COMP 652: Machine Learning

Lecture 7

Today

- “Constructive” methods for determining ANN architecture
- Other types of networks and structures

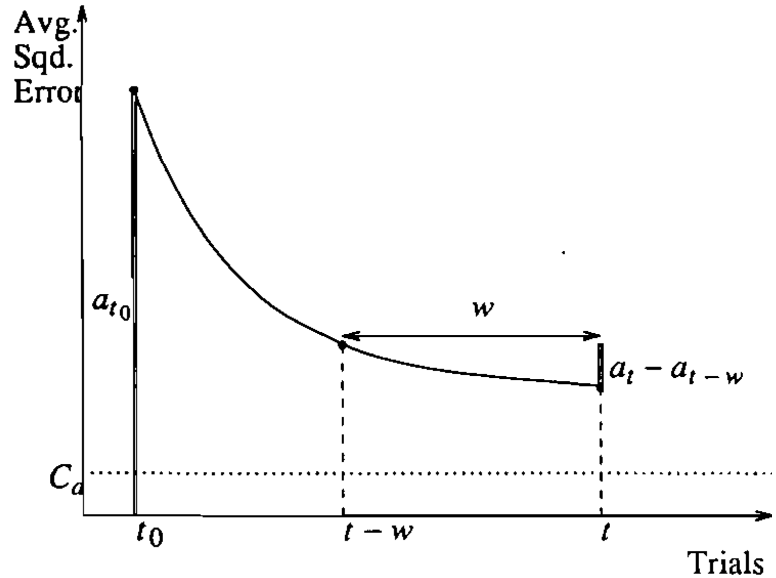
Constructive methods

- How many hidden layers should my network have?
- How many hidden units in each layer?
- Too few hidden units $\Rightarrow$ cannot approximate the desired input-output function well
- Too many hidden units $\Rightarrow$ danger of overfitting; difficult to optimize weights
- *Constructive* methods attempt to find the right answers to these questions, by:

  1. Starting with a network that is almost certainly too small.
  2. "Growing" it somehow, adding hidden nodes as needed

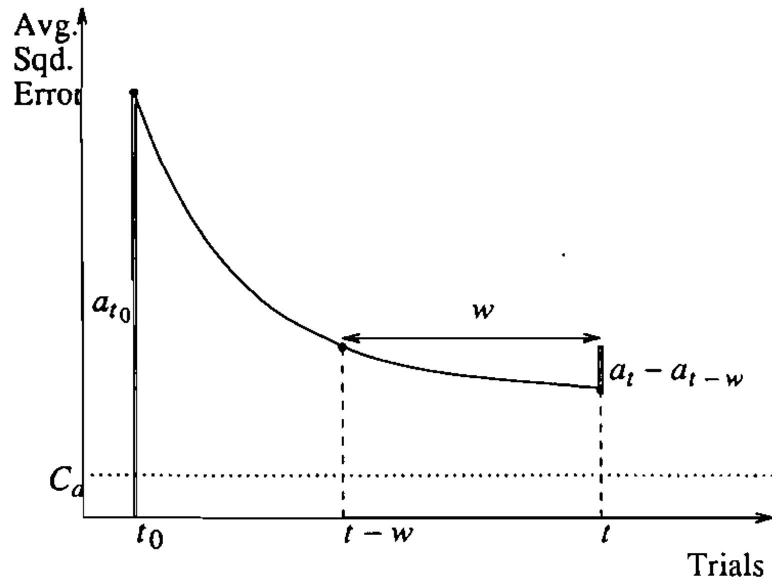
Dynamic node creation (Ash, 1989)

- Start with one hidden unit
- Train using backprop
- If at any time, the learning curve is "flattening out too much", add another hidden unit and keep training



Dynamic node creation (Ash, 1989)

- Start with one hidden unit
- Train using backprop
- If at any time, the learning curve is "flattening out too much", add another hidden unit and keep training



