Block Practical: Connectionist models and cognitive processes

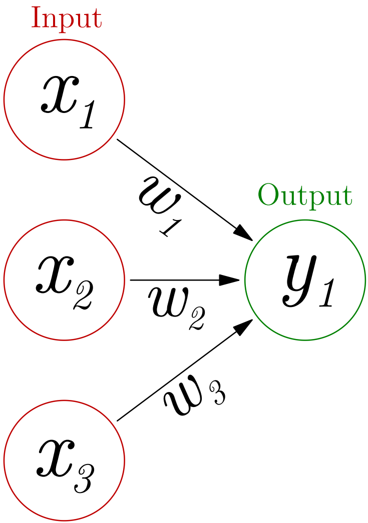
Part 3: **Feedforward Networks**

Olivia Guest

2-layer Perceptron

Simplest form of network

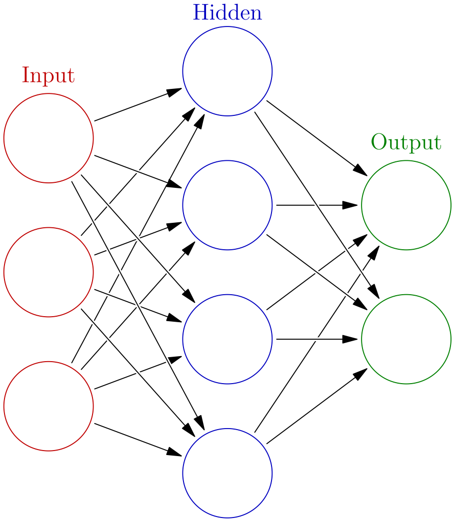
- Linearly separable datasets only
- Cannot solve XOR-like problems
- We need something more human-like in problem solving abilities!



3-layer Perceptron

Just add hidden units!

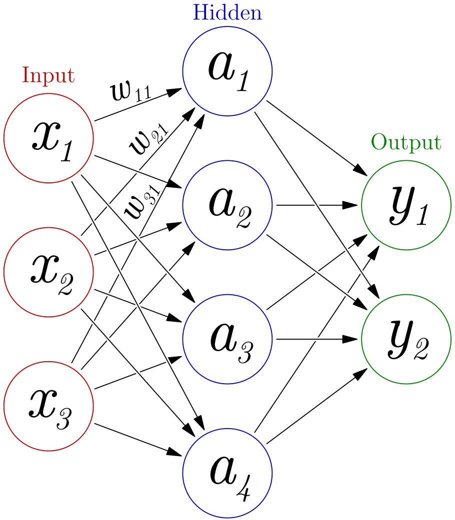
- Can solve most problems



3-layer Perceptron

Just add hidden units!

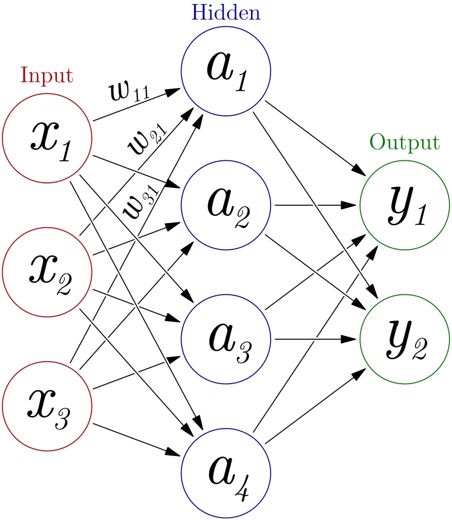
- Can solve most problems
- As before we use targets to find the error for the output states...
  $\delta_i = y_i - t_i$



3-layer Perceptron

Just add hidden units!

