Artificial neural networks

Susanne Still
University of Hawaii at Manoa

Part 1

Feed forward artificial neural networks

- Feed forward artificial neural nets are most often used to solve supervised learning problems.
- Recall: Supervised learning methods make choices about:
  - error function
  - hypothesis class H
  - algorithm used for error minimization

Training Error vs. Generalization Error

- The goal of learning is to build a hypothesis that generalizes well to data drawn from the same process, which we have not seen yet. I.e. to make good predictions! (drawing)
- Error can be evaluated only on the given training data set. (Training Error).
- Want to minimize the generalization error, but can not measure it.

How to deal with overfitting / model complexity control?

- Absolutely crucial -> at the heart of machine learning methods.
- Recall: derive bounds on the generalization error which depend only on things we can measure, and use those bounds to minimize best estimate of actual risk.

Cross-Validation

- Empirical way to estimate out-of-sample error
- Versatile (finds use in many different applications)
- Has some problems (we won't discuss today, but keep that in mind!)

Cross-Validation

- Split data into 3 sets:
- Training set (used to train the algorithm, to learn the model h)
- Validation set (used for cross-validation)
- Test set (used only to test the resulting performance)

Cross-Validation

- Decide on a hypothesis class H
- Train algorithm on the Test set (Test error is minimized) -> find best hypothesis h in the class H.
- Evaluate the error that h makes on the validation set
- Do this procedure for all possible splits of the Data set (Training Set + Validation Set).

Cross-Validation

- Can split in half or can do "leave-one-out" cross-validation.
- Compute the average error, averaged over all possible splits.
- Use this to decide between different hypothesis classes.
- Chose the one with lowest cross-validation error

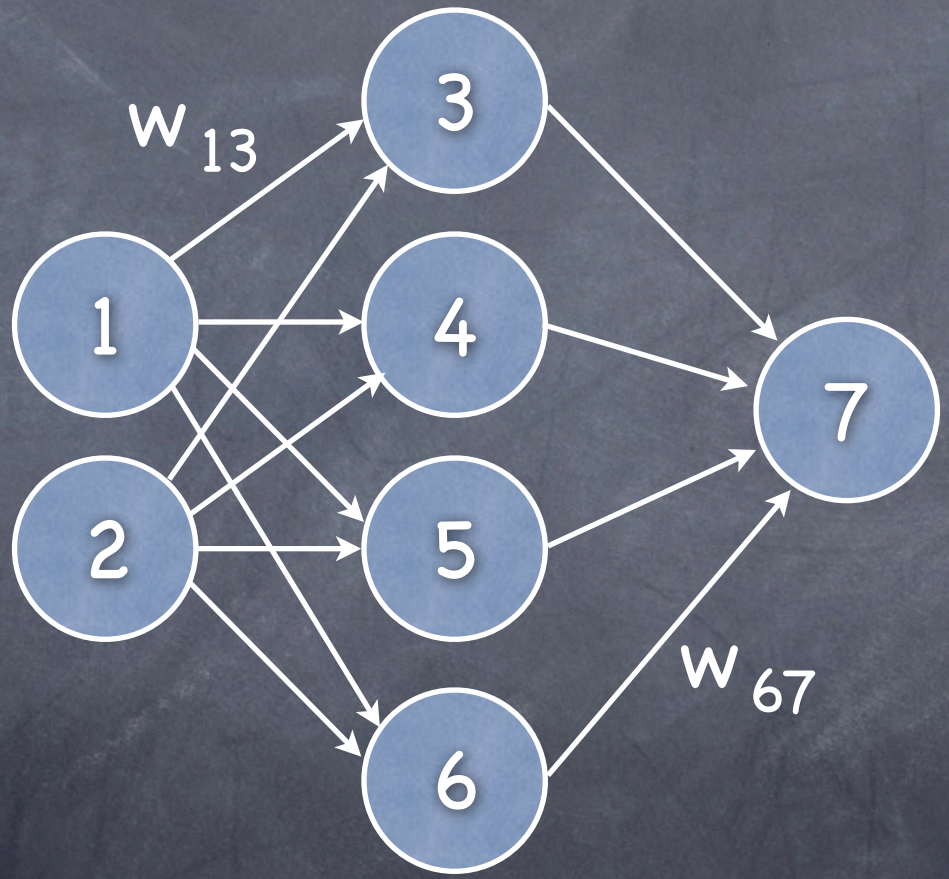
Recall

- Artificial neuron = neuron gets approximated mathematically by a threshold units
- Example: Perceptron
- Single artificial neuron implements linear hypothesis class
- "Linear classifier"

Non-linear data

- Linear classifier not able to separate the data.
- Project data into a higher dimensional feature space in which they are linearly separable (-> Kernel trick...)
- Use a non-linear hypothesis class.

