Knowledge Processing with Neural Networks

Lecture 2: The Neuron and its Models

http://www.informatik.uni-hamburg.de/WTM/

Overview

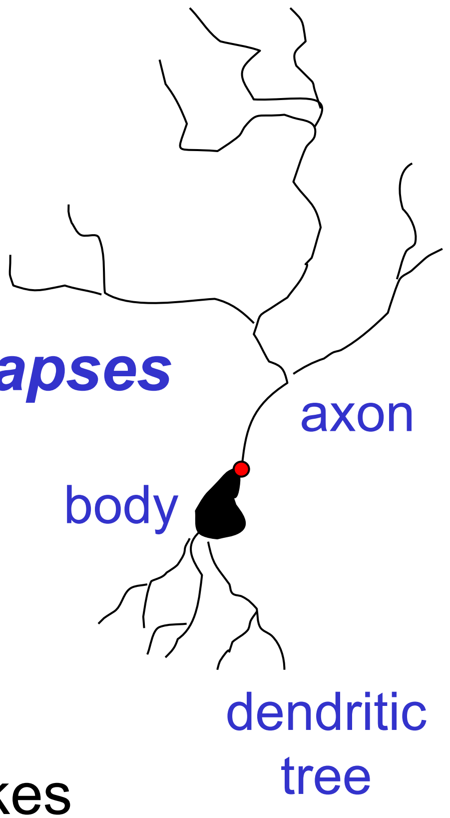
- Introduction, motivation, neural architectures
- General Background Sources
  - S. Haykin: Neural networks and Learning Machines, 2009
  - R. Rojas: Neural networks 1996 (book online available) http://page.mi.fu-berlin.de/rojas/neural/neuron.pdf
- Thanks to first three Introduction to neural networks slide sets from Jeff Hinton for introduction material, which is here enhanced with some images and video footage for this lecture

The goals of neural computation

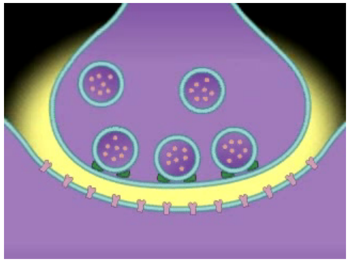
- To understand how the brain actually works
- To understand a new style of computation
  - Inspired by neurons and their adaptive connections
  - Very different style from sequential computation
    - should be good for things that brains are good at (e.g. speech. vision, navigation)
    - Should be bad for things that brains are bad at (e.g. 23 x 71)
- To solve practical problems by developing novel learning algorithms
  - Learning algorithms can be very useful even if they have nothing to do with how the brain works

A typical cortical neuron

- Gross physical structure:
  - There is one axon that branches
  - There is a ***dendritic*** tree that collects input from other neurons
- Axons typically contact dendritic trees at ***synapses***
  - A spike of activity in the axon causes charge to be injected into the post-synaptic neuron
- Spike generation:
  - There is an ***axon*** that generates outgoing spikes whenever enough charge has flowed in at synapses to depolarize the cell membrane

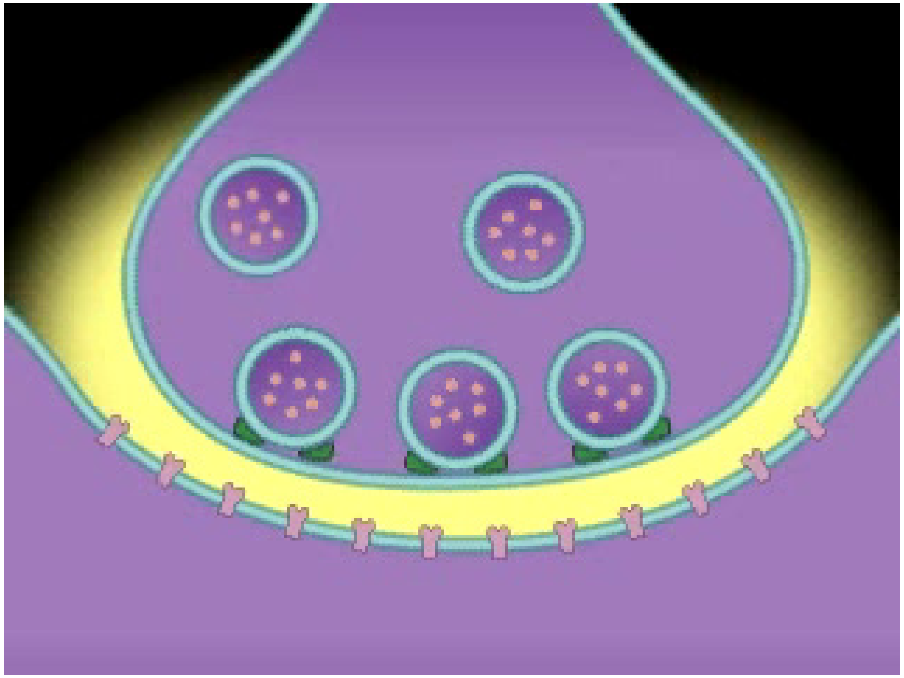


Synapses



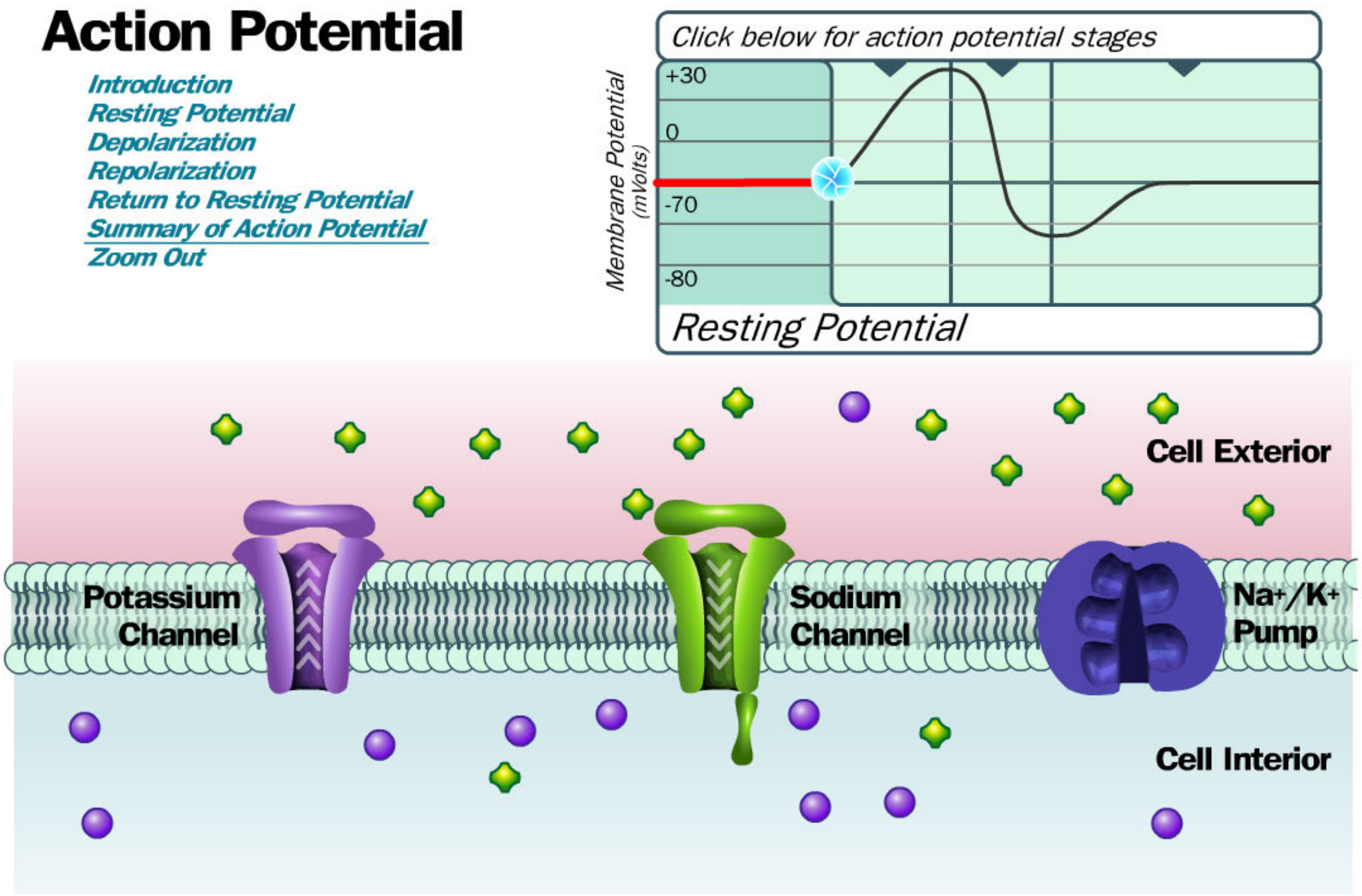
- When a spike travels along an axon and arrives at a synapse it causes vesicles of ***transmitter*** chemical to be released
  - There are several kinds of transmitter
- The transmitter molecules diffuse across the ***synaptic cleft*** and bind to receptor molecules in the membrane of the post-synaptic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

