Neural networks

We shall envision the mind (or brain) as composed of many partially autonomous "agents"as a "Society" of smaller minds. Each sub-society of mind must have its own internal epistemology and phenomenology, with most details private, not only from the central processes, but from one another. (Minsky, K-Lines; 1980)

Lesson in neuronal politics:

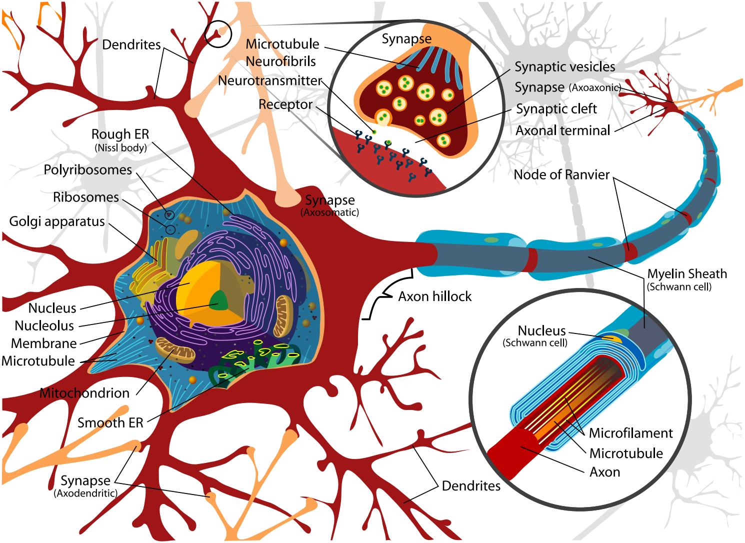
Strong local/individual policies have many strengths: sustainable, realistic, flexible, robust, and fault-tolerant

Neural networks: Objectives

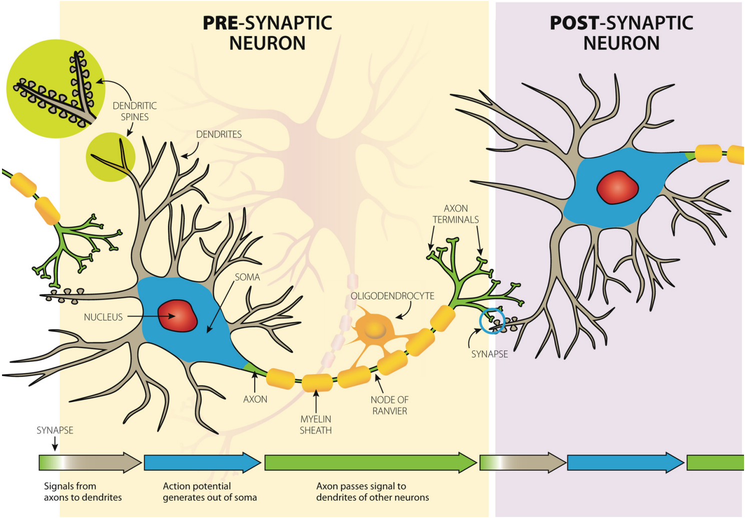
At the end of this section you should be able to:

- Detail the basic features of biological neurons
- Draw and formulate the equations for a basic neuron and its structure
- Describe various network structures
- Understand various learning rules and their limitations

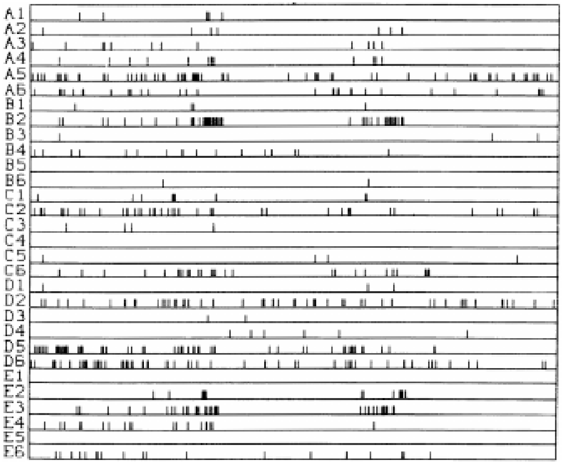
Real neurons



Pre- and Post- synaptic

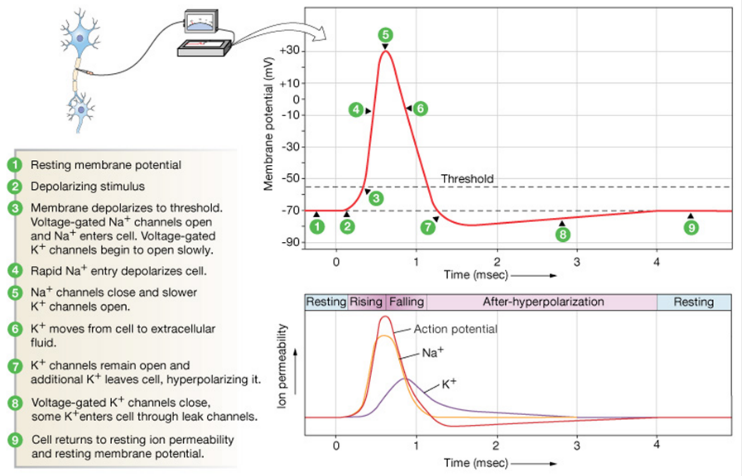


Action potentials



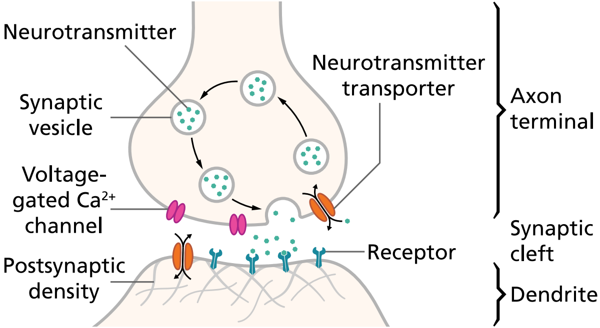
A 4 second recording of the neural activity recording from 30 neurons of the visual cortex of a monkey. Each vertical bar indicates a spike. The human brain can recognize a face within 150ms, which correlates to less than 3mm in this diagram; dramatic changes in firing frequency occur in this time span, neurons have to rely on information carried by solitary spikes.

Neurons spike to “think” (mostly)



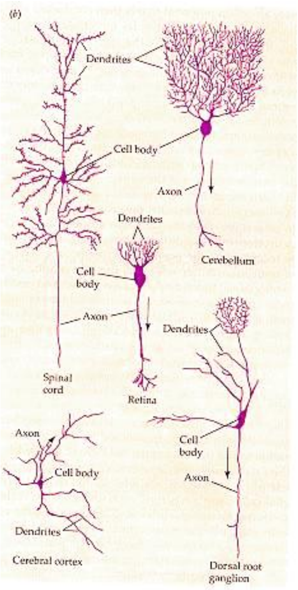
Neurons are unequivocally the basis of human/animal thinking, learning, consciousness, etc.