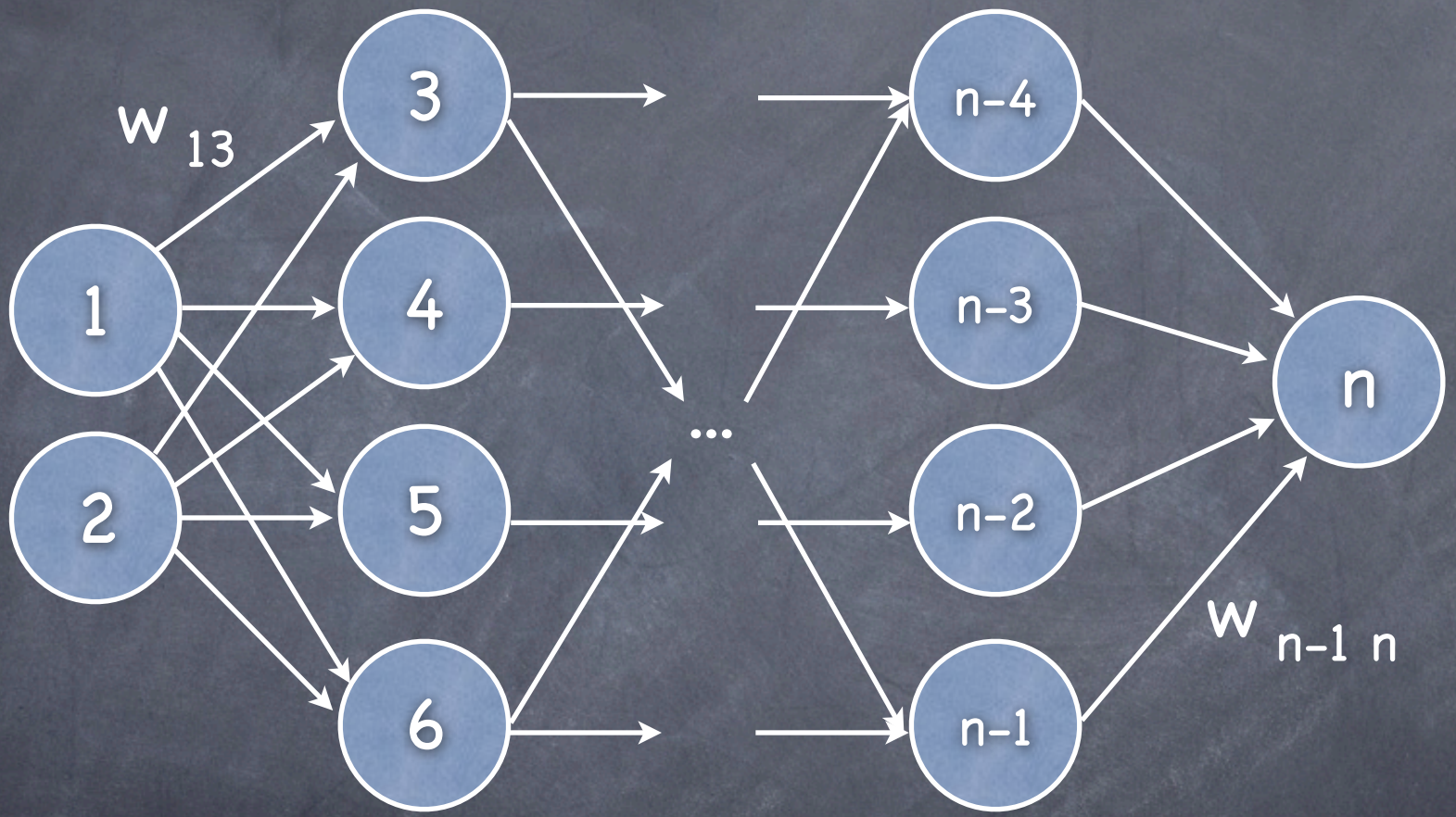
Feed forward artificial neural networks



Feed-forward ANN: Architecture

- Network can have one or more hidden layers (indexed starting with the input layer, ending with the output layer).
- Each hidden layer has some number of neurons called units. Often: sigmoidal units (sigmoidal transfer function), because differentiable.
- Neurons in the hidden layer are called "hidden units".
- Usually one output neuron.

Architecture



Input Layer

k=1

Hidden Layer

k=2

Hidden Layer

k=K-1

Output Layer

k=K

Feed-forward ANN: Connections

- Neurons are connected by weights
- The connections are uni-directional
- Weight on the connection from neuron i to neuron j: w_{ij}
- All connections go from layer k to layer k+1. There are no connections backwards.
- There are no connections between units in one layer.
- Hence the name "feed-forward"

Feed Forward Artificial Neural Networks

- Representational Power
- Designing feed forward neural networks
- Learning in feed forward neural networks
  - Gradient decent
  - Backpropagation algorithm

Feed-forward ANN; Representational Power

- A single artificial neuron can express the boolean functions AND, OR, and NOT, but not XOR. It can separate linearly separable data. This is true for both choices of transfer functions, step function or sigmoid alike.
- A feed-forward artificial neural network with one hidden layer can express any boolean function. This might require a number of hidden units exponential in the number of inputs.

