Chapter 8

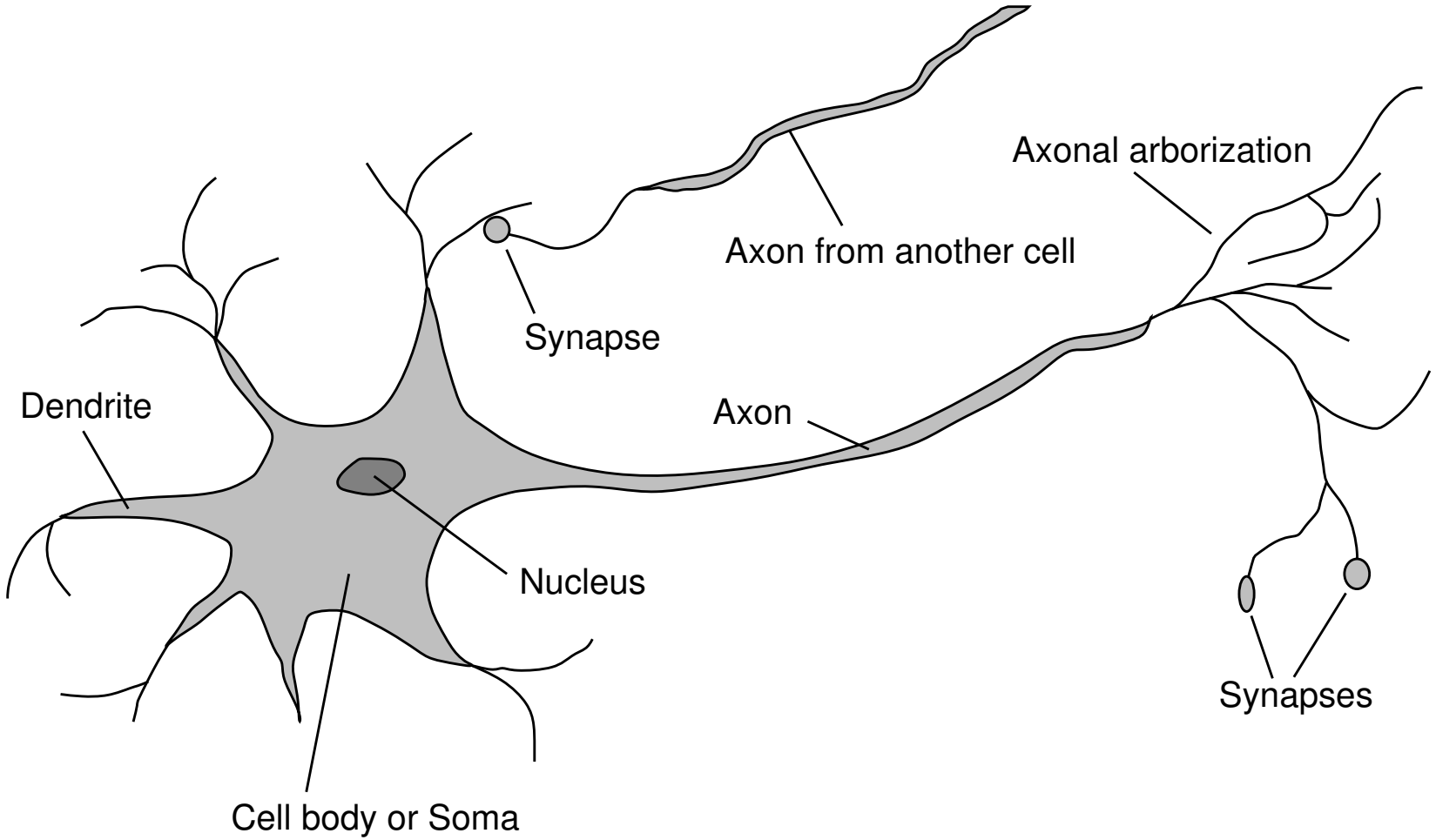
Neural Networks

Outline

- Brains
- Neural networks
- Feed-forward networks
  - Single-layer networks
  - Multi-layer networks
- Recurrent networks
  - Elman networks
- Learning
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning

Brains

- A neuron is a brain cell whose function is to collect and process electrical signals



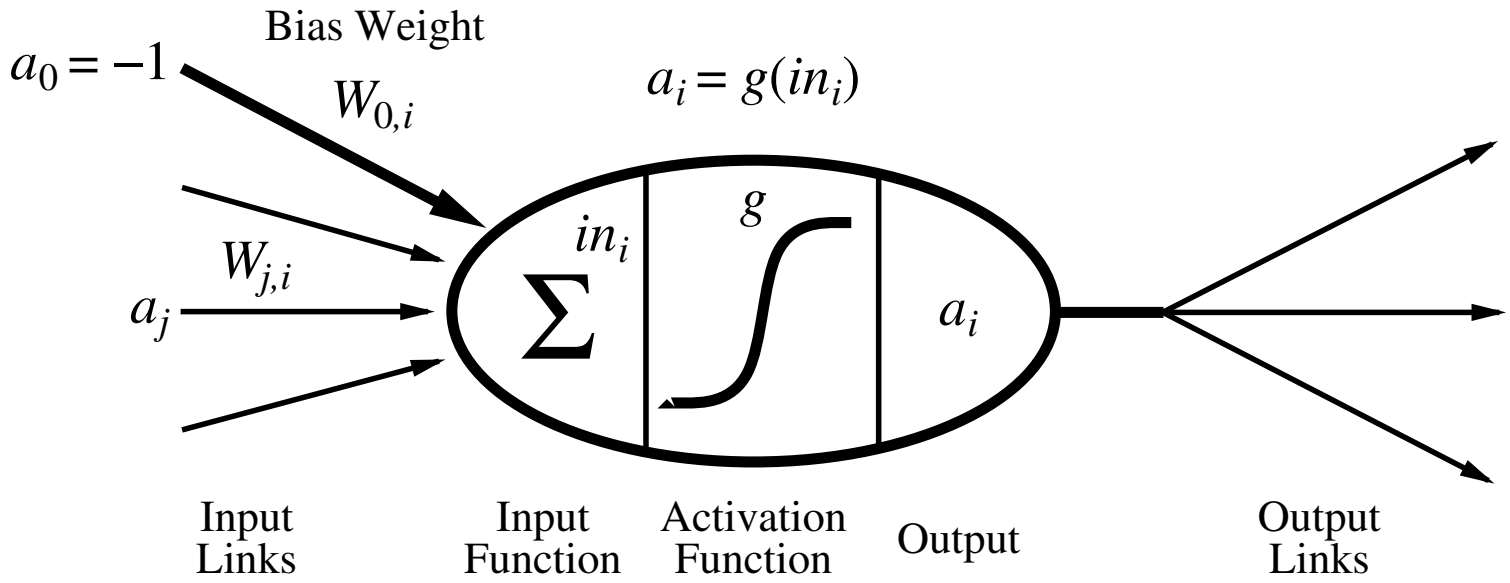
Brains(2)