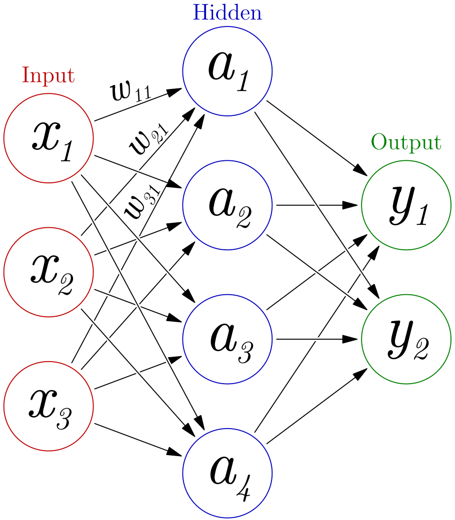
- Can solve most problems
- As before we use targets to find the error for the output states...
  $\delta_i = y_i - t_i$
- But how do we train the hidden units?



3-layer Perceptron

Feedforward phase

- Run network as normal, i.e., feed activations forwards
- When at output calculate error: $\delta_i = y_i - t_i$
- But how do we calculate error for the hidden units?



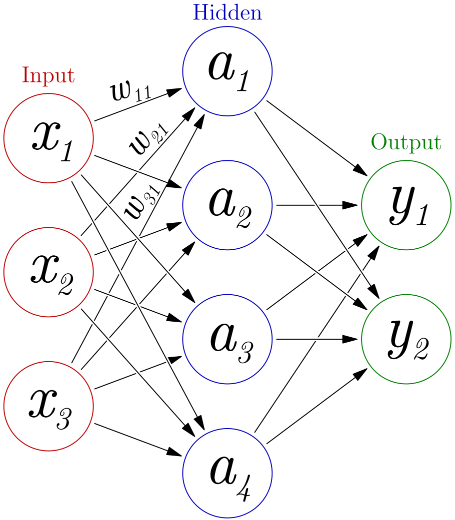
3-layer Perceptron

Backpropagation phase

- Now the output is the starting point
- When at output calculate error:

$$\delta_i = y_i - t_i$$

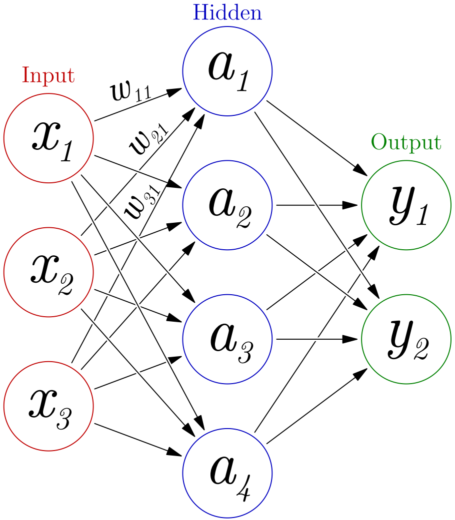
- Now run the network backwards!



3-layer Perceptron

Backpropagation phase

- The hidden targets are:
  $t_i = \sum_j \delta_j w_{ij}$



3-layer Perceptron

Backpropagation phase

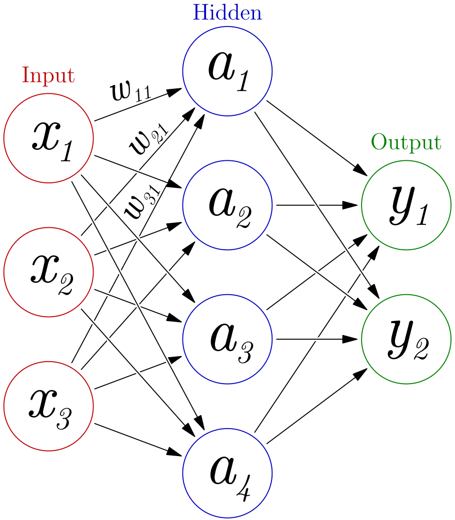
- The hidden targets are: $t_i = \sum_j \delta_j w_{ij}$
- Now we can calculate errors!

  Output error:

  $\delta_i = y_i - t_i$

  Hidden error:

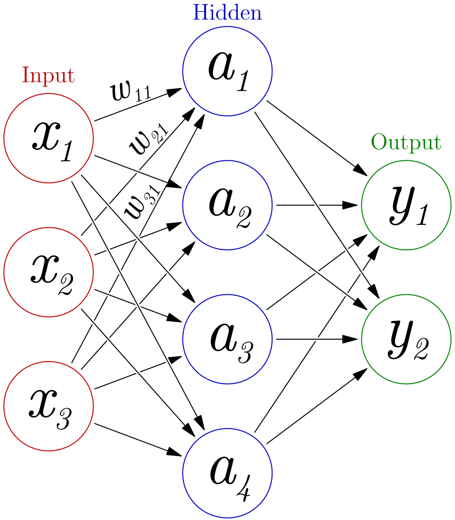
  $\delta_i = a_i(1 - a_i)t_i$



3-layer Perceptron

Backpropagation phase

- Now we have our δ_is, we can calculate weight updates!

