COGS300

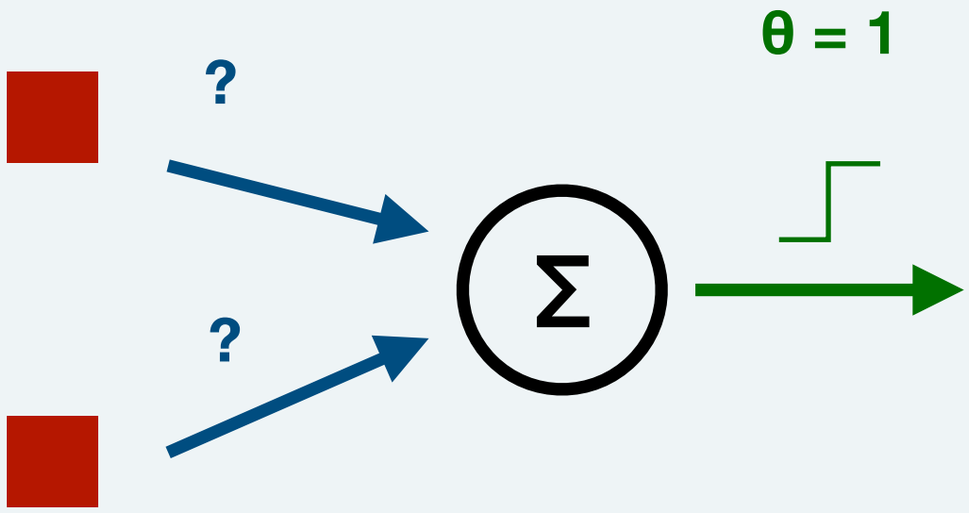
Introducing connectionism

Instructor: Márton Sóskuthy
marton.soskuthy@ubc.ca

TAs: Daichi Furukawa · Victoria Lim · Amy Wang
cogs.300@ubc.ca

What is the role of neural network models in understanding and modelling the expression of feelings and emotions in cognitive processes?