- The brain's information-processing capacity is thought to emerge primarily from networks of neurons
- There are approx. 10^{11} neurons in human brain, connected via approx. 10^{14} synapses
- 1ms-10ms cycle time
- Some of the earliest AI work aimed to create artificial neural networks

Artificial Neuron

- McCulloch and Pitts devised simple mathematical model of a neuron
- Gross oversimplification of real neurons
- Its purpose was to develop understanding of what networks of simple units can do

Artificial Neuron(2)

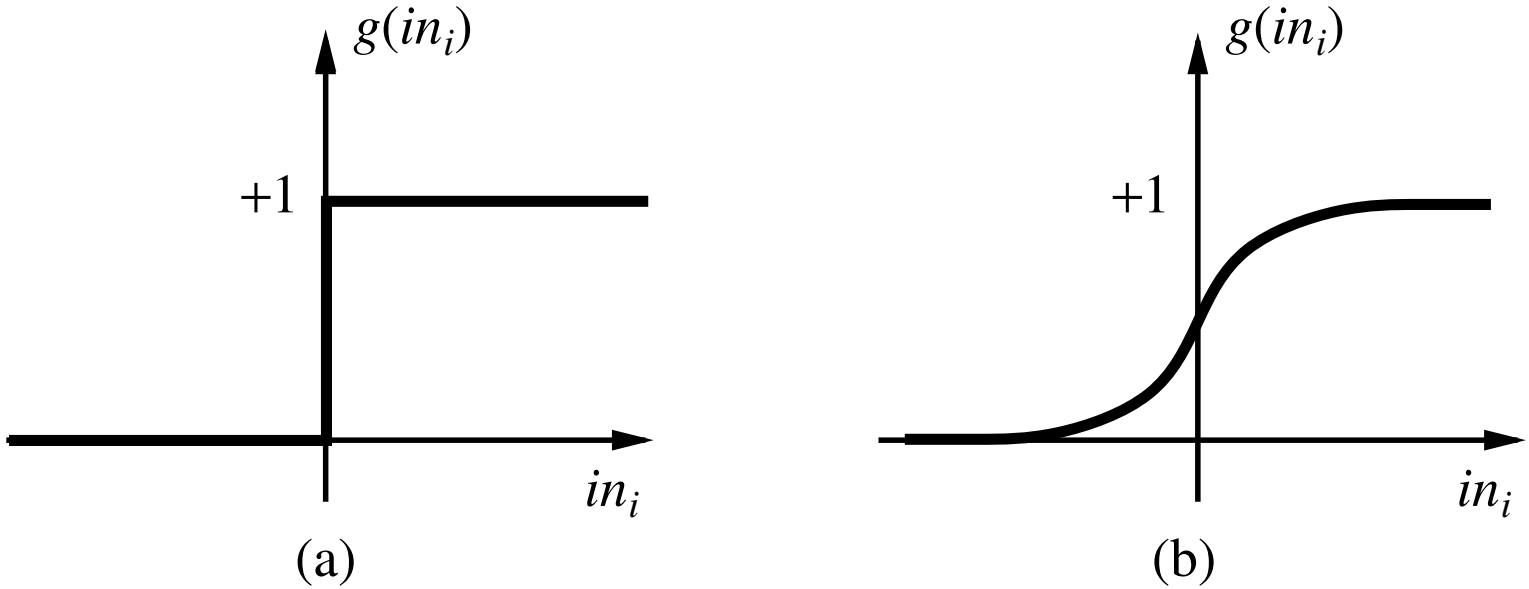


- Each unit i first computes weighted sum of its inputs:
  $in_i = \sum_{j=0}^{n} W_{j,i} a_j$
- Then applies activation function g to derive output:
  $a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$

Activation Function

- Activation function g is designed to meet two desiderata:
  - Unit should be “active” (near 1) when “right” inputs are given and “inactive” (near 0) when the “wrong” inputs are given
  - Activation needs to be nonlinear, otherwise entire neural network collapses into a simple linear function
- Two typical activation functions are
  - Threshold function (or step function)
  - Sigmoid function
- In general, activation functions are monotonically increasing

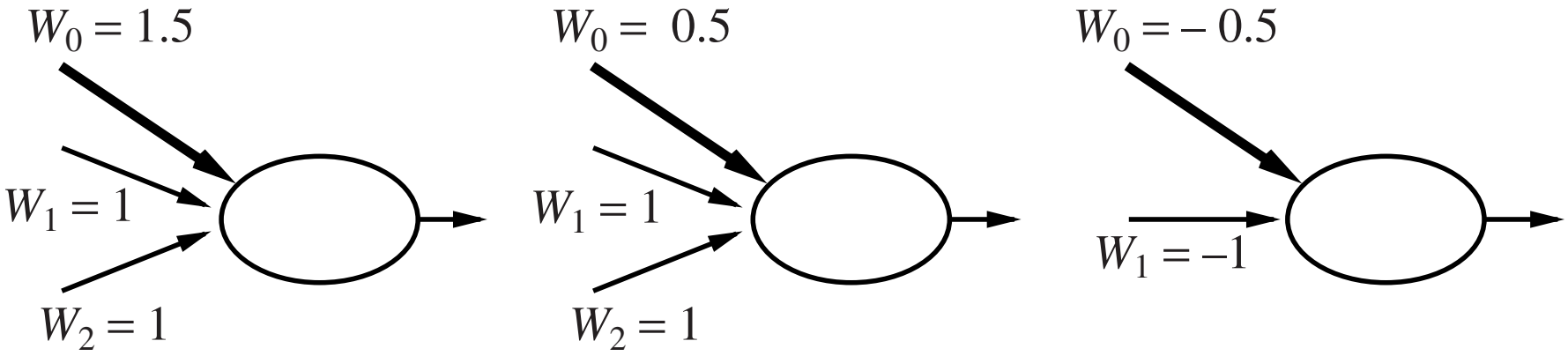
Activation Function(2)



- (a) is threshold function $g(x) = 1$ for $x > 0, = 0$ otherwise
- (b) is sigmoid function $g(x) = 1/(1 + e^{-x})$
- Usually, bias weight $W_{0,i}$ is used to move threshold location: $g(in_i) = g(x - W_{0,i})$

What Can We Do?