Feed-forward ANN; Representational Power

- A feed-forward artificial neural network with one hidden layer that is large enough can approximate any bounded continuous function to arbitrary precision.
- A feed-forward artificial neural network with two hidden layers can approximate any function to arbitrary accuracy
- Reference: Cybenko 1988/9; Hornik et al. 1989

Some applications

- Speech recognition
- Speech synthesis (Nettalk)
- Image classification
- Digit recognition (LeNet)
- Computer Vision
- Finance

When to use FF ANNs?

- Target function unknown
- High-dimensional input; discrete or real valued
- Noisy data
- Training time is not too important
- Interpretability of result is not important
- Calculation of output from input has to be fast

Disadvantages

- Poor generalization when the number of examples is small (compared to the dimensionality of the input space)
- Overfitting has to be addressed; lots of parameters -> sloppy resulting models
- The gradient decent training method can get trapped in an unfavorable local minimum
- modern FFANNs have tricks to address these!

Designing FF ANNs

- choosing the number of units:
  too few -> concept can not be learned.
  too many -> overfitting
  for n binary inputs, log(n) units are a good heuristic.

- choosing the number of layers:
  always start with one layer; never go beyond two, unless network architecture requires it (e.g. convolutional nets). BUT (careful): functions that can be compactly represented by a depth k architecture might require an exponential number of computational elements to be represented by a depth k – 1 architecture !!!

See: http://www.iro.umontreal.ca/~lisa/publications/index.php?page=publication&kind=single&ID=209

Network design

- destructive methods: start with a large network, remove (prune) connections: put the weight to 0 and look at the effect on the error:
- train network -> solution corresponds to local minimum
- approximate the impact of every unit on the performance of the network (calculate Hessian)
- take away the weakest unit

Network design

- constructive methods:

1. dynamic node creation (Ash): (one hidden layer)
   - start with one hidden unit; train
   - if error is large, add another unit
   - repeat.
2. meiosis networks (Hanson)

Meiosis networks

- start with one hidden unit; train
- compute variance of each weight during training
- if a unit has at least one high variance weight, replace unit by two new units, and perturb the weights
- -> create functionally different units

ANNs for time series data

- input is a function of time: x(t)
- possible tasks include:
  - predicting a class label (e.g. speech recognition)
  - predicting future data, time-series prediction (e.g. in finance)