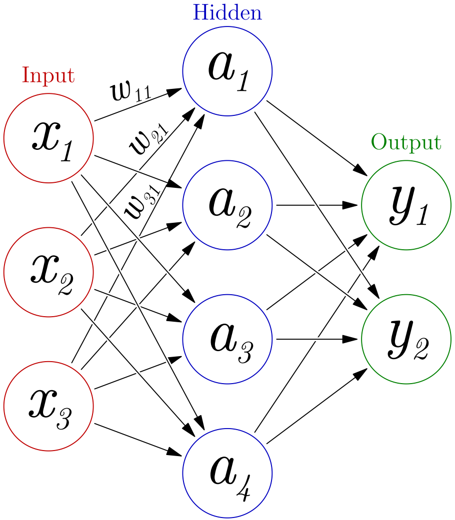


3-layer Perceptron

Backpropagation phase

- Now we have our δ_is, we can calculate weight updates!

$$\Delta w_{ij} =$$

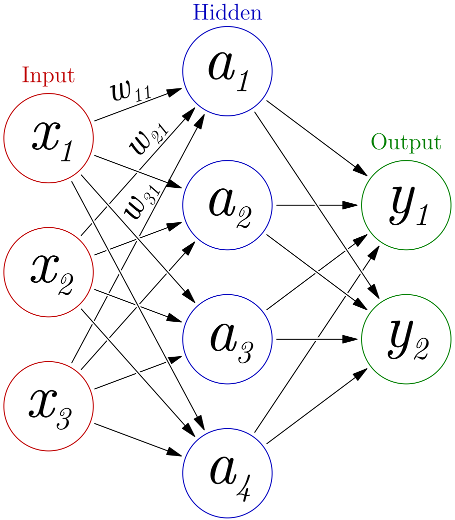


3-layer Perceptron

Backpropagation phase

- Now we have our δ_is, we can calculate weight updates!

$$\Delta w_{ij} = \qquad \delta_j$$

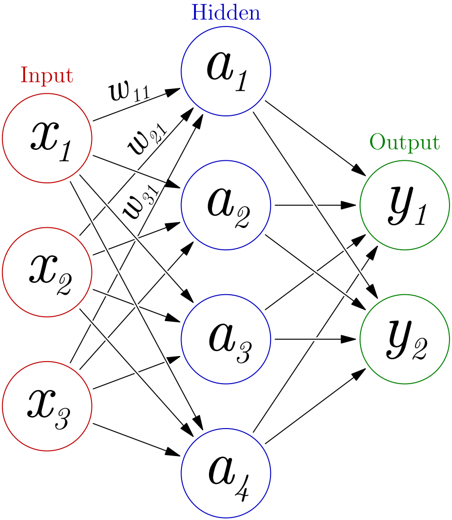


3-layer Perceptron

Backpropagation phase

- Now we have our δ_is, we can calculate weight updates!

$$\Delta w_{ij} = \qquad \delta_j \; s_i$$

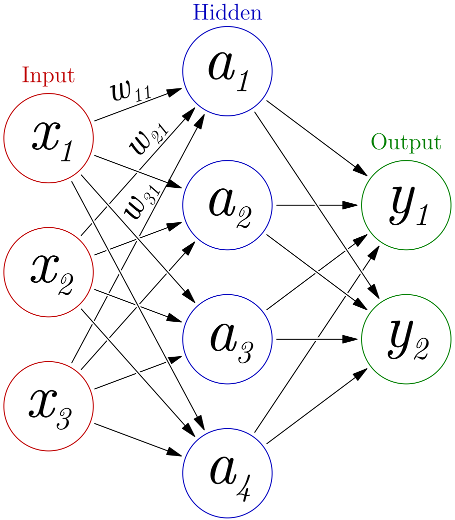


3-layer Perceptron

Backpropagation phase

- Now we have our δ_is, we can calculate weight updates!