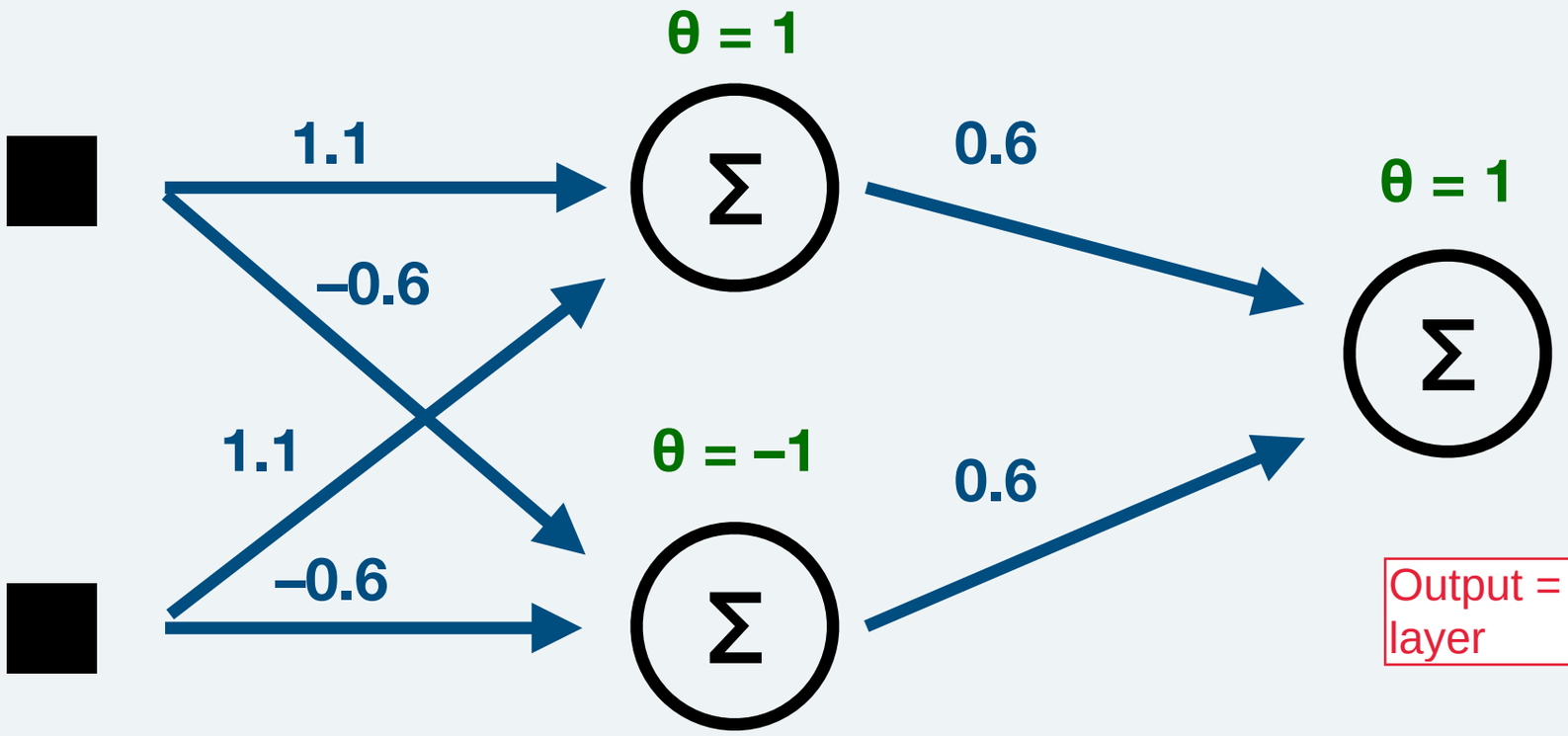
Back to the XOR problem



		XOR
0	0	0
0	1	1
1	0	1
1	1	0

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

Back to the XOR problem

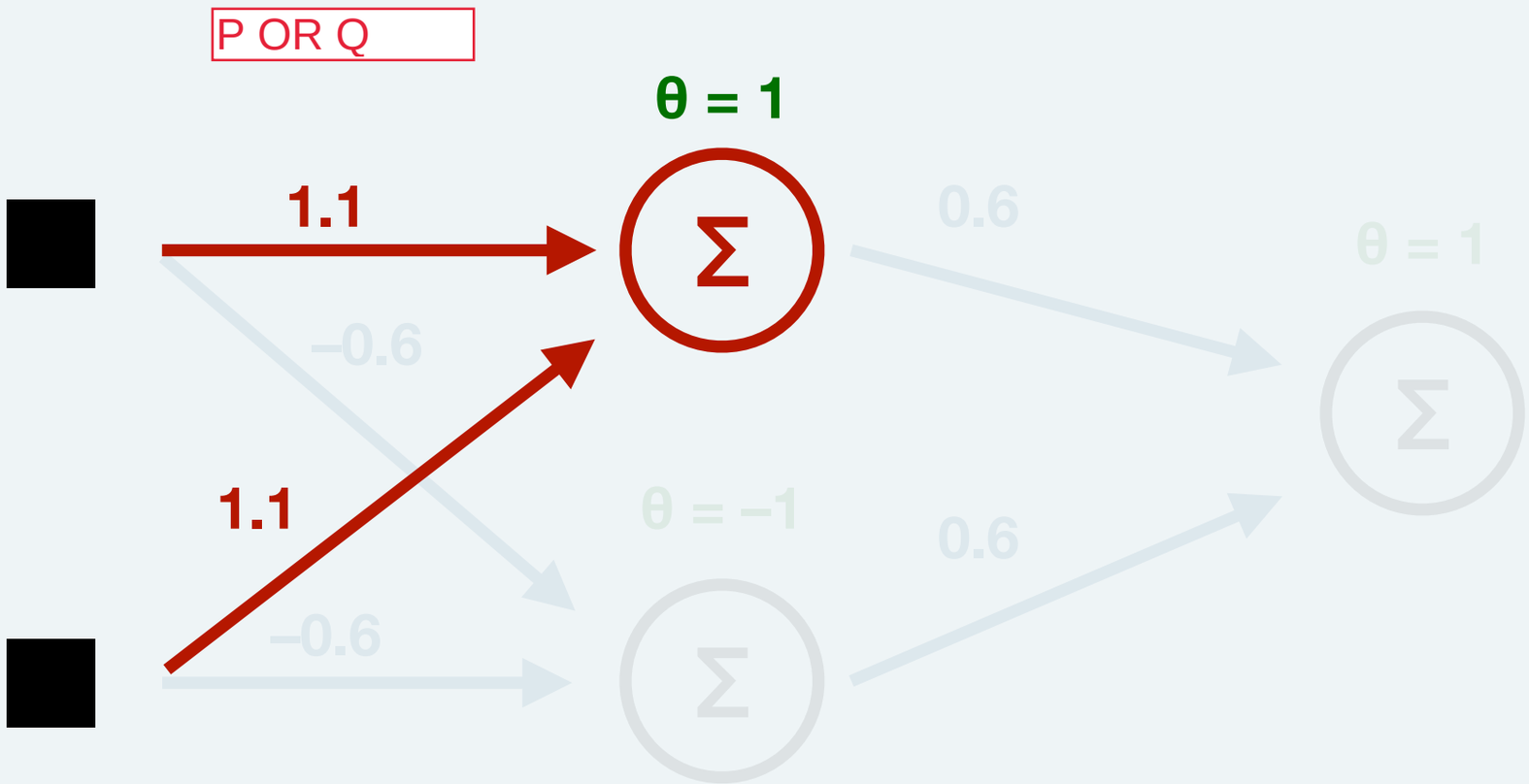


XOR

0	0	0
0	1	1
1	0	1
1	1	0

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

Back to the XOR problem

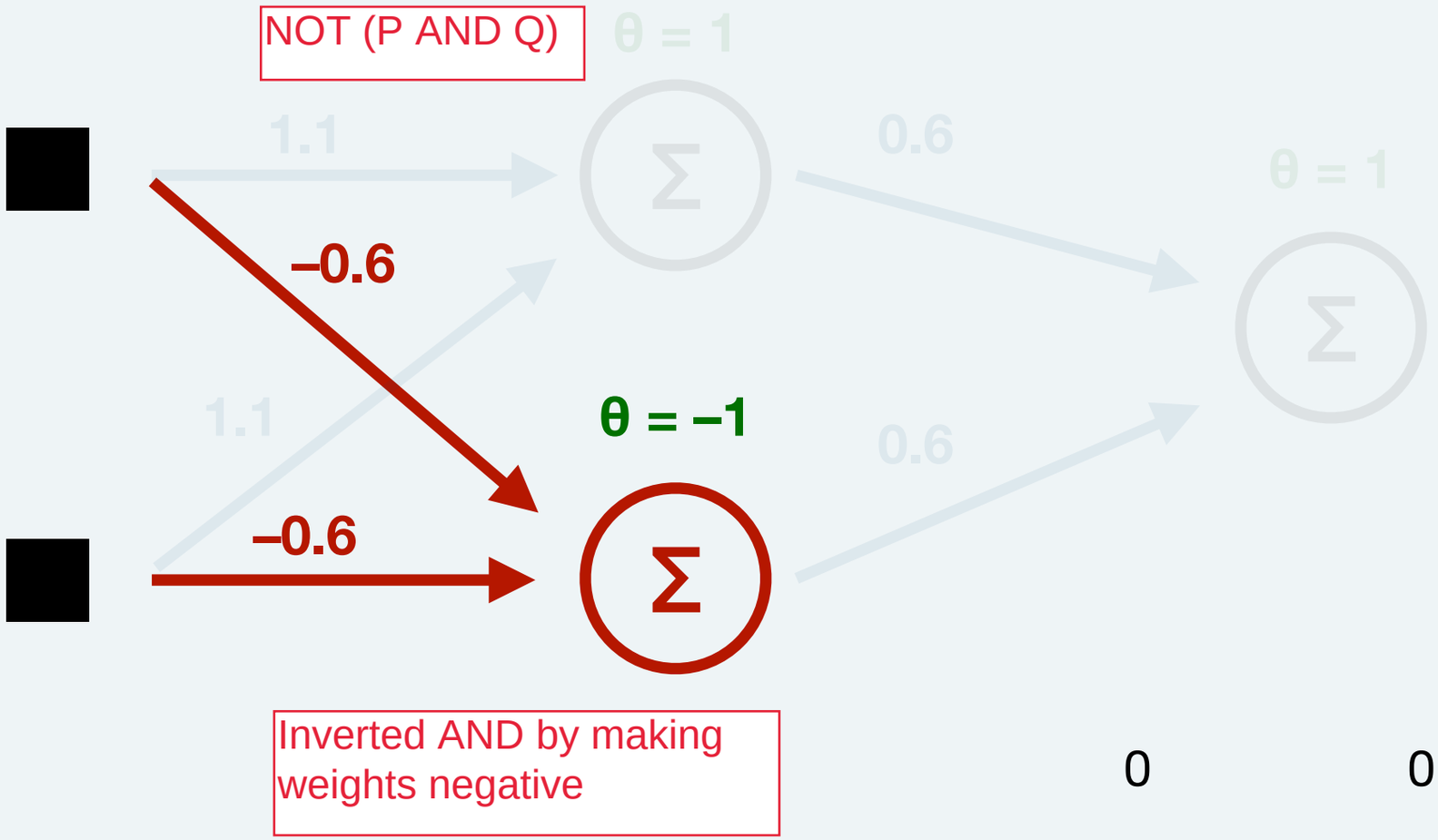


OR

0	0	0
0	1	1
1	0	1
1	1	1

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

Back to the XOR problem



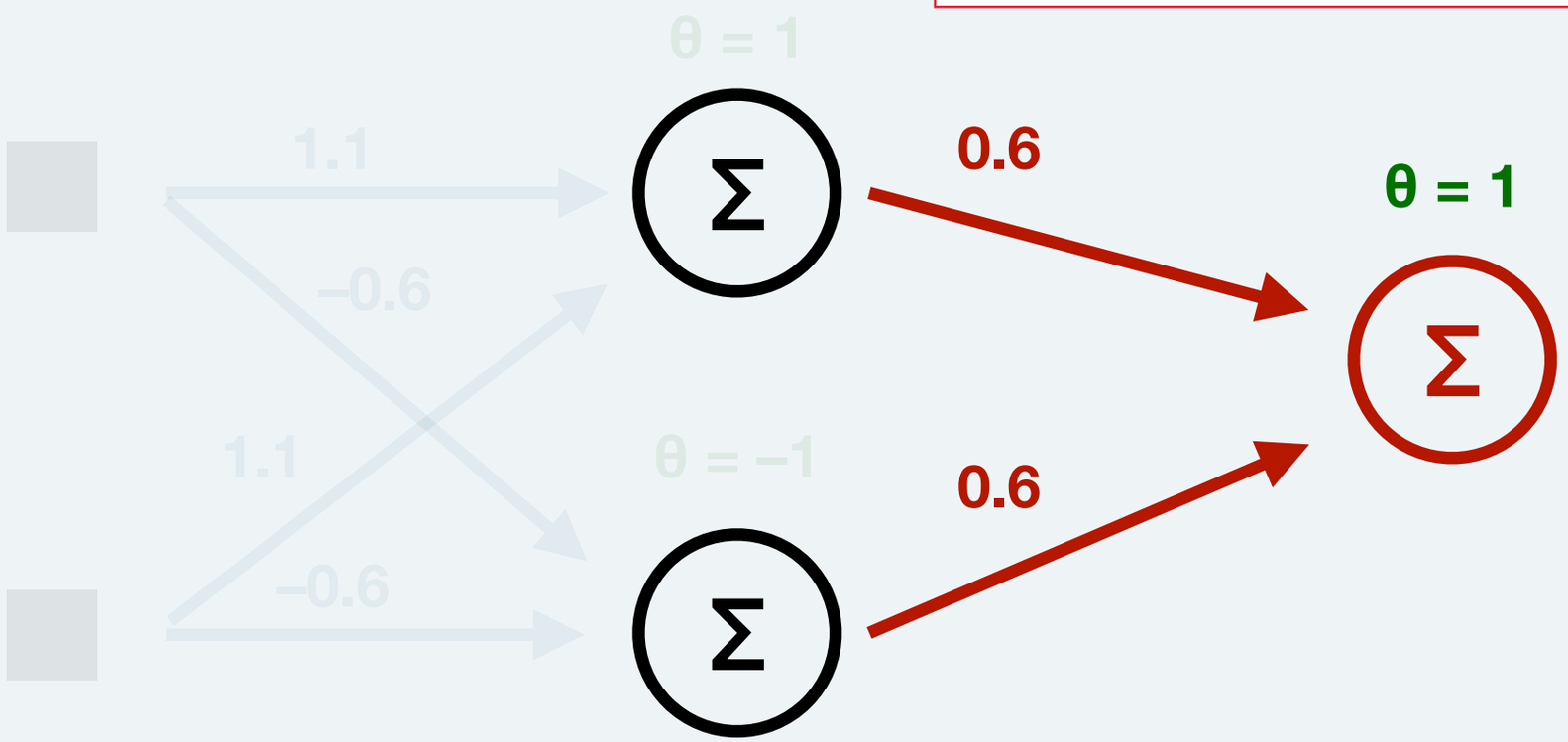
NOT AND

0	0	1
0	1	1
1	0	1
1	1	0

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

Back to the XOR problem

(P OR Q) AND (NOT (P AND Q))

