CS 388: Natural Language Processing: Neural Networks

Raymond J. Mooney

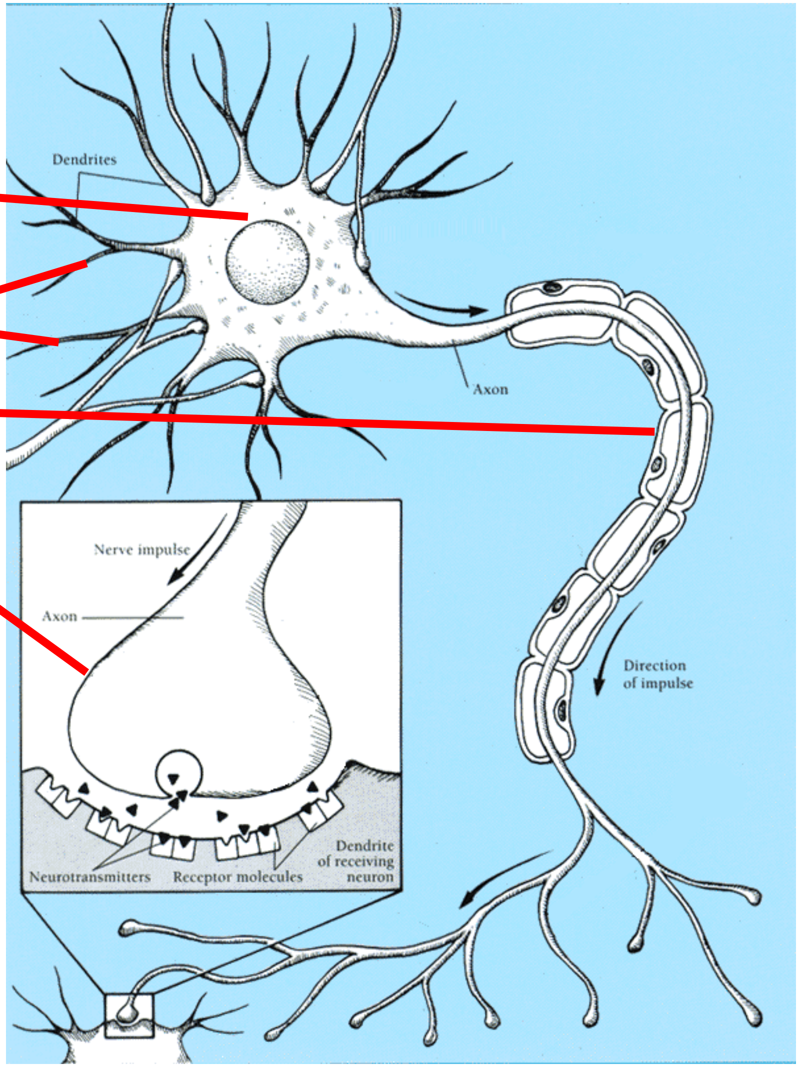
University of Texas at Austin

Neural Network Learning

- Model learning approach berdasarkan sistem syaraf pada biologi.
- Perceptron: Algoritma dasar untuk neural network sederhana (single layer), dikembangkan pada kisaran tahun 1950.
- Backpropagation: Algoritma yang lebih kompleks untuk belajar melalui multi-layer neural network, dikembangkan pada tahun 1980an.

