T acts as the (pseudo-) temperature

6.2 How to compute the entries of a state transition matrix

STM $P_T = \{P_{ij}\}$ of a Boltzmann machine with n units is the $2^n * 2^n$ matrix, whose elements p_{ij} represent the probability of a transition from state i to the state j in a single step at the temperature T.



What is the probability of transitioning from 00 to 01 at a temperature of T=1?

$$
\begin{aligned}
p_{00\to 01} &= P(\text{update 2nd unit}) p(x_2^t = 1 | x_1^{t-1} = 0, x_2^{t-1} = 0) \\
&= 0.5 \frac{1}{1 + \exp(-(\sum_j w_{ij} x_j + \theta_i)/T)} \\
&= 0.5 \frac{1}{1 + \exp(-(-1 \cdot 0 + \theta_i)/1)} \\
&= 0.5 \frac{1}{1 + \exp(-0.5)} = 0.31
\end{aligned}
$$

Figure 5: Caption

6.3 What is meant by the stable distribution of a Boltzmann machine

- The moment when the vector with probabilities doesn't chance anymore

6.4 How to write the probability of being in a state x using the Boltzmann distribution

6.5 What does the learning rule for fully observed EBMs look like

7 Unsupervised deep learning

7.1 What is meant by receptive fields and efficient coding?

- Receptive field: properties of a sensory stimulus that generates a strong response from a neuron
- Efficient coding: The goal is to represent images as faithfully and efficiently as possible using neurons with receptive fields

7.2 What is meant by unsupervised learning?

- Finding hidden structure in unlabelled data

7.3 What does the energy function of an RBM looks like?

$$E(v,h) = -b^T v - c^T h - h^T W v$$

7.4 What is meant by the factorization property of RBMs

$$P(h|v) = \prod_i P(h_i|v)$$

- where $P(h_i = 1|v) = \sigma(c_i + W_i v)$
- with $\sigma(x) = \frac{1}{1+exp(-x)}$ the sigmoid activation function

$$P(v|h) = \prod_i P(v_i|h)$$

- where $P(v_i = 1|h) = \sigma(b_i + (W^T)_i h)$
- when updating v or h each unit can be considered separately

7.5 What is meant by contrastive divergence?

- A heuristic approximation
- assumes that we start at the training examples and only take k block Gibbs sampling steps

7.6 Write down the Learning rule for RBMs based on CD-1

7.7 Explain how to do deep learning with RBMs

7.8 Explain the notion of deep auto encoders

8 RBFs (Radial Basis Function Networks) and SOMs (Self-Organizing Maps)

8.1 Understand the functional form of the Gaussian RBF network

RBF networks are feed-forward network consisting of:

- A hidden layer of radial kernels. This layer performs a non-linear transformation of input space (typically of higher dimensionality than the input space)
- An output layer of linear neurons. This layer performs linear regression to predict the desired targets
- The reason for using a non-linear transformation followed by a linear one: "A complex pattern-classification problem cast in a high-dimensional space non-linearly is more likely to be linearly seperable than in a low-dimensional space".

8.2 Understand how the RBF parameters are estimated

8.3 Know what is meant by a topographic map

- Neurobiological studies indicate that sensory inputs (motor, visual, auditory, etc.) up to higher-order semantic categories are mapped onto corresponding areas of the cerebral cortex in an orderly fashion

8.4 Be able to give an example of a topographic map in the brain

- the visual field
- the auditory field

8.5 Be able to explain in words the four stages of the self-organising map algorithm

- **Initialization**: All connection weights are initialized with small random values.

- **Competition**: For each input pattern only one neuron is the winner (winner-takes-all)
- **Cooperation**: The winning neuron determines how neighbouring neurons cooperate.
- **Adaptation**: Neurons adapt their connection weights according to their excitation