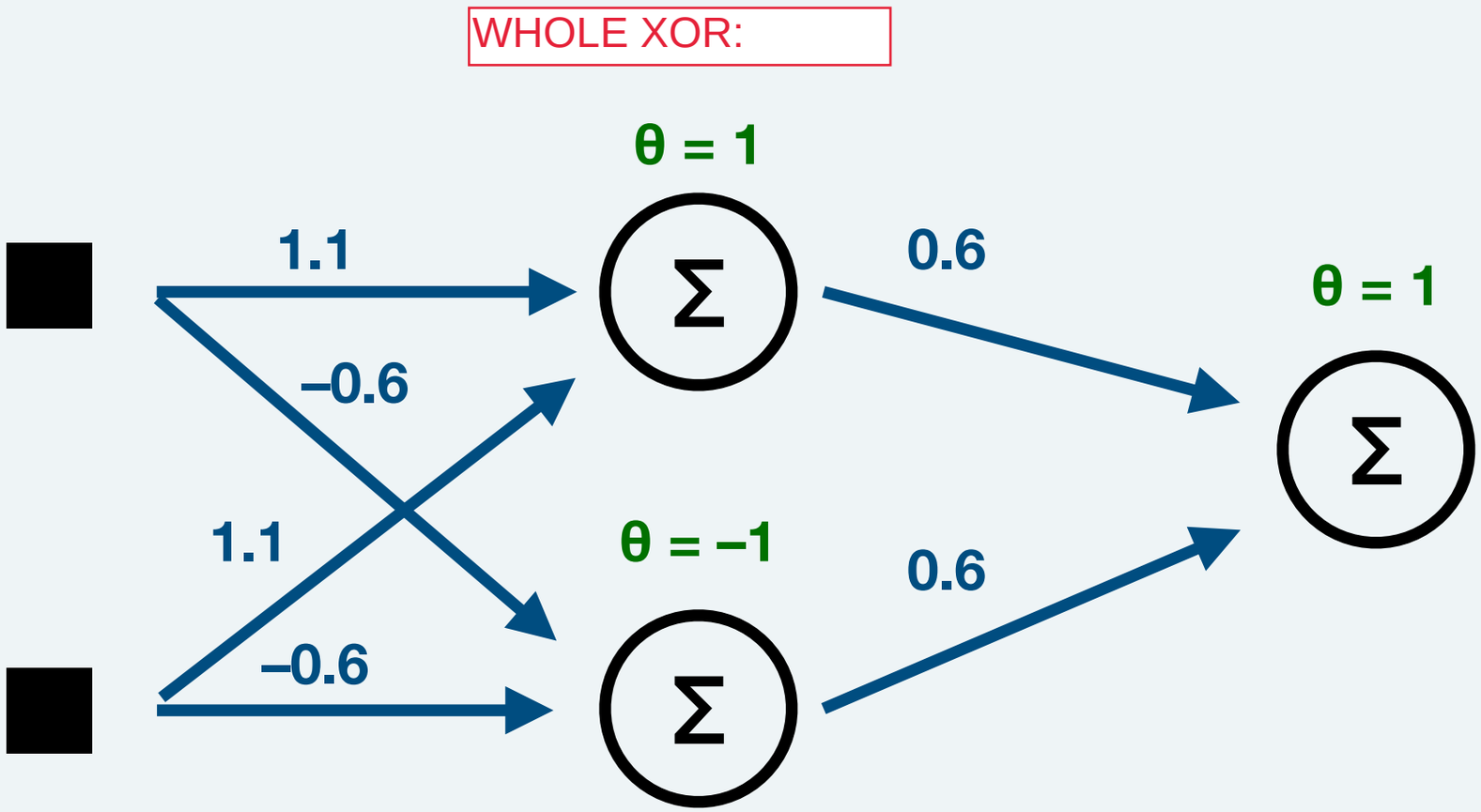


AND

0	0	0
0	1	0
1	0	0
1	1	1

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

Back to the XOR problem

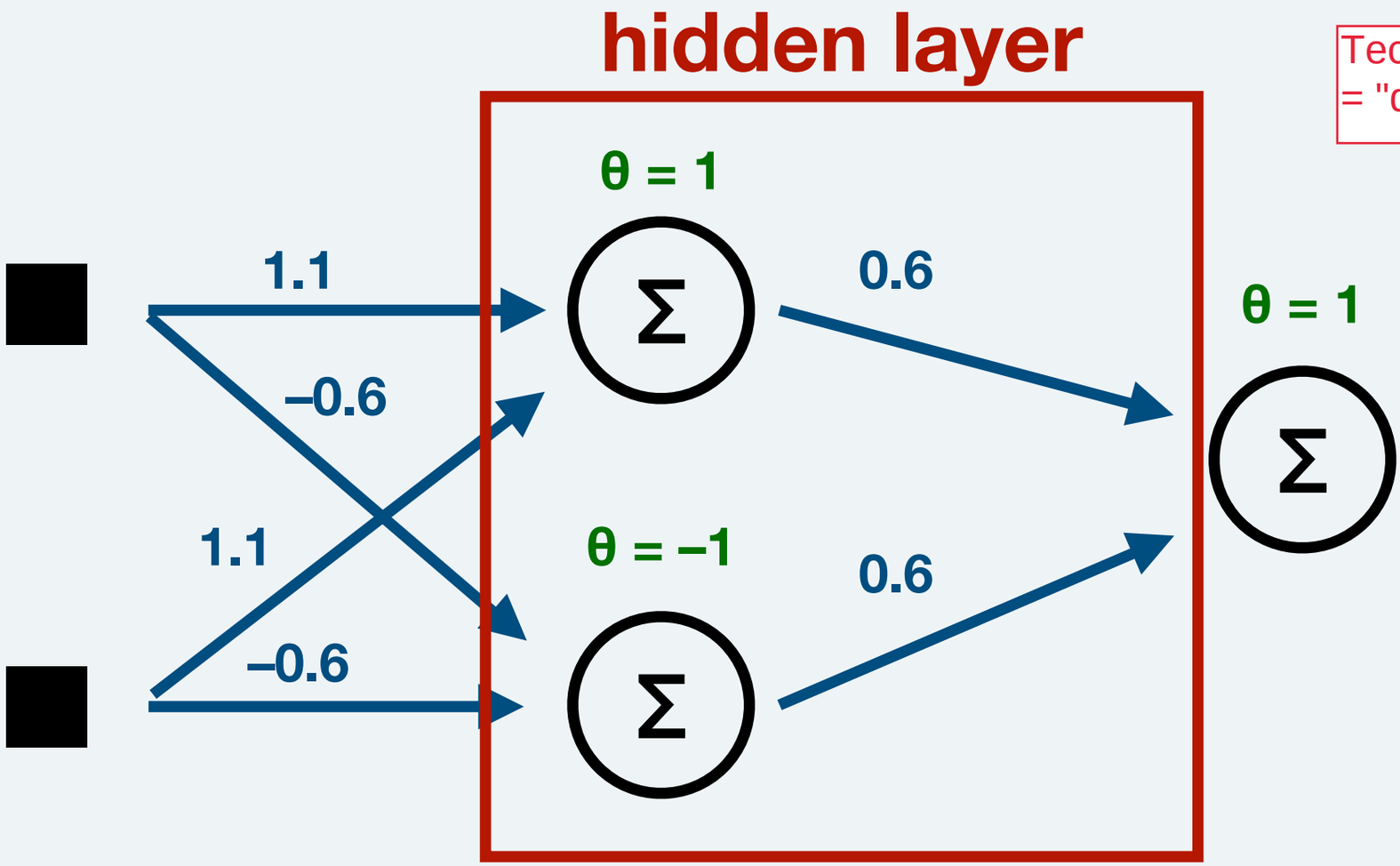


XOR

0	0	0
0	1	1
1	0	1
1	1	0

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

Back to the XOR problem



Technically ≥ 1 hidden layer
= "deep neural network"

Deep NN allows us to implement non-linear operations (e.g. XOR)