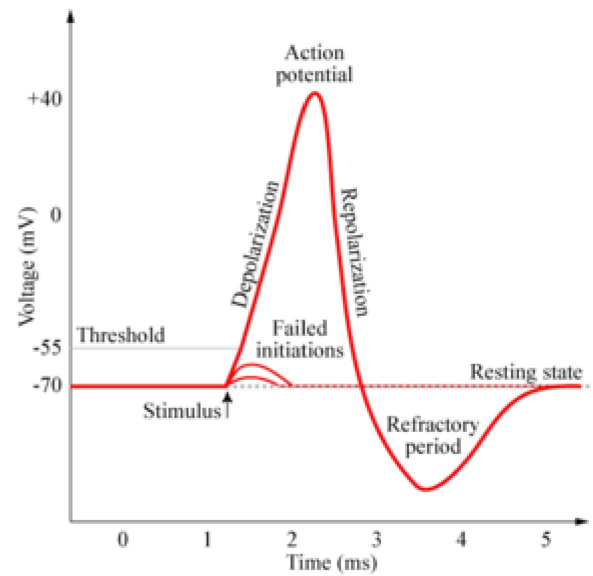
Real Neurons

- Cell structures
  - Cell body
  - Dendrites
  - Axon
  - Synaptic terminals



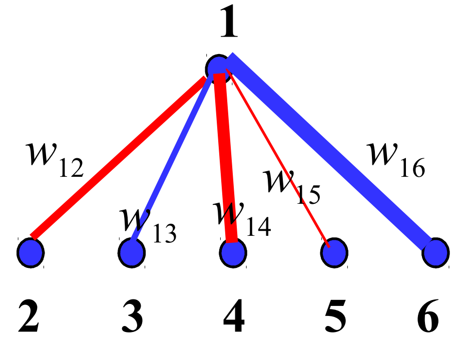
Neural Communication

- Electrical potential across cell membrane exhibits spikes called action potentials.
- Spike originates in cell body, travels down axon, and causes synaptic terminals to release neurotransmitters.
- Chemical diffuses across synapse to dendrites of other neurons.
- Neurotransmitters can be excititory or inhibitory.
- If net input of neurotransmitters to a neuron from other neurons is excititory and exceeds some threshold, it fires an action potential.



Simple Artificial Neuron Model (Linear Threshold Unit)

- Model network as a graph with cells as nodes and synaptic connections as weighted edges from node *i* to node *j*, w_{ji}



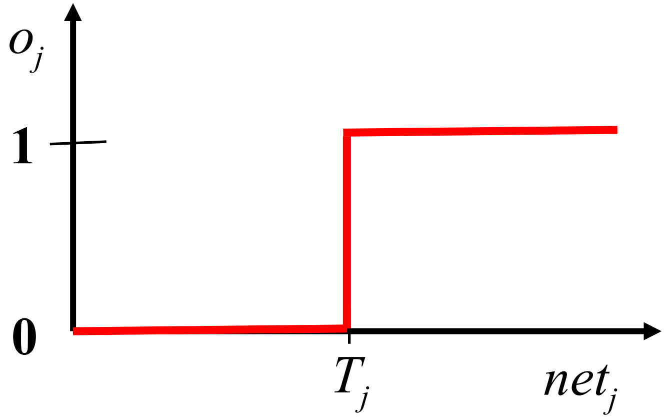
- Model net input to cell as

$$net_j = \sum_i w_{ji} o_i$$

- Cell output is:

$$o_j = \begin{matrix} 0 \text{ if } net_j < T_j \\ 1 \text{ if } net_i \geq T_j \end{matrix}$$

(T_j is threshold for unit *j*)



Perceptron Training

- Assume supervised training examples giving the desired output for a unit given a set of known input activations.
- Learn synaptic weights so that unit produces the correct output for each example.
- Perceptron uses iterative update algorithm to learn a correct set of weights.

Perceptron Learning Rule

- Update weights by:

$$w_{ji} = w_{ji} + \eta(t_j - o_j)o_i$$

where η is the "learning rate"

t_j is the teacher specified output for unit j.

- Equivalent to rules:
  - If output is correct do nothing.
  - If output is high, lower weights on active inputs
  - If output is low, increase weights on active inputs
- Also adjust threshold to compensate:

$$T_j = T_j - \eta(t_j - o_j)$$

Perceptron Learning Algorithm

- Iteratively update weights until convergence.

Initialize weights to random values
Until outputs of all training examples are correct
    For each training pair, *E*, do:
        Compute current output o_j for *E* given its inputs
        Compare current output to target value, t_j, for *E*
        Update synaptic weights and threshold using learning rule

- Each execution of the outer loop is typically called an *epoch*.