What biological synapses do…



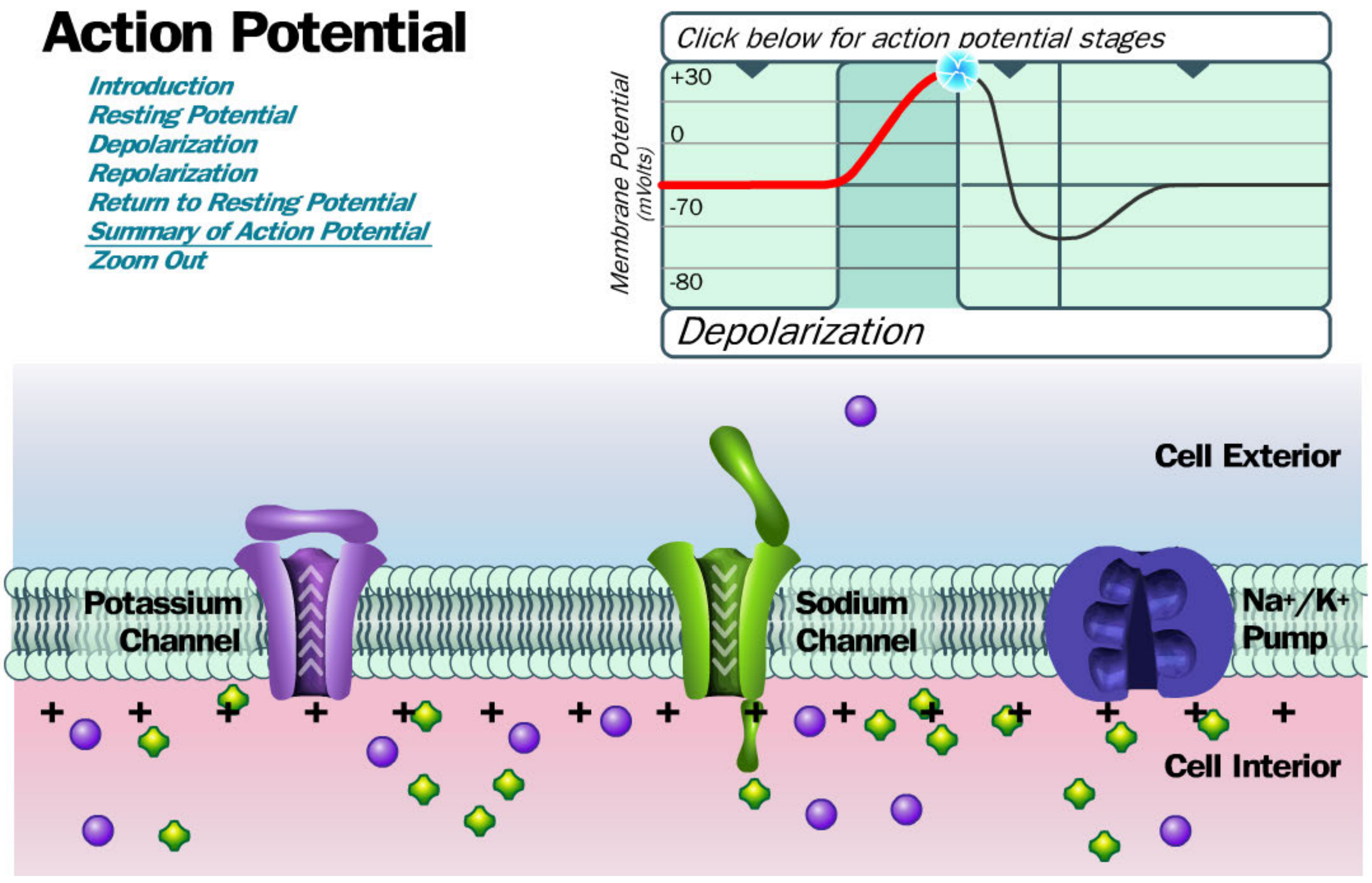
Transmitters are "universal", e.g. they work in the human brain the same as transmitters in the mouse brain

Action Potential: Resting Potential



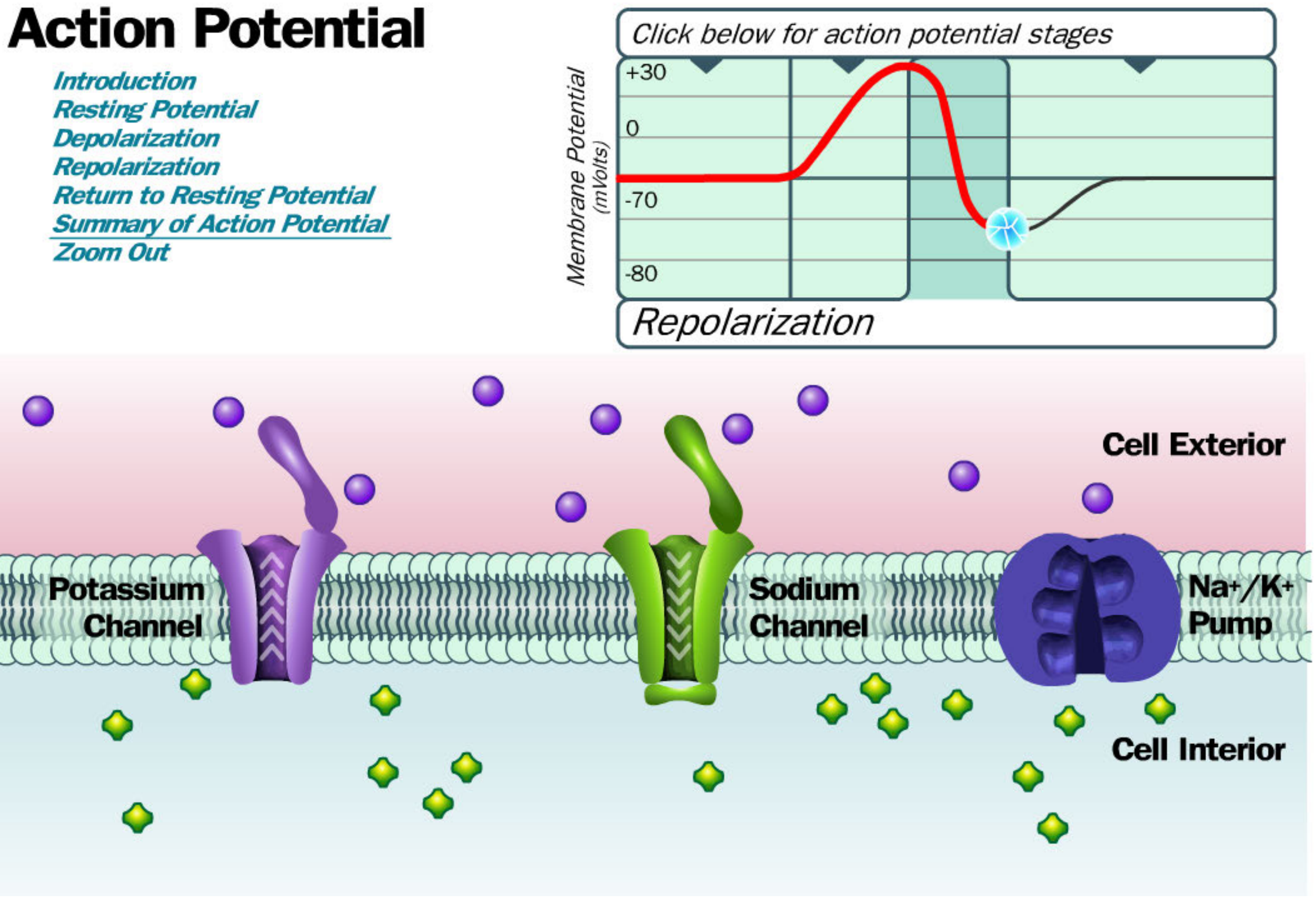
[http://outreach.mcb.harvard.edu/animations/actionpotential_short.swf]

Action Potential: Depolarization



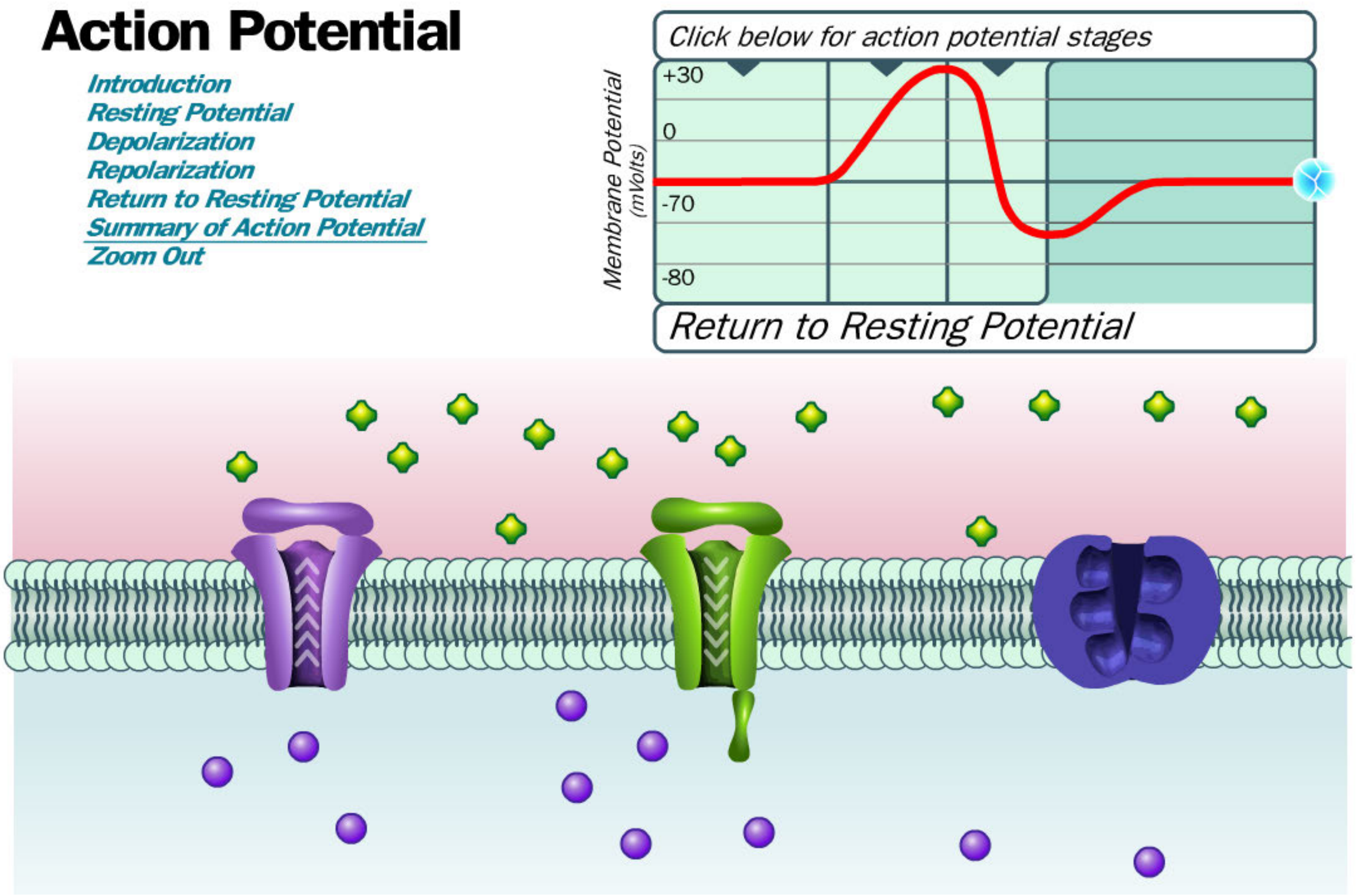
[http://outreach.mcb.harvard.edu/animations/actionpotential_short.swf]