XOR

0	0	0
0	1	1
1	0	1
1	1	0

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

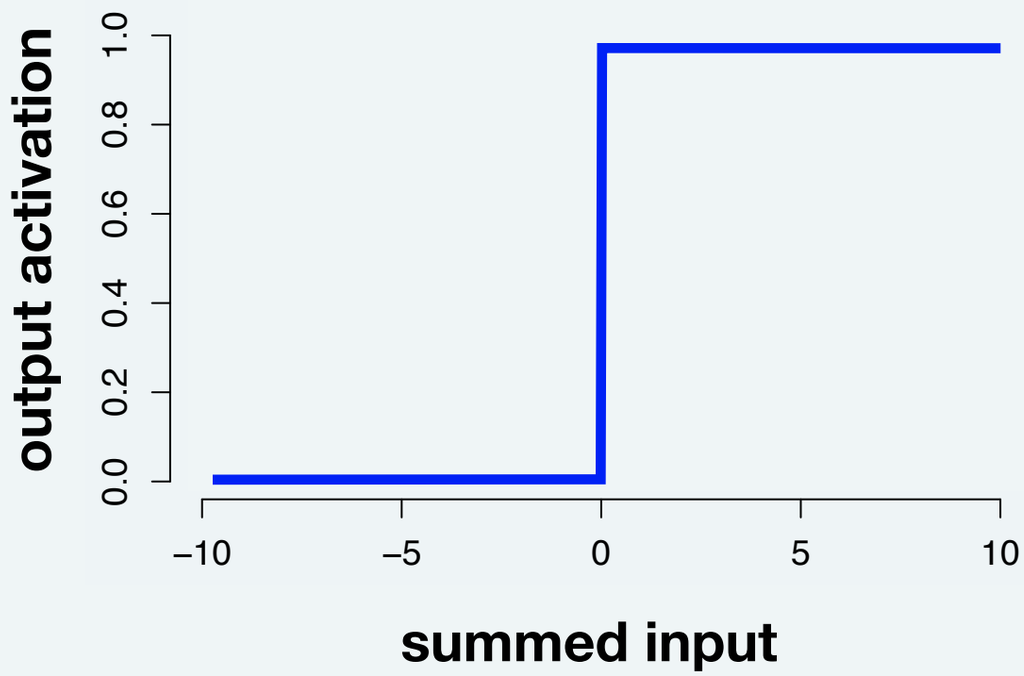
But how do you train such a network?

A more complex learning algorithm is needed – the perceptron learning rule just doesn't cut it!

The actual algorithm is pretty complex, so we'll only look at the main intuition behind it.

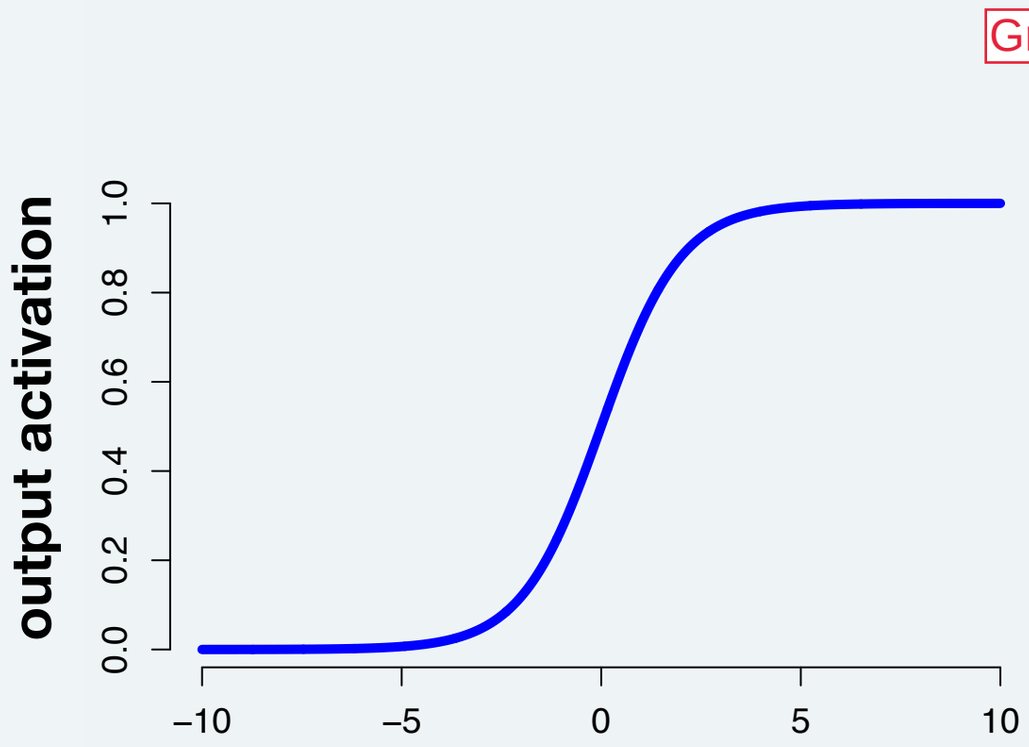
But how do you train such a network?

Changing the activation function



But how do you train such a network?

Changing the activation function



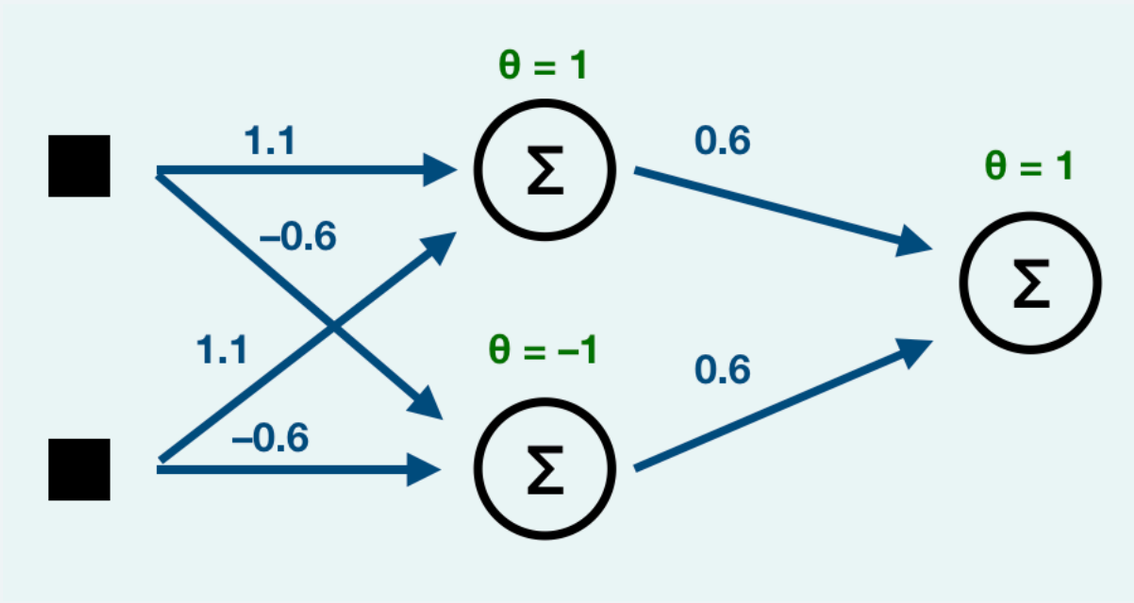
Gradient

Altho nowadays people don't always use continuous functions

Sigmoid

But how do you train such a network?

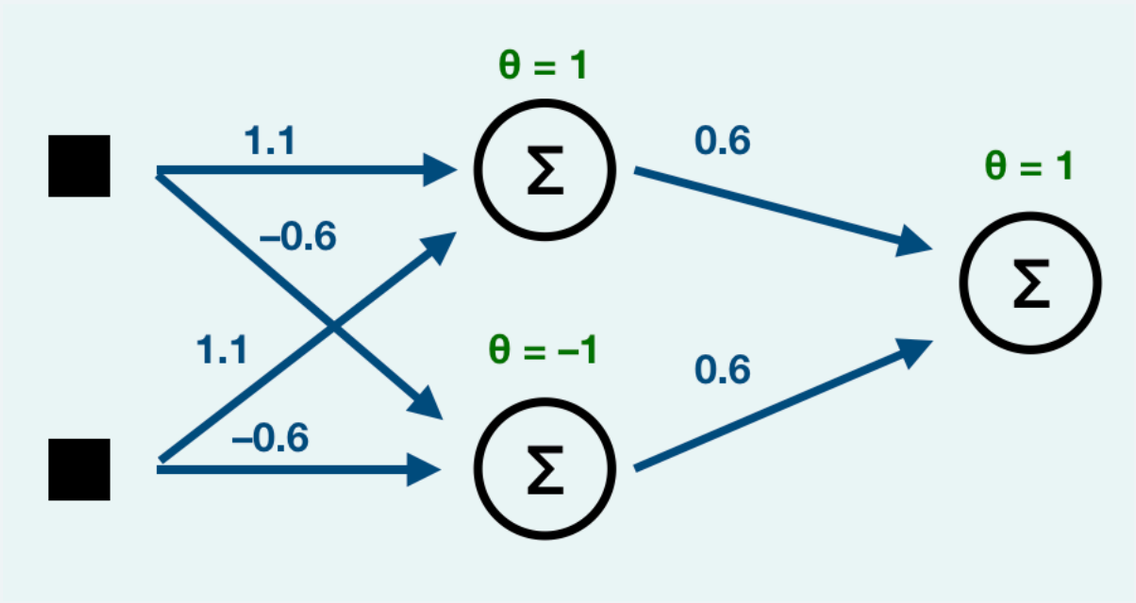
The loss function –
the error to be minimised during learning



→ **0.98 instead of 0**

But how do you train such a network?