Threshold to "Bias"

- Threshold can be converted to an additional "bias" weight on an additional constant 1 input (o_0=1)

$$\sum_i w_{ji} o_i > T_j$$

$$\sum_i w_{ji} o_i - T_j > 0$$

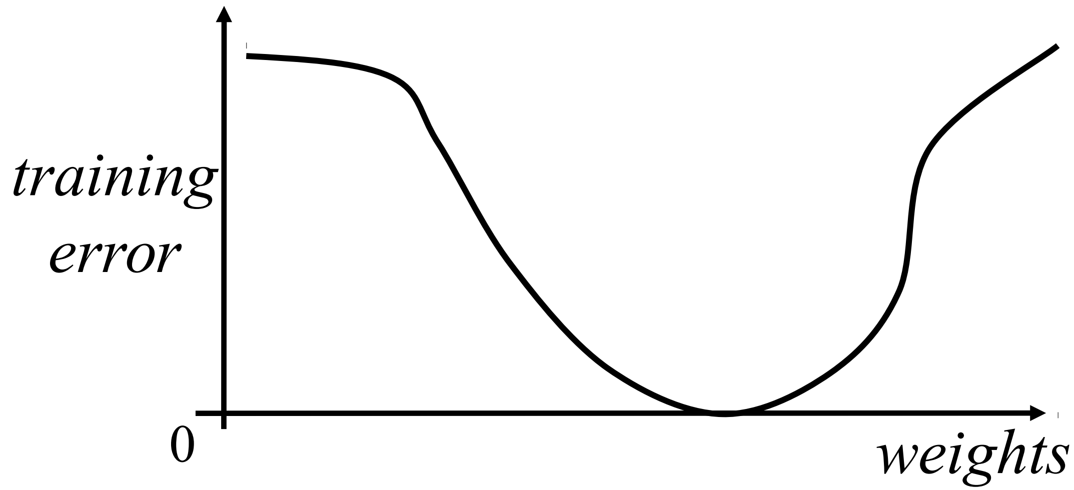
$$\sum_i w_{ji} o_i + b_j > 0$$

$$\sum_i w_{ji} o_i > 0$$

Where sum now includes i=0 and $w_{j0}=b_j$

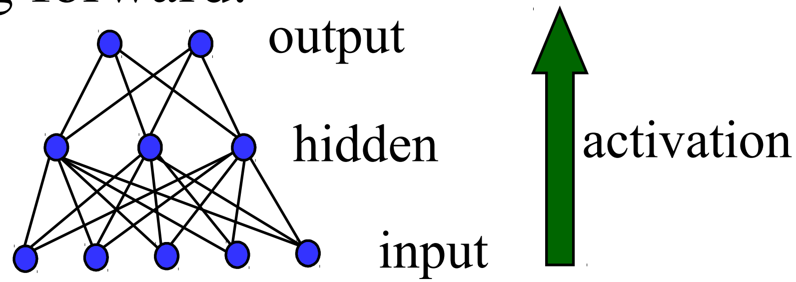
Perceptron as Hill Climbing

- The hypothesis space being search is a set of weights and a threshold.
- Objective is to minimize classification error on the training set.
- Perceptron effectively does hill-climbing (gradient descent) in this space, changing the weights a small amount at each point to decrease training set error.
- For a single model neuron, the space is well behaved with a single minima.



Multi-Layer Feed-Forward Networks

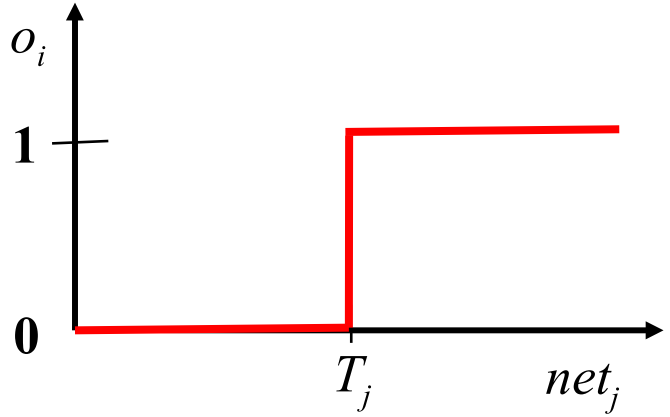
- Multi-layer networks can represent arbitrary functions, but an effective learning algorithm for such networks was thought to be difficult.
- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.