Synapses: inter-neuron signaling / learning



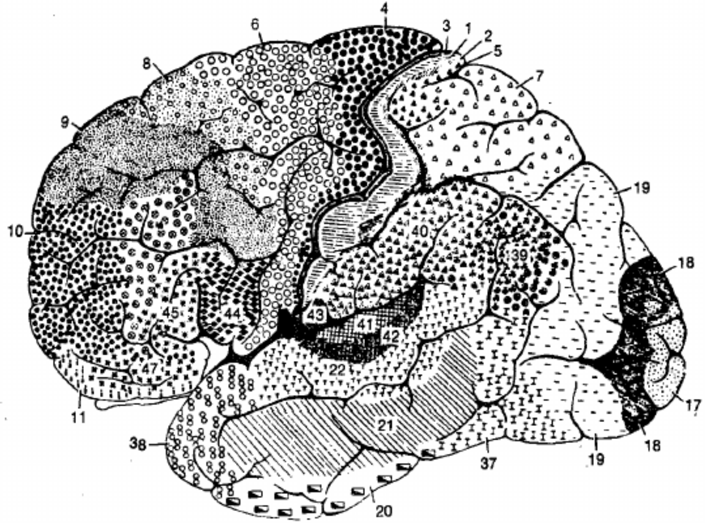
- Rate-limited step is transmission between neurons
- Learning is mostly rooted in the synapses
- Neurons change their reactivity and weights to learn

Diversity of neuron types



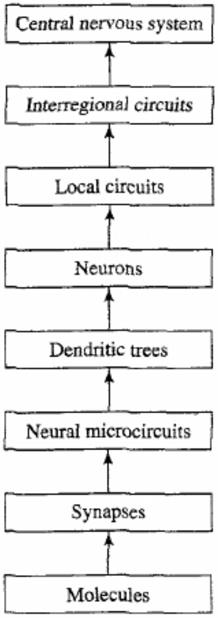
What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity (and size), not from any single, perfect principle. (Marvin Minsky, Society of Mind; 1987)

Diversity of neuron types cont...



Network structure varies on a macro scale.

Level of abstraction



Which level of abstraction to model?

Neurons are slow (compared to computers) and fairly small...

typical time-scales	
action potential:	$\sim 1msec$
reset time:	$\sim 3msec$
synapses:	$\sim 1msec$
pulse transport:	$\sim 5m/sec$

typical sizes	
cell body:	$\sim 50\mu m$
axon diameter:	$\sim 1\mu m$
synapse size:	$\sim 1\mu m$
synaptic cleft:	$\sim 0.05\mu m$

Brains vs. Computers

conventional computers	biological neural networks
processors *operation speed* $\sim 10^8 Hz$ *signal/noise* $\sim \infty$ *signal velocity* $\sim 10^8 m/sec$ *connections* ~ 10	neurons *operation speed* $\sim 10^2 Hz$ *signal/noise* ~ 1 *signal velocity* $\sim 1 m/sec$ *connections* $\sim 10^4$
sequential operation program & data external programming	parallel operation connections, neuron thresholds self-programming & adaptation
hardware failure: fatal no unforseen data	robust against hardware failure messy, unforseen data

	process elements	element size	speed	computation	robust	learns	intelligent, conscious
Brain	10^{14} synapses	10e-6m	100Hz	parallel, distr	yes	yes	usually
Computer	10^8 transistors	10e-6m	10^9Hz	serial, central	no	a little	Debateably yes

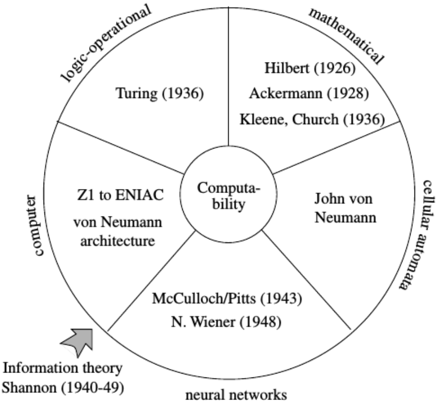
Brains vs. Computers: Robustness

- performance degrades gracefully under partial damage. In contrast, most programs and engineered systems are brittle: if you remove some arbitrary parts, very likely the whole will cease to function.
- brain reorganizes itself from experience.
- it performs massively parallel computations extremely efficiently. For example, complex visual perception occurs within less than 30 ms, that is, 10 processing steps!
- Flexible, and can adjust to new environments
- Can tolerate (well) information that is fuzzy, inconsistent, probabalistic, noisy, or inconsistent
- Small and very energy efficient

Brains vs. Computers: function

- Traditional computing excels in many areas, but not in others.
- **A great definition:** AI is the the development of algorithms or paradigms that require machines to perform cognitive tasks at which humans are currently better.
- Symbolic rules don't reflect processes actually used by humans

Types of computation



- Neural networks can be universal general purpose computers, and in some app-specific hardware instances do better than Turing machines.

Types of computation

- The use of neural networks may seem to challenge the physical symbol system hypothesis, which relies on symbols having meaning.
- Although meaning is attached to the input and output units, the designer does not associate a meaning with the hidden units.
- What the hidden units actually represent is something that is learned.
- After a neural network has been trained, it is often possible to look inside the network to determine what a particular hidden unit actually represents.
- Arguably, the computer has an internal meaning; it can explain its internal meaning by showing how examples map into the values of the hidden unit.

(Artificial) Neural networks

- Massively parallel distributed processor made up of simple units, which has a natural propensity for storing and using experiential knowledge.
- Knowledge is acquired by the network from its environment through learning
- Interconnection strengths (synaptic weights) store acquired knowledge

Domains studying NNs

- Machine learning:
  - Having a computer program itself from a set of examples so you don't have to program it yourself.
  - **Optimization:** given a set of constraints and a cost function, how do you find an optimal solution? E.g. traveling salesman problem.
  - **Classification:** grouping patterns into classes: i.e. handwritten characters into letters.
  - **Associative memory:** recalling a memory based on a partial match.
  - **Regression:** function mapping

Domains studying NNs

- **Cognitive science:**
  - Modelling higher level reasoning: language, problem solving
  - Modelling lower level reasoning: vision, audition speech recognition, speech generation
- **Neurobiology:** Modelling models of how the brain works.
  - neuron-level
  - higher levels: vision, hearing, etc. Overlaps with cognitive folks.
- **Mathematics:**
  - Nonparametric statistical analysis and regression.

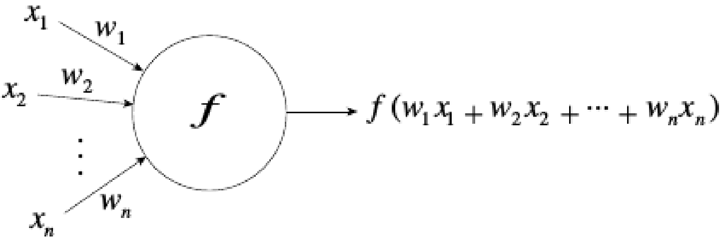
Applications

- Signal processing: suppress line noise, with adaptive echo canceling, blind source separation
- Control: e.g. backing up a truck: cab position, rear position, and match with the dock get converted to steering instructions. Manufacturing plants for controlling automated machines.
- Siemens successfully uses neural networks for process automation in basic industries, e.g., in rolling mill control more than 100 neural networks do their job, 24 hours a day
- Robotics - navigation, vision recognition
- Pattern recognition, i.e. recognizing handwritten characters, e.g. Apple's Newton used a neural net
- Medicine, i.e. storing medical records based on case information
- Speech production: reading text aloud (NETtalk)
- Speech recognition
- Vision: face recognition , edge detection, visual search engines
- Business,e.g.. rules for mortgage decisions are extracted from past decisions made by experienced evaluators, resulting in a network that has a high level of agreement with human experts.
- Financial Applications: time series analysis, stock market prediction
- Data Compression: speech signal, image, e.g. faces
- Game Playing: backgammon, chess, go, ...