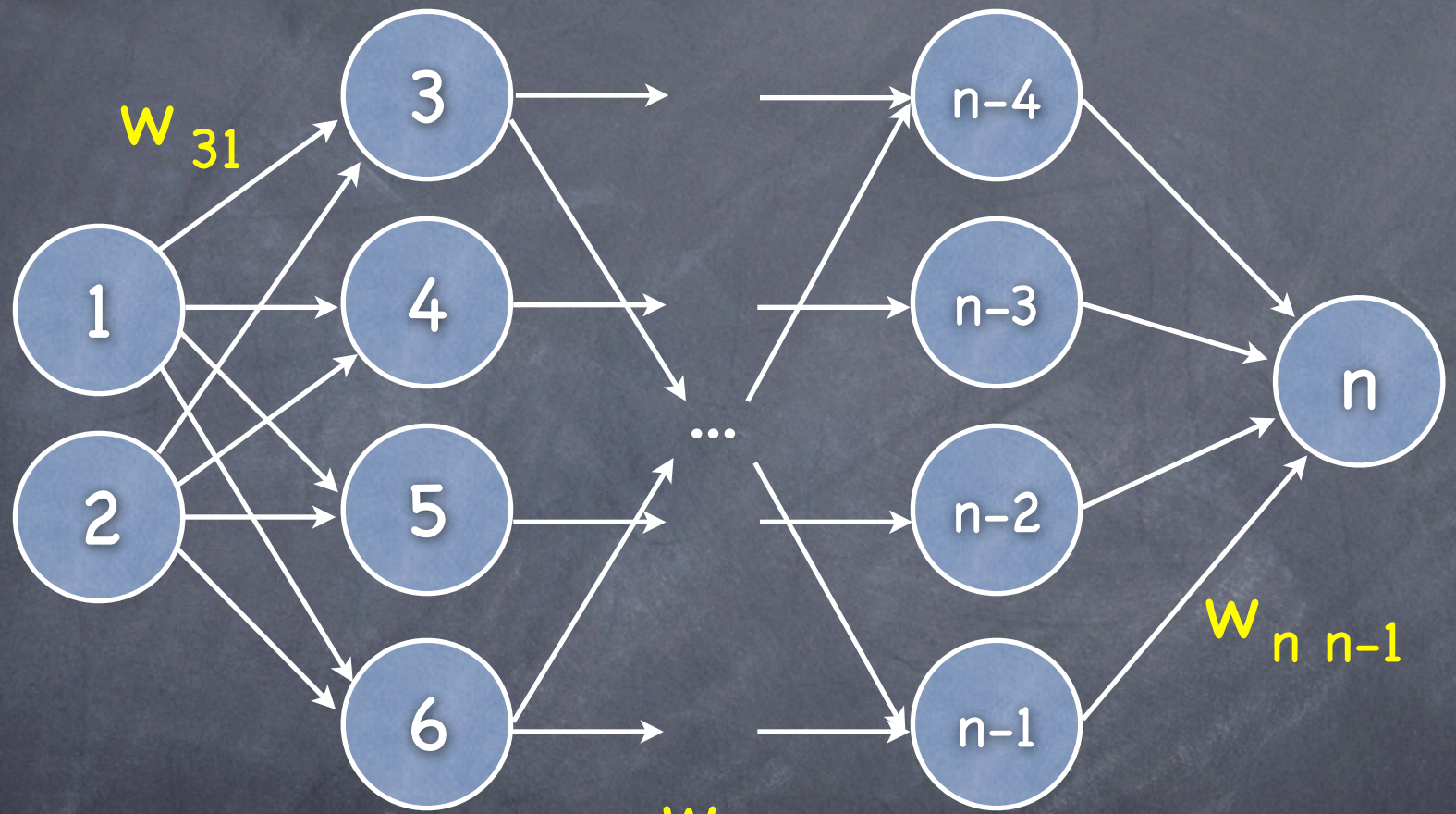
Time-delay NNs

(Waibel)

- Set a time frame T
- take all inputs that occur between t and t+T and feed them into the network
- train using backprop
- shift the window and repeat
- on-line algorithm (in the extreme, we can take one data point at a time)

- This is great. But how do we deal with complexity control? (later...)
- And how do we train a feed-forward ANN?
  - Gradient descent
  - Backpropagation algorithm

How do we train a FFANN?



Learning

- Problem: Complex hypothesis, thus no direct calculation of optimal weights possible.
- Instead: adjust weights incrementally (learning). How to adjust the weights?
- Idea: Gradient decent.
- Energy function
- Calculate gradient w.r.t. the weights
- Adjust weights proportional to gradient

Learning

- Use sigmoidal units: Output of units is given by (i = 1, ..., n):

Input to neuron i

$$o_i = \sigma(\vec{w}_i \vec{x}_i)$$

- Recall: $\sigma(u) = \dfrac{1}{1+e^{-u}}$
- Derivative:

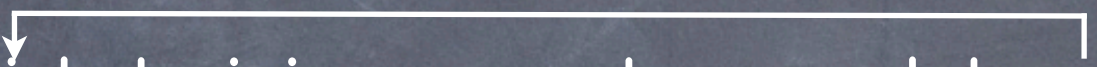
$$\frac{d\sigma(u)}{du} = \sigma(u)\big(1-\sigma(u)\big)$$

Backpropagation algorithm

- gradient decent over all weights in network
- Forward pass: compute outputs of all units starting from the input layer, ending with the output layer
- Backward pass: compute the updates Δ_i starting from the output layer
- update the weights according to

$$w_{ij} \leftarrow w_{ij} + \alpha_{ij} \Delta_i x_{ij}$$

Backpropagation algorithm



- pick training example -> update weights (stochastic gradient decent)
- loop through all samples -> update weights (batch)
- one pass through the data set is called an epoch.

Backpropagation algorithm

- incremental (stochastic) gradient decent
- initialize weights
- repeat until convergence:
- pic training example, use it as input to the network and compute the outputs
- for each unit i, compute Δ_i
- update each network weight:

$$w_{ij} \leftarrow w_{ij} + \alpha_{ij} \Delta_i x_{ij}$$

Backpropagation algorithm

- guaranteed to converge to local minimum if learning rates are small enough
- local minimum can be much worse than global minimum
- there can be many local minima

Learning rate

- The backpropagation algorithm is sensitive to the size of the learning rate.
- too small -> very slow
- too large -> divergence
- learning rate has effect on the ability to escape local optima
- each unit has its own optimal rate (one for all is in general not optimal)

Learning rate

- Adjusting learning rates: Heuristic method Delta-bar-delta
- Idea:
- if the gradient direction does not change -> increase learning rate
- if gradient switches sign -> decrease