- The weights determine the function computed. Given an arbitrary number of hidden units, any boolean function can be computed with a single hidden layer.

Hill-Climbing in Multi-Layer Nets

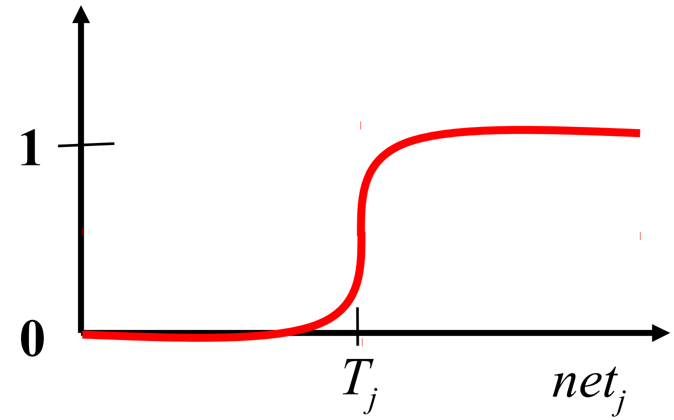
- Since “greed is good” perhaps hill-climbing can be used to learn multi-layer networks in practice although its theoretical limits are clear.
- However, to do gradient descent, we need the output of a unit to be a differentiable function of its input and weights.
- Standard linear threshold function is not differentiable at the threshold.



Differentiable Output Function

- Need non-linear output function to move beyond linear functions.
  - A multi-layer linear network is still linear.
- Standard solution is to use the non-linear, differentiable sigmoidal "logistic" function:

$$o_j = \frac{1}{1 + e^{-(net_j - T_j)}}$$



Can also use tanh or Gaussian output function

Gradient Descent

- Define objective to minimize error:

$$E(W) = \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2$$

where D is the set of training examples, K is the set of output units, t_{kd} and o_{kd} are, respectively, the teacher and current output for unit k for example d.

- The derivative of a sigmoid unit with respect to net input is:

$$\frac{\partial o_j}{\partial net_j} = o_j(1 - o_j)$$

- Learning rule to change weights to minimize error is:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

Backpropagation Learning Rule

- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j (1 - o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j (1 - o_j) \sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

where η is a constant called the learning rate

t_j is the correct teacher output for unit j

δ_j is the error measure for unit j

Error Backpropagation

- First calculate error of output units and use this to change the top layer of weights.