9 Recurrent neural networks

9.1 Be able to show in a diagram how an RNN differs from a feedforward neural network

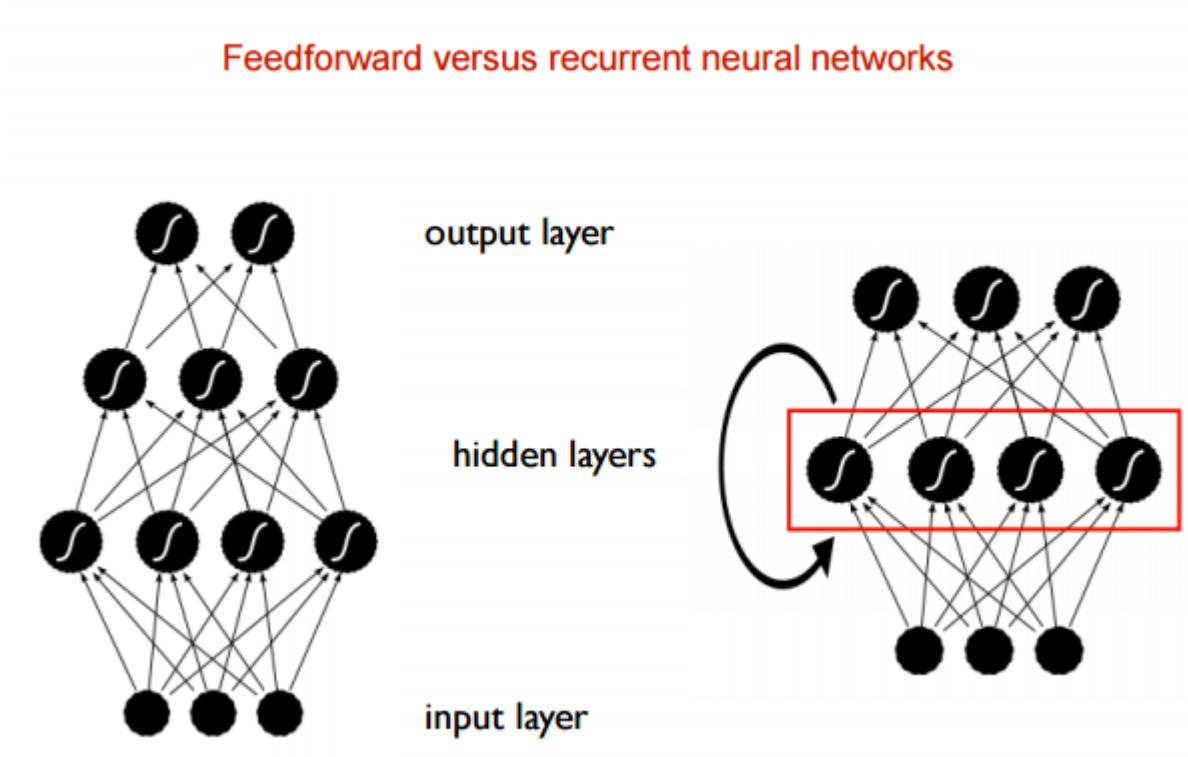


Figure 6: Caption

9.2 Be able to write down the update equation for the hidden unit states in an RNN

$$h(n) = f(W^i x(n) + W^h h(n-1))$$

9.3 Be able to explain in your own words how BPTT (Back Propagation Through Time) works (no need to remember all the equations)

- Generalizes backprop for feedforward networks to the recurrent case
- Done by unrolling the network so all cycles between units are removed
- Weights at each time point are identical.
- Steps:
  - Forward pass
    - Take a training example and push it through the unrolled network
    - At each n-th layer:
      - Read the input $x(n)$
      - Compute $h(n)$ from $x(n)$ and $h(n-1)$
      - compute $y(n)$
  - Computation of error terms
  - Weight updates

9.4 Understand the vanishing gradient issue in RNNs

- RNNs can be seen as infinitely deep neural networks with the difference that eacch layer gets its own input and output. This means that the vanishing gradient problem can have a major influence.
- This makes it hard for RNNs to model long-range dependencies that are too far in the past

9.5 Be able to explain in your own words why LSTM is useful

- It is useful because Long Short-Term Memory is an RNN architecture designed to have a better memory
- It uses linear memory cells surrounded by multiplicative gate units to store read, write and reset information
- These **memory blocks** replace the non-linear units in an RNN
- Can be trained using backpropagtion using somewhat more involved gradients

9.6 Recall that feedforward NNs implement functions, RNNs dynamical systems and NTMs programs

9.7 Know what is meant with reservoir computing

- Reservoir computing assumes the existence of a (possibly random) reservoir of connected units
- Desired outputs can be generated by reading out appropriate combinations of unit activations in the reservoir
- Used to generate sequences
- Inputs can be added to modify sequences

9.8 be able to replicate the Echo State Property

- "If the network has been run for a very long time then the current network state is uniquely determined by the history of the input and the (teacher-forced) output".

9.9 Be able to explain learning in ESNs in you own words

- For each time step, clamp output units to their observed state
- Feed the output activity into the reservoir via the backprojection weights Z (known as teacher forcing)
- Collect the reservoir states for each time step into a T x N matrix M where T is the number of time steps and N the number of hidden units
- Collect the output states into a T x L matrix T where T is the number of time steps and L the number of output units
- For these matrices the initial K time steps should be removed since they depend on an arbitrary initial network state

9.10 Recall that FORCE (First-Order Reduced and Controlled Error) learning is a more stable form of learning in ESNs (Echo State Networks)

10 Spiking neurons

10.1 Understand the difference between rate coding and temporal coding

- Rate coding:

  - Information is carried by the spike rate: the number of spikes arriving in a temporal interval
  - Can be expressed as a real number
  - All neural network models discussed so far make use of a rate code
  - Adrian showed that the firing rate of stretch receptor neurons in the muscles is related to the force applied to the muscle
  - Information is carried by the joint activity of a pool of neurons (population coding) can implement an instantaneous rate code (spike density code)
  - Transmits $log_2(n+1)$ bits, since only $n+1$ different events can occur in a fixed T ms time window.

- Temporal coding:
- Information is carried by individual spikes (action potentials)
- Exact spike timing has been shown to underly e.g. echo-localization in bats
- For a 1 ms precision, $nlog_2(T)$ bits can be transmitted in a T ms time window

10.2 Be able to explain the role of components of the equivalent circuit diagram for the Hodgkin & Huxley model

10.3 Explain the trade-off between biological realism and computational efficiency of spiking neuron models