Benefits of neural networks

- Nonlinearity: distributed throughout the network
- Input-output mapping: supervised learning
- Adaptivity: learn via synaptic weights
- Evidential response: give probability/confidence in decision
- Contextual information: distributed store of info, association
- Fault tolerance: individual neurons can be damaged
- VLSI implementability: hardware networks
- Standardized design, analysis, and theoretical literature
- Neurobiological analogy: much reciprocity between fields

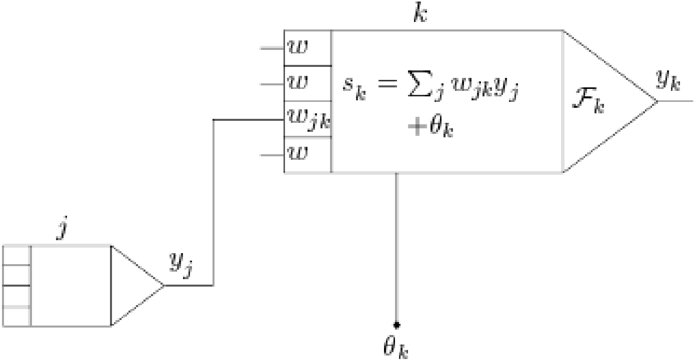
Basic neuron model



Neuron operations:

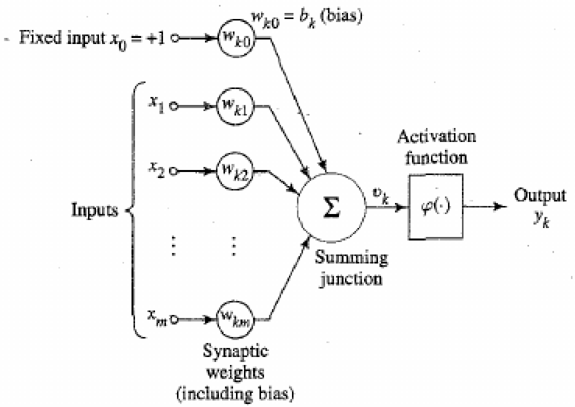
1. Sum (inputs x weights)
2. Apply activation function
3. Transmit signal

Basic neuron model



- Often a bias θ can be applied/learned

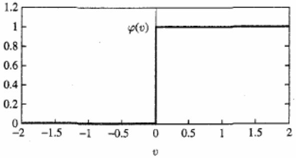
Basic neuron model



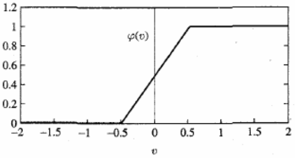
$$v_k = \sum_{j=0}^{m} w_{kj} x_j$$

$$y_k = \varphi(v_k)$$

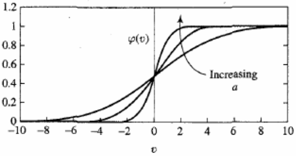
Activation functions: many types



(a)



(b)



$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases}$$

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

Note: $exp(x)$ is e^x

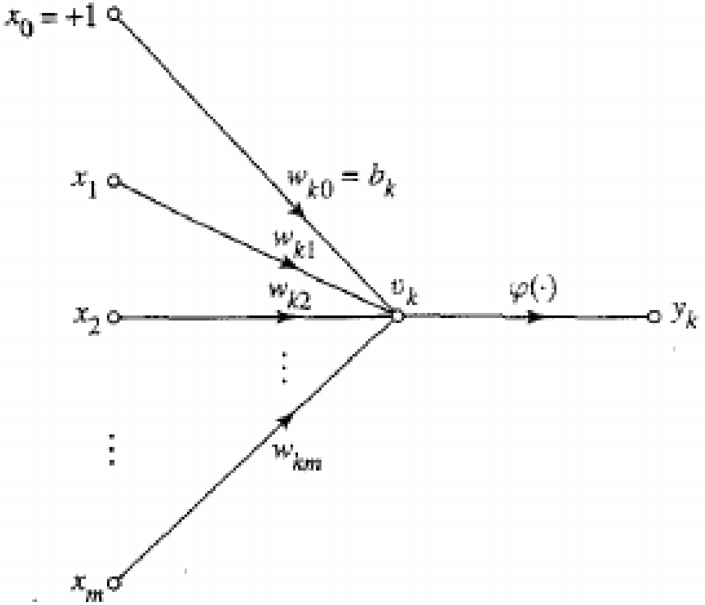
Alternative: Probability-based firing

$$x = \begin{cases} +1 & \text{with probability } P(v) \\ -1 & \text{with probability } 1 - P(v) \end{cases}$$

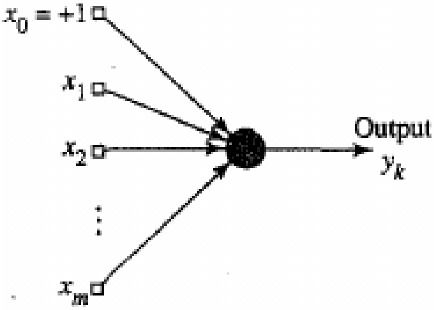
$$P(v) = \frac{1}{1 + \exp(-v/T)}$$

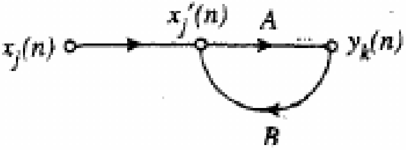
T is pseudo temperature used to control noise level (uncertainty)

Signal flow diagram

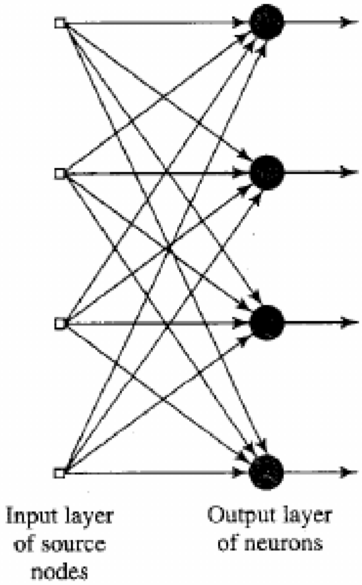


Architectural graphs and recurrence



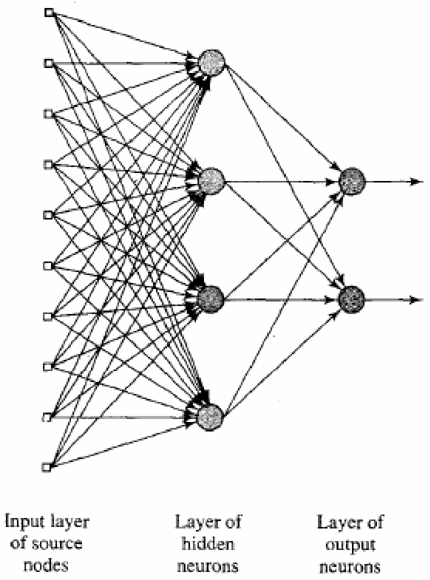


Single layer network

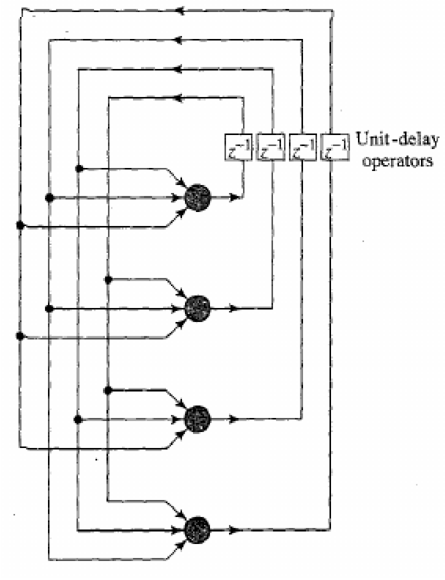


Brains
Neurons
Connections
Signals
Diversity
Levels
Scale
vs. Computers
Computation
Neural networks
Applications
Models
Activation func
Stochasticity
Signal flow
Graph structure
Learning
Unsupervised
Hebbian
Associative
Credit
Supervised
Competitive
Error Corr.
Multi-layer
Error
Backprop
Reinforcement
Overfitting