Break

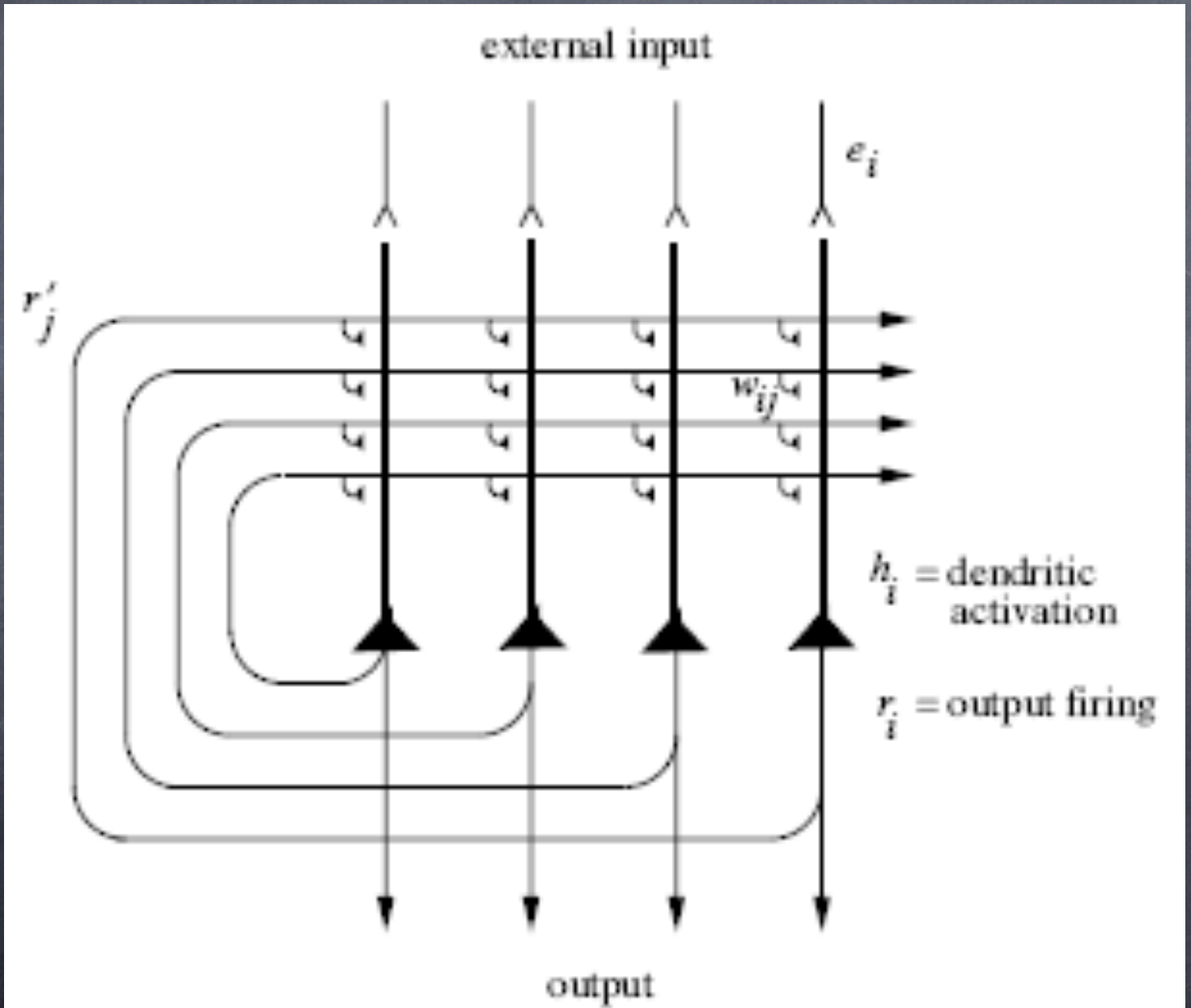
Recurrent Neural Networks

General Architecture
Hopfield NN (Intro)
Hebbian Learning

Architecture

- connections within layers
- there is no structure in the layers
- network can be fully connected (all neurons to all)
- some connections could be missing

Architecture



http://www.oftnai.org/images/memory1.gif

Hopfield Neural Net

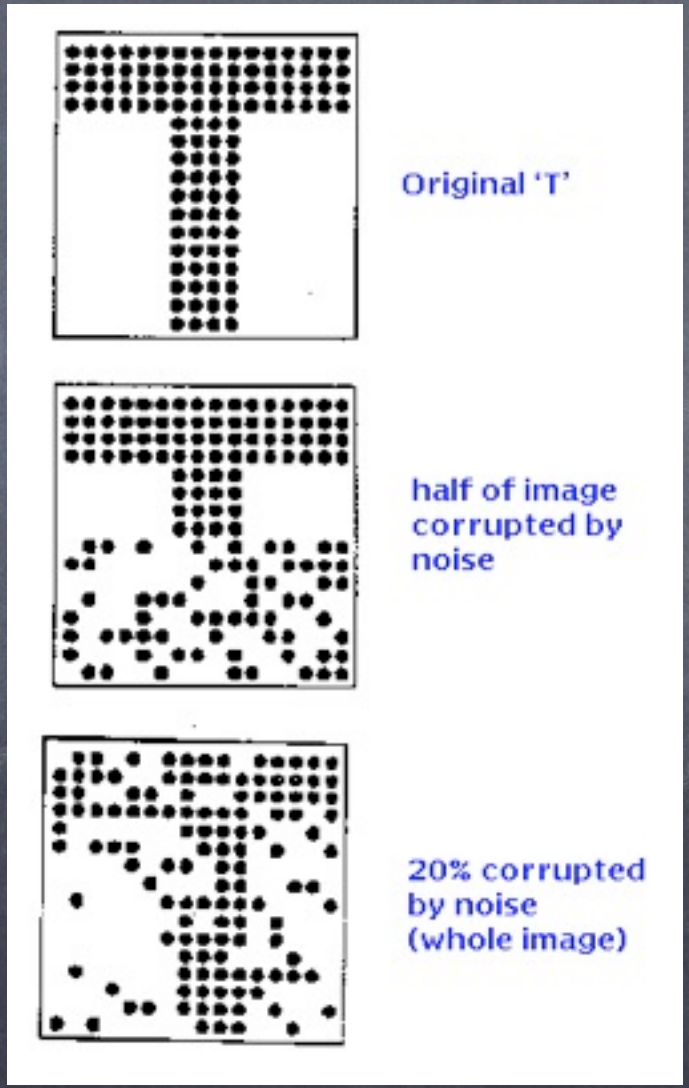
J. Hopfield: Neural networks and physical systems with emergent collective computational abilities. PNAS 79, 2554, 1982.