- With single neurons, Boolean functions can be implemented



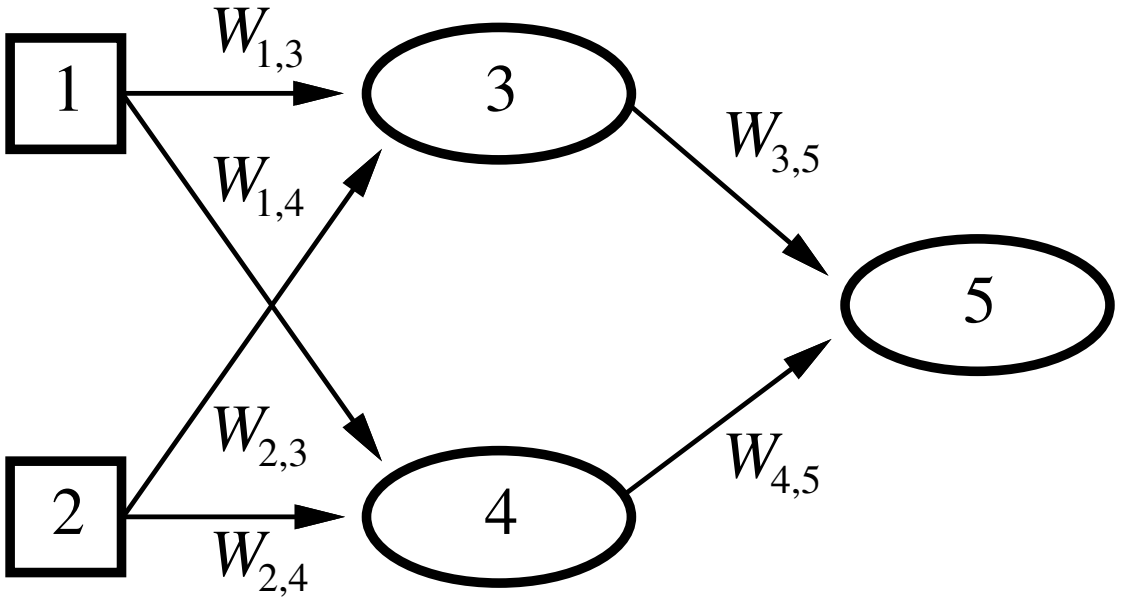
AND **OR** **NOT**

- Using neurons, we can build a network to compute any Boolean function of the inputs

Network Structures

- Two main categories of neural networks structures
  - Feed-forward networks
    - Represents a function of its current input
    - No internal states other than weights
  - (Cyclic or) recurrent networks
    - Feeds outputs back into inputs
    - Dynamical system (may reach stable state, exhibit oscillations or even chaotic behavior)
    - Can support short-term memory
    - This makes them more interesting, but also harder to understand

Feed-forward Example



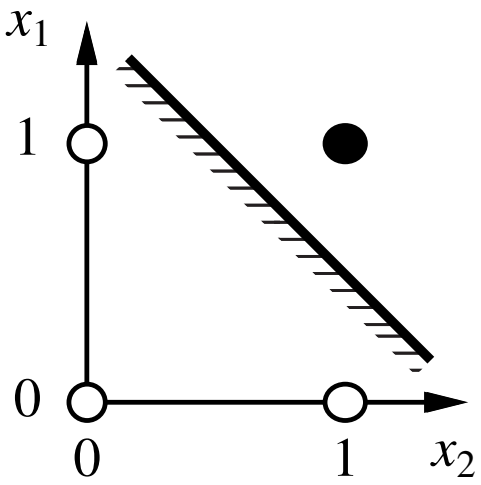
- Simple neural network with two inputs, one hidden layer of two units, and one output
- Feed-forward networks are usually arranged in layers (each unit receives input from the immediately preceding layer)

Single-layer Networks

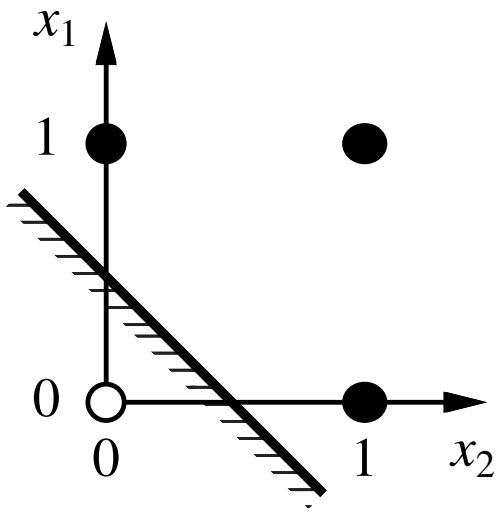
- Network with all inputs connected directly to the outputs is called a single-layer neural network (or perceptron network)
- Each output unit is independent of the others, so we look at a single output unit
- We start by examining the expressiveness of perceptrons
- As already seen, simple Boolean functions are possible
- Majority function (outputs 1 if more than half of inputs are 1) is also possible: $W_j = 1$, threshold $W_0 = n/2$

Expressiveness

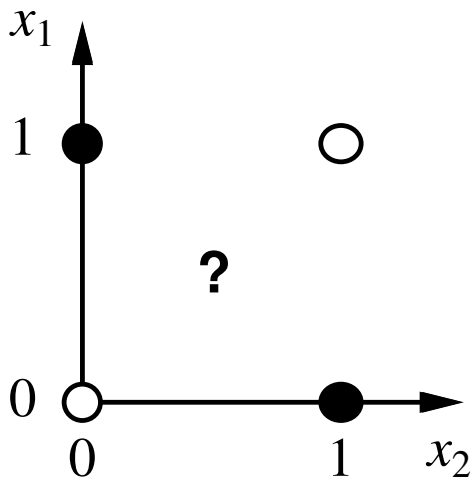
- Consider perceptron with threshold function
- Can represent AND, OR, NOT, majority, but e.g. not XOR:



(a) x_1 **and** x_2



(b) x_1 **or** x_2

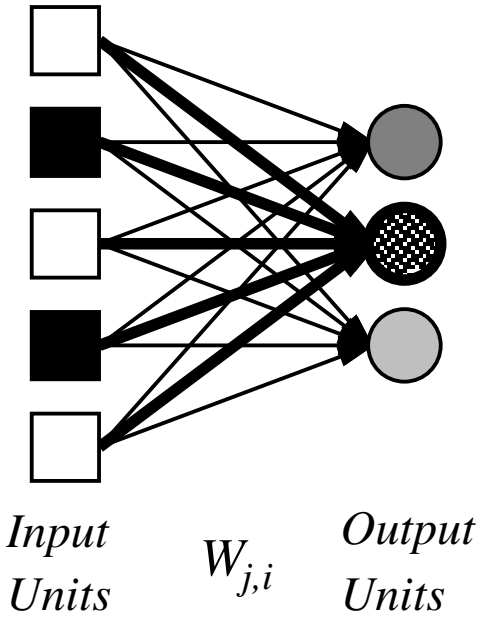


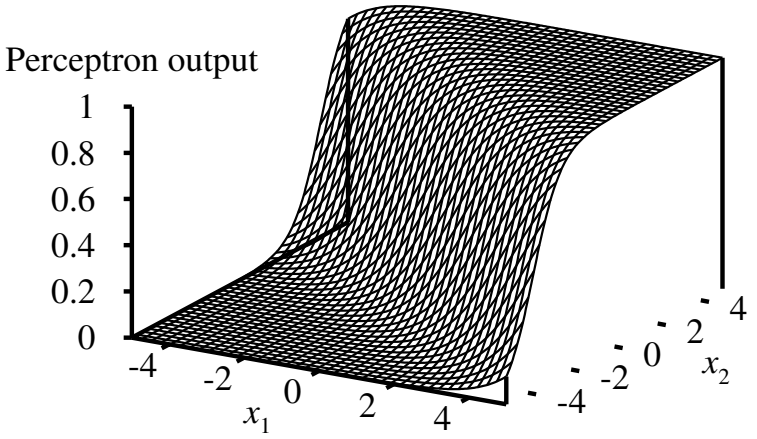
(c) x_1 **xor** x_2

Expressiveness(2)

- Perceptron represents a linear separator in input space
- Threshold perceptron returns 1, iff the weighted sum of its inputs is positive: $\sum_{j=0}^{n} W_j x_j > 0$
- Or, interpreting the W_js and x_js as a vector, $\vec{W} \circ \vec{x} > 0$
- This defines a hyperplane in the input space, perceptron returns 1 if input is on one side of that plane
- For this reason, threshold perceptron is called linear separator

Expressiveness(3)





- Output units all operate separately, no shared weights
- Adjusting weights moves the location, orientation, and steepness of “cliff”