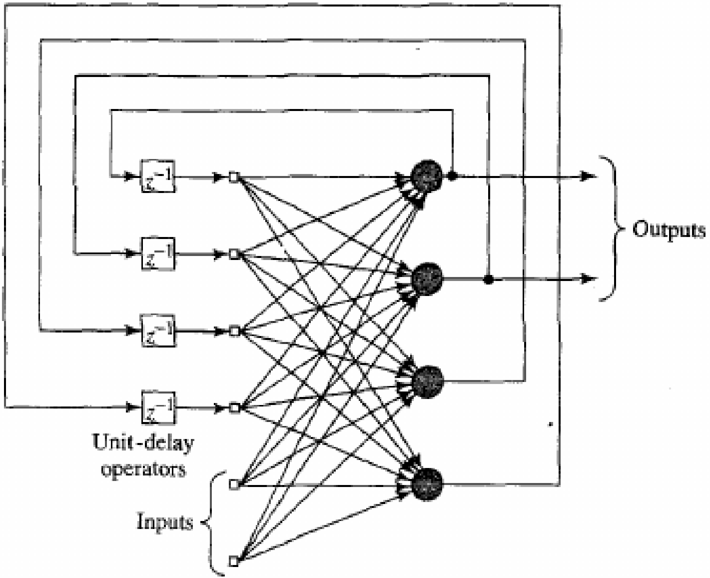
Multi-layer feed forward fully connected



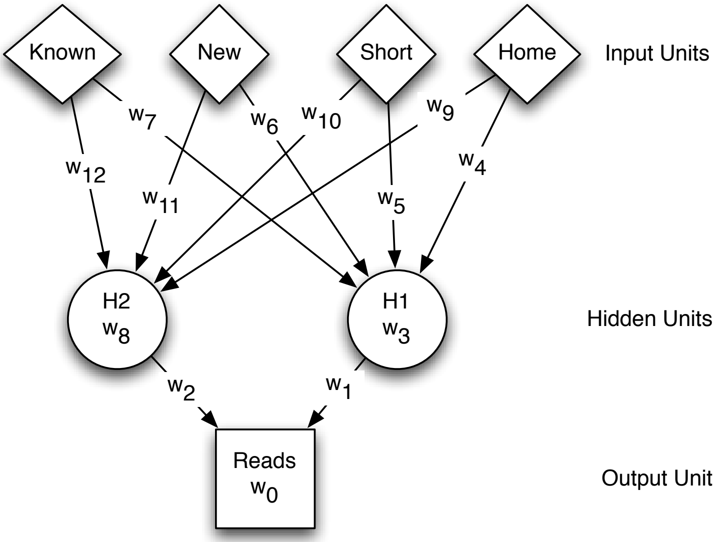
Recurrent network with no self feedback



Recurrent network with hidden neurons



Knowledge representation? newsgroup example

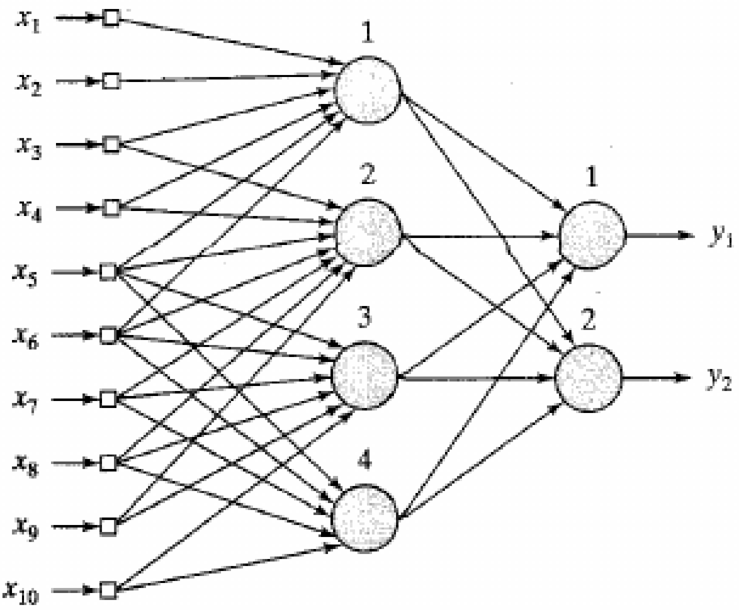


Knowledge? distributed / learned

Knowledge refers to stored information used to interpret, predict, or respond to the outside world. In a neural network:

- Similar inputs should elicit similar activations/representations in the network
- The inverse: dissimilar items should be represented very differently
- Important features should end up dominating the network
- Prior information can be built into the network, though it is not required, e.g., receptive fields

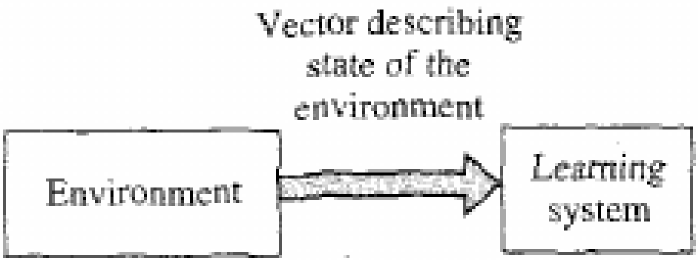
Receptive fields: What is different here?



Learning in NN

- **Learning** is a process by which the free parameters (**synaptic weights**) of the network are adapted through a process of stimulation/activation by the environment in which the network is embedded.
- The type of learning is determined by the ways the parameters are changed: e.g., Supervised (with sub-types), Unsupervised (with sub-types), and Reinforcement learning.
- A set of well-defined rules for updating weights is defined as a **learning algorithm**
- The mapping from environment to network to task is often coined the **learning paradigm**

Unsupervised learning



- E.g., clustering, auto-associative, etc

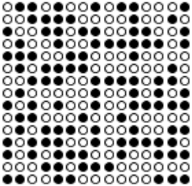
Hebbian learning:

- Hebbian theory is a theory in neuroscience that proposes an explanation for the adaptation of neurons in the brain during the learning process.
- "Fire together, wire together"
- $\Delta w_i = \eta x_i y$
  or the change in the *ith* synaptic weight w_i is equal to a learning rate η times the *ith* input x_i times the postsynaptic response y. Weights updated after every training example
- Variants of this are very successful at clustering problems, and can provably perform ICA, PCA, etc.