- 1982: John Hopfield, Physicist, working on spin-systems, proposed fully connected ANN to solve an associative memory task
- models of physical systems can be used to solve computational problems
- Refer to as "HNN".

Problem

- Store K patterns, $X^\mu = \{x_i^\mu\}$
  such that when presented with a new pattern, the system associates this new pattern to the most similar stored pattern.
- i = 1, ...,N: number of sites/pixels
  = number of units/neurons
- μ = 1, ..., K: number of different patterns
- $x_i^\mu \in \{-1; 1\}$

Associative memory



http://www.shef.ac.uk/psychology/gurney/notes/l5/Ts.gif

Applications

- Information retrieval
- Image recognition
- Image reconstruction
- Content-addressable memory (CAM)

Hardware application: CAM content-addressable memory

- Random access memory (RAM):
  - Input: memory address
  - Returns: data word stored at that address.
- CAM:
  - Input: data word
  - Returns: List of storage addresses where the data word was found
- CAM much faster for search operations!

CAM applications

- Computer networking devices: network switch
- Network routers
- CPU cache controllers
- Database engines
- Data compression hardware

How does it work?

1. The Ingredients:

- Hopfield used the transfer function:
  g = sgn (x).
  Let us set the bias to zero.
- Units ("neurons"): $s_i = sgn(\sum_j w_{ij} s_j)$
- Architecture: fully connected

2. Learning

- Hebbian learning
- Synchronous update (requires clock)
- Asynchronous update:
  1. random update
  2. each unit chooses to update with some probability

Hypothesis by Hebb ('49)

- changes in synaptic strength proportional to the correlation between the activity of pre- and post-synaptic neuron.
- "Neurons that fire together, wire together."
- Original quote: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

Learning in the brain models thereof

- LTP (long term potentiation):

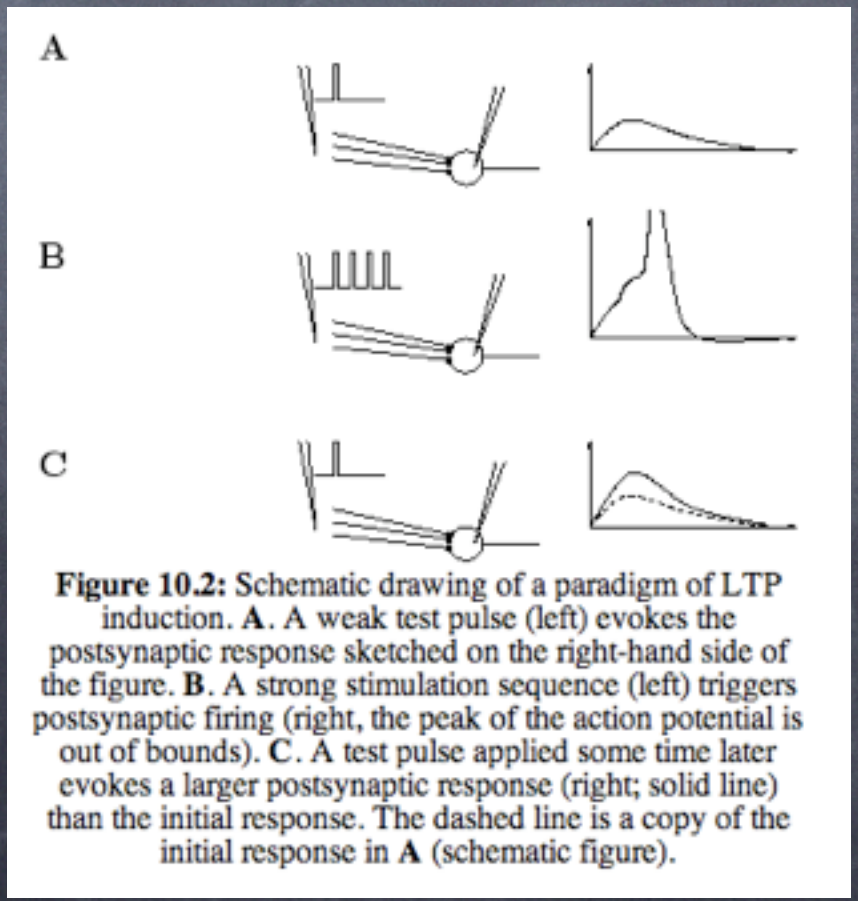


Figure 10.2: Schematic drawing of a paradigm of LTP induction. **A**. A weak test pulse (left) evokes the postsynaptic response sketched on the right-hand side of the figure. **B**. A strong stimulation sequence (left) triggers postsynaptic firing (right, the peak of the action potential is out of bounds). **C**. A test pulse applied some time later evokes a larger postsynaptic response (right; solid line) than the initial response. The dashed line is a copy of the initial response in **A** (schematic figure).