Action Potential: Repolarization



[http://outreach.mcb.harvard.edu/animations/actionpotential_short.swf]

Modelling Neurons: Resting Potential



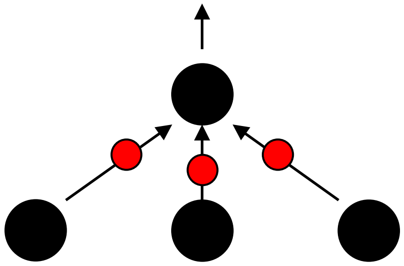
[http://outreach.mcb.harvard.edu/animations/actionpotential_short.swf]

Hodgkin-Huxley Model

- Take a real neuron (from a giant squid)
- Take lots of measurements of chemical concentrations
- Monitor the ***membrane potential***
- Write down the differential equations
- Model is too time consuming for larger networks

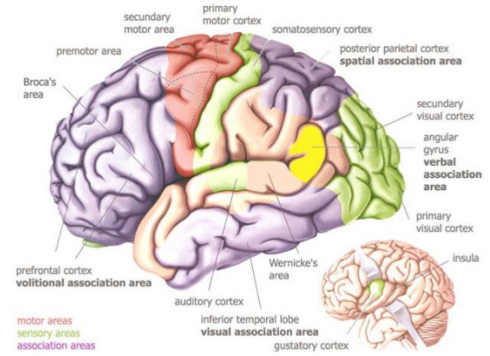
How the brain works

- Each ***neuron receives inputs*** from other neurons
  - Some neurons also connect to receptors
  - Cortical neurons use ***spikes*** to communicate
  - The timing of spikes is important
- The effect of each input line on the neuron is controlled by a ***synaptic weight***
  - The weights can be positive or negative



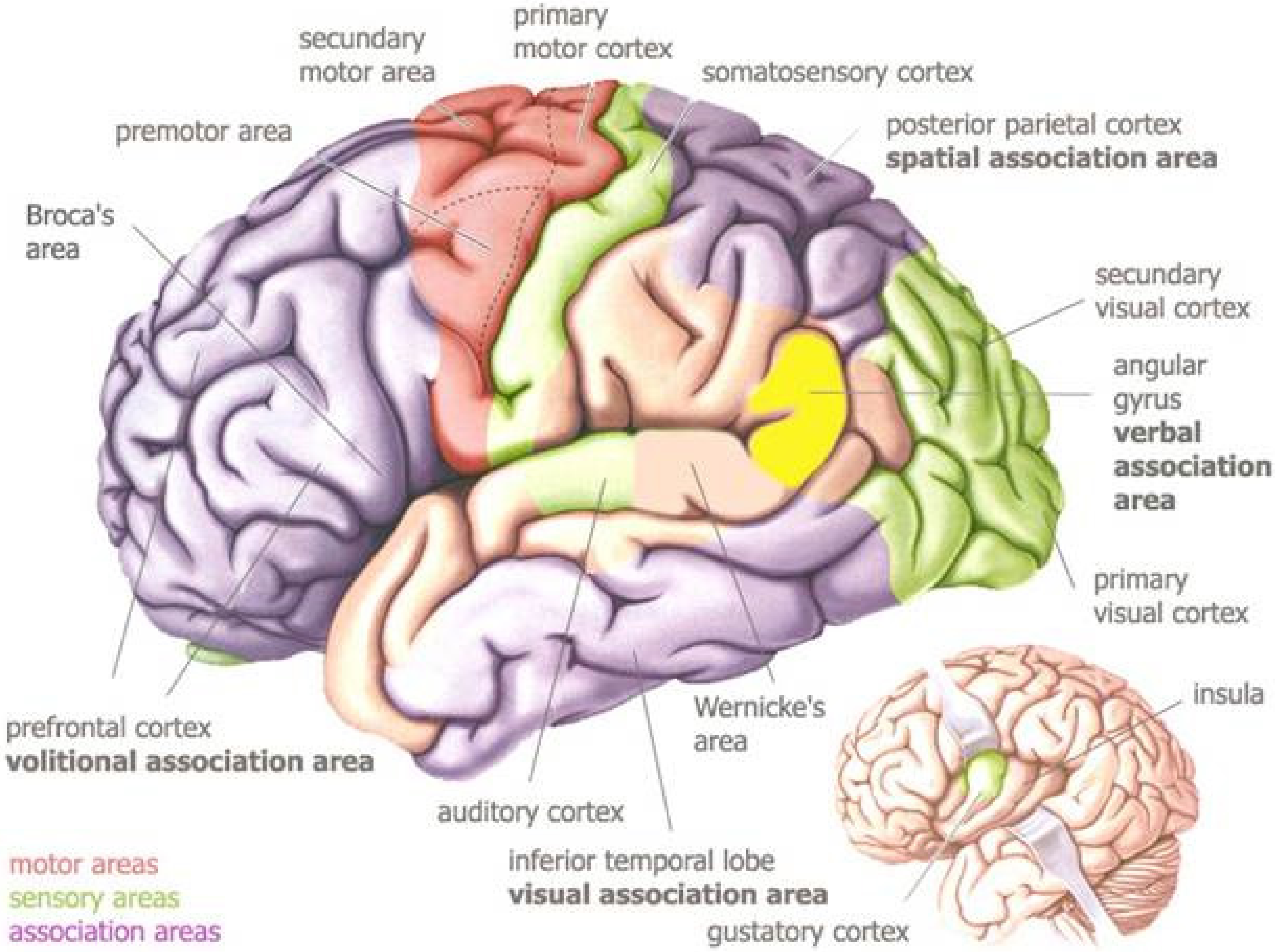
- The synaptic ***weights adapt*** so that the whole network learns to perform useful computations
  - Recognizing objects, understanding language, making plans, controlling the body
- You have about 10^{12} neurons each with about 10^3 weights
  - A huge number of weights can affect the computation in a very short time.

Modularity and the brain



- Parts of the cortex do ***different jobs***.
  - Local damage to the brain has specific effects
  - Specific tasks increase the blood flow to specific regions.
- But cortex looks ***about the same*** all over.
  - Early brain damage makes functions relocate
- Cortex is made of ***general purpose hardware*** that has the ability to turn into special purpose hardware in response to experience.
  - This gives rapid ***parallel*** computation plus flexibility.
  - **Conventional computers** get flexibility by having stored programs, but this requires very fast central processors that perform large computations sequentially.

Modularity and the brain



www.marketoracle.co.uk/images/dorn_4_11_07b.jpg

Idealized neurons

- To ***model things*** we have to ***idealize*** them (e.g. atoms)
  - Idealization removes complicated details that are not essential for understanding the main principles
  - Allows us to apply mathematics and to make analogies to other, familiar systems.
  - Once we understand the basic principles, its easy to add complexity to make the model more faithful
- It is often worth ***understanding models*** that are known to be simplified (but we must not forget that they are simplified!)
  - E.g. neurons that communicate real values rather than discrete spikes of activity.

Linear neurons

- These are simple but computationally limited
  - If we can make them learn we **may** get insight into more complicated neurons

$$y = b + \sum_i x_i w_i$$

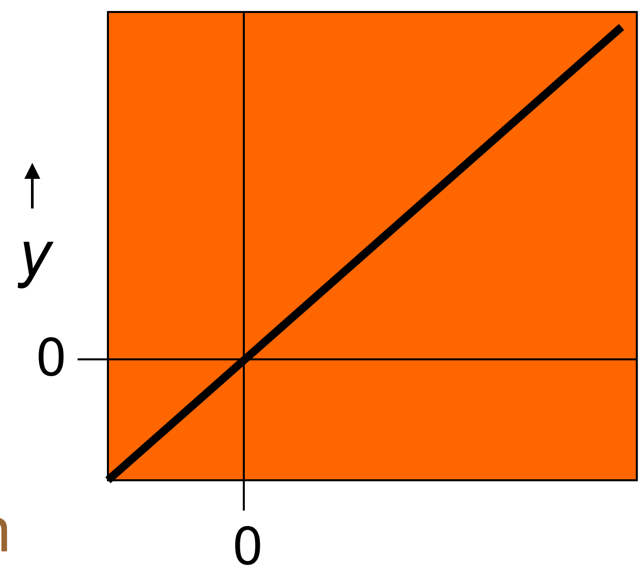
bias

i th input

output

index over input connections

weight on ith input



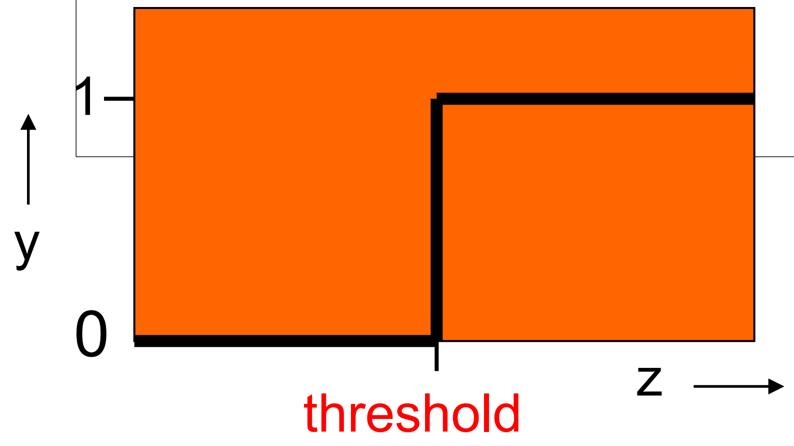
$$b + \sum_i x_i w_i \rightarrow$$

Binary threshold neurons

- McCulloch-Pitts (1943): influenced von Neumann!
  - First compute a ***weighted sum*** of the inputs from other neurons
  - Then send out a fixed size spike of activity if the weighted sum exceeds a ***threshold***.
  - McCulloch & Pitts: each spike is ***like the truth value*** of a proposition and each neuron combines truth values to compute the truth value of another proposition.

$$z = \sum_i x_i w_i$$

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



Linear threshold neurons

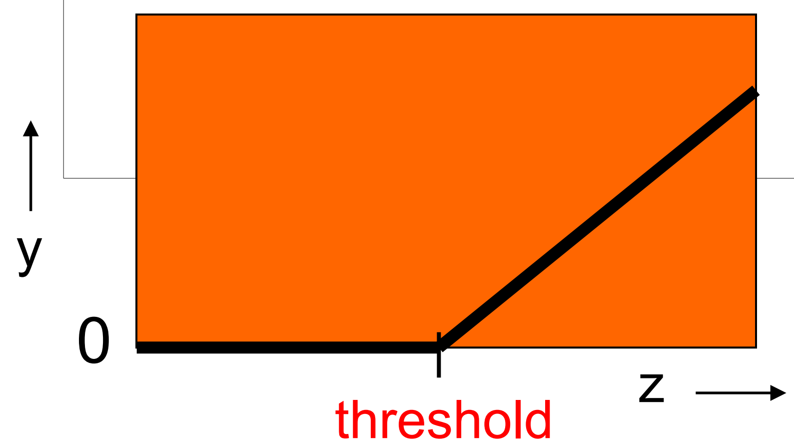
These have a confusing name.