Associative learning (can be supervised)

$$w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) > 0 : \quad S_i(t+1) = 1$$
$$w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) < 0 : \quad S_i(t+1) = -1$$

to be depicted as



$\bullet : \quad S_i = 1 \quad$ (neuron i firing)

$\circ : \quad S_i = -1 \quad$ (neuron i at rest)

$$input_i > 0 : \quad S_i \to 1$$
$$input_i < 0 : \quad S_i \to -1$$

$$input_i = w_{i1}S_1 + \ldots + w_{iN}S_N$$

Hebbian-like rule: $\quad S_i = S_j : \quad w_{ij} \uparrow \qquad S_i \neq S_j : \quad w_{ij} \downarrow \qquad\qquad w_{ij} \to w_{ij} + S_i S_j$

$t=0 \qquad t=1 \qquad t=2 \qquad t=3 \qquad t=4$

After learning, activate original from noisy version.

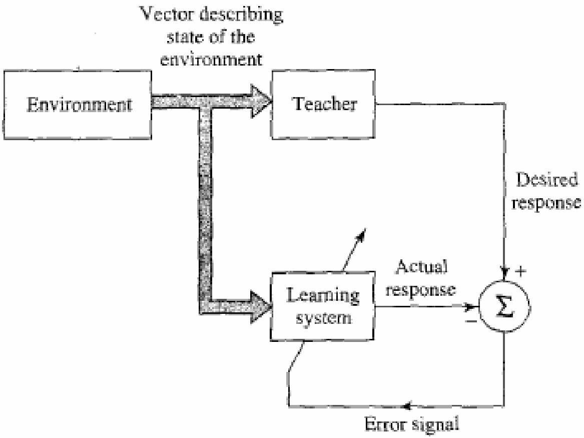
Clustering

We'll go over a little more in clustering, with spiking networks Thursday

Credit assignment problem

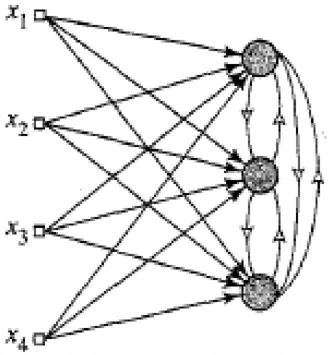
- **Structural:** Which weights need changing due to good/bad outcome?
- **Temporal:** Which preceding internal decisions resulted in the delayed reward?

Learning with a teacher



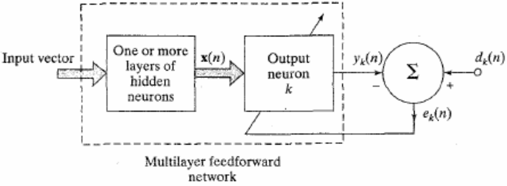
- Supervised learning: attempts to minimize the error between the actual outputs, i.e., the activation at the output layer and the desired or target activation, by changing the values of the weights.

Competitive learning

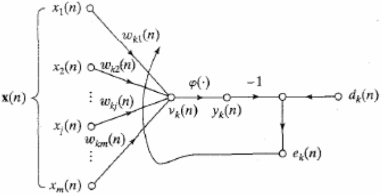


- Winner-takes all based weight updates (inhibition of lateral neighbors). Similar to functions in retina

Basic error correction learning



(a) Block diagram of a neural network, highlighting the only neuron in the output layer



(b) Signal-flow graph of output neuron

Error:

$$e_k(n) = d_k(n) - y_k(n)$$

Minimize:

$$\mathcal{E}(n) = \frac{1}{2} e_k^2(n)$$

Update via:

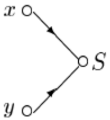
$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n)$$

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

AND, OR, NOT

AND:

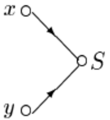
x	y	$x \wedge y$	$x+y-\frac{3}{2}$	S
0	0	0	$-3/2$	0
0	1	0	$-1/2$	0
1	0	0	$-1/2$	0
1	1	1	$1/2$	1



$$w_1 = w_2 = 1$$
$$\theta = \frac{3}{2}$$

OR:

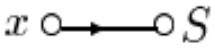
x	y	$x \vee y$	$x+y-\frac{1}{2}$	S
0	0	0	$-1/2$	0
0	1	1	$1/2$	1
1	0	1	$1/2$	1
1	1	1	$3/2$	1



$$w_1 = w_2 = 1$$
$$\theta = \frac{1}{2}$$

NOT:

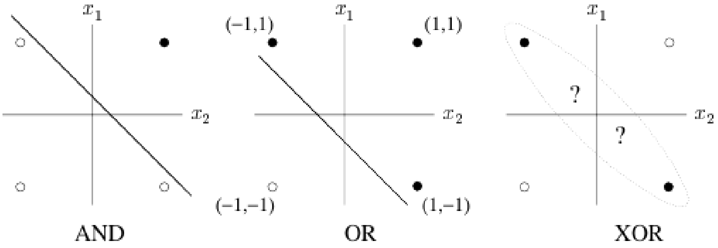
x	$\neg x$	$-x+\frac{1}{2}$	S
0	1	$1/2$	1
1	0	$-1/2$	0



$$w_1 = -1$$
$$\theta = -\frac{1}{2}$$

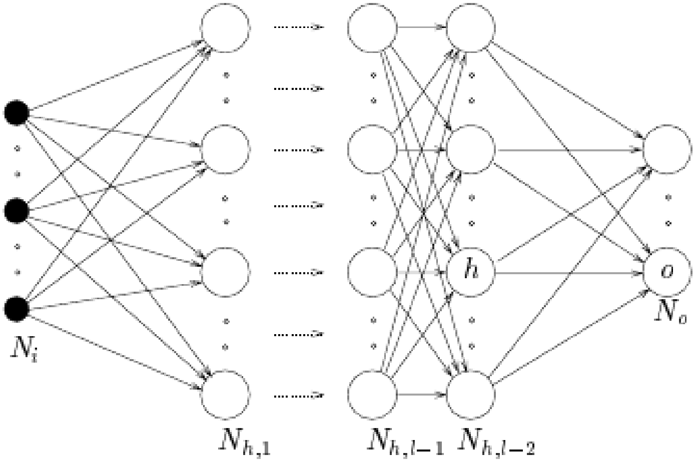
- Easy for linear single layer network with 2 neurons and a bias, with step activation.