Multilayer Networks

- Layers are usually fully connected

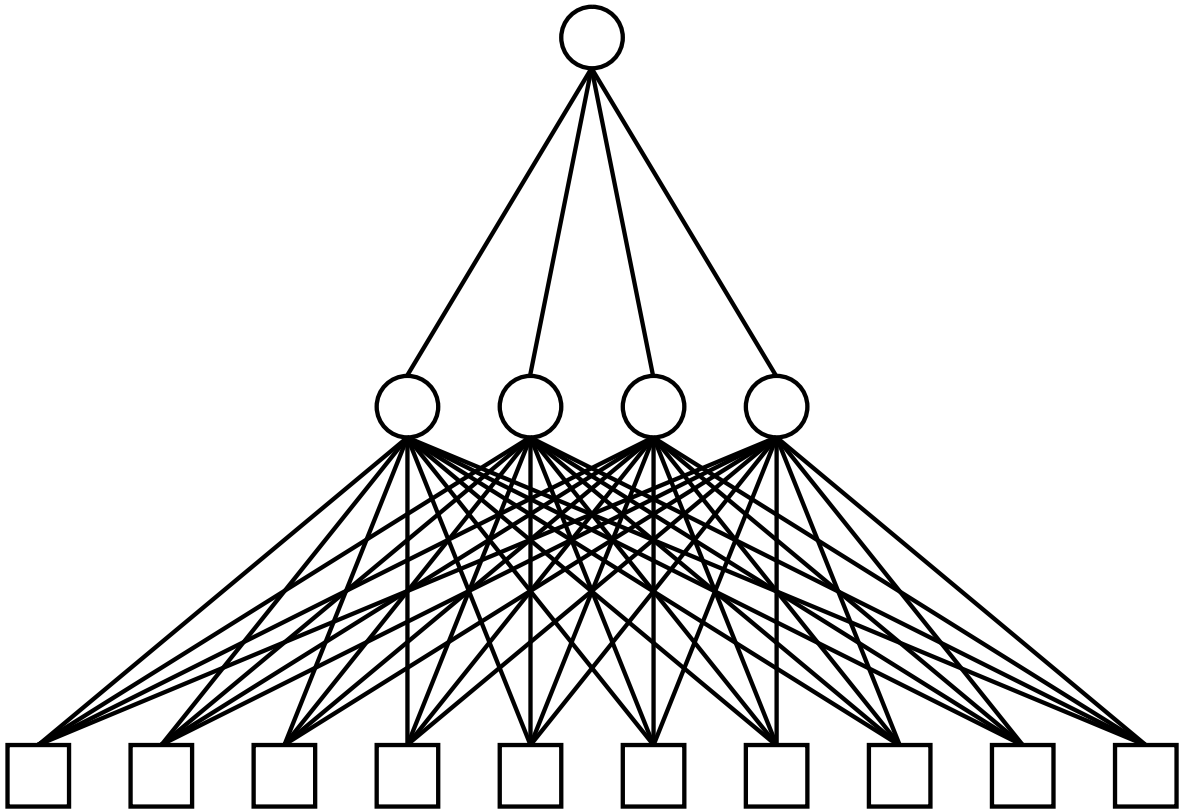
Output units O_i

$W_{j,i}$

Hidden units a_j

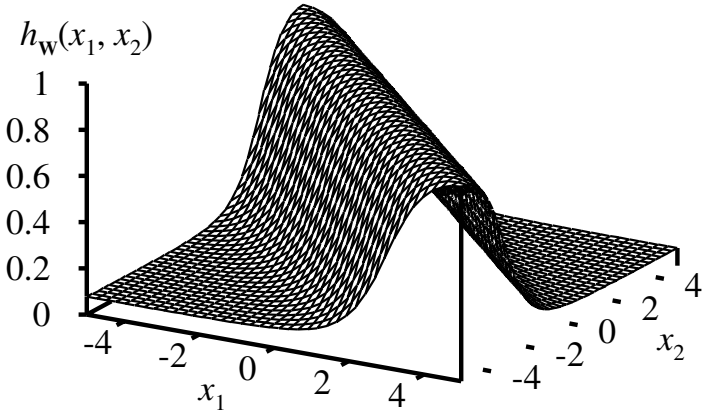
$W_{k,j}$

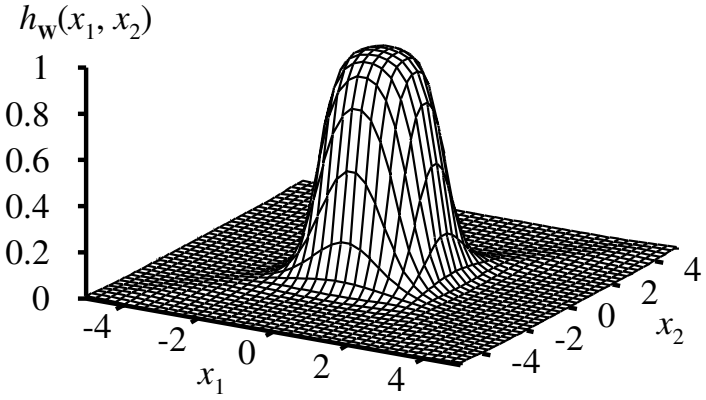
Input units I_k



Expressiveness

- All continuous functions with 2 layers, all functions with 3 layers



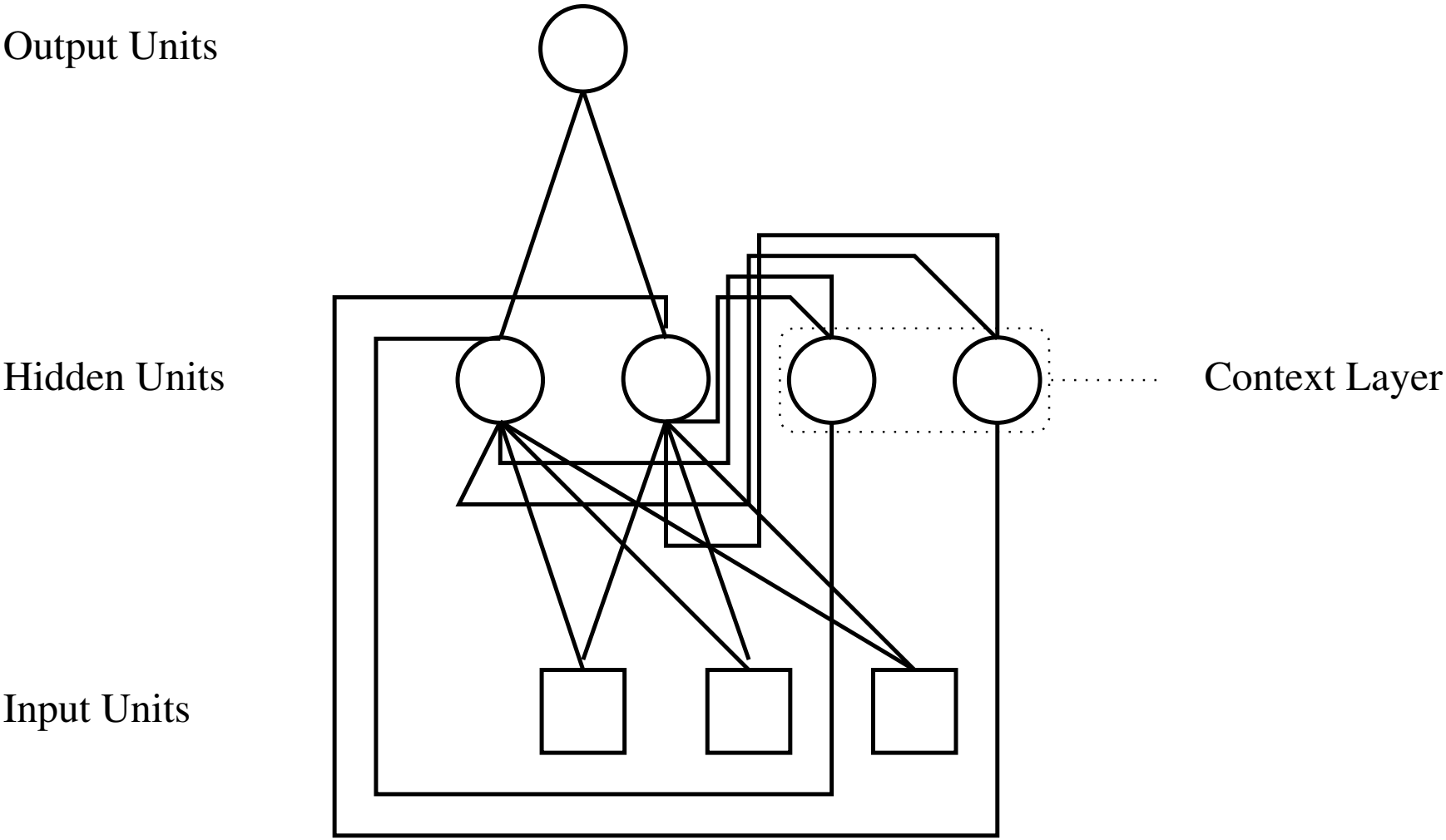


- Combine two opposite-facing threshold functions to make ridge
- Combine two perpendicular ridges to make bump
- Add various bumps to fit any surface

Recurrent Networks

- Recurrent neural networks have feedback connection (to store information over time)
- Elman network is a simple one that makes a copy of the hidden layer
- This copy is called context layer
- Context layer stores the previous state of the hidden layer

Elman network



Elman Network(2)

- The context layer feeds previous network states into the hidden layer
- Input vector: $\vec{x} = (\underbrace{x_1, \ldots, x_n}_{\text{actual inputs}} , \underbrace{x_{n+1}, \ldots, x_{2n}}_{\text{context units}})$
- Connections from each hidden unit to corresponding context unit has weight 1
- Context units are fully interconnected with all hidden units (not necessarily with weight 1)