Current output: o_j=0.2

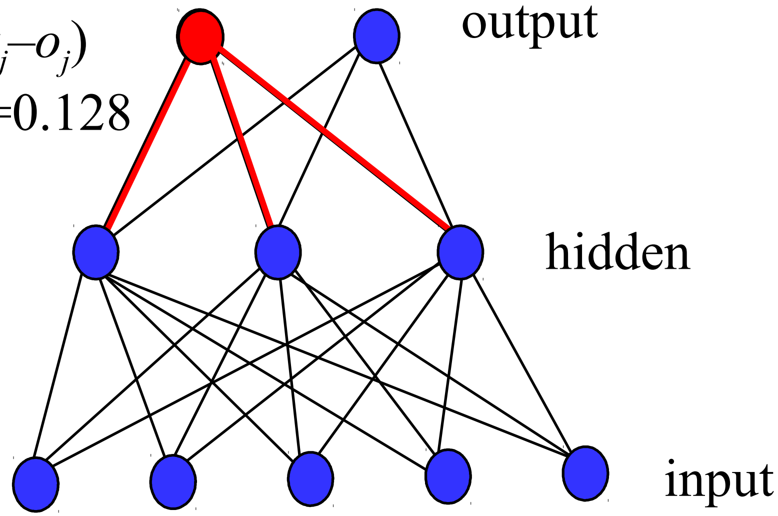
Correct output: t_j=1.0

Error $\delta_j = o_j(1-o_j)(t_j-o_j)$

$0.2(1-0.2)(1-0.2)=0.128$

Update weights into j

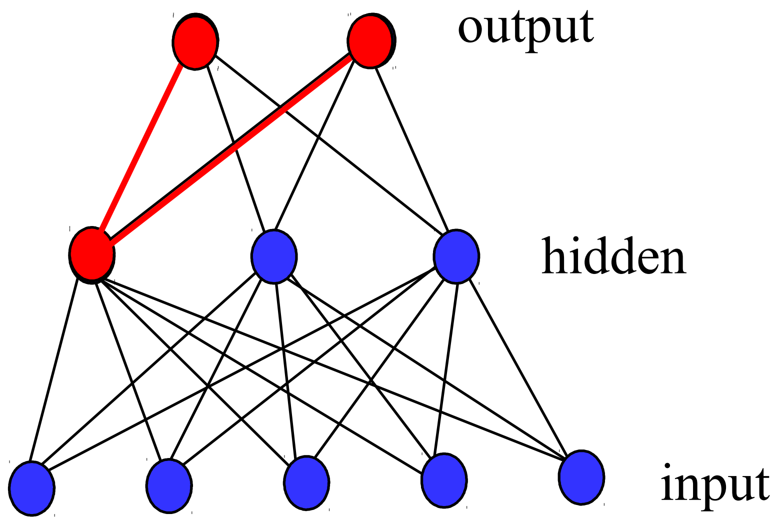
$$\Delta w_{ji} = \eta \delta_j o_i$$



Error Backpropagation

- Next calculate error for hidden units based on errors on the output units it feeds into.

$$\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{kj}$$



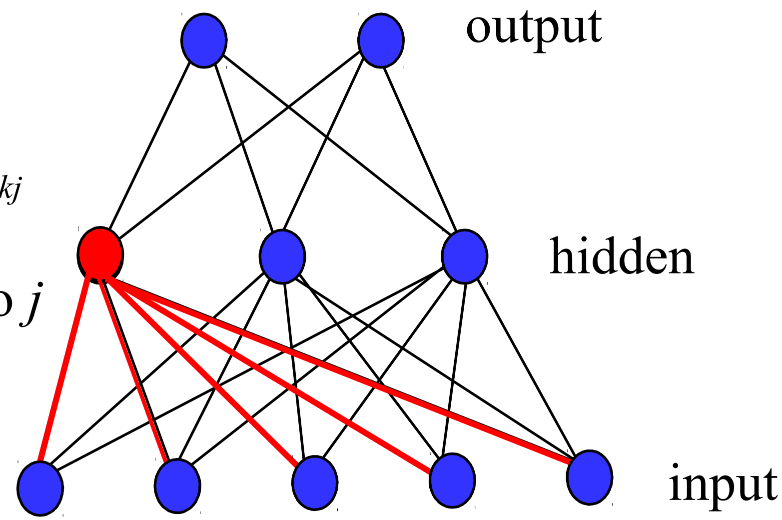
Error Backpropagation

- Finally update bottom layer of weights based on errors calculated for hidden units.

$$\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{kj}$$

Update weights into j

$$\Delta w_{ji} = \eta \delta_j o_i$$



Backpropagation Training Algorithm

Create the 3-layer network with *H* hidden units with full connectivity between layers. Set weights to small random real values.

Until all training examples produce the correct value (within ε), or mean squared error ceases to decrease, or other termination criteria:

Begin epoch

For each training example, *d*, do:

Calculate network output for *d*'s input values

Compute error between current output and correct output for *d*

Update weights by backpropagating error and using learning rule

End epoch

Comments on Training Algorithm

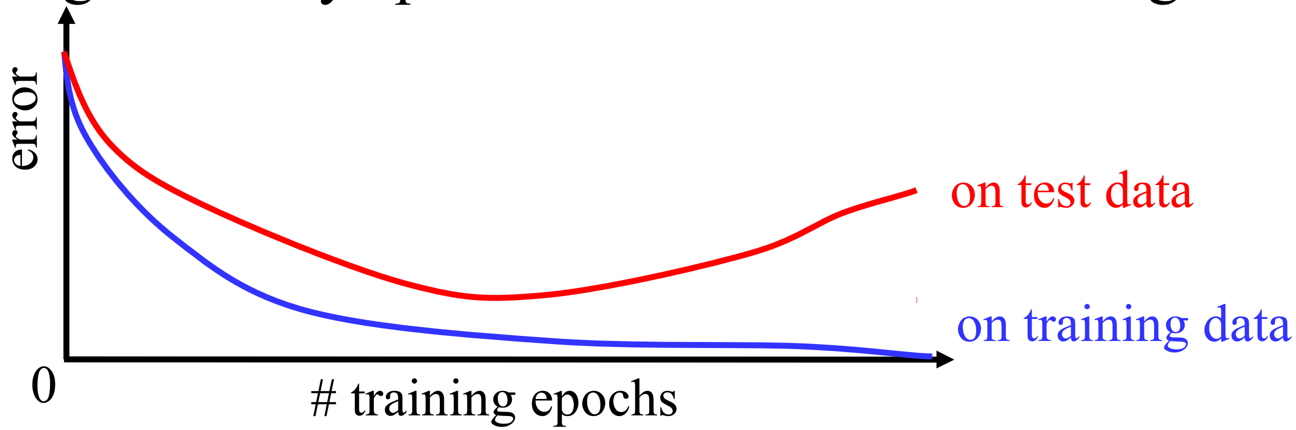
- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data.
- Many epochs (thousands) may be required, hours or days of training for large networks.
- To avoid local-minima problems, run several trials starting with different random weights (*random restarts*).
  - Take results of trial with lowest training set error.
  - Build a committee of results from multiple trials (possibly weighting votes by training set accuracy).

Hidden Unit Representations

- Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
- On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc..
- However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.

Over-Training Prevention

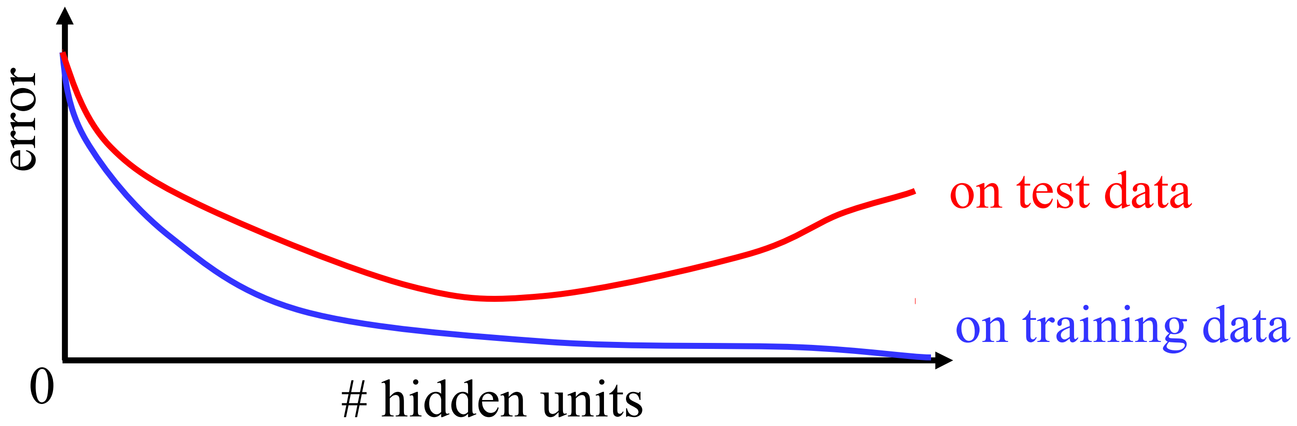
- Running too many epochs can result in over-fitting.



- Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.
- To avoid losing training data for validation:
  - Use internal 10-fold CV on the training set to compute the average number of epochs that maximizes generalization accuracy.
  - Train final network on complete training set for this many epochs.

Determining the Best Number of Hidden Units

- Too few hidden units prevents the network from adequately fitting the data.
- Too many hidden units can result in over-fitting.