They compute a linear weighted sum of their inputs

The output is a non-linear function of the total input

$$z_j = b_j + \sum_i x_i w_{ij}$$

$$y_j = \begin{cases} z_j & \text{if } z_j \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

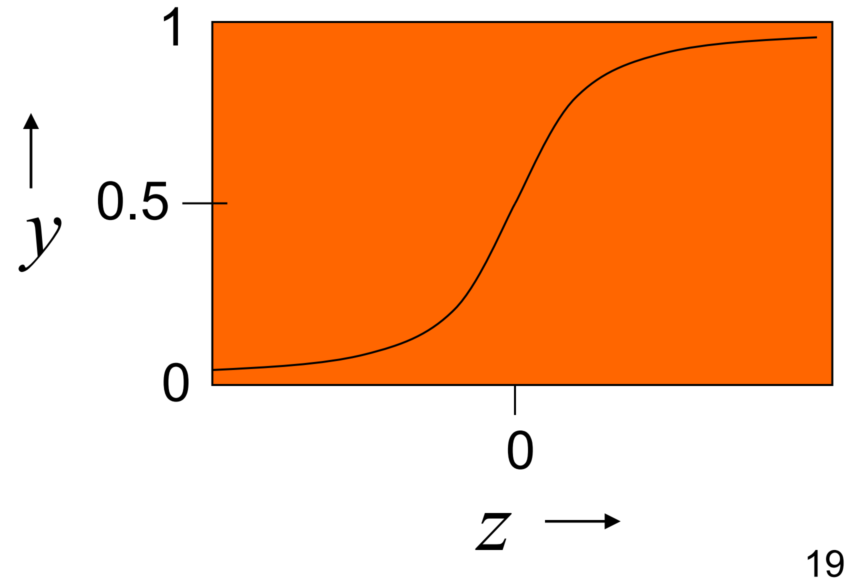


Sigmoid neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - Typically they use the ***logistic function***

$$z = b + \sum_i x_i w_i$$

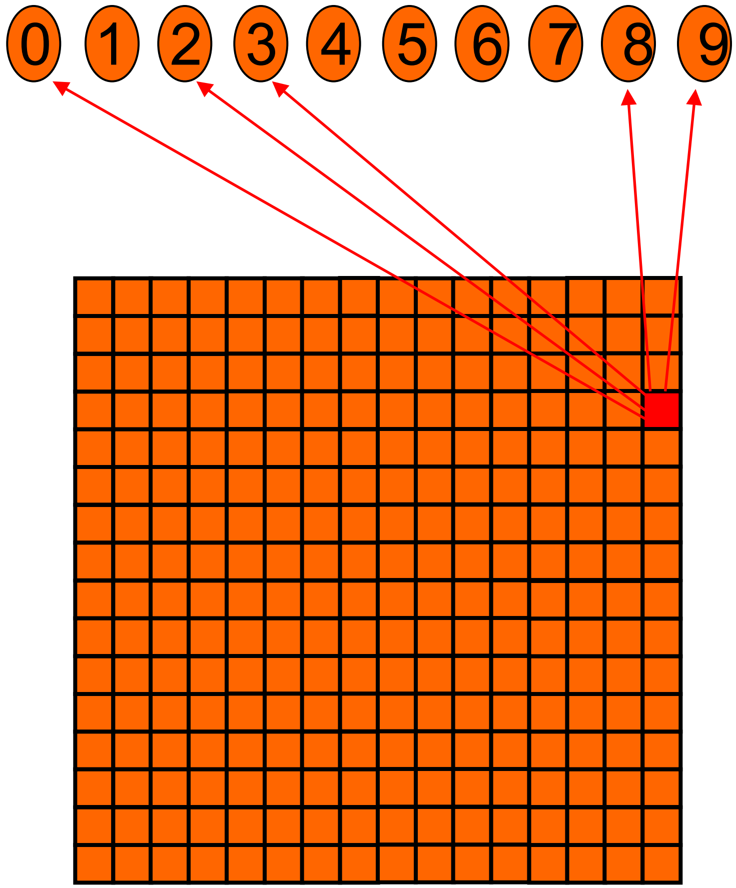
$$y = \frac{1}{1 + e^{-z}}$$



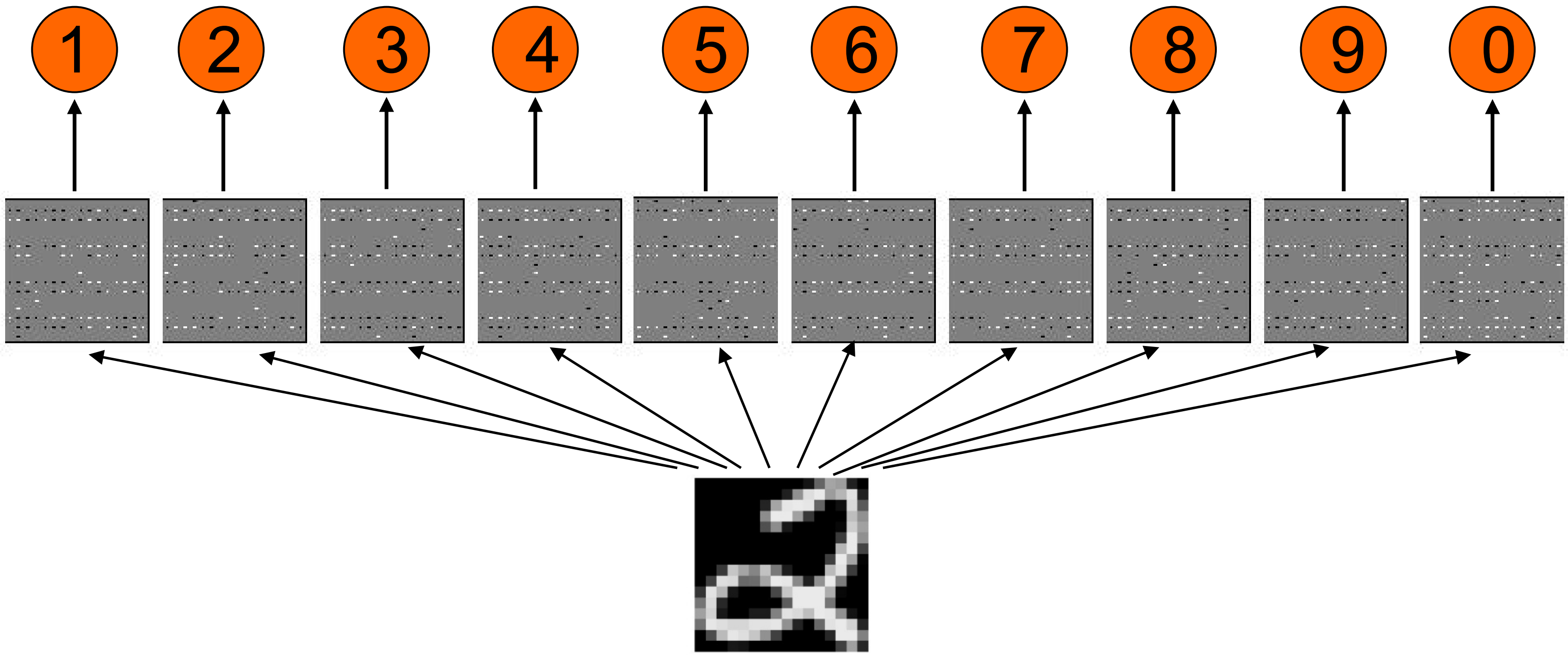
- If we treat y as a ***probability*** of producing a spike, we get ***stochastic binary neurons***.

A very simple way to recognize handwritten shapes

- Consider a neural network with two layers of neurons.
  - neurons in the top layer represent known shapes.
  - neurons in the bottom layer represent pixel intensities.
- A pixel gets to vote if it has ink on it.
  - Each inked pixel can vote for several different shapes.
- The shape that gets the most votes wins.