$$\Delta w_{ij} = \sum_j \delta_j \; s_i$$



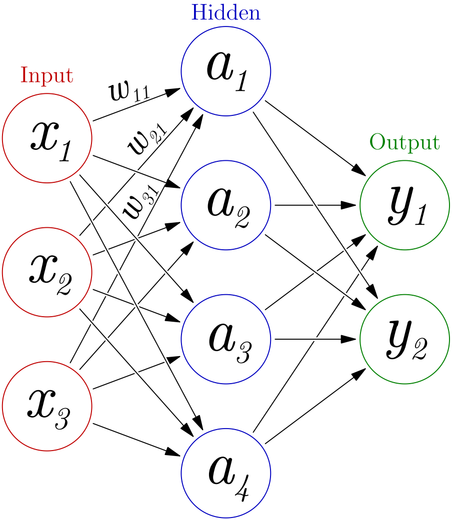
3-layer Perceptron

Backpropagation phase

- Now we have our δ_is, we can calculate weight updates!

$$\Delta w_{ij} = \sum_j \delta_j \; s_i$$

- But — before applying these updates — we want to avoid local minima, so we have a few options...



3-layer Perceptron

Just before applying Δw_{ij}s consider...

- Learning rate (too low too slow, too high too imprecise)

3-layer Perceptron

Just before applying Δw_{ij}s consider...

- Learning rate (too low too slow, too high too imprecise)
- Initialisation of weights (random $\rightarrow$ different results)

3-layer Perceptron

Just before applying Δw_{ij}s consider...

- Learning rate (too low too slow, too high too imprecise)
- Initialisation of weights (random $\rightarrow$ different results)
- Momentum: move in a similar direction to last time; avoids noise affecting updates

3-layer Perceptron

Just before applying Δw_{ij}s consider...

- Learning rate (too low too slow, too high too imprecise)
- Initialisation of weights (random $\rightarrow$ different results)
- Momentum: move in a similar direction to last time; avoids noise affecting updates
- Patternwise or in a batch? How to define a training epoch?

3-layer Perceptron

Just before applying Δw_{ij}s consider...

- Learning rate (too low too slow, too high too imprecise)
- Initialisation of weights (random $\rightarrow$ different results)
- Momentum: move in a similar direction to last time; avoids noise affecting updates
- Patternwise or in a batch? How to define a training epoch?
- How to present patterns? In a random order? Serially?

3-layer Perceptron

Applying weight changes!

- General equation for learning:

3-layer Perceptron

Applying weight changes!

- General equation for learning:

$$w_{ij}^{\varepsilon+1} =$$

3-layer Perceptron

Applying weight changes!

- General equation for learning:

$$w_{ij}^{\varepsilon+1} = w_{ij}^{\varepsilon}$$

3-layer Perceptron

Applying weight changes!

- General equation for learning:

$$w_{ij}^{\varepsilon+1} = w_{ij}^{\varepsilon} \qquad\qquad -\mu \Delta w_{ij}^{\varepsilon}$$

3-layer Perceptron

Applying weight changes!

- General equation for learning:

$$w_{ij}^{\varepsilon+1} = w_{ij}^{\varepsilon} + \nu \Delta w_{ij}^{\varepsilon-1} - \mu \Delta w_{ij}^{\varepsilon}$$

3-layer Perceptron

Applying weight changes!

- General equation for learning:

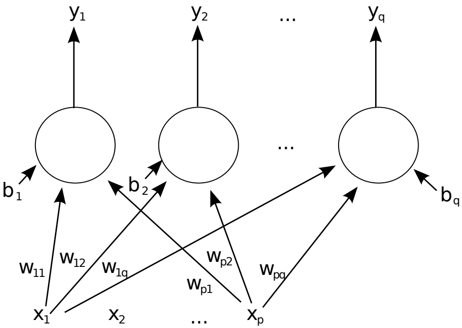
$$w_{ij}^{\varepsilon+1} = w_{ij}^{\varepsilon} + \nu \Delta w_{ij}^{\varepsilon-1} - \mu \Delta w_{ij}^{\varepsilon}$$

where ε denotes the epoch, the momentum is ν, and the learning rate is μ — latter two can be either variable or constant.