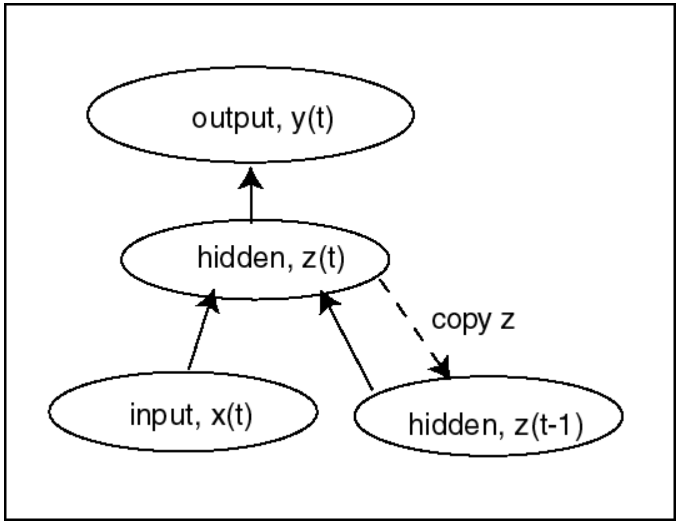
- Use internal cross-validation to empirically determine an optimal number of hidden units.

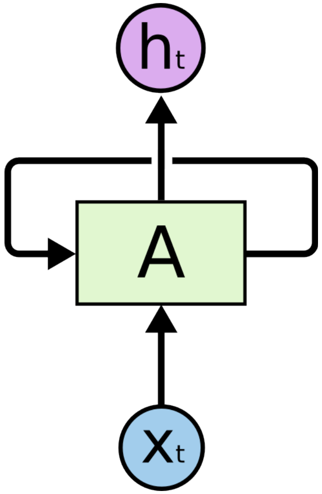
Recurrent Neural Networks (RNN)

- Add feedback loops where some units' current outputs determine some future network inputs.
- RNNs can model dynamic finite-state machines, beyond the static combinatorial circuits modeled by feed-forward networks.

Simple Recurrent Network (SRN)

- Initially developed by Jeff Elman ("*Finding structure in time*," 1990).
- Additional input to hidden layer is the state of the hidden layer in the previous time step.