XOR



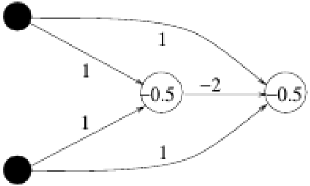
- **Problem:** Requires a hidden layer (for non-linearity)

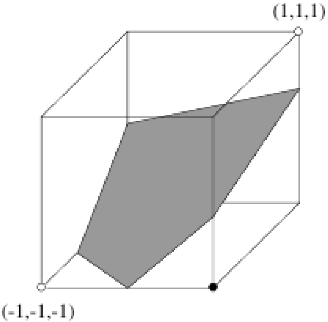
Solution: N-layer network



- **Solution:** Can solve any non-linear function

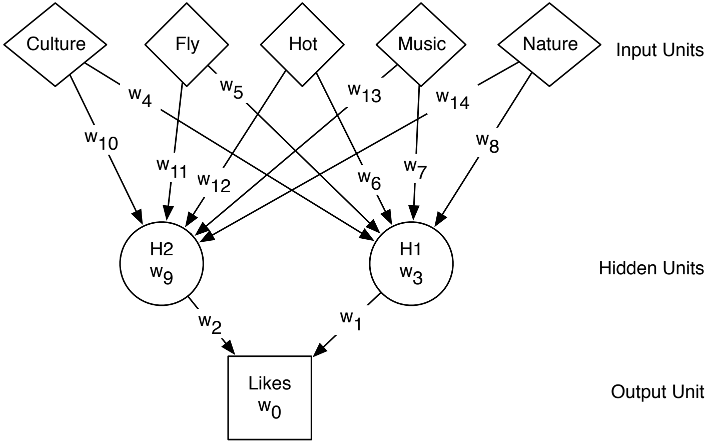
XOR



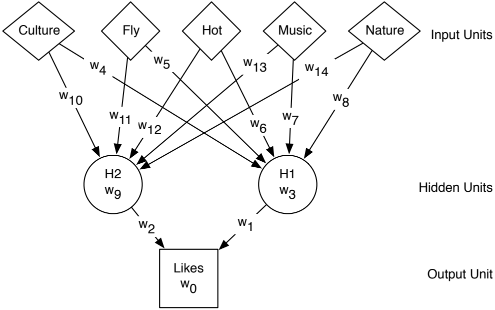


- Separation into 3D via hidden layer allows solving XOR
- **Problem:** How to solve for errors in hidden layer??

Neural network for traveling example



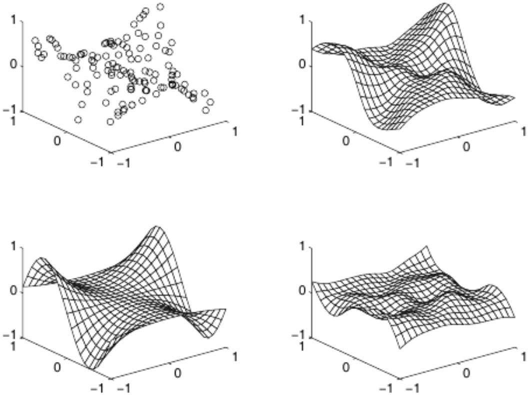
Neural network for traveling example



Given input example, e, what is output prediction?

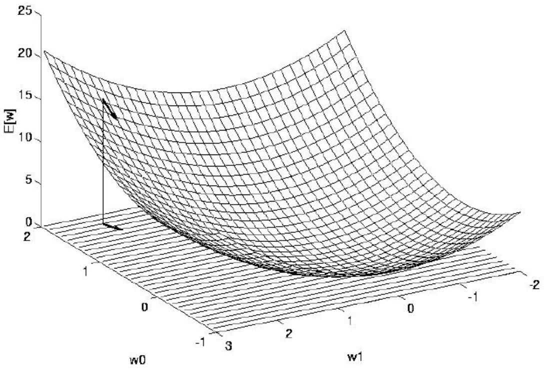
- $val(e, H1) = f(w_3 + w_4 val(e, Culture) + w_5 val(e, Fly) + w_6 val(e, Hot) + w_7 val(e, Music) + w_8 val(e, Nature)$
- $val(e, H2) = f(w_9 + w_10 val(e, Culture) + w11 val(e, Fly) + w_12 val(e, Hot) + w_13 val(e, Music) + w_14 val(e, Nature))$
- $pval(e, Likes) = f(w_0 + w_1 val(e, H1) + w2 val(e, H2))$

Error gradients



- **Top left:** original samples; **Top right:** network approximation;
- **Bottom left:** true function which generated samples; **Bottom right:** raw error

Error gradients: simple



- Error (vertical) as function of 2 weights (x_1 and x_2)

Error

- How much should we change each weight?
- In proportion to its influence on the error.
- The bigger the influence of weight w_m , the greater the reduction of error that can induced by changing it
- This influence wouldn't be the same everywhere: changing any particular weight will generally make all the others more or less influential on the error, including the weight we have changed.

Solution: Error backpropagation

Step 1: Propagation: Each propagation involves the following:

- Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
- Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (difference between the input and output values) of all output and hidden neurons.

Solution: Error backpropagation

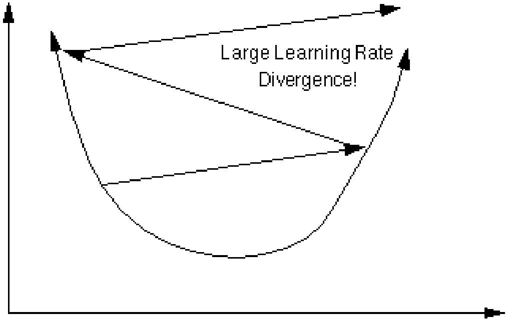
Step 2: Weight update: For each weight-synapse do the following:

- Multiply its output delta and input activation to get the gradient of the weight.
- Subtract a ratio (percentage) of the gradient from the weight.

The ratio (percentage) influences the speed and quality of learning; it is called the learning rate. The greater the ratio, the faster the neuron trains; the lower the ratio, the more accurate the training is. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

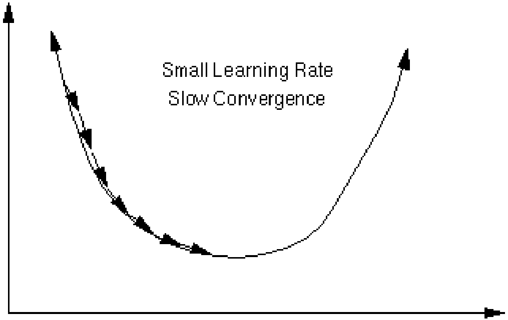
Finally: Repeat step 1 and 2 until the performance of the network is satisfactory.

Learning rate



- Learning rate is too large

Learning rate



- Learning rate is too small

Solution: Error backpropagation

Overview and basic idea:

```
initialize network weights (often small random values)
do
    for Each training example ex
        prediction = neural-net-output(network, ex) // forward pass
        actual = teacher-output(ex)
        compute error (prediction − actual) at the output units, as △
        Starting with output layer, repeat until layer I (input):
                propagate △ values back to previous layer
                update network weights between the two layers
until all examples classified correctly or another stopping criterion satisfied
return the network
```

Backprop(from ArtInt)

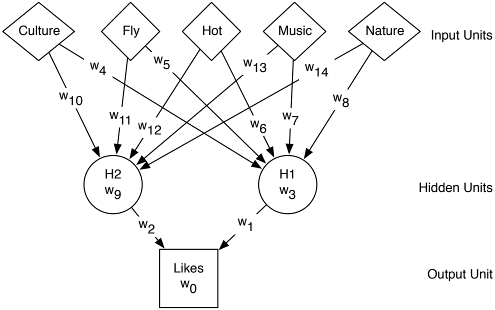
```
Procedure BackPropagationLearner(X,Y,E,n_h,η)
      Inputs
            X: set of input features, X={X_1,...,X_n}
            Y: set of target features, Y={Y_1,...,Y_k}
            E: set of examples from which to learn
            n_h: number of hidden units
            η: learning rate
      Output
            hidden unit weights hw[0:n,1:n_h]
              output unit weights ow[0:n_h,1:k]
       Local
              hw[0:n,1:n_h] weights for hidden units
              ow[0:n_h,1:k] weights for output units
              hid[0:n_h] values for hidden units
              hErr[1:n_h] errors for hidden units
              out[1:k] predicted values for output units
              oErr[1:k] errors for output units
       initialize hw and ow randomly
       hid[0]←1
       repeat
              for each example e in E do
                     for each h ∈{1,...,n_h} do
                            hid[h] ← f(∑_{i=0}^{n} hw[i,h] ×val(e,X_i))
                     for each o ∈{1,...,k} do
                            out[o] ← f(∑_{h=0}^{n} hw[i,h] ×hid[h])
                            oErr[o] ←out[o]×(1-out[o])×(val(e,Y_o)-out[o])
                     for each h ∈{0,...,n_h} do
                            hErr[h] ←hid[h]×(1-hid[h])×∑_{o=0}^{k} ow[h,o] ×oErr[o]
                            for each i ∈{0,...,n} do
                                   hw[i,h]←hw[i,h] + η×hErr[h]×val(e,X_i)
                            for each o ∈{1,...,k} do
                                   ow[h,o]←ow[h,o] + η×oErr[o]×hid[h]
       until termination
       return w_0,...,w_n
```