How to learn the weights

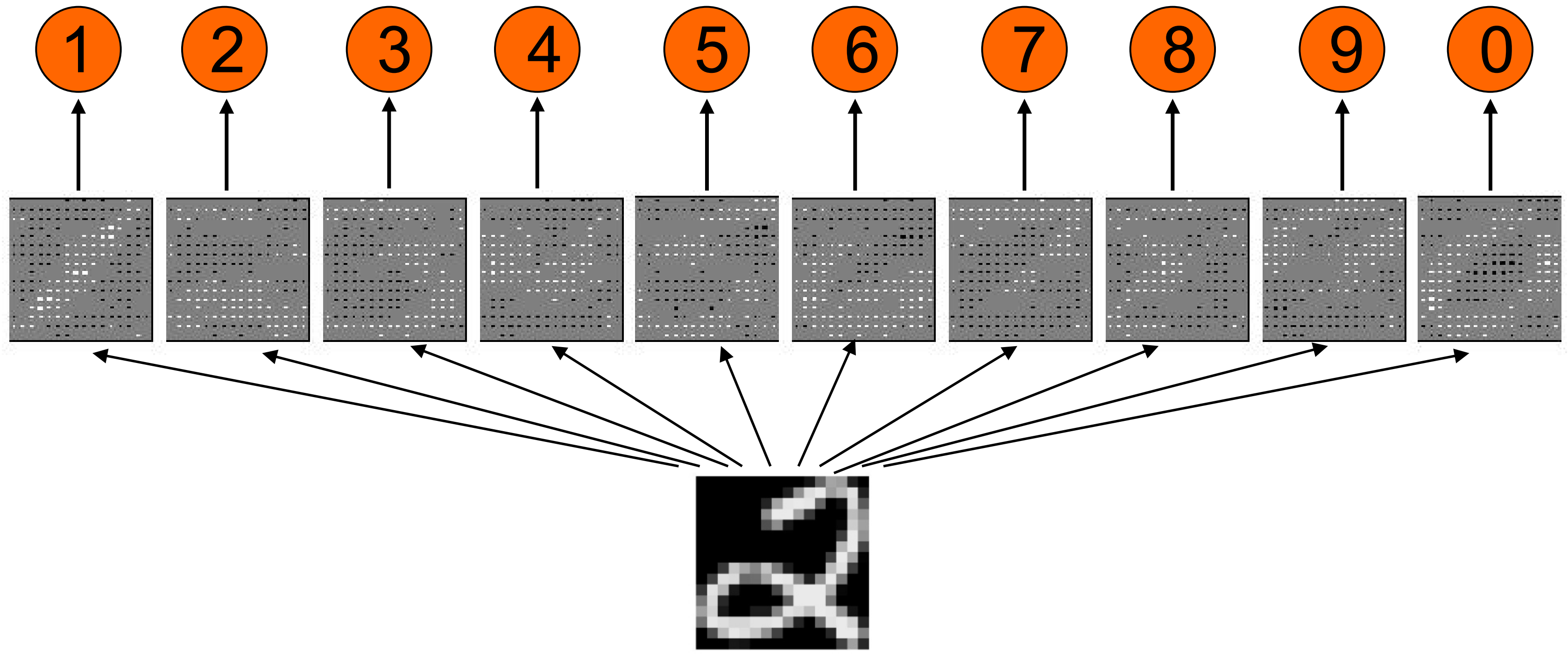


The image

Show the network an image and increment the weights from active pixels to the correct class. (desired class)

Then decrement the weights from active pixels to whatever class the network guesses (computed class).

How to learn the weights

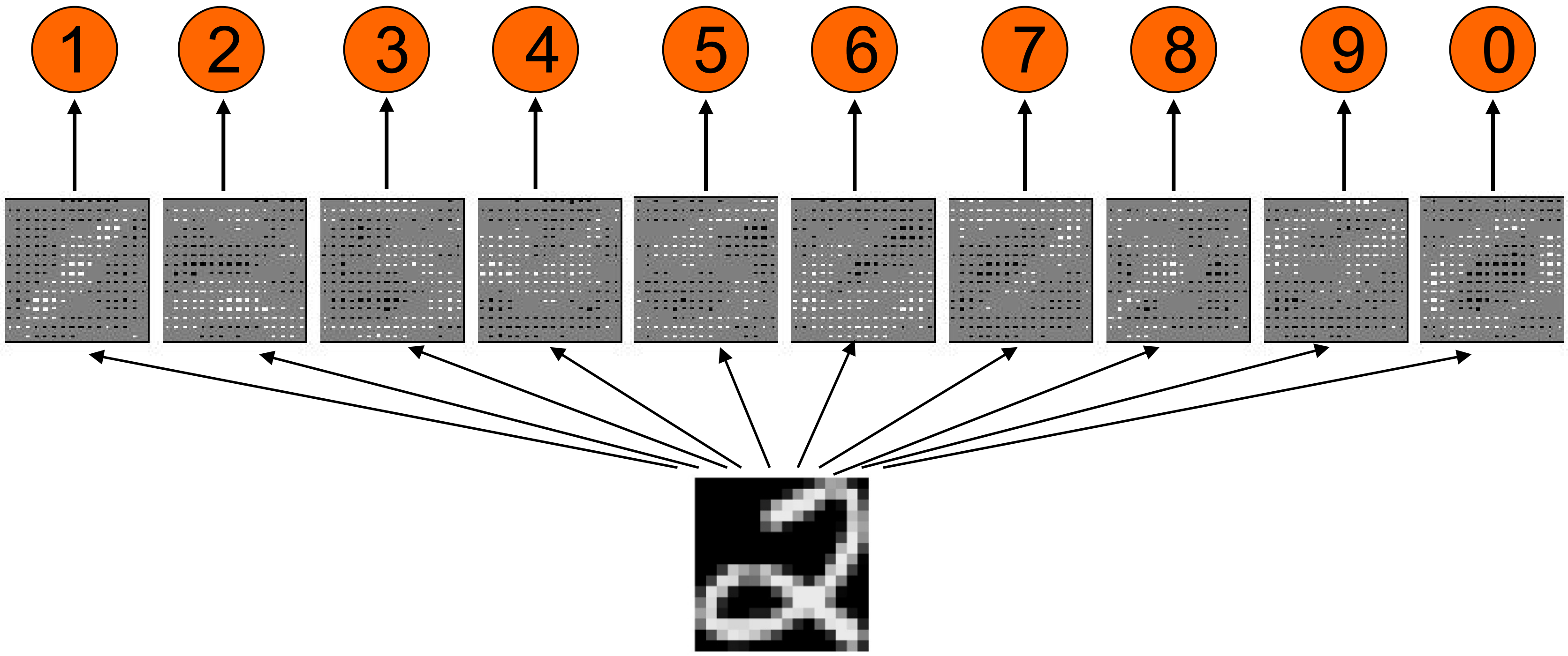


The image

Show the network an image and increment the weights from active pixels to the correct class.

Then decrement the weights from active pixels to whatever class the network guesses.

How to learn the weights

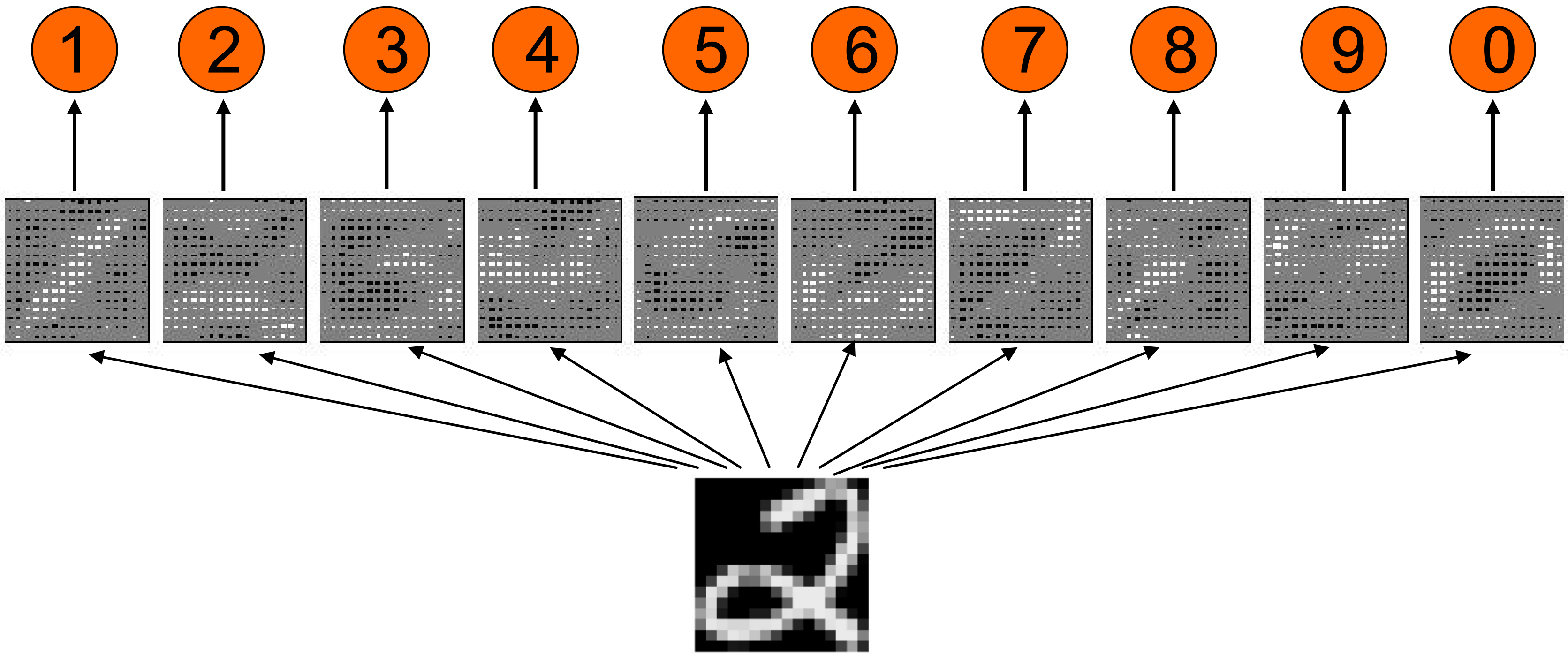


The image

Show the network an image and increment the weights from active pixels to the correct class.

Then decrement the weights from active pixels to whatever class the network guesses.