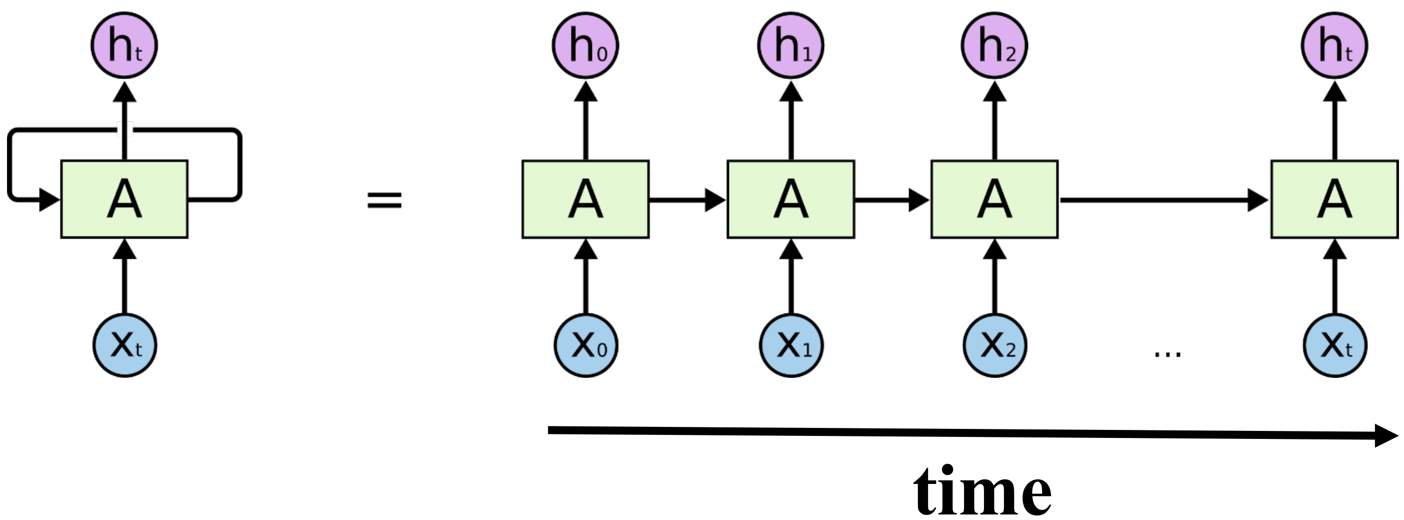




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

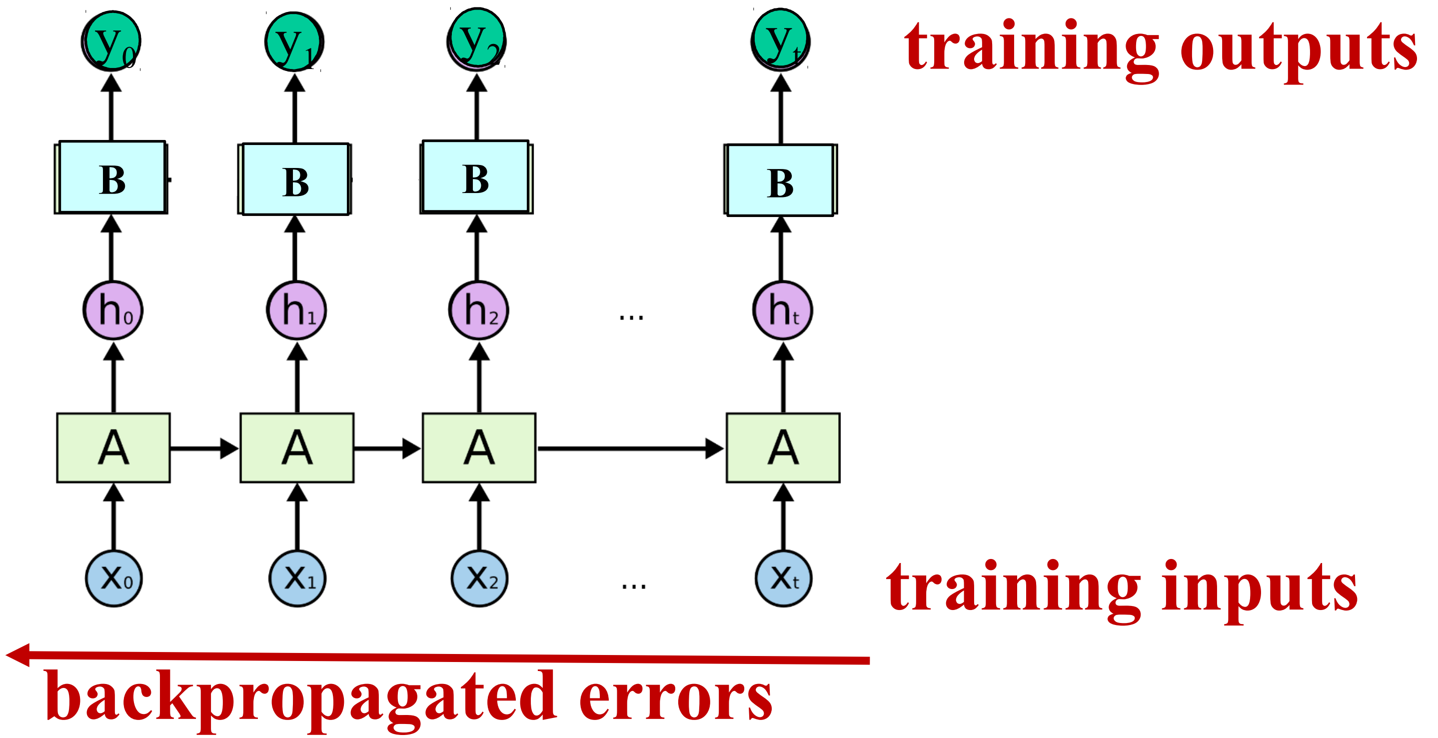
Unrolled RNN

- Behavior of RNN is perhaps best viewed by "unrolling" the network over time.



Training RNN's

- RNNs can be trained using "backpropagation through time."
- Can viewed as applying normal backprop to the unrolled network.



Conclusions

- "Feed forward" neural networks are a powerful machine learning technique for feature-vector classification.
- Training becomes increasingly difficult as the number of neural layers increases.
  - Perceptron for training a single layer network
  - Backpropagation for multi-layer networks
- Recurrent neural networks can perform sequence modeling and labeling, but backpropagation thru time has problems training unrolled networks that are "deep in time."