This approach assumes *n* input features, *k* output features, and *nh* hidden units. Both *hw* and *ow* are two-dimensional arrays of weights. Note that $0 : nk$ means the index ranges from 0 to *nk* (inclusive) and $1 : nk$ means the index ranges from 1 to *nk* (inclusive). This algorithm assumes that $val(e, X_0) = 1$ for all *e*

Backprop (from AIMA)

function BACK-PROP-LEARNING(examples, network, α) **returns** a neural network
inputs: *examples*, each of which has input vector $\mathbf{x}$ and output vector $\mathbf{y}$
network with L layers, weights $w_{i,j}$, activation function g
α: learning rate
local variables: $\triangle$, a vector of errors, indexed by network node
repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x},\mathbf{y})$ in *examples* **do**
    //Propagate the inputs forward to compute the outputs//
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $l = 2$ to L **do**
      for each node j in layer l **do**
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_i \leftarrow g(in_j)$
    //Propagate deltas backward from output layer to input layer//
    for each node j in the output layer **do**
      $\triangle[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $l = L - 1$ **to** 1 **do**
      for each node i in layer l **do**
        $\triangle[i] \leftarrow g'(in_i) \sum_j w_{i,j} \triangle[j]$
    //Update every weight in the network using deltas//
    for each weight in $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \triangle[j]$
until some stopping criterion is satisfied
return network

Neural network for traveling example



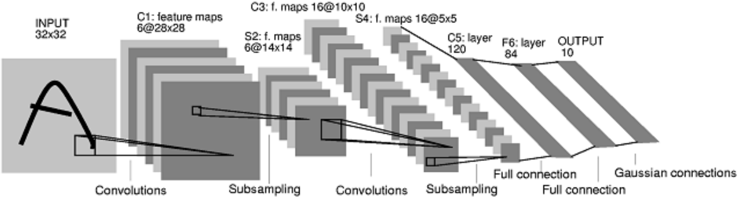
One hidden layer containing two units, trained on the travel data, can perfectly fit. One run of back-propagation with the learning rate =0.05, and taking 10,000 steps, gave weights that accurately predicted the training data:

- $H1 = f(-2.0Culture - 4.43Fly + 2.5Hot + 2.4Music - 6.1Nature + 1.63)$
- $H2 = f(-0.7Culture + 3.0Fly + 5.8Hot + 2.0Music - 1.7Nature - 5.0)$
- $Likes = f(-8.5H1 - 8.8H2 + 4.36)$

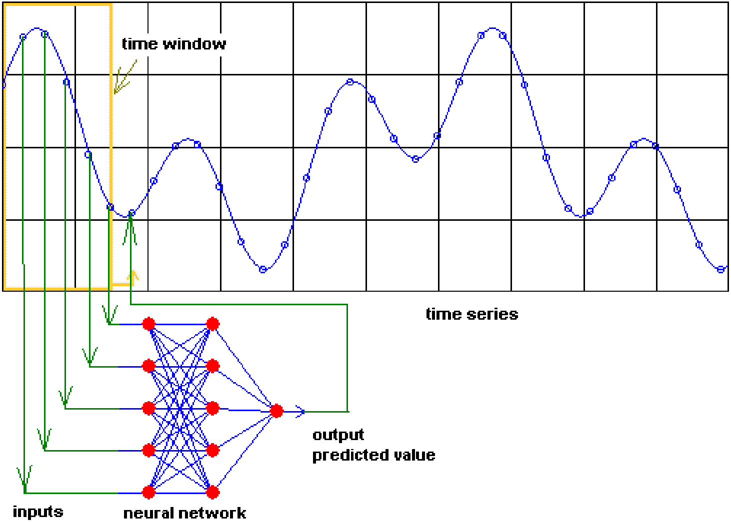
Comparison: digit recognition

	3 NN	300 Hidden NN	LeNet	Boosted LeNet	SVM	Virtual SVM	Shape match
Error rate	2.4	1.6	0.9	0.7	1.1	0.56	0.63
Run time	1000	10	30	50	2000	200	
Memory req	12	.49	0.012	0.21	11		
Training time	0	7	14	30	10		
% rejected to reach 0.5%	8.1	3.2	1.8	0.5	1.8		

- 3-nearest neighbor (memory)
- 300 hidden, fully connected, 123,00 weights
- LeNet (below) a convolution net
- 3 copies of LeNet
- SVM, Virtual SVM, Shape match

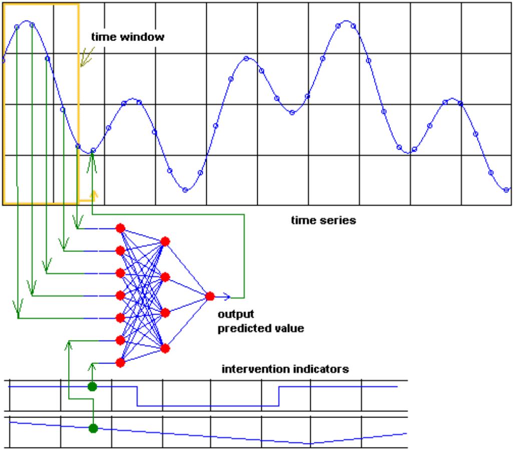


Prediction!



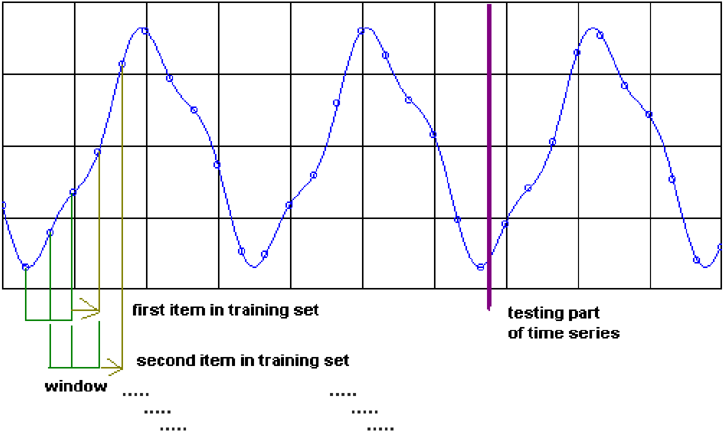
- Neural networks can predict complex time-series, e.g., prices, economies, etc

Prediction!