How to learn the weights

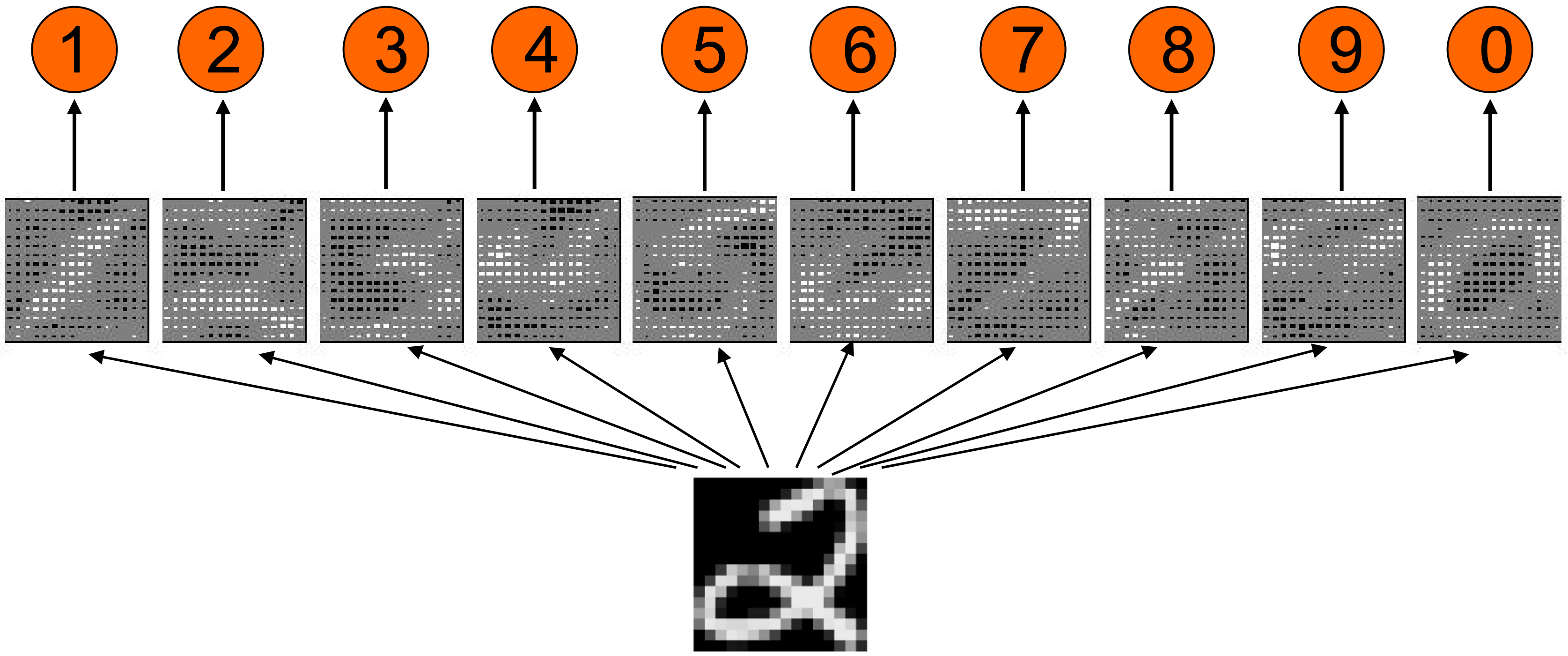


The image

Show the network an image and increment the weights from active pixels to the correct class.

Then decrement the weights from active pixels to whatever class the network guesses.

How to learn the weights

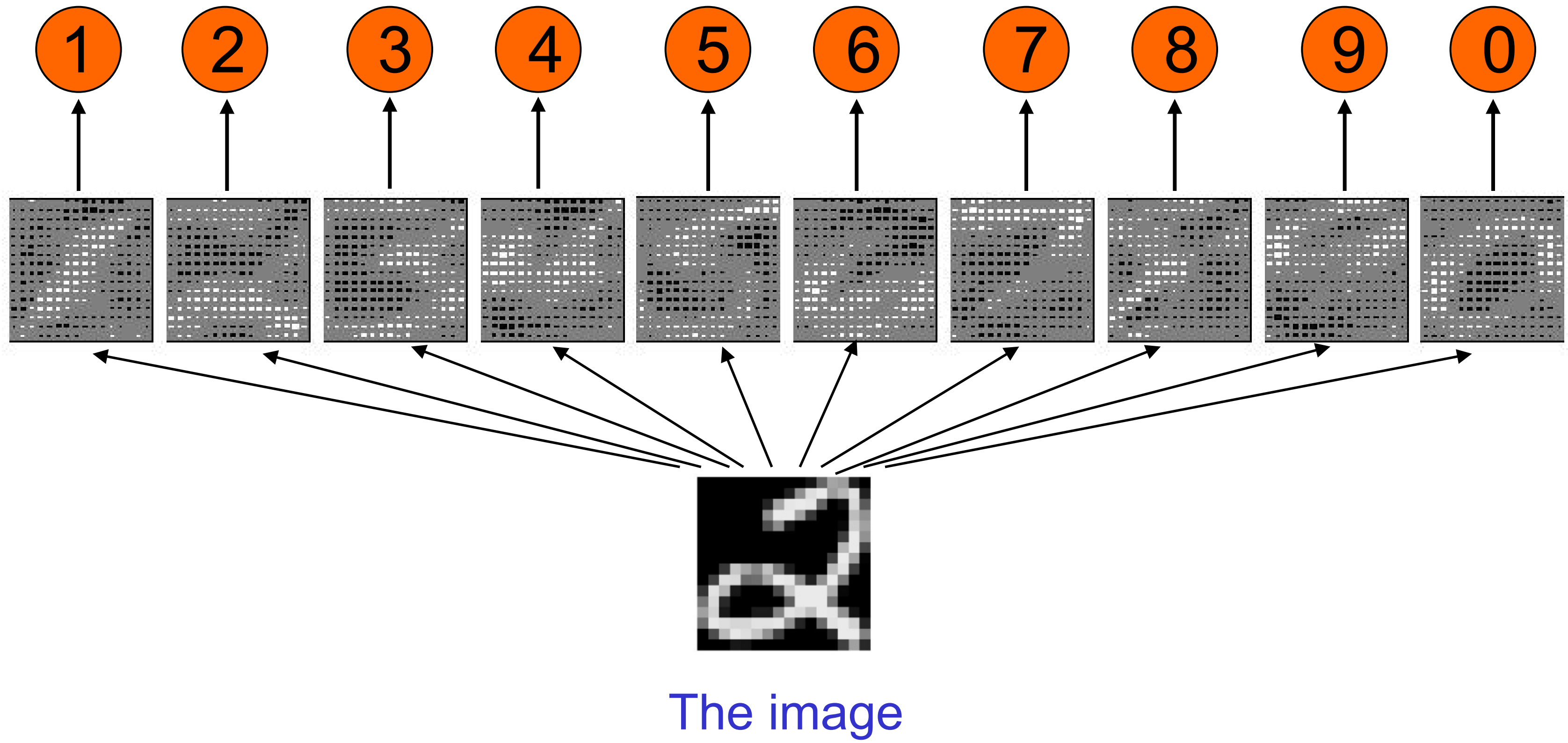


The image

Show the network an image and increment the weights from active pixels to the correct class.

Then decrement the weights from active pixels to whatever class the network guesses.

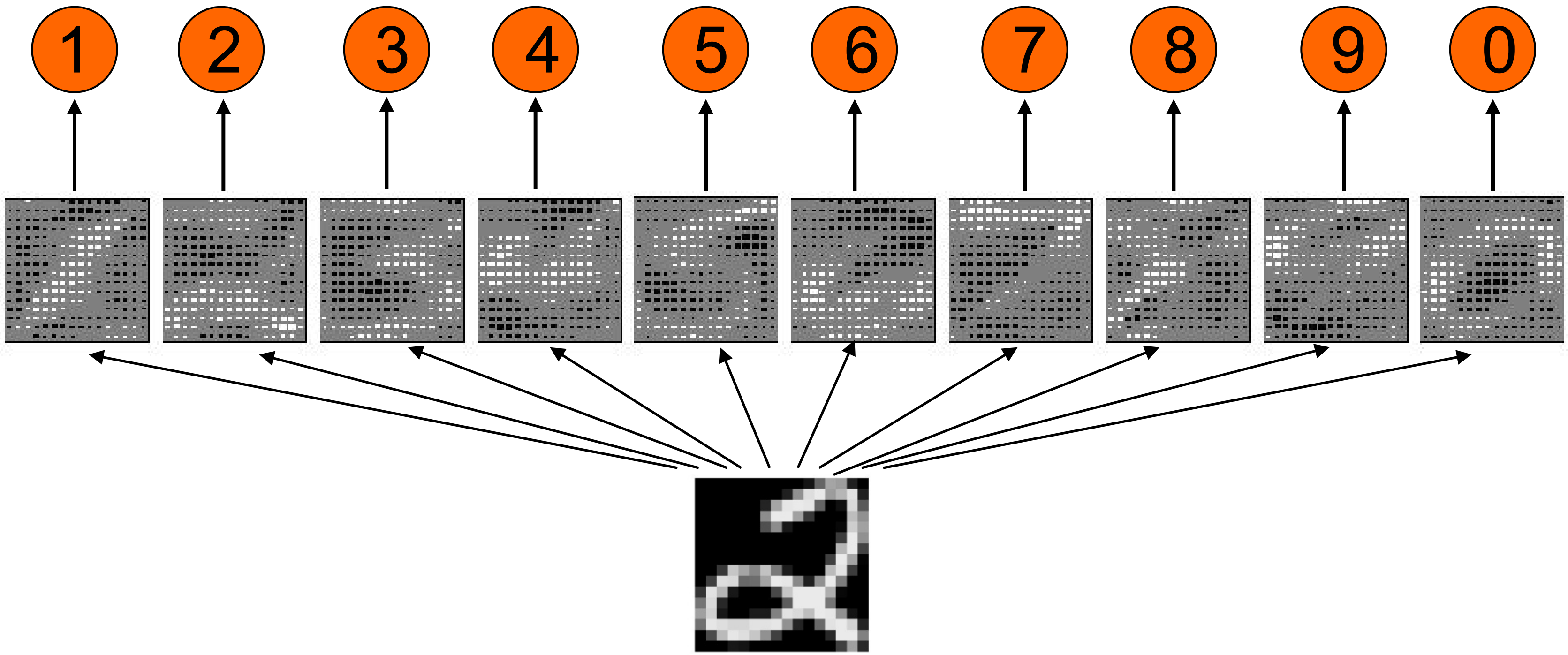
How to learn the weights



Show the network an image and increment the weights from active pixels to the correct class.

Then decrement the weights from active pixels to whatever class the network guesses.