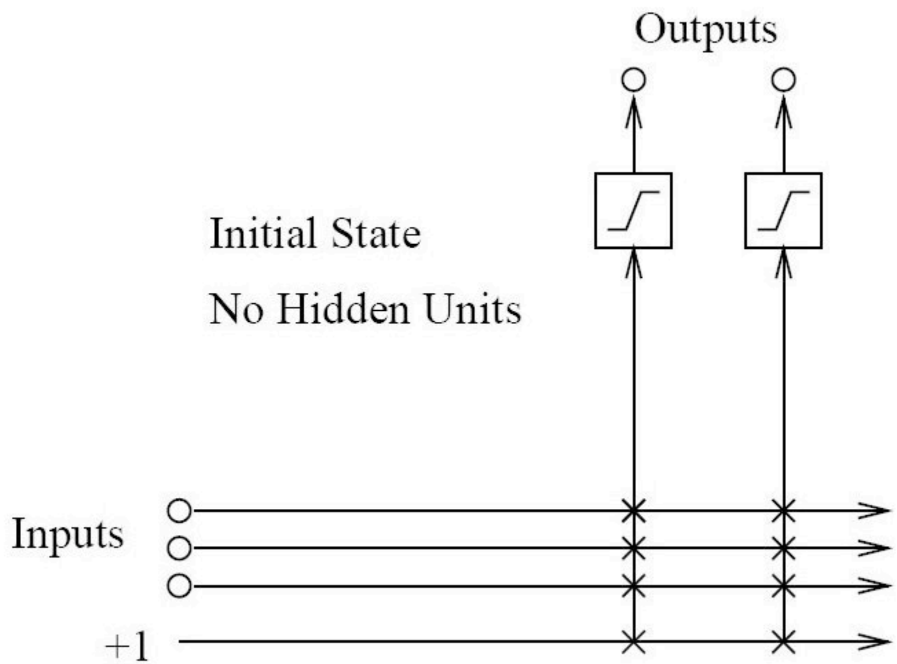
- Criteria: $\frac{a_t - a_{t-w}}{a_w} < \Delta_T$ (??) and $t - w \geq t_0$, where t_0 is time at which last node was added, w is time "window" size, Δ_T is slope-trigger threshold, and a_x is SSQ at time x.

Dynamic node creation (Ash, 1989)

- Start with one hidden unit
- Train using backprop
- If at any time, the learning curve is "flattening out too much", add another hidden unit and keep training
- Criteria: $\frac{a_t - a_{t-w}}{a_w} < \Delta_T$ (??) and $t - w \geq t_0$, where t_0 is time at which last node was added, w is time "window" size, Δ_T is slope-trigger threshold, and a_x is SSQ at time x.
- Node addition stops when SSQ less than a user-defined threshold, or the squared error on every instance is below a user-defined threshold.
- Found minimum or near-minimum size networks for a variety of problems: autoencoding, binary addition, parity.

Cascade correlation (Fahlman & Lebiere, 1991)

- Start with no hidden units, just the inputs connected to the outputs



- Train using backpropagation until the error is stable. (This is just logistic regression, really.)
- If the error is small enough, stop, otherwise consider adding a new hidden unit. (User choice? XVal?)

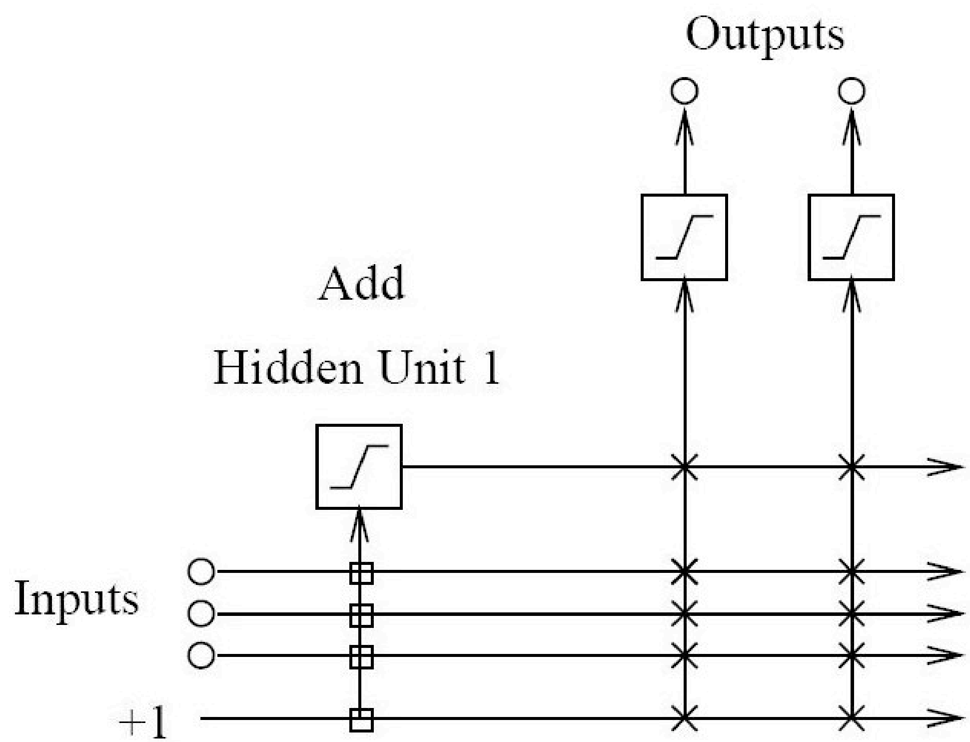
Cascade correlation (II)