Learning

- For simple Boolean or majority functions it is easy to find appropriate weights
- Generally, by adjusting the weights, we change the function that a network represents
- That is how learning occurs in neural networks
- When we have no prior knowledge about the function – except for data – we have to learn values for W_j from this data

Learning(2)

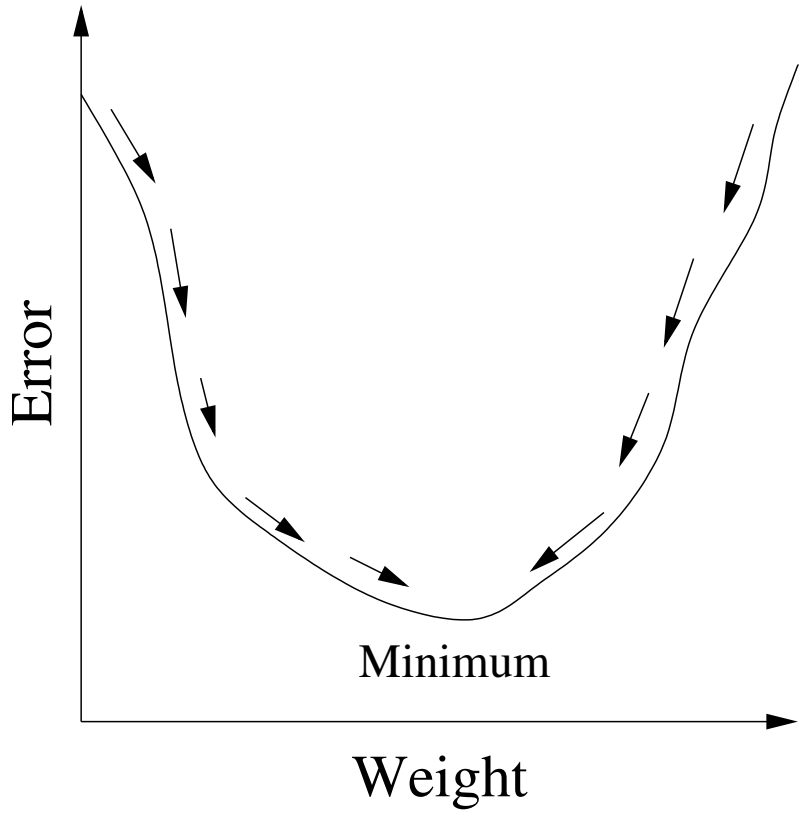
- Three main types of learning (we are looking at first two)
  - Supervised learning
    - Network is provided with a data set of input vectors and desired output (training set)
    - Adjust the weights so that the error between the real output and the desired output is minimized
  - Unsupervised learning
    - Clusters the training set to discover patterns or features in the input data
  - Reinforcement learning
    - Reward the network for good performance, penalize it for bad performance

Supervised Learning

- Gradient descent is widely popular approach to train (single-layer) networks
- Idea: adjust the weights in of the network to minimize some measure of the error on training set
- “Classical” measure of error is sum of squared errors
- Squared error for a single training example with input $\vec{x}$ and desired output y is $E = \frac{1}{2}Err^2 = \frac{1}{2}(y - h_{\vec{W}}(\vec{x}))^2$ (where $h_{\vec{W}}(\vec{x})$ is output of perceptron)

Gradient Descent

- Depending on the gradient of the error, we increase or decrease the weight



Gradient Descent(2)

- For calculating the gradient, we need some calculus
- We need to determine a partial derivative of E with respect to each weight:

$$\begin{aligned}
\frac{\partial E}{\partial W_j} &= \frac{\partial \frac{1}{2} Err^2}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} \\
&= Err \times \frac{\partial}{\partial W_j} \left(y - g \left(\sum_{j=0}^{n} W_j \times x_j \right) \right) \\
&= Err \times (-g'(in) \times x_j)
\end{aligned}$$

- g' = derivative of activation function (e.g. for sigmoid, $g' = g(1-g)$)

Gradient Descent(3)

- In the gradient descent algorithm, W_js are updated as follows:

$$W_j = W_j + \alpha \times Err \times g'(in) \times x_j$$

- α is the learning rate:
  - Size of the steps taken in the negative direction of the gradient

Gradient Descent(4)

- Complete algorithm runs training examples through the net one at a time (adjusting the weights slightly)
- Each cycle is called an epoch
- Epochs are repeated until some stopping criterion is reached
  - E.g. weight changes become very small
  - Only converges for linearly separable data set

Gradient Descent(5)

- Different variants for cycling through training examples:
  - Batch: adding up all gradient contributions and adjusting weights at end of epoch
  - Stochastic: select examples randomly
- There are many other methods besides gradient descent:
  - Widrow-Hoff
  - Generalized Delta
  - Error-Correction
  - …

Back-propagation Learning

- Learning in a multi-layer network is a little different
- Minor difference: we now have several outputs and an output vector $\vec{h}_{\vec{W}}(\vec{x})$
- Major difference: error at output layer is clear, error in hidden layers is unclear
- Idea: back-propagate error from output layer to hidden layers

Back-propagation Learning(2)

- At output layer, weight update is identical to gradient descent
- We have multiple output units, so let Err_i be the i-th component of the error vector $\vec{y} - \vec{h}_{\vec{W}}(\vec{x})$, so

$$W_{j,i} = W_{j,i} + \alpha \times \underbrace{Err_i \times g'(in_i)}_{\Delta_i} \times x_j$$

- Now we need to connect the output units to the hidden units

Back-propagation Learning(3)

- Idea: hidden node j is "responsible" for some fraction of the error in the nodes to which it connects
- Δ_i values are divided according to the weights of the connections:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

- Now we can use same weight-update rule for the hidden nodes

$$W_{k,j} = W_{k,j} + \alpha \times \Delta_j \times x_k$$

Back-propagation Learning(4)

- Back-propagation process can be summarized as follows:
  - Compute the Δ values for the output units (using the observed error)
  - Starting with output layer, repeat for each layer until earliest layer is reached:
    - Propagate the Δ values back to the previous layer
    - Update the weights in the previous layer

