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

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
Pre- and Post- synaptic

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
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
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: \( \approx 1 \text{msec} \)
- reset time: \( \approx 3 \text{msec} \)
- synapses: \( \approx 1 \text{msec} \)
- pulse transport: \( \approx 5 \text{m/sec} \)

**typical sizes**
- cell body: \( \approx 50 \mu\text{m} \)
- axon diameter: \( \approx 1 \mu\text{m} \)
- synapse size: \( \approx 1 \mu\text{m} \)
- synaptic cleft: \( \approx 0.05 \mu\text{m} \)
# Brains vs. Computers

<table>
<thead>
<tr>
<th></th>
<th>Conventional Computers</th>
<th>Biological Neural Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors</td>
<td>operation speed $\sim 10^8$ Hz</td>
<td>operation speed $\sim 10^2$ Hz</td>
</tr>
<tr>
<td></td>
<td>signal/noise $\sim \infty$</td>
<td>signal/noise $\sim 1$</td>
</tr>
<tr>
<td></td>
<td>signal velocity $\sim 10^8$ m/sec</td>
<td>signal velocity $\sim 1$ m/sec</td>
</tr>
<tr>
<td>Connections</td>
<td>$\sim 10$</td>
<td>$\sim 10^4$</td>
</tr>
<tr>
<td>Sequential Operation</td>
<td></td>
<td>Parallel Operation</td>
</tr>
<tr>
<td>Program &amp; Data</td>
<td></td>
<td>Connections, neuron thresholds</td>
</tr>
<tr>
<td>External Programming</td>
<td></td>
<td>Self-programming &amp; adaptation</td>
</tr>
<tr>
<td>Hardware Failure: Fatal</td>
<td></td>
<td>Robust against hardware failure</td>
</tr>
<tr>
<td>No Unforeseen Data</td>
<td></td>
<td>Messy, unforeseen data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Process Elements</th>
<th>Element Size</th>
<th>Speed</th>
<th>Computation</th>
<th>Robust</th>
<th>Learns</th>
<th>Intelligent, Conscious</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brain</td>
<td>$10^{14}$ synapses</td>
<td>10e-6m</td>
<td>100Hz</td>
<td>parallel, distr</td>
<td>yes</td>
<td>yes</td>
<td>usually</td>
</tr>
<tr>
<td>Computer</td>
<td>$10^8$ transistors</td>
<td>10e-6m</td>
<td>$10^9$ Hz</td>
<td>serial, central</td>
<td>no</td>
<td>a little</td>
<td>Debateably yes</td>
</tr>
</tbody>
</table>

**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
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, probabilistic, 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 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
Neural networks can be universal general purpose computers, and in some app-specific hardware instances do better than Turing machines.
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
Often a bias $\theta$ can be applied/learned
Basic neuron model

- Fixed input $x_0 = +1$
- Synaptic weights (including bias)
- $v_k = \sum_{j=0}^{m} w_{kj} x_j$
- $y_k = \varphi(v_k)$
Activation functions: many types

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

\[ x_0 = +1 \]

\[ x_1 \]

\[ w_{0k} = b_k \]

\[ x_2 \]

\[ w_{1k} \]

\[ w_{2k} \]

\[ \vdots \]

\[ w_{mk} \]

\[ v_k \]

\[ \varphi(\cdot) \]

\[ y_k \]
Architectural graphs and recurrence
Single layer network
Multi-layer feed forward fully connected

Input layer of source nodes
Layer of hidden neurons
Layer of output neurons

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
Recurrent network with no self feedback
Recurrent network with hidden neurons
Knowledge representation? newsgroup example

- Known
- New
- Short
- Home

Input Units
- H2
  - w8
  - w2
  - Reads
  - w0

Hidden Units
- H1
  - w3
  - w1

Output Unit
- w4
- w5
- w9
- w10
- w6
- w11
- w7
- w12
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 \( i \)th synaptic weight \( w_i \) is equal to a learning rate \( \eta \) times the \( i \)th 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

- : \( S_i = 1 \) (neuron \( i \) firing)
- : \( S_i = -1 \) (neuron \( i \) at rest)

input \( i \) > 0 : \( S_i \rightarrow 1 \)
input \( i \) < 0 : \( S_i \rightarrow -1 \)

input \( i \) = \( w_{i1}S_1 + \ldots + w_{iN}S_N \)

Hebbian-like rule:

\( S_i = S_j : \quad w_{ij} \uparrow \)
\( S_i \neq S_j : \quad w_{ij} \downarrow \)
\( w_{ij} \rightarrow w_{ij} + S_iS_j \)

After learning, activate original from noisy version.
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?
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

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x \land y</th>
<th>x + y - \frac{3}{2}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3/2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1/2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1/2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x \lor y</th>
<th>x + y - \frac{1}{2}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1/2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3/2</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>x</th>
<th>\neg x</th>
<th>\neg x + \frac{1}{2}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

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

- Easy for linear single layer network with 2 neurons and a bias, with step activation.
**Problem**: Requires a hidden layer (for non-linearity)
Solution: N-layer network

- **Solution:** Can solve any non-linear function
Separation into 3D via hidden layer allows solving XOR

**Problem:** How to solve for errors in hidden layer??
Neural network for traveling example

Culture

Fly

Hot

Music

Nature

Input Units

H2

H1

Likes

Output Unit

Hidden Units

w_0

w_1

w_2

w_3

w_4

w_5

w_6

w_7

w_8

w_9

w_10

w_11

w_12

w_13

w_14

Neural networks

Applications
Models
Activation function
Stochasticity
Signal flow
Graph structure

Learning

Unsupervised
Hebbian
Associative
Credit
Supervised
Competitive
Error correction
Multi-layer
Error
Backprop
Reinforcement learning
Overfitting

Brains

Neurons
Connections
Signals
Diversity
Levels
Scale
vs. Computers
Computation
Given input example, $e$, what is output prediction?

- $val(e, H1) = f(w_3 + w_4 val(e, Culture) + w_5 val(e, Fly) + w_6 val(e, Hot) + w_7 val(e, Music) + w_8 val(e, Nature))$
- $val(e, H2) = f(w_9 + w_{10} val(e, Culture) + w_{11} val(e, Fly) + w_{12} val(e, Hot) + w_{13} val(e, Music) + w_{14} val(e, Nature))$
- $pval(e, Likes) = f(w_0 + w_1 val(e, H1) + w_2 val(e, H2))$
Error gradients

- **Top left:** original samples; **Top right:** network approximation;
- **Bottom left:** true function which generated samples; **Bottom right:** raw error
Error (vertical) as function of 2 weights \((x_1 \text{ and } x_2)\)
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 is too large
Learning rate is too small
Solution: Error backpropagation

Overview and basic idea:

1. Initialize network weights (often small random values)
2. do
   3. for Each training example ex
   4. prediction = neural-net-output(network, ex) // forward pass
   5. actual = teacher-output(ex)
   6. compute error \((prediction - actual)\) at the output units, as \(\triangle\)
   7. Starting with output layer, repeat until layer 1 (input):
      7. propagate \(\triangle\) values back to previous layer
      9. update network weights between the two layers
8. until all examples classified correctly or another stopping criterion satisfied
9. return the network
Backprop (from ArtInt)

1: **Procedure** BackPropagationLearner($X, Y, E, n_h, \eta$)

**Inputs**

3: $X$: set of input features, $X = \{X_1, ..., X_n\}$
4: $Y$: set of target features, $Y = \{Y_1, ..., Y_k\}$
5: $E$: set of examples from which to learn
6: $n_h$: number of hidden units
7: $\eta$: learning rate

**Output**

9: hidden unit weights $hw[0:n, 1:n_h]$  
10: output unit weights $ow[0:n_h, 1:k]$

**Local**

12: $hw[0:n, 1:n_h]$ weights for hidden units
13: $ow[0:n_h, 1:k]$ weights for output units
14: $hid[0:n_h]$ values for hidden units
15: $hErr[1:n_h]$ errors for hidden units
16: $out[1:k]$ predicted values for output units
17: $oErr[1:k]$ errors for output units

18: initialize $hw$ and $ow$ randomly
19: $hid[0] \leftarrow 1$

20: repeat

for each example $e$ in $E$ do

21: for each $h \in \{1, ..., n_h\}$ do

22: $hid[h] \leftarrow f(\sum_{i=1}^{n} hw[i, h] \times val(e, X_i))$

23: for each $o \in \{1, ..., k\}$ do

24: $out[o] \leftarrow f(\sum_{h=1}^{n_h} hw[i, h] \times hid[h])$

25: $oErr[o] \leftarrow out[o] \times (1-out[o]) \times (val(e, Y_o) - out[o])$

26: for each $h \in \{0, ..., n_h\}$ do

27: $hErr[h] \leftarrow hid[h] \times (1-hid[h]) \times \sum_{o=1}^{k} ow[h, o] \times oErr[o]$