- If adding a hidden unit, it will take input from all input units.
- Before connecting it to the output layer, train its incoming weights to maximize the *correlation to the error* of the previous network (using gradient descent):

$$\sum_i (J_i - \bar{J}_i)(o_i - \bar{o})$$

  where o_i is the output of the neuron for pattern i and J_i is the error on pattern i, $\bar{o}$ and $\bar{J}$ are averages over all patterns.
- Freeze the weights coming into the unit and train only the weights going into the output neuron

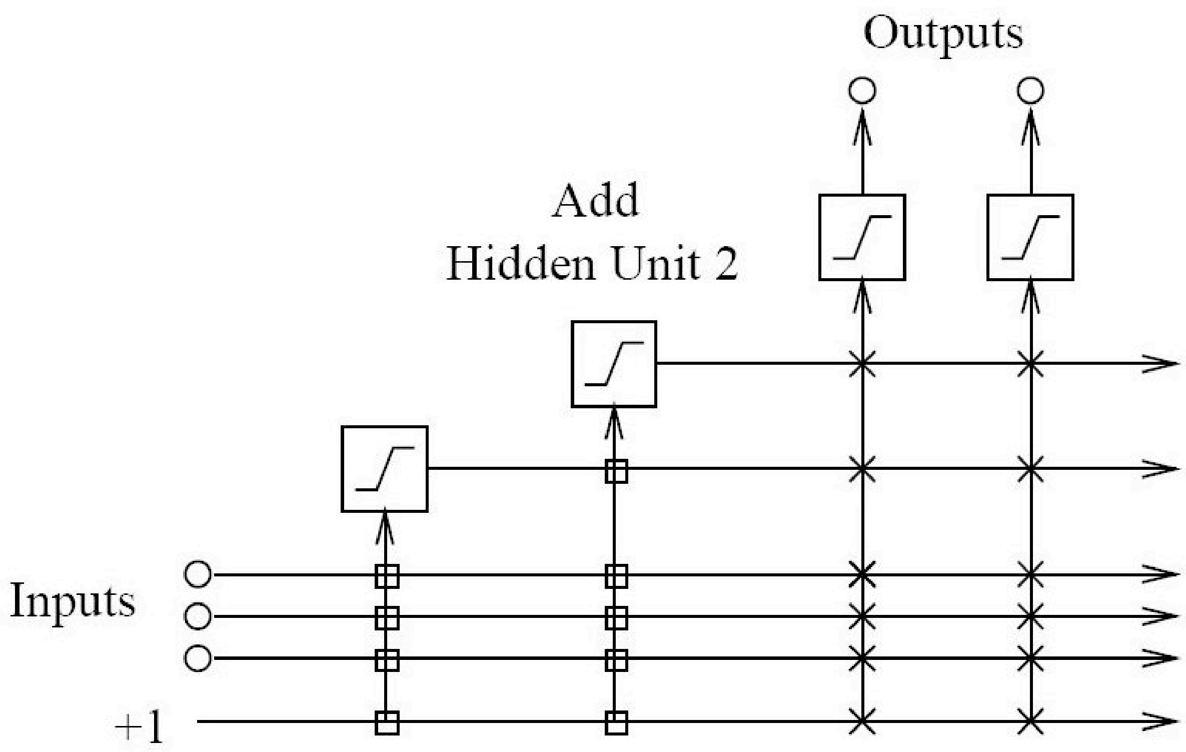
Network after adding one hidden unit



Cascade correlation (III)

- If you want to continue adding hidden units:
- Create a new unit taking input from whole input layer as well as all previous hidden units
- Train weights to maximize correlation with output errors
- Freeze weights and add to network
- Train weight going into output layer

Network after adding two hidden units



Cascade correlation (IV)