Bias Units aka Thresholds

What is it?

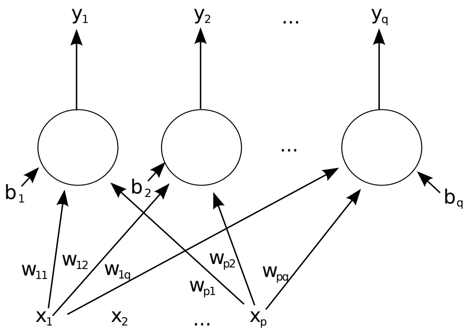
- A weight attached to a "unit" that is always on



Bias Units aka Thresholds

What is it?

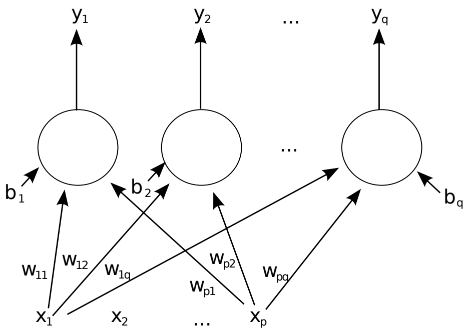
- A weight attached to a "unit" that is always on
- Trained identically to the weights



Bias Units aka Thresholds

What is it?

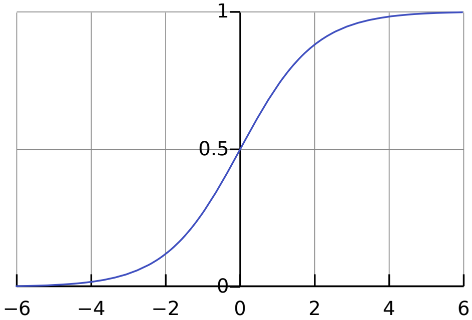
- A weight attached to a "unit" that is always on
- Trained identically to the weights
- Moves activation function left and right



Activation Function

Logistic, squashing, sigmoid, activation, step, etc., functions

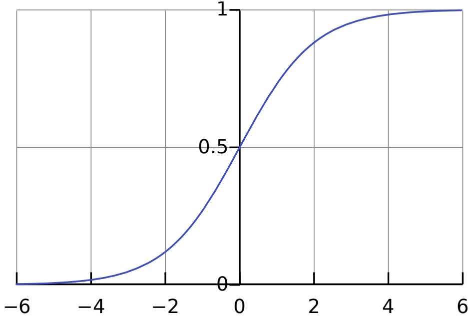
- Generic names: f, activation function, etc.



Activation Function

Logistic, squashing, sigmoid, activation, step, etc., functions

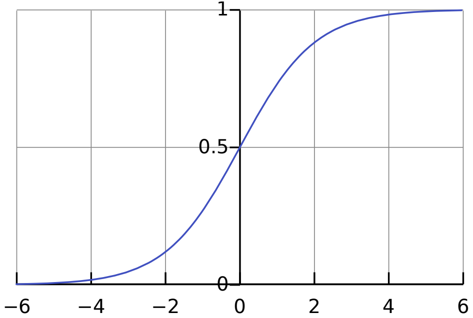
- Generic names: f, activation function, etc.
- Specific names: logistic function, hyperbolic tangent function, etc.



Activation Function

Logistic, squashing, sigmoid, activation, step, etc., functions

- Generic names: f, activation function, etc.
- Specific names: logistic function, hyperbolic tangent function, etc.
- Takes pre-synaptic input to a unit: $\eta_i = \sum_j s_j w_{ji} + b_i$



Activation Function

Logistic, squashing, sigmoid, activation, step, etc., functions

- Generic names: f, activation function, etc.
- Specific names: logistic function, hyperbolic tangent function, etc.
- Takes pre-synaptic input to a unit: $\eta_i = \sum_j s_j w_{ji} + b_i$
- More on this in exercises!