The learned weights



The image

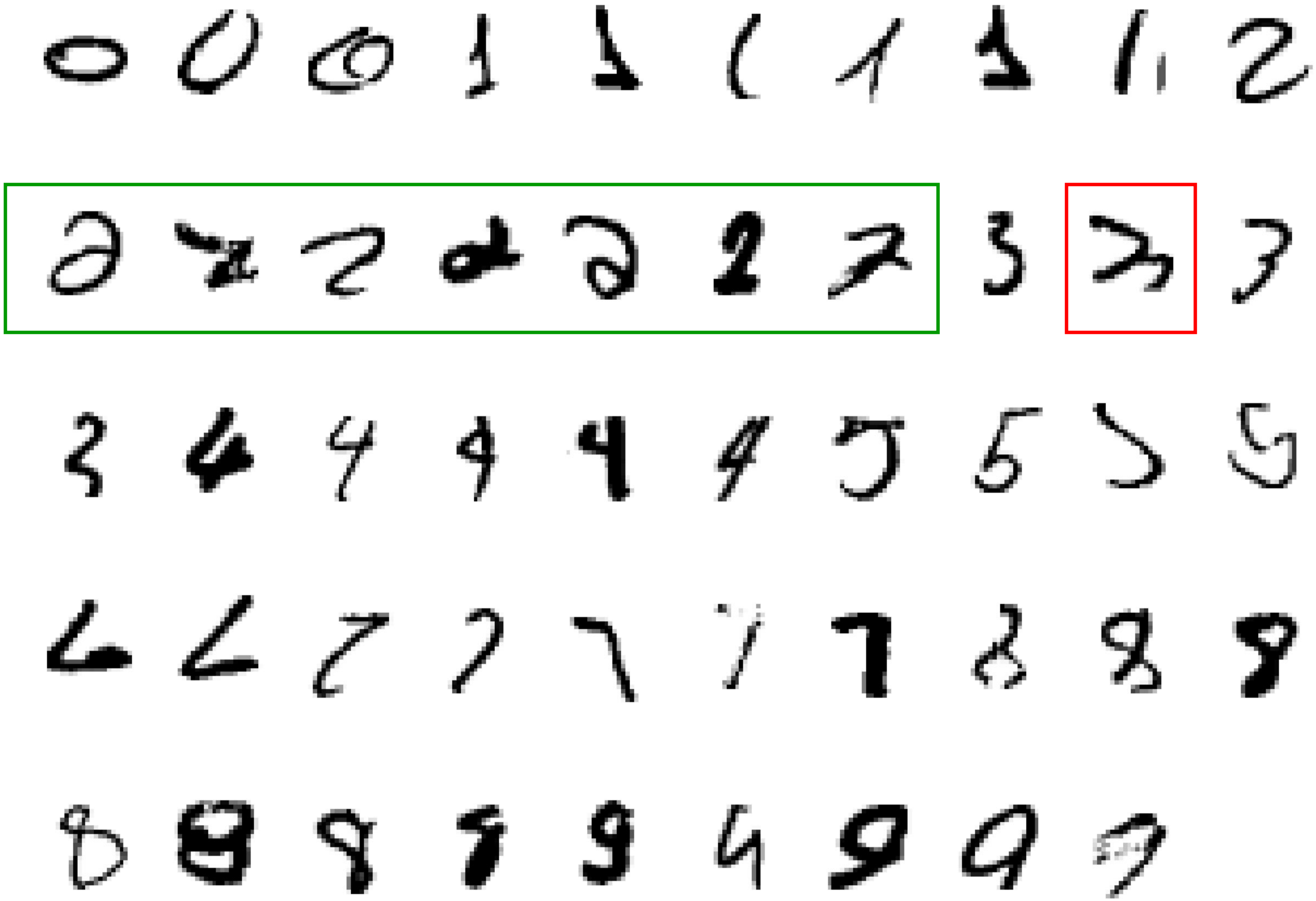
The precise details of the learning algorithm will be explained in future lectures.

Why the simple system does not work

- A two layer network with a single winner in the top layer is equivalent to having a ***rigid template*** for each shape.
  - The winner is the template that has the biggest overlap with the ink.

- The ways in which ***shapes vary*** are much too complicated to be captured by simple template matches of whole shapes.
  - To capture all the allowable variations of a shape we ***need to learn the features*** that it is composed of.

Examples of handwritten digits that need to be recognized correctly the first time they are seen



Supervised Learning

- Each ***training case*** consists of an input vector x and a desired output y (there may be multiple desired outputs but we will ignore that for now)
  - Regression: Desired output is a real number
  - Classification: Desired output is a class label (1 or 0 is the simplest case).
- ***Learning*** usually means adjusting the parameters to reduce the discrepancy between the desired output on each training case and the actual output produced by the model.

Linear neurons

- The neuron has a real-valued output which is a weighted sum of its inputs

$$\hat{y} = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

weight vector

input vector

Neuron's estimate of the desired output

- The aim of learning is to ***minimize the discrepancy between the desired output and the actual output***
  - How do we measure the discrepancies?
  - Do we update the weights after every training case?
  - Why don't we solve it analytically?

A motivating example

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and beer.
  - You get several portions of each
- The cashier only tells you the total price of the meal
  - After several days, you should be able to figure out the price of each portion.
- Each meal price gives a linear constraint on the prices of the portions:

$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{beer} w_{beer}$$

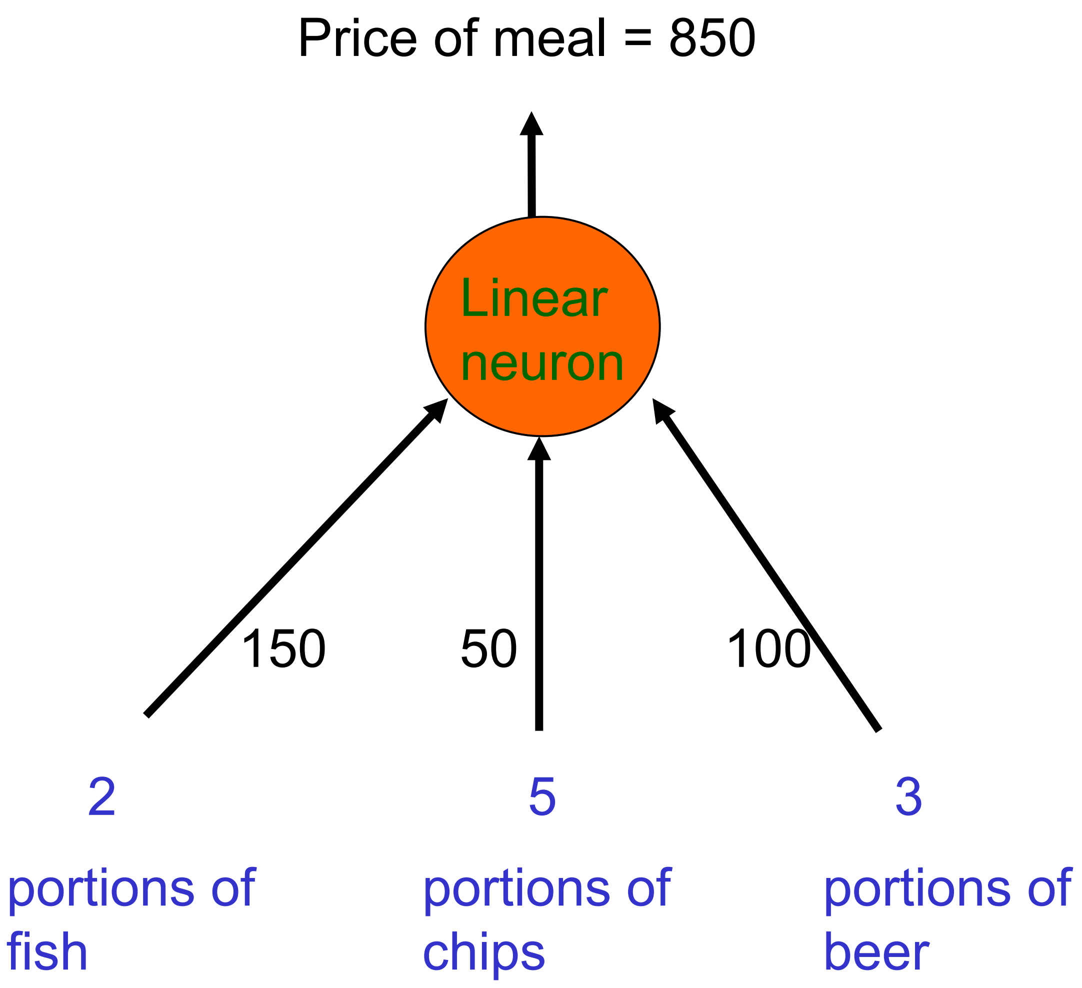
Two ways to solve the equations

- The obvious approach is just to solve a set of ***simultaneous linear equations***, one per meal.
- But we want a method that could be implemented in a ***neural network***.
- The prices of the portions are like the weights in a linear neuron.

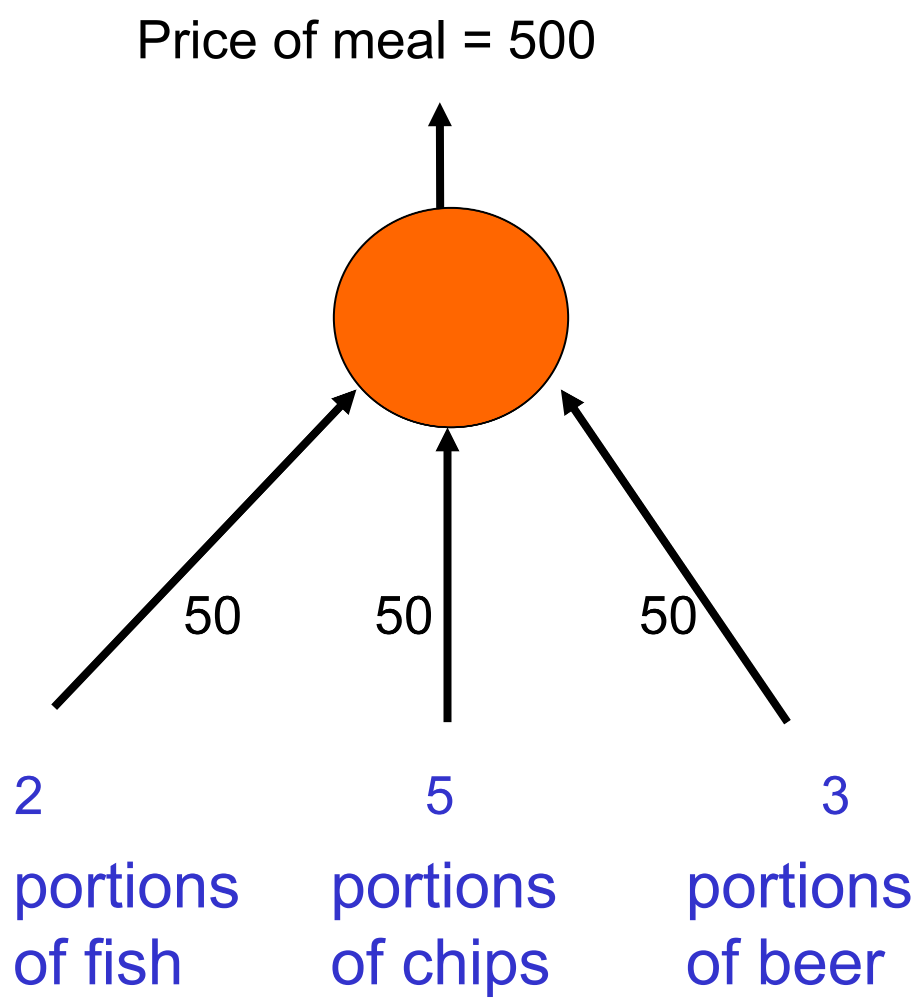
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{beer})$$

- We will start with guesses for the weights and then ***adjust the guesses to give a better fit*** to the prices given by the cashier.