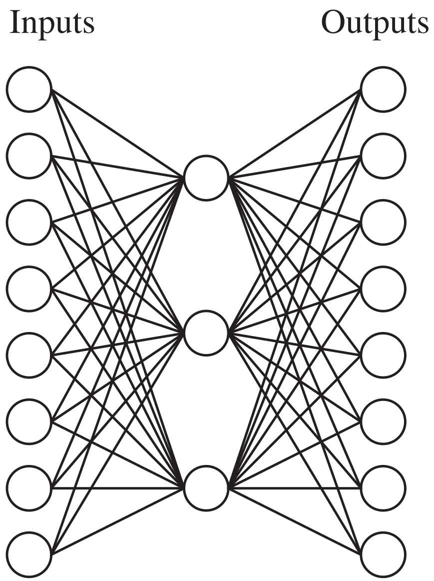
- The algorithm alternates between two phases:
  - Input phase: Training hidden units
  - Output phase: Training the weights that connect the units to the output

  until the error is below a desired threshold
- Because only a small set of weights is trained at one time, training is very fast despite the depth of the network (Logistic regression)
- Variations allow for the new neurons to go either in the same layer or in a new layer.

Some special ANN architectures

Auto-encoders

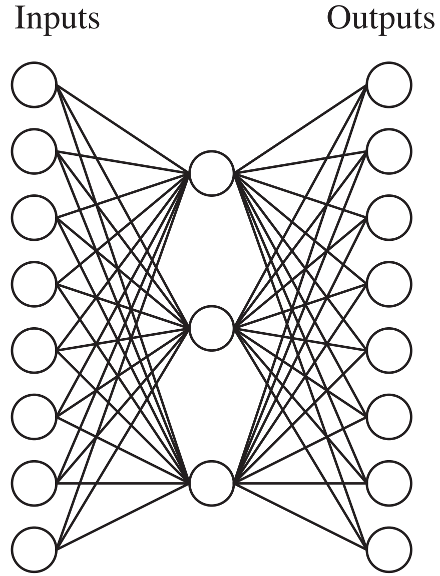
- Autoencoder ANNs are trained to reproduce the input at the output layer. (I.e., training data is of the form .$\{\langle \mathbf{x_i}, \mathbf{x_i} \rangle\}$).
- The hidden layer is smaller than the input and output layers.



- Why would you do that?

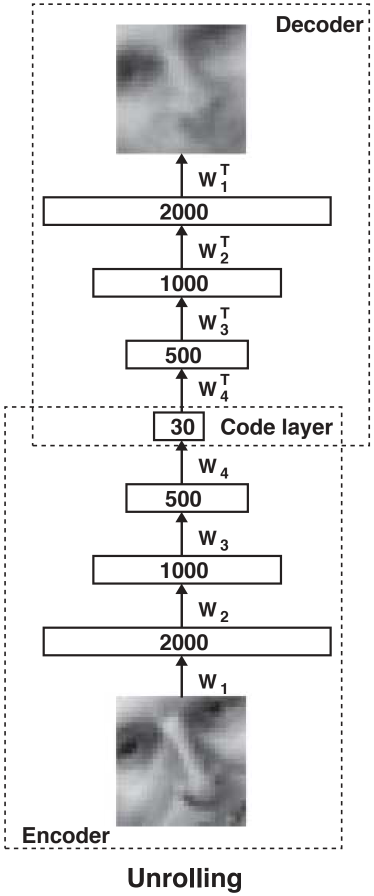
Auto-encoders

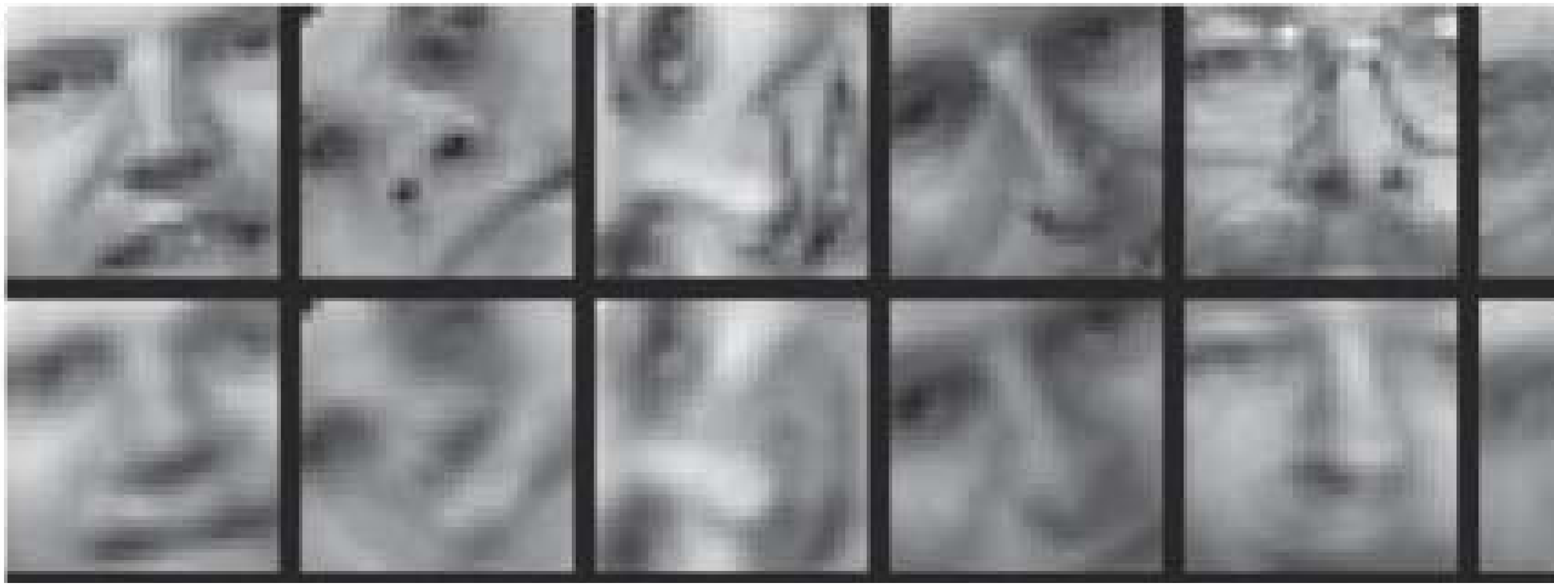
- Autoencoder ANNs are trained to reproduce the input at the output layer. (I.e., training data is of the form $.\{\langle \mathbf{x_i}, \mathbf{x_i} \rangle\}$).
- The hidden layer is smaller than the input and output layers.



- Why would you do that?
- Answers: Dimensionality reduction (which has various benefits), error correction, pattern completion . . .

Example: Face reconstruction (Hinton, 2006)



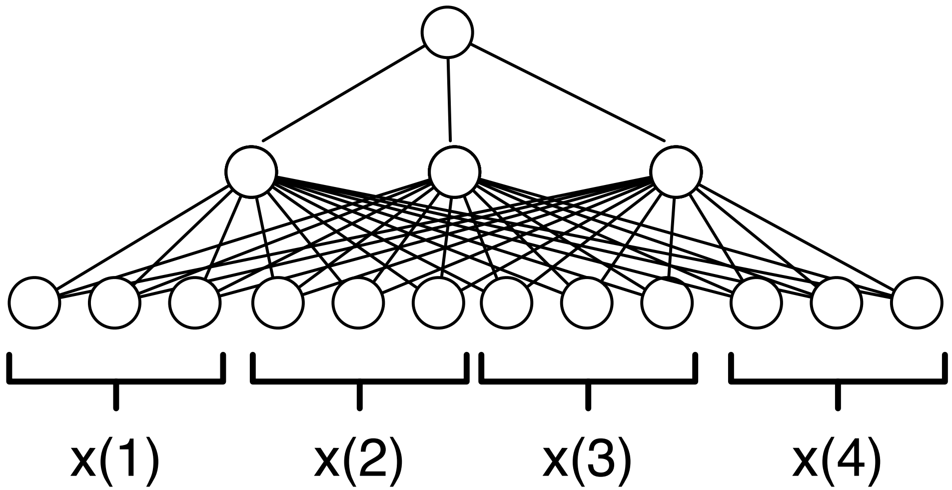


Neural networks for temporal data

- In temporal data, we have a one or succession of inputs at different times $\mathbf{x}(t)$ – or possibly a set of such time series.
- There can be several tasks:
  - Predicting a class label
    E.g., Speech recognition – what phoneme is being spoken at the moment?
    E.g., Prognosis – will / is the patient responding to therapy? Is therapy warranted?
  - Time series predictions E.g., What will be the next state, $\mathbf{x}(t+1)$?
    E.g., Endpoint prediction – where is this trajectory leading?
    (Relevant in medicine, finance, user modeling, etc.)
- Prediction may need to be performed as the data is received
- Exactly what to use as input no longer clear! Past as well as current information may be relevant.

Prediction based on the whole trajectory

- When predicting an outcome based on the whole trajectory, can simply input the whole thing!
  (This assume you have multiple time series, each with its own associated target value y.)



- Could have many weights to fit, depending on length of time series and length of $\mathbf{x}(t)$.
- Problematic if different time series have different lengths.

Time-delay neural networks (Waibel)

- Suppose we have a single time series $\mathbf{x}(t)$ with targets $y(t)$ associated to each time point.
- Fix a time window T
- Collect the inputs $\mathbf{x}(t), \ldots \mathbf{x}(t-T)$ and feed them together to the network.
- Shift the window and repeat
- Train using standard backpropagation. Data set is:

$$\begin{array}{ccccccc} \mathbf{x}(1) & \mathbf{x}(2) & \ldots & \mathbf{x}(T) & \rightarrow & y(T) \\ \mathbf{x}(2) & \mathbf{x}(3) & \ldots & \mathbf{x}(T+1) & \rightarrow & y(T+1) \\ \mathbf{x}(3) & \mathbf{x}(4) & \ldots & \mathbf{x}(T+2) & \rightarrow & y(T+2) \end{array}$$

Time series modeling

- Given one or more time series $\mathbf{x}(t)$, we may want to model the dynamics of the system generating the time series
- One obvious approach is to solve the prediction problem: $\mathbf{x}(t) \rightarrow \mathbf{x}(t+1)$. (Or more generally, we may include the time delay approach from the previous slide.)
- Suppose we minimize the sum squared error:

$$J_{\mathbf{w}} = \sum_t (\mathbf{x}(t+1) - h_{\mathbf{w}}(\mathbf{x}(t)))^2$$

- This corresponds to a maximum likelihood hypothesis under what noise model?

Time series modeling

- Given one or more time series $\mathbf{x}(t)$, we may want to model the dynamics of the system generating the time series
- One obvious approach is to solve the prediction problem: $\mathbf{x}(t) \rightarrow \mathbf{x}(t+1)$. (Or more generally, we may include the time delay approach from the previous slide.)
- Suppose we minimize the sum squared error:

$$J_{\mathbf{w}} = \sum_{t} (\mathbf{x}(t+1) - h_{\mathbf{w}}(\mathbf{x}(t)))^2$$

- This corresponds to a maximum likelihood hypothesis under what noise model?

Time series modeling

- Given one or more time series $\mathbf{x}(t)$, we may want to model the dynamics of the system generating the time series
- One obvious approach is to solve the prediction problem: $\mathbf{x}(t) \rightarrow \mathbf{x}(t+1)$. (Or more generally, we may include the time delay approach from the previous slide.)
- Suppose we minimize the sum squared error:

$$J_{\mathbf{w}} = \sum_t (\mathbf{x}(t+1) - h_{\mathbf{w}}(\mathbf{x}(t)))^2$$

- This corresponds to a maximum likelihood hypothesis under what noise model?

$$\mathbf{x}(t+1) = h_{\mathbf{w}}(\mathbf{x}(t)) + \epsilon_t$$

  where the ϵ_t are vectors of i.i.d. Gaussian with common standard deviation σ.
- Interpretation: Actual next state, $\mathbf{x}(t+1)$ is noisy version of expected next state $h_{\mathbf{w}}(\mathbf{x}(t))$.

An alternative, trajectory-based, error criterion

- For hypothesis $h_{\mathbf{w}}$, suppose we generate a trajectory as follows:

$$\begin{aligned}\mathbf{z}(1) &= \mathbf{x}(1)\\ \mathbf{z}(t+1) &= h_{\mathbf{w}}(\mathbf{z}(t))\end{aligned}$$

- Suppose we minimize the sum squared error:

$$J_{\mathbf{w}} = \sum_t (\mathbf{x}(t) - \mathbf{z}(t))^2$$

- This corresponds to a maximum likelihood hypothesis under what noise model?

An alternative, trajectory-based, error criterion

- For hypothesis $h_{\mathbf{w}}$, suppose we generate a trajectory as follows:

$$\begin{aligned}\mathbf{z}(1) &= \mathbf{x}(1)\\ \mathbf{z}(t+1) &= h_{\mathbf{w}}(\mathbf{z}(t))\end{aligned}$$

- Suppose we minimize the sum squared error:

$$J_{\mathbf{w}} = \sum_t (\mathbf{x}(t) - \mathbf{z}(t))^2$$

- This corresponds to a maximum likelihood hypothesis under what noise model?

$$\mathbf{x}(t) = \mathbf{z}(t) + \epsilon(t)$$

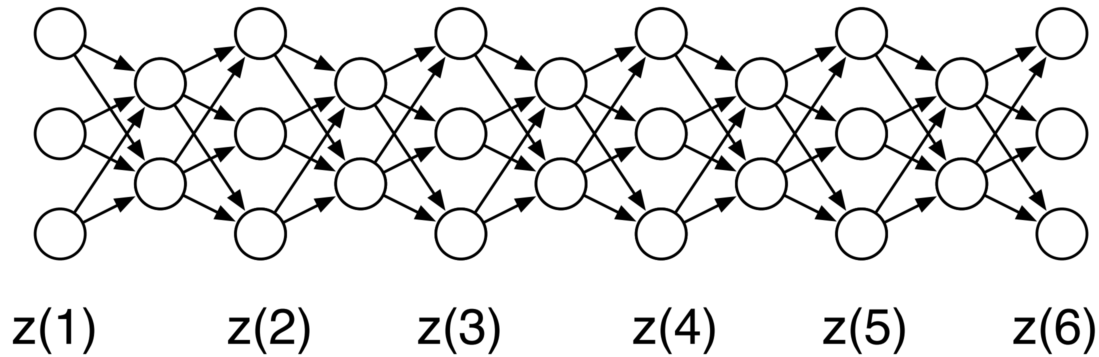
  where the ϵ_t are vectors of i.i.d. Gaussian with common standard deviation σ.
- Interpretation: The actual trajectory was the deterministic $\mathbf{z}(t)$, and we observed a noisy version of that.

How to minimize trajectory-based error?

How to minimize trajectory-based error?

Backpropagation through time (Rumelhart, Hinton and Williams)

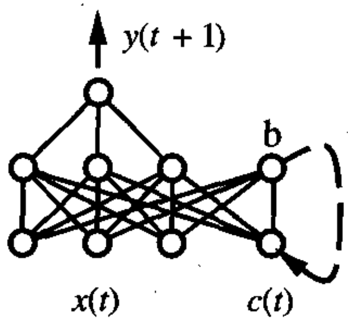
- Consider the contribution to $J_{\mathbf{w}}$ at one time point: $J_{\mathbf{w}}(t) = (\mathbf{x}(t) - \mathbf{z}(t))^2$.
- $\mathbf{z}(t) = h_{\mathbf{w}}(h_{\mathbf{w}}(\ldots h_{\mathbf{w}}(\mathbf{z}(0))\ldots))$. It is computed by chaining together, or "unrolling", $t-1$ copies of the ANN:



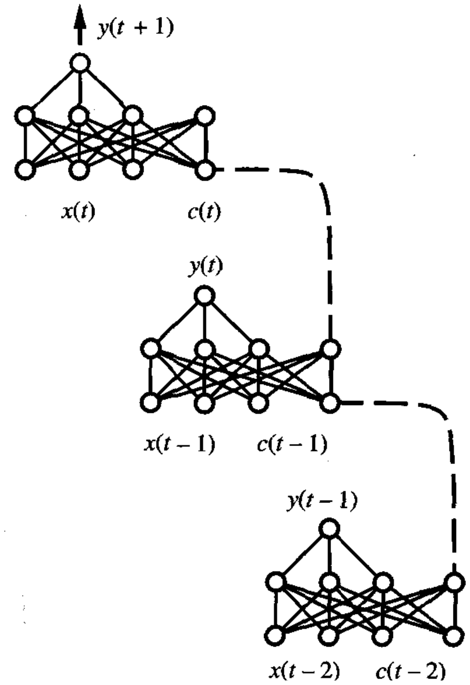
- Can apply standard backprop to this network, except every weight occurs $t-1$ times.
- Just sum up error partials for every weight occurrence.
- Results in very deep nets that are hard to train, but it *can* work.

Recurrent neural networks

- More generally, we can allow some of the network's hidden or output units to feed back into the next iteration



- Training by backpropagation through time
- Intuition: Hidden feedback