28: for each $i \in \{0, ..., n\}$ do

29: $hw[i, h] \leftarrow hw[i, h] + \eta \times hErr[h] \times val(e, X_i)$

30: for each $o \in \{1, ..., k\}$ do

31: $ow[h, o] \leftarrow ow[h, o] + \eta \times oErr[o] \times hid[h]$

32: until termination

33: return $w_{\omega, ..., \omega_{n_h}}$

This approach assumes $n$ input features, $k$ output features, and $n_h$ 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 $e$. 
function BACK-PROP-LEARNING(examples, network, α) returns a neural network

inputs: examples, each of which has input vector x and output vector y

network with L layers, weights w_{i,j}, activation function g

α: learning rate

local variables: Δ, a vector of errors, indexed by network node

repeat

for each weight w_{i,j} in network do

w_{i,j} ← a small random number

for each example (x,y) in examples do

//Propagate the inputs forward to compute the outputs/

for each node i in the input layer do

    a_{i} ← x_{i}

for l = 2 to L do

    for each node j in layer l do

        in_{j} ← \sum_{i} w_{i,j}a_{i}

        a_{i} ← g(in_{j})

//Propagate deltas backward from output layer to input layer/

for each node j in the output layer do

    Δ[j] ← g'(in_{j}) × (y_{j} - a_{j})

for l = L - 1 to 1 do

    for each node i in layer l do

        Δ[i] ← g'(in_{i}) \sum_{j} w_{i,j} Δ[j]

//Update every weight in the network using deltas/

for each weight in w_{i,j} in network do

    w_{i,j} ← w_{i,j} + α × a_{i} × Δ[j]

until some stopping criterion is satisfied

return network
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:

- $H1 = f(-2.0\text{Culture} - 4.43\text{Fly} + 2.5\text{Hot} + 2.4\text{Music} - 6.1\text{Nature} + 1.63)$
- $H2 = f(-0.7\text{Culture} + 3.0\text{Fly} + 5.8\text{Hot} + 2.0\text{Music} - 1.7\text{Nature} - 5.0)$
- $\text{Likes} = f(-8.5H1 - 8.8H2 + 4.36)$
Comparison: digit recognition

<table>
<thead>
<tr>
<th></th>
<th>3 NN</th>
<th>300 Hidden NN</th>
<th>LeNet</th>
<th>Boosted LeNet</th>
<th>SVM</th>
<th>Virtual SVM</th>
<th>Shape match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error rate</td>
<td>2.4</td>
<td>1.6</td>
<td>0.9</td>
<td>0.7</td>
<td>1.1</td>
<td>0.56</td>
<td>0.63</td>
</tr>
<tr>
<td>Run time</td>
<td>1000</td>
<td>10</td>
<td>30</td>
<td>50</td>
<td>2000</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Memory req</td>
<td>12</td>
<td>0.49</td>
<td>0.012</td>
<td>0.21</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training time</td>
<td>0</td>
<td>7</td>
<td>14</td>
<td>30</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>% rejected to reach 0.5%</td>
<td>8.1</td>
<td>3.2</td>
<td>1.8</td>
<td>0.5</td>
<td>1.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

- Input can be given by experts via intervention indicators
Training via a shifting window
Like other methods, training, validation, and testing sets help
Reinforcement learning

- Temporal credit assignment problem
- More to come with spiking networks Thursday
Overfitting 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
- $\lambda$ defined as a constant to penalize higher order during the error calculation (for neurons)
Regularization: too little or too much

- dotted = train, solid = test
- $y=\text{error}, x=\lambda$, such that either too low or high order is worse, with a happy medium in the middle.
Regularization: Bayesian

- Pre-specify your hypothesis about $\lambda$
- Left, $\lambda$ 1000
- Right, $\lambda$ 1
\[
p(w | \lambda, H) \propto \exp\left[ -\frac{\lambda}{2} w^2 \right]
\]
\[
p(w | D, \lambda, H) = \frac{p(D | w, \gamma, H)p(w | \lambda, H)}{p(D | \lambda, H)} \quad \text{such that } D \text{ are data}
\]
\[
p(w | D, \lambda, H) = p(D | w) \propto \prod_u \exp\left[ -\frac{1}{2} (y^u - f(x^u - w))^2 \right]
\]