The loss function –
the error to be minimised during learning

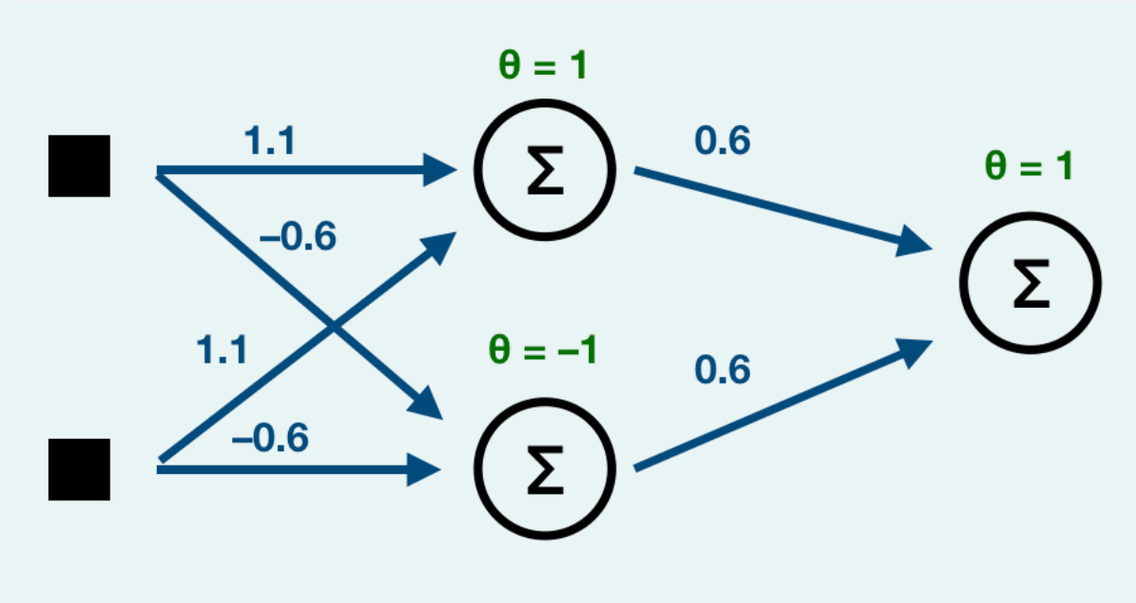


→ **0.98 instead of 0**

E = target – actual?

But how do you train such a network?

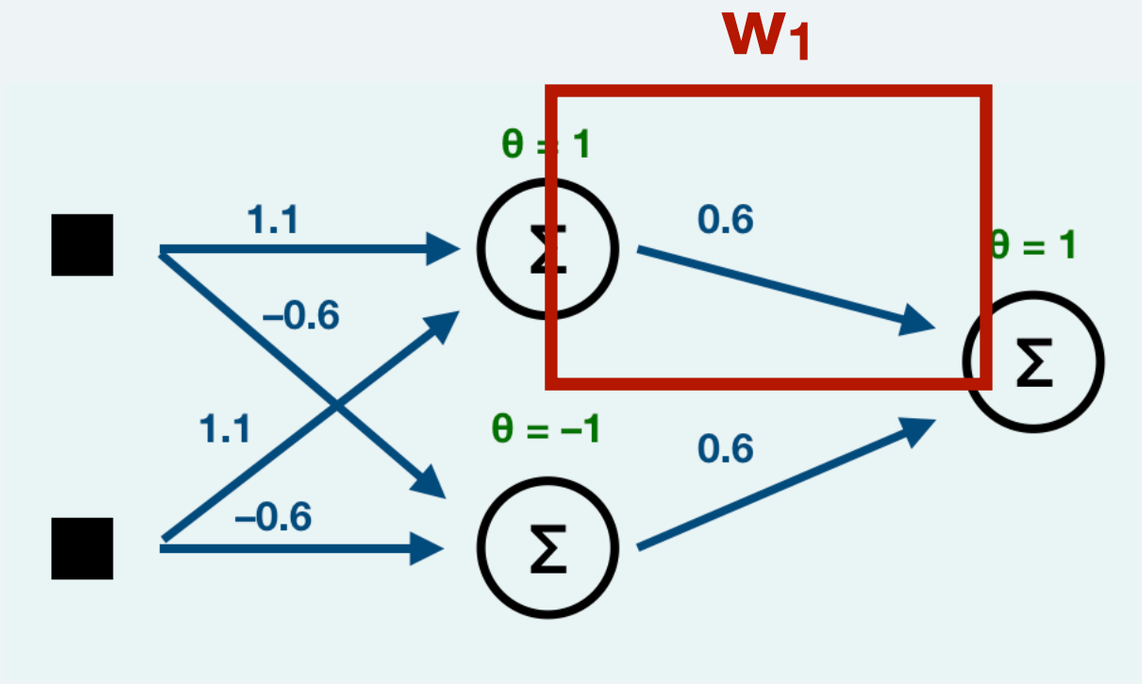
The loss function –
the error to be minimised during learning

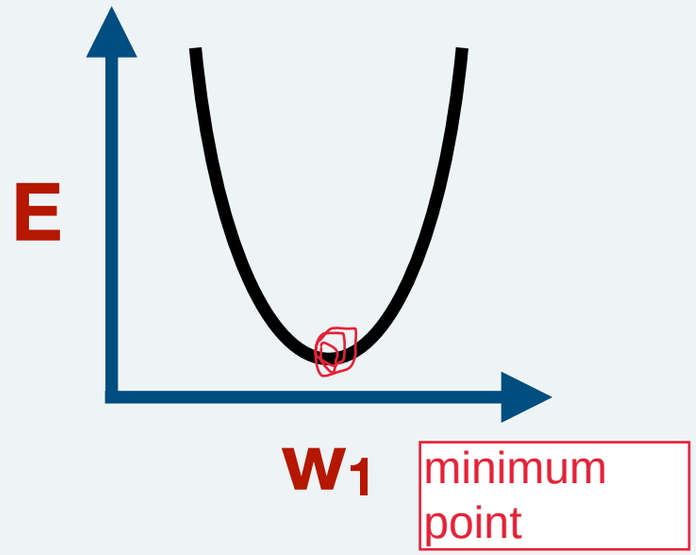


→ **0.98 instead of 0**

$$E = (\text{target} - \text{actual})^2$$

But how do you train such a network?