http://icwww.epfl.ch/~gerstner/SPNM/node71.html#Fig-ch2-7

Hopfield Neural Net

Storing one pattern
Storing many patterns
Dynamics
Stability
Storage capacity

One pattern

- a pattern is stable when:

$$sgn(\sum_j w_{ij} x_j) = x_i \qquad \forall i$$

- "Hebbian learning": choose $w_{ij} \propto x_i x_j$

$$\Rightarrow sgn(x_i \sum_{j=1}^{N} \underbrace{x_j x_j}_{=1}) \quad = \quad sgn(N x_i) = x_i$$

- choose $w_{ij} = \frac{1}{N} x_i x_j$

Energy function

- Hamiltonian (Lyaponov-/ cost-/ objective-function)

$$H = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j$$

- function of the configuration: $\{s_i\}$ some configurations have higher/lower energy
- want the energy to be minimal when the overlap is largest:

$$H = -\frac{1}{2N}(\sum_{i=1}^{N} s_i x_i)^2$$

Many patterns: sum over them

$$H = -\frac{1}{2N}\sum_{\mu=1}^{K}(\sum_{i=1}^{N} s_i x_i^\mu)^2$$

make all patterns local minima!

$$= -\frac{1}{2N}\sum_{\mu=1}^{K}(\sum_{i=1}^{N} s_i x_i^\mu)(\sum_{j=1}^{N} s_j x_j^\mu)$$

$$= -\frac{1}{2}\sum_{ij}(\frac{1}{N}\sum_{\mu=1}^{K} x_i^\mu x_j^\mu) s_i s_j$$

$$\Rightarrow w_{ij} = \frac{1}{N}\sum_{\mu=1}^{K} x_i^\mu x_j^\mu$$

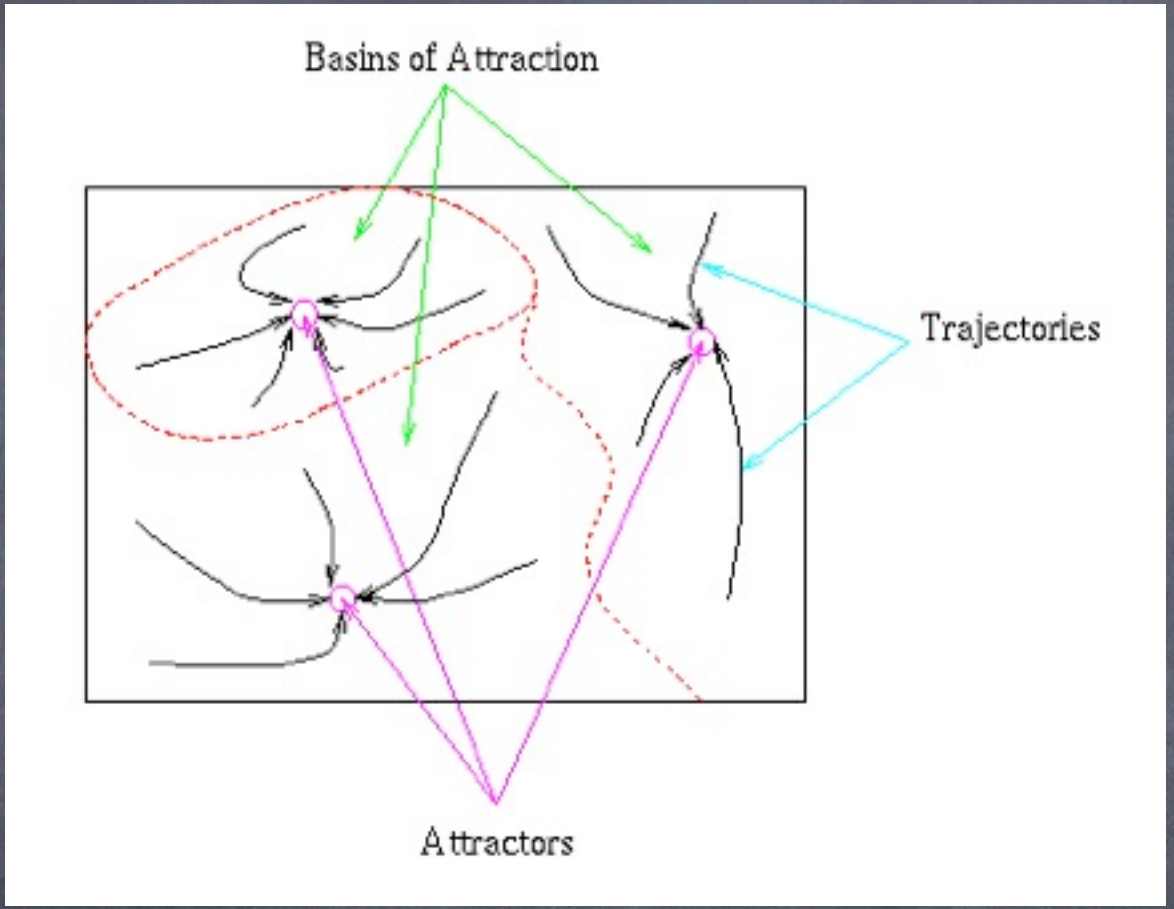
HEBB RULE (generalized)

Energy $H = -\frac{1}{2} \sum_{ij} w_{ij} s_i s_j$

- symmetric connections
- dynamics or the network: the energy can only go down!