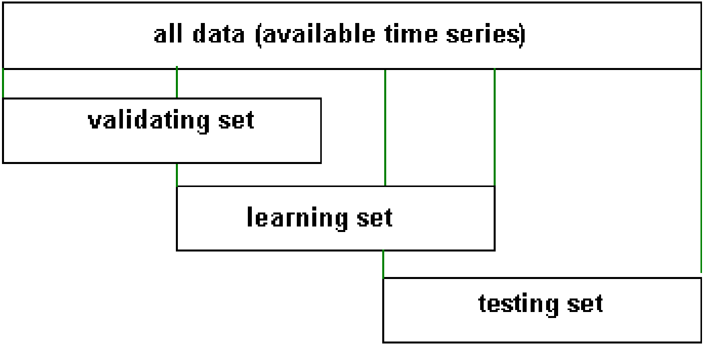
- Input can be given by experts via intervention indicators

Prediction!



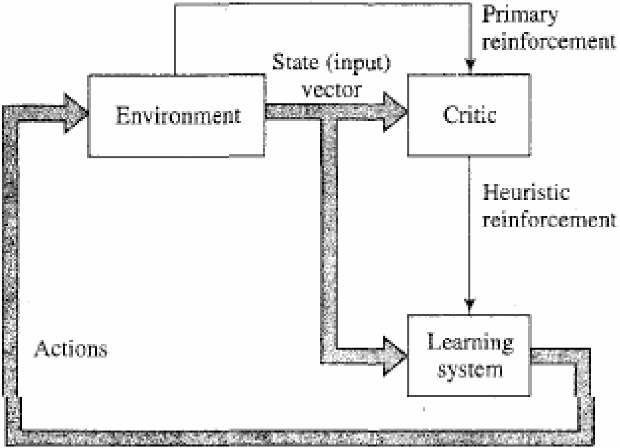
- Training via a shifting window

Prediction!



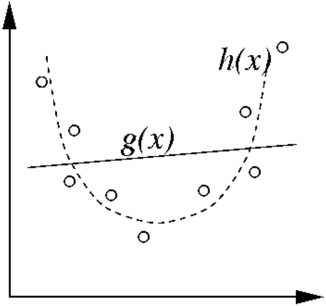
- Like other methods, training, validation, and testing sets help

Reinforcement learning



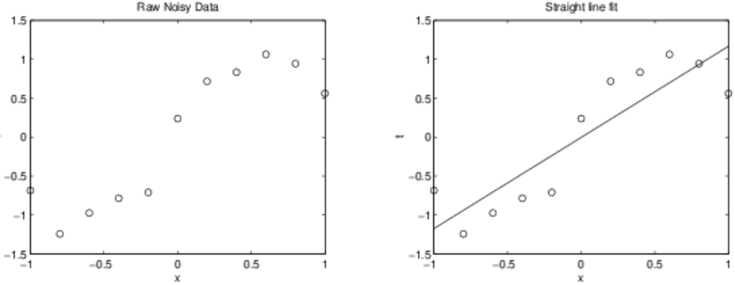
- Temporal credit assignment problem
- More to come with spiking networks Thursday

Overfitting



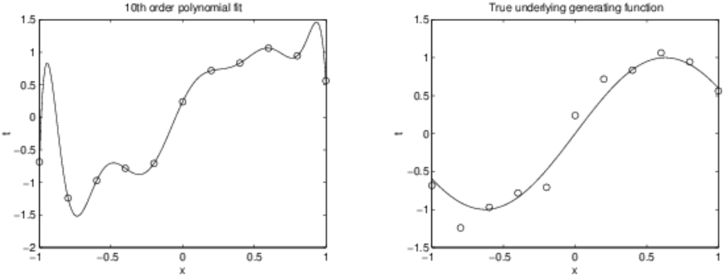
- Over-fitting impedes generalization

Regularization



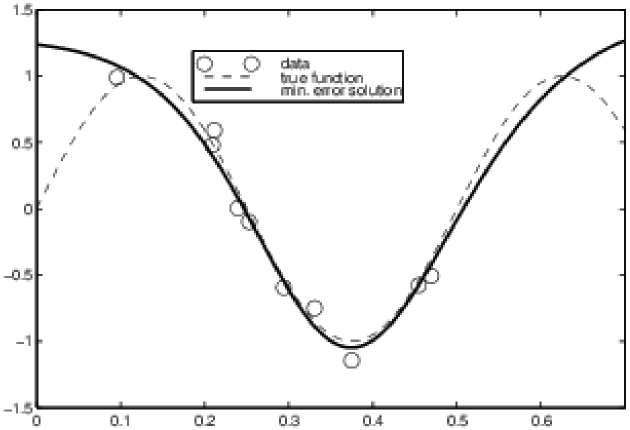
- Straight line might be an underfit to these data points

Regularization



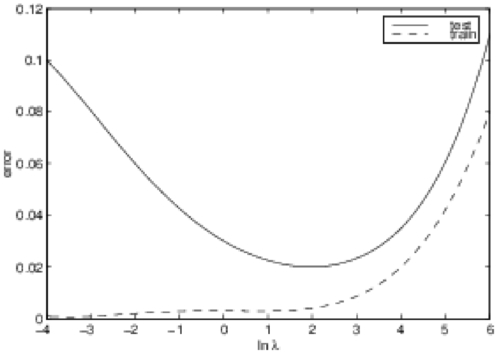
- Left, 10th order might be an overfit.
- Right, the true function from which the data were sampled

Regularization



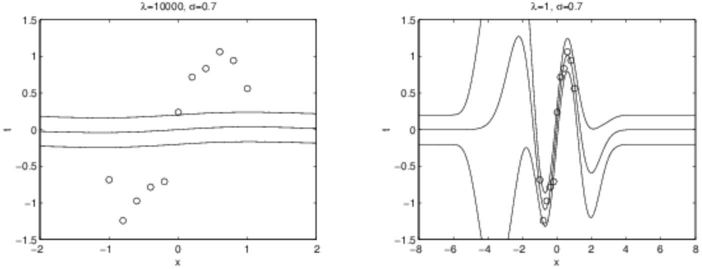
- λ defined as a constant to penalize higher order during the error calculation (for neurons)

Regularization: too little or too much



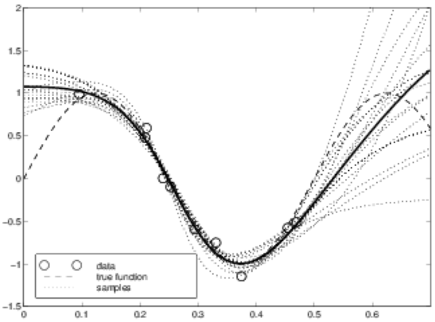
- dotted = train, solid = test
- y=error, x= λ, such that either too low or high order is worse, with a happy medium in the middle.

Regularization: Bayesian



- Pre-specify your hypothesis about λ
- Left, λ 1000
- Right, λ 1

Regularization: Bayesian



- $p(w|\lambda, H) \propto exp[-\frac{\lambda}{2}w^2]$
- $p(\mathbf{w}|D, \lambda, H) = \frac{p(D|\mathbf{w},\gamma,H)p(\mathbf{w}|\lambda,H)}{p(D|\lambda,H)}$ such that D are data
- $p(\mathbf{w}|D, \lambda, H) = p(D|\mathbf{w}) \propto \prod_u exp[-\frac{1}{2}(y^u - f(x^u - \mathbf{w}))^2]$