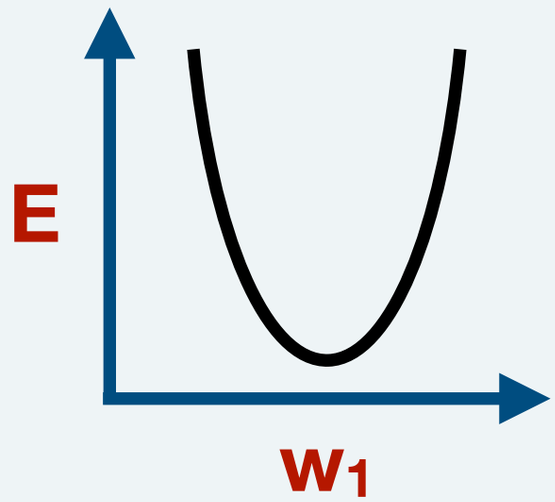


→ **0.98 instead of 0**

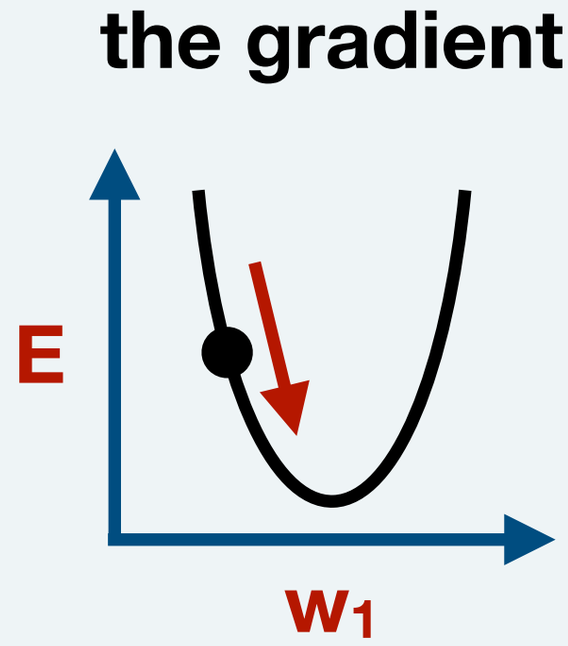
$$E = (\text{target} - \text{actual})^2$$

But how do you train such a network?

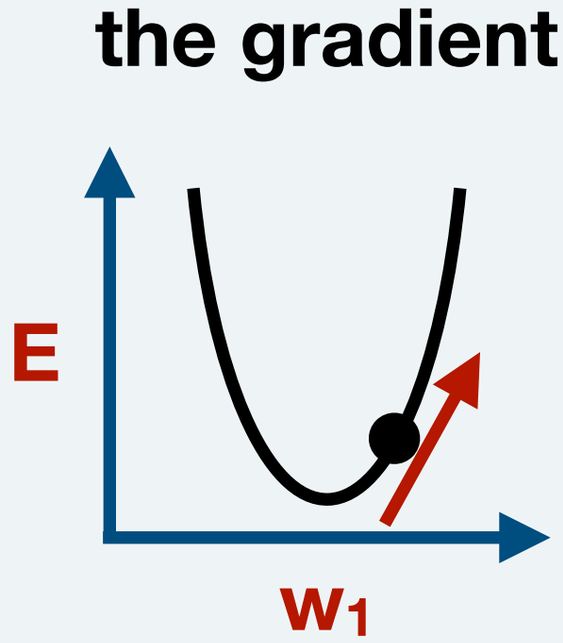
Larger number of features: need more efficient guessing method



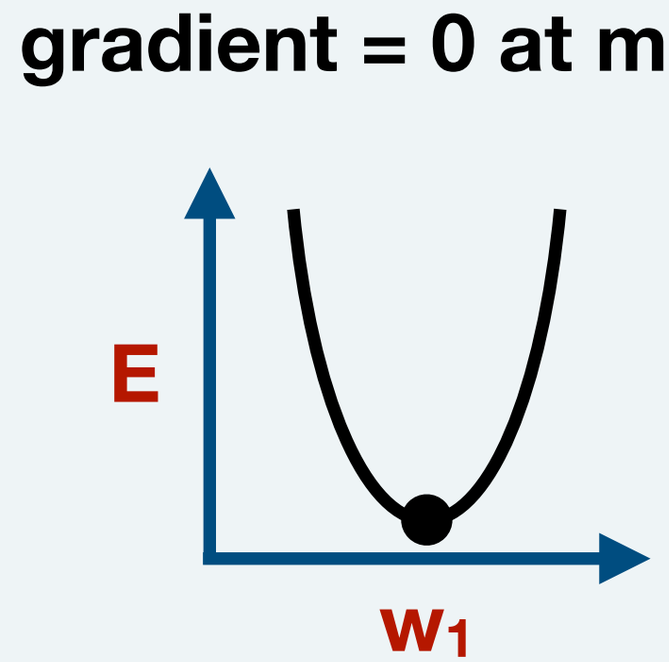
But how do you train such a network?



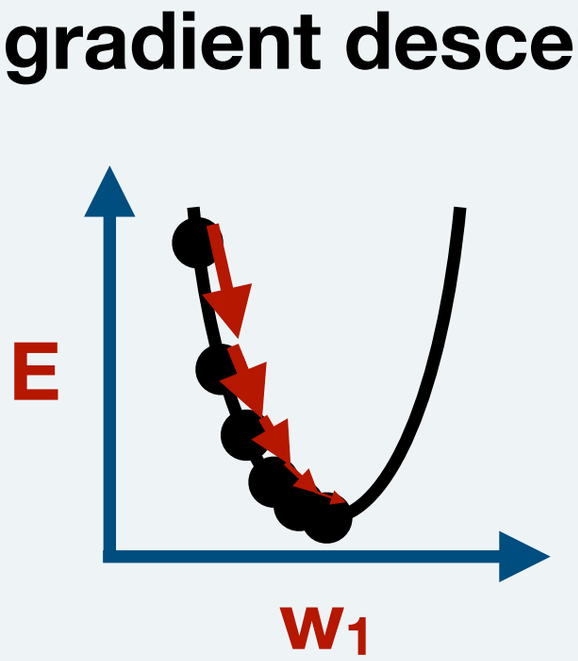
But how do you train such a network?



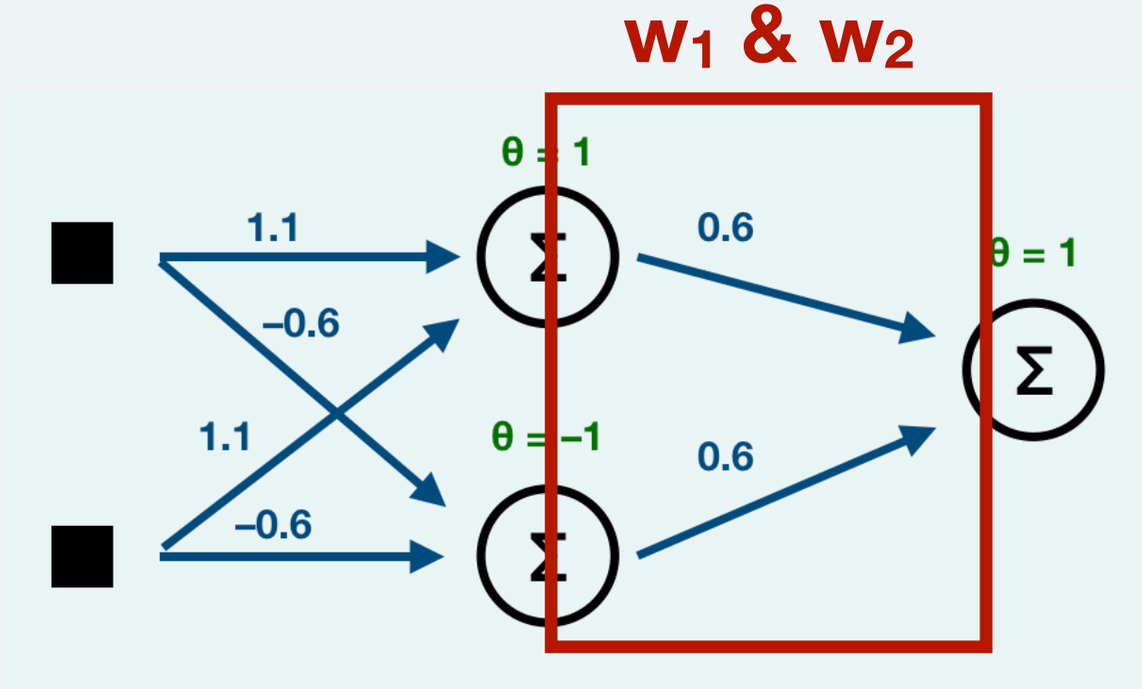
But how do you train such a network?

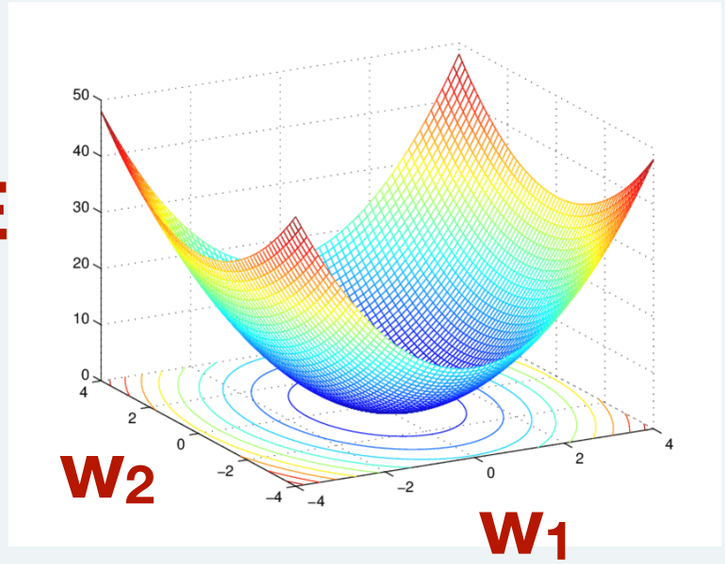


But how do you train such a network?