  => network converges to a fixed point attractor
- true for sequential updating, random updating, almost holds for parallel updating

Dynamics



http://www.itee.uq.edu.au/~cogs2010/cmc/chapters/Hopfield/Attractors.gif

- dynamics <=> motion of a particle on the energy surface moving due to gravity
- attractors = memorized patterns = local minima

Stability of pattern x_i^ν

Condition: $sgn(h_i^\nu) = x_i^\nu$

with: $$h_i^\nu = \sum_j w_{ij} x_j^\nu = \frac{1}{N} \sum_j \sum_\mu x_i^\mu x_j^\mu x_j^\nu$$

$$= x_i^\nu + \frac{1}{N} \sum_j \sum_{\mu \neq \nu} x_i^\mu x_j^\mu x_j^\nu$$

Crosstalk term

$$= x_i^\nu (1 - C_i^\nu)$$

If $C_i^\nu < 1 \Rightarrow$ does no harm (same sign as x_i^ν)

Storage capacity

$$C_i^\nu = -x_i^\nu \frac{1}{N} \sum_j \sum_{\mu \neq \nu} x_i^\mu x_j^\mu x_j^\nu$$

Need to worry about $C_i^\nu > 1$

Consider:

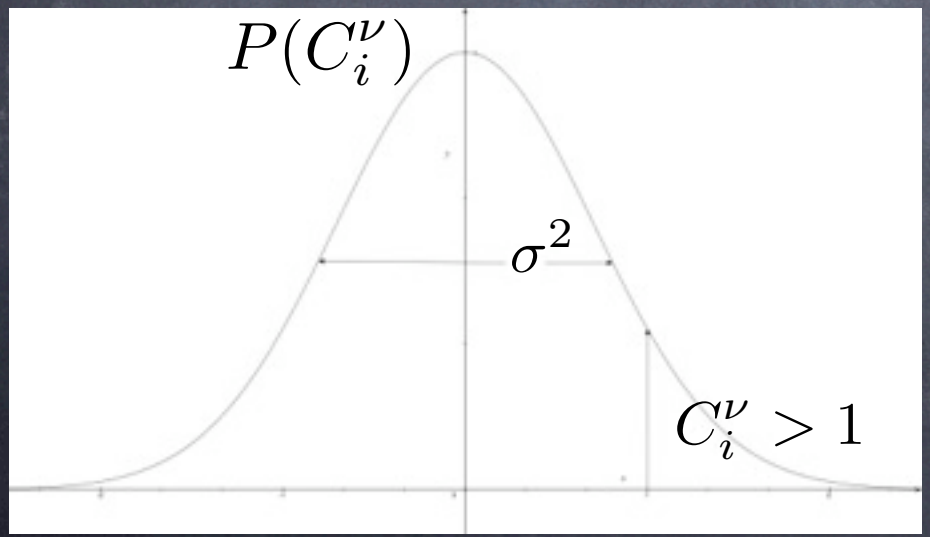
- orthogonal patterns
- random patterns

- consider orthogonal patterns: $\sum_j x_j^\mu x_j^\nu = 0$
- no crosstalk
- capacity: $K_{\max} = N$
- because at most N mutually orthogonal bit strings of length N can be stored.

Random patterns

- consider random patterns with equal probability for all bits (independent). $p(x_i^\mu = 1) = p(x_i^\mu = -1)$ Assume large N and K. Then C_i^ν is distributed according to a normal distribution.
- Probability that a given bit is unstable:

$$P_{\text{error}} = P(C_i^\nu > 1) = \frac{1}{\sqrt{2\pi\sigma}} \int_1^\infty dx\, e^{-\frac{x^2}{2\sigma^2}}$$



$$= \frac{1}{2}(1 - erf(\frac{1}{\sqrt{2\pi\sigma}}))$$

$$\sigma = \sqrt{\frac{K}{N}}$$

- can choose a criterion on the error (e.g. make the error probability smaller than 1%) and from that get a condition on the maximum number of patterns that can be stored.

- "perfect" memory on all patterns: to get all N bits of K patterns right with 99% accuracy, we need

$$P_{\text{error}} < 0.01/NK$$

- $N \to \infty \Rightarrow K/N \to 0$

- use asymptotic expansion:

$$1 - erf(x) \overset{x \to \infty}{\to} \frac{e^{-x^2}}{\sqrt{\pi} x}$$

$$\log(P_{\text{error}}) \simeq \log 2 - \frac{N}{2K} - \frac{1}{2}\log\pi - \frac{1}{2}\log(\frac{N}{2K})$$

$$\overset{!}{<} \log(\frac{0.01}{NK})$$

- take leading order terms for large N:

$$\frac{N}{2K} > \log(NM) \simeq \log(N^2) = 2\log(N)$$

- Capacity for large N: $K_{\max} = \frac{N}{4}\log(N)$
- Similarities between patterns reduce capacity
- Sparse patterns give better capacity