Unsupervised Learning

- In supervised learning, supervisor (or teacher) presents an input pattern and a desired response
- Neural networks try to learn functional mapping between input and output
- Unsupervised learning's objective is to discover patterns of features in input data
- This is done with no help or feedback from teacher
- No explicit target outputs are prescribed, however, similar inputs will result in similar outputs

Hebbian Learning Rule

- Developed by Donald Hebb, a neuropsychologist, is one of the oldest learning rules (1949)
- Idea: when neuron A repeatedly participates in firing neuron B, the strength of the action of A onto B increases
- Formally speaking, this means

$$W_{j,i} = W_{j,i} + \alpha \times g(in_i) \times g(in_j)$$

- α is learning rate and neuron j “feeds” neuron i

Hebbian Learning Rule(2)

- Summary of Hebbian learning rule
  - Initialize all weights (e.g. small random values)
  - For each input pattern, compute corresponding output vector
  - Adjust the weights as shown on last slide
  - Repeat from step 2 until stopping criterion has been reached

Hebbian Learning Rule(3)

- Problem with Hebbian learning: repeated presentations of input patterns leads to an unlimited growth in weight values
- Solution: impose limit on increase in weight
- One type of limit is to introduce a nonlinear "forgetting factor" β:

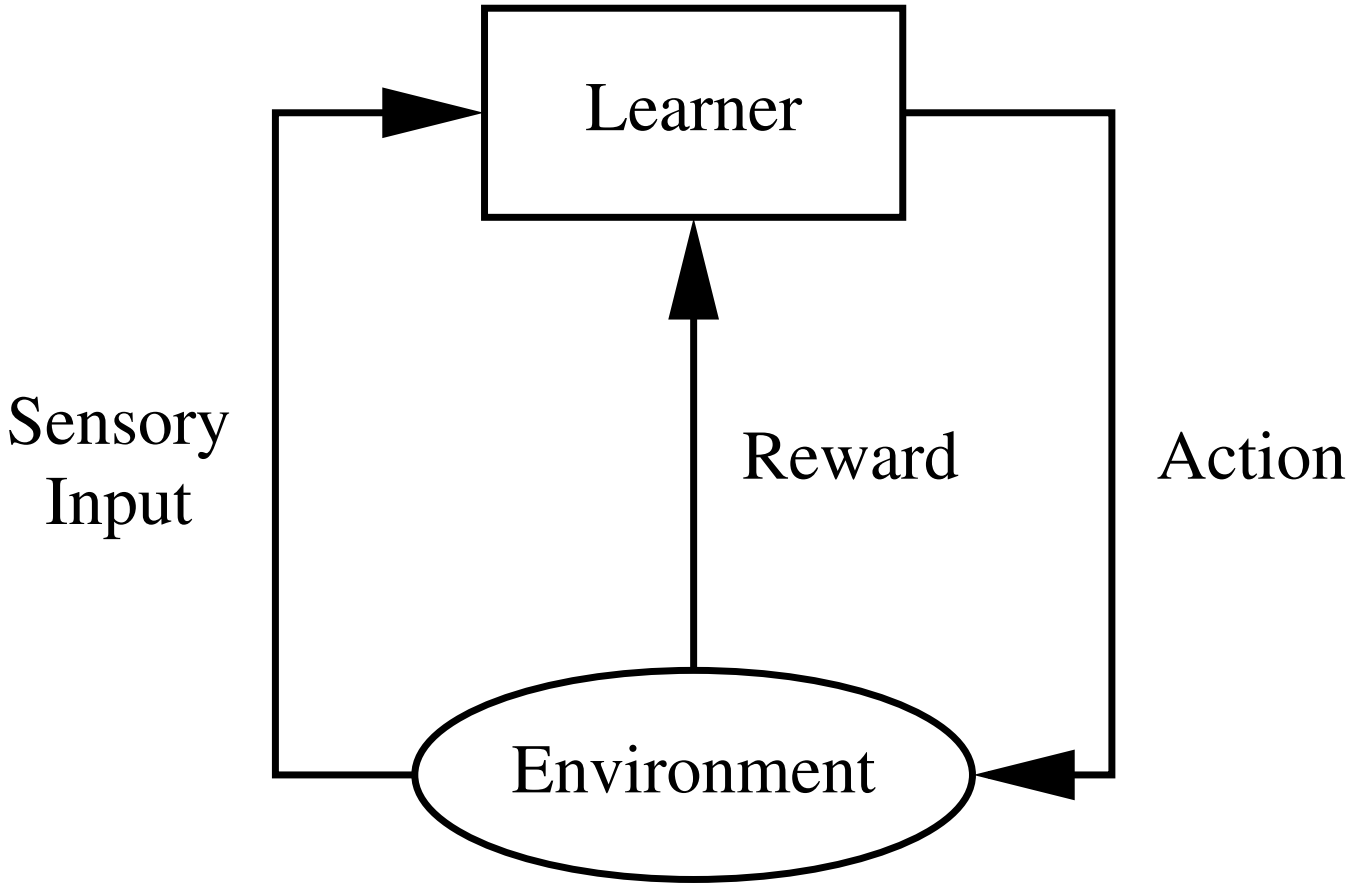
$$W_{j,i} = W_{j,i} + \alpha \times g(in_i) \times g(in_j) - \beta \times g(in_j) \times W_{j,i}$$

Reinforcement Learning

- In supervised learning an input data set and a full set of desired outputs is presented
- In reinforcement learning the feedback is not as elaborate
  - Desired output is not described explicitly
  - Learning network only gets feedback whether output was a success or not
  - Learning with a critic (rather than learning with a teacher)
- Main objective is to maximize the (expected) reward or reinforcement signal

Reinforcement Learning(2)

- General situation:



Learning Rule

- Neural network reinforcement learning usually requires a multi-layer architecture
- An external evaluator is needed to decide whether network has scored a success or not
- Every node in the network receives a scalar reinforcement signal r representing quality of output
- r is between 0 and 1, 0 meaning maximum error, 1 meaning optimal
- Compared to back-propagation (where output nodes receive error signal, which is propagated backward), here every node receives same signal

Learning Rule(2)

- Mazzoni et al. presented following weight-update algorithm (based on Hebbian learning):

$$W_{j,i} = W_{j,i} + \alpha \times \quad (\quad (g(in_i) - p_i) \times g(in_j) \times r + \\ \lambda \times (1 - g(in_i) - p_i) \times g(in_j) \times (1 - r))$$

- where λ is a constant, and p_i is the probability of neuron i firing
- A correct response (large r) will strengthen connections that were active during the response
- An incorrect response (small r) will weaken active synapses