But how do you train such a network?



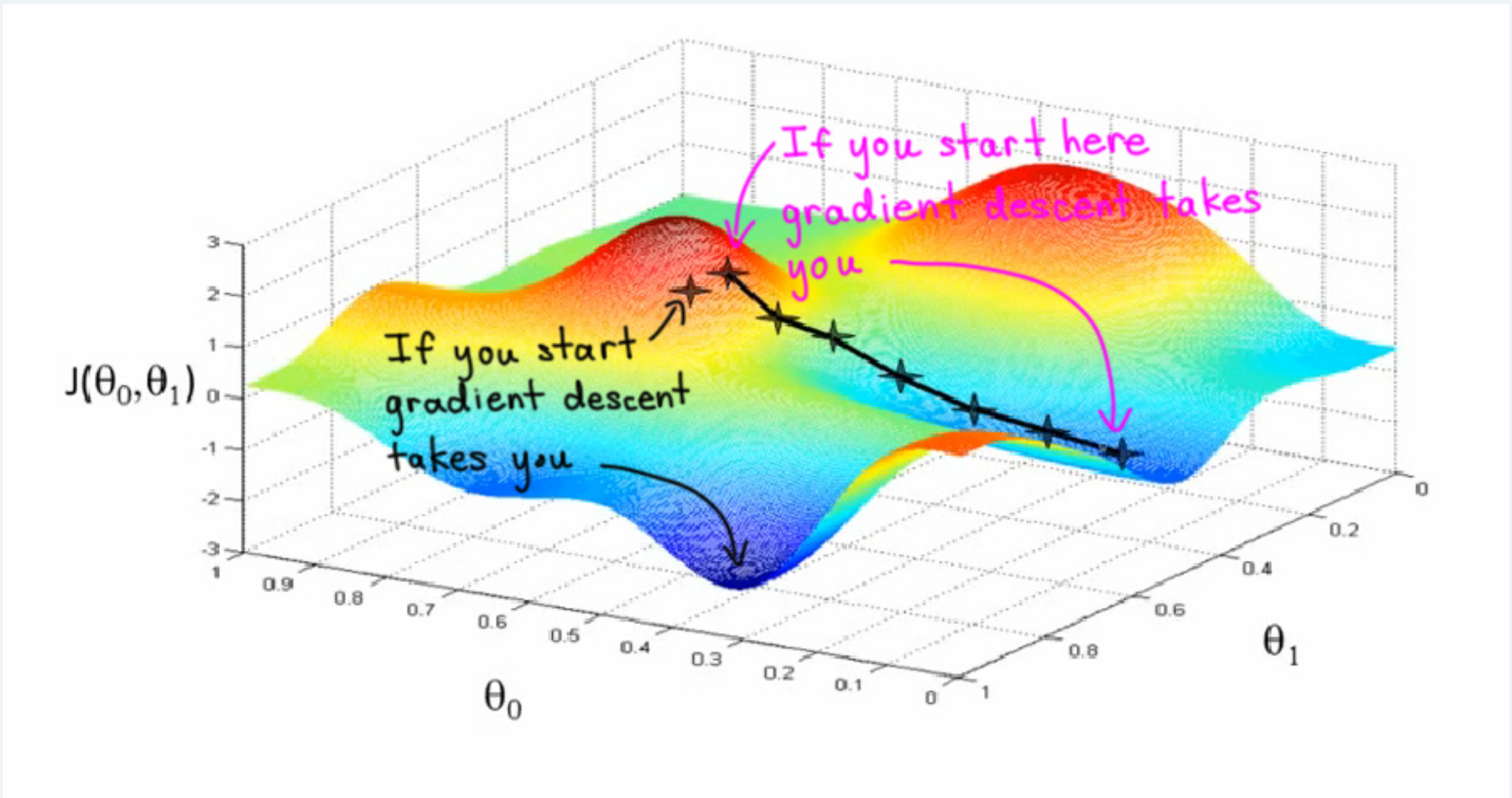


each parameter = a new dimension

$$E = (\text{target} - \text{actual})^2$$

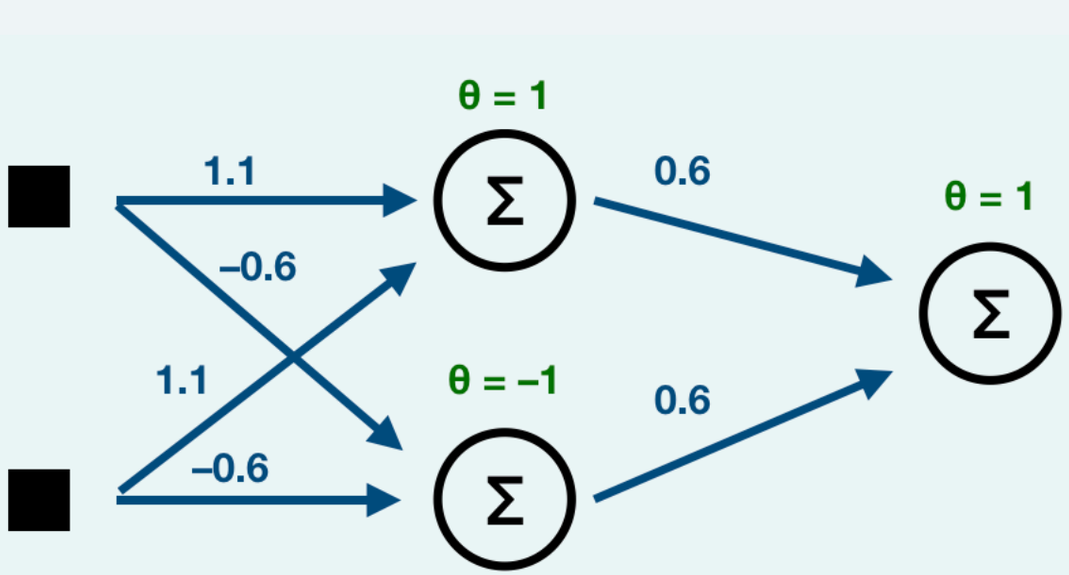
But how do you train such a network?

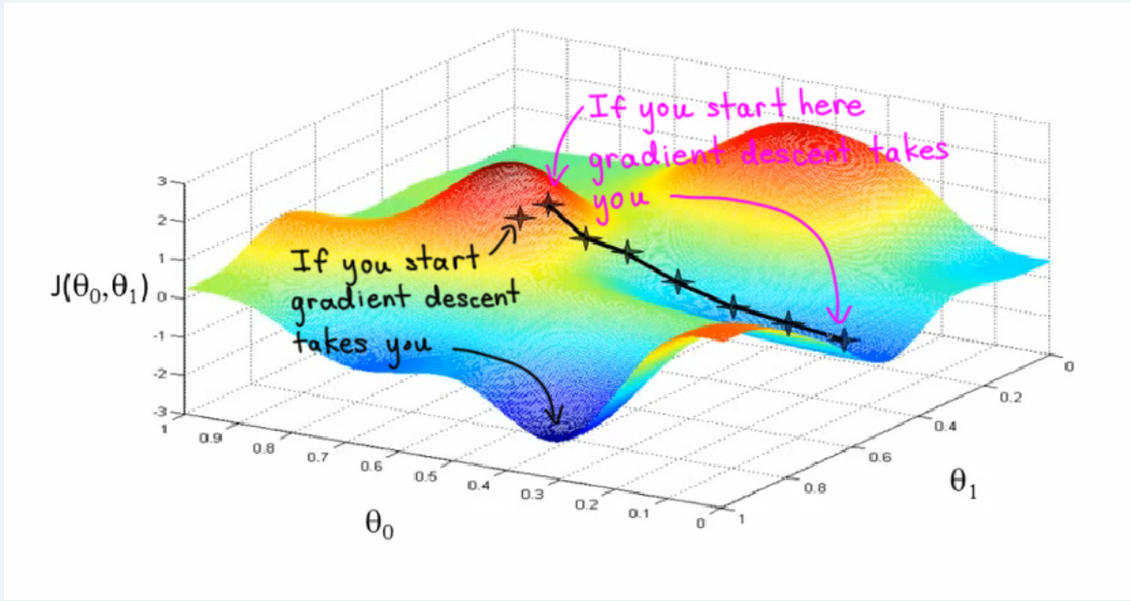
Problem: get stuck in local minimum



But how do you train such a network?

Back propagation is an algorithm that implements gradient descent for neural networks with hidden layers.





https://www.youtube.com/watch?v=Ilg3gGewQ5U

learning rate
= step size
in gradient
descent

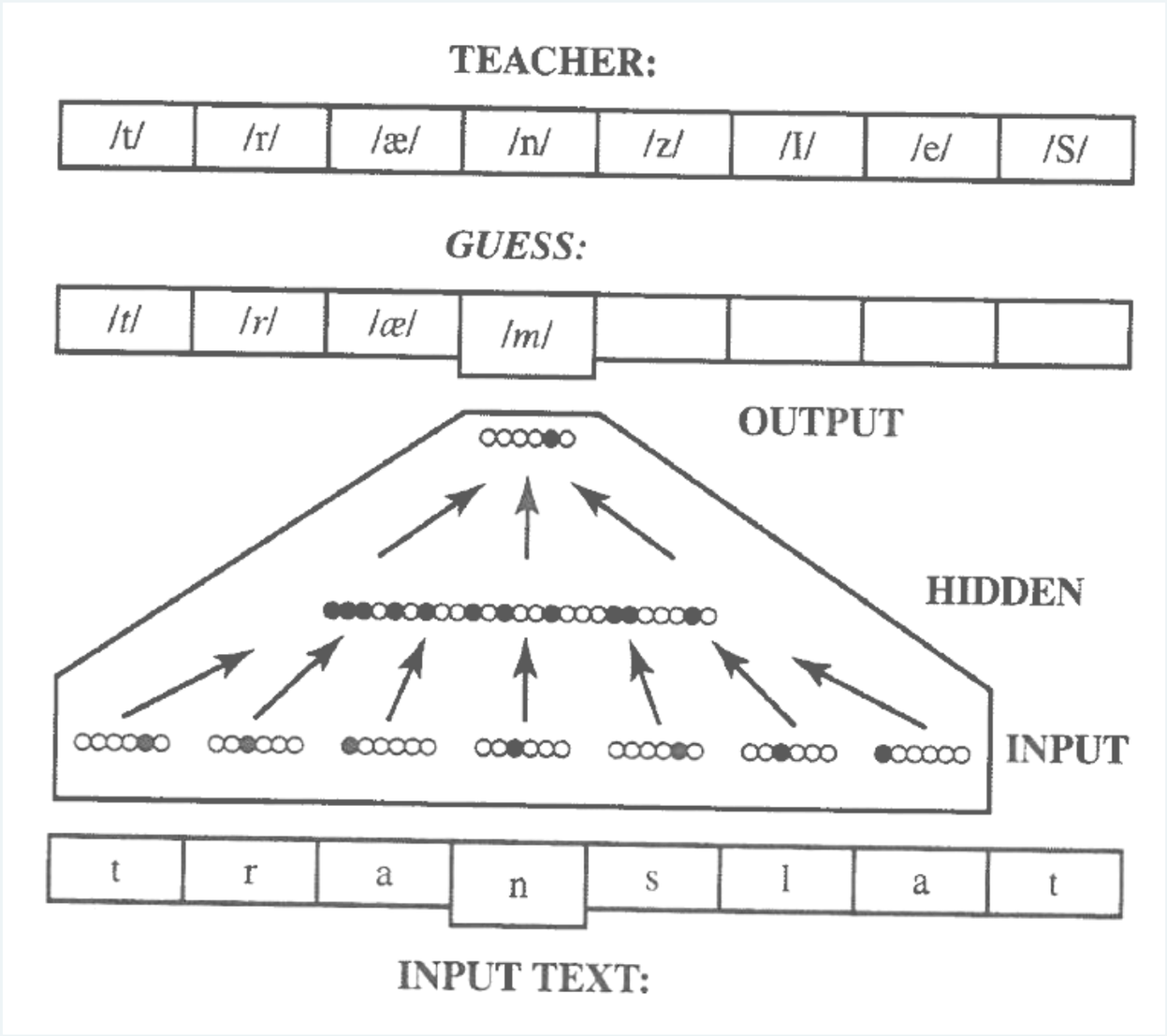
training loss
always lower
than test loss

Let’s have a play around with the following website:

https://playground.tensorflow.org

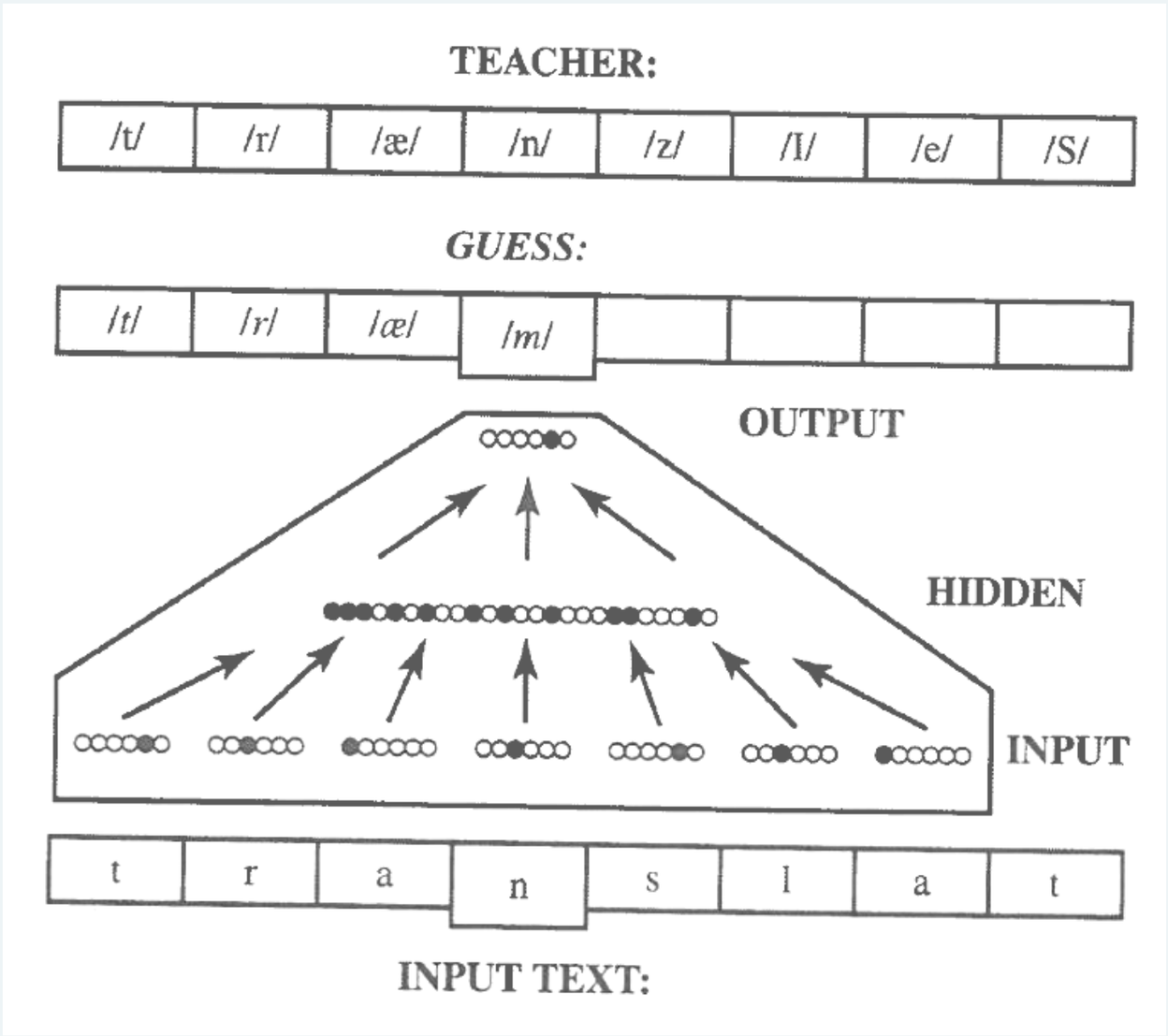
Examples of connectionist networks

NETtalk



Examples of connectionist networks

NETtalk



- **distributed, superpositional & subsymbolic representations**

➡ **graceful degradation**

➡ **generalisation**

Examples of connectionist networks

Recurrent networks