The cashier's brain



A model of the cashier's brain with arbitrary initial weights



- Residual error = 350
- The learning rule is:

$$\Delta w_i = \varepsilon \, x_i \, (y - \hat{y})$$

- With a learning rate ε of 1/35, the weight changes are +20, +50, +30
- This gives new weights of 70, 100, 80
- Computed price then 880 which is closer to the correct price

Behaviour of the iterative learning procedure

- Do the updates to the weights always make them get closer to their correct values?

  No! - Notice that the weight for chips got worse!
- Does the online version of the learning procedure eventually get the right answer?

  Yes, if the learning rate gradually decreases in the appropriate way.
- How quickly do the weights converge to their correct values?

  It can be very slow if two input dimensions are highly correlated (e.g. ketchup and chips).
- Can the iterative procedure be generalized to much more complicated, multi-layer, non-linear nets? YES! To come…

Deriving the delta rule

- Define the error as the squared residuals summed over all training cases:

$$E = \sum_n \tfrac{1}{2}(y_n - \hat{y}_n)^2$$

- Now differentiate to get error derivatives for weights

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial \hat{y}_n}{\partial w_i} \boxed{\frac{\partial E_n}{\partial \hat{y}_n}}$$

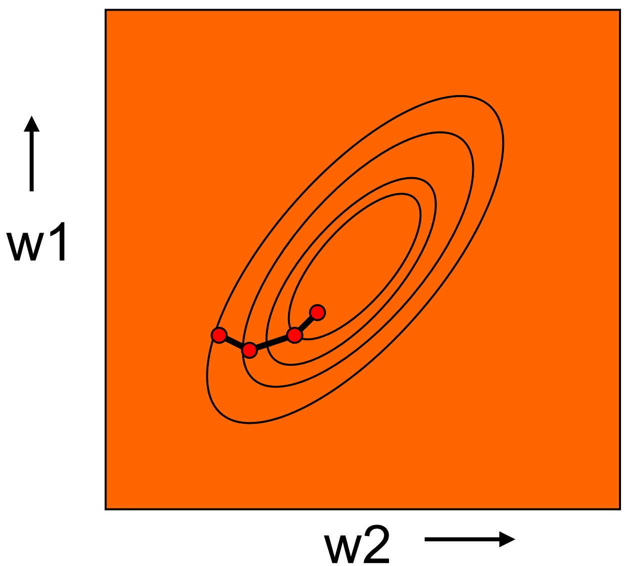
$$= \boxed{-} \sum_n x_{i,n} \boxed{(y_n - \hat{y}_n)}$$

- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases

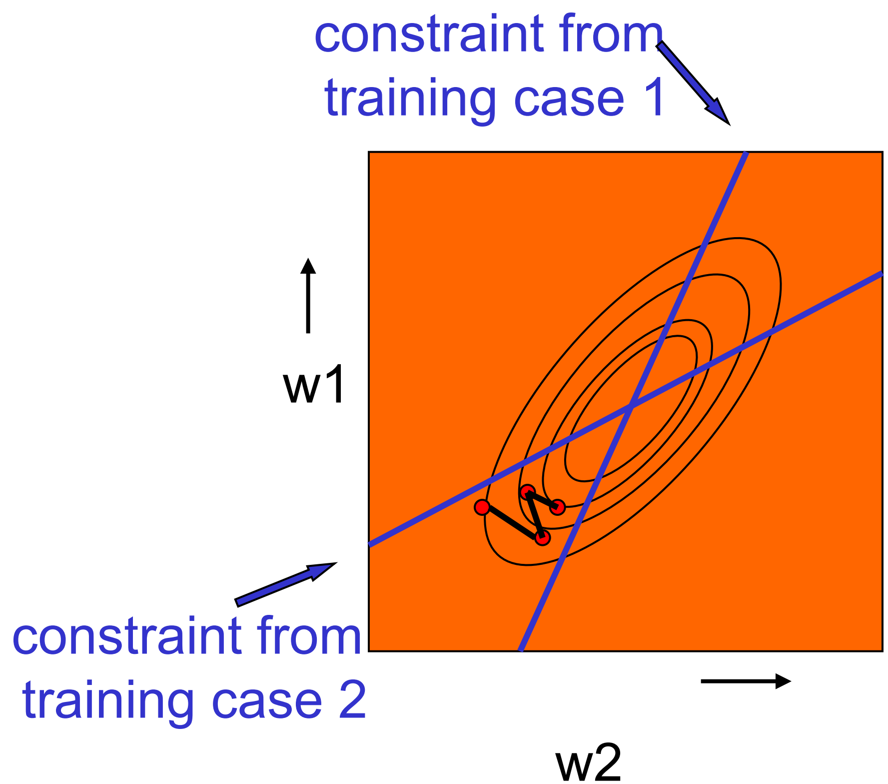
$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i}$$

Batch versus online (incremental) learning

- Batch learning does steepest descent on the error surface
- Update after whole epoch over all training cases



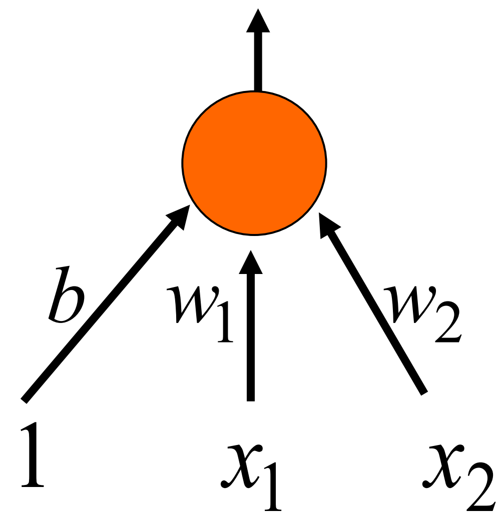
- Online learning zig-zags around the direction of steepest descent



Adding biases for thresholds

- A linear neuron is a ***more flexible*** model if we include a bias.
- We can avoid having to figure out a separate learning rule for the bias by using a trick:
  - A bias is exactly equivalent to a weight on an extra input line that always has an activity of 1.

$$\hat{y} = b + \sum_i x_i w_i$$

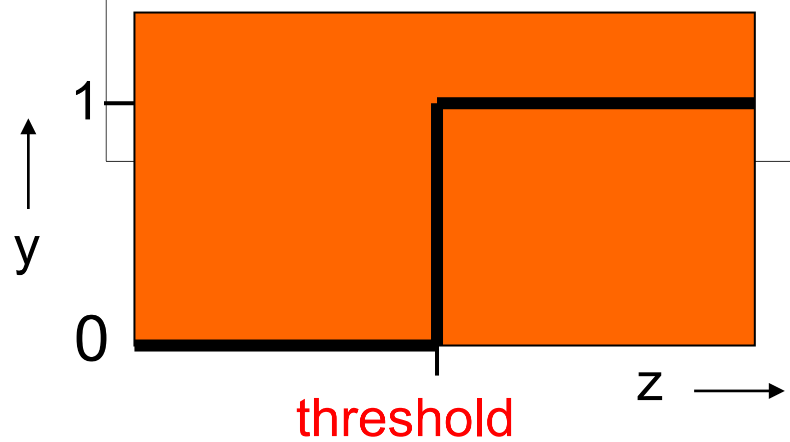


Special case: Binary threshold neurons revisited

- McCulloch-Pitts (1943)
  - First compute a weighted sum of the inputs from other neurons
  - Then output a 1 if the weighted sum exceeds the threshold.

$$z = \sum_i x_i w_i$$

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



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 suitable set of weights if any such set exists.

What binary threshold neurons can and cannot learn

- A binary threshold output unit cannot tell if two single bit numbers are the same!

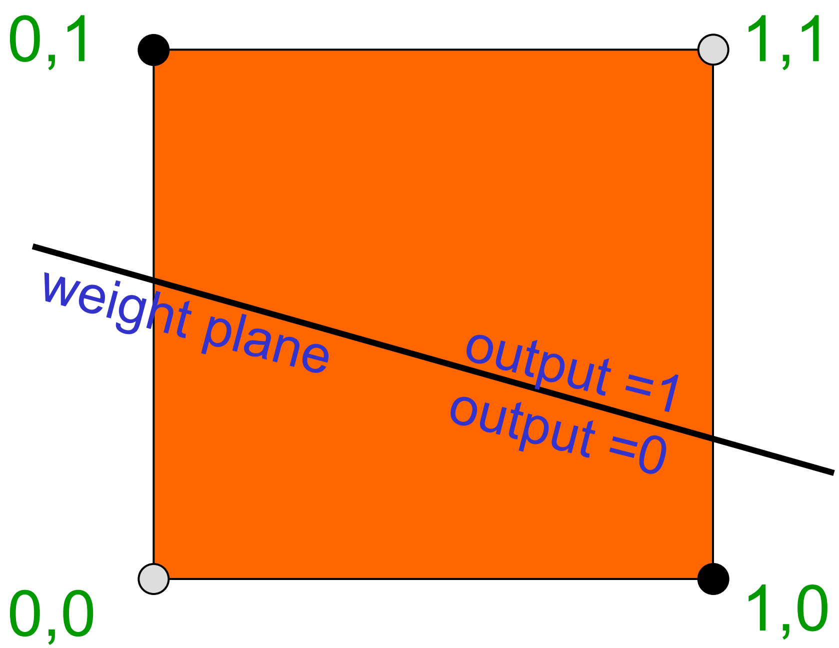
  Same: (1,1) → 1; (0,0) → 1

  Different: (1,0) → 0; (0,1) → 0

- The four input-output pairs give four inequalities that are impossible to satisfy:

$$w_1 + w_2 \geq \theta, \quad 0 \geq \theta$$
$$w_1 < \theta, \qquad w_2 < \theta$$

Data Space
(not weight space)



The positive and negative cases cannot be separated by a plane

Reading

- Background
  - R. Rojas: Neural networks 1996 (book online available)
    http://page.mi.fu-berlin.de/rojas/neural/neuron.pdf

- Knowledge Technology website
  http://www.informatik.uni-hamburg.de/WTM/