Learning Structures

- So far, we have only looked at learning weights (given a fixed network structure)
- How do we find the best network structure?
- Choosing a network that is too small:
  - May not be powerful enough to get task done
- Choosing a network that is too big:
  - Problem of overfitting: network memorizes examples rather than generalizing

Learning Structures(2)

- If we stick to fully connected networks, the only choices are:
  - The number of hidden layers
  - The number of neurons in each
- Usual approach: try several and keep the best
  - Try to keep it small to avoid overfitting

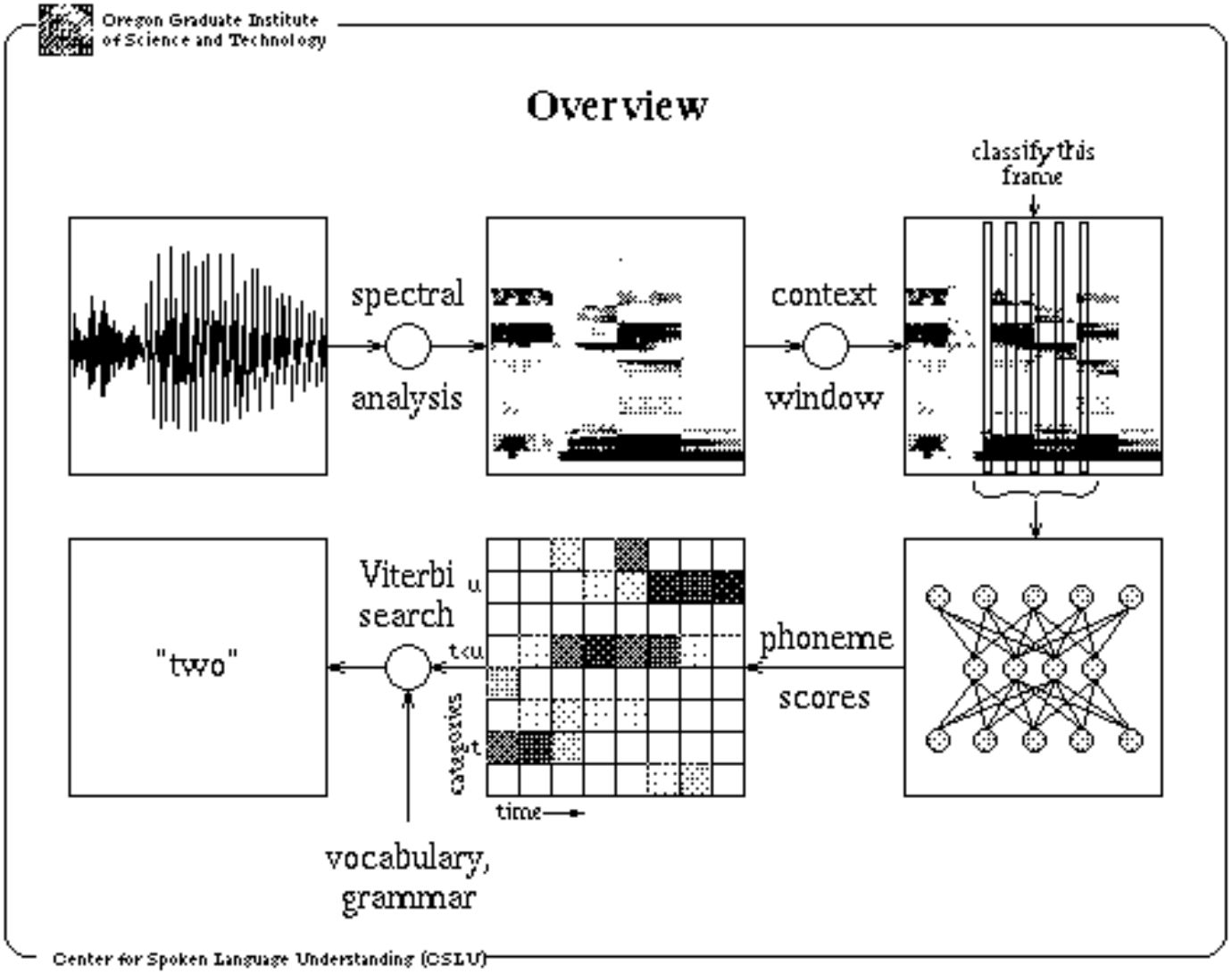
Learning Structures(3)

- Let us now consider networks that are not fully connected
- We need some effective search method to “weed out” connections
- One approach is optimal brain damage algorithm
  - Starts with a fully connected network and removes connections from it
  - After first training an information-theoretic approach identifies a selection of connections to be dropped
  - Network is retrained and if performance has not decreased, process is repeated
  - It is also possible to remove neurons that are not contributing much to result

Learning Structures(4)

- Several algorithms for growing larger network from smaller one
- Tiling algorithm starts with a single unit that tries its best
- Subsequent units are added to take care of examples that first unit got wrong
- Algorithm adds only as many units as are needed to cover all examples

Applications:Speech Recognition



Applications:Handwriting Recognition

- 400-300-10 unit network: 1.6% error
- 768-192-30-10 unit LeNet: 0.9% error

Applications:Fraud Detection

- Banks are using AI software (including neural networks) to detect fraud
- Have the ability to detect fraudulent behavior by analyzing transactions and alerting staff
- Credit card fraud losses in the UK fell for the first time in nearly a decade in 2003 (by more than 5% to 402.4m pounds)
- Barclays reported that after installing a system in 1997, fraud was reduced by 30% by 2003

Applications:CNC

- Neural networks are also used in computer numerically controlled (CNC) machines
- E.g. Siemens SINUMERIK 840D controller for drilling, turning, milling, grinding and special-purpose machines

Applications:Drug Design

- Used for testing if certain anti-inflammatory drugs cause adverse reactions
- The rate of these reactions is about 10% (with 1% serious and 0.1% fatal)
- Three-layer, backpropagated network was used to predict serious reactions
- Predicted rate matched within 5% of observed rate

Chapter Summary

- Neural networks are an AI technique modeled on the brain
- Single-layer feed-forward networks can represent linearly separable functions
- Multi-layer feed-forward networks can represent any function (given enough units)
- Recurrent networks can store information over time
- Many different techniques to train networks
- Neural networks have been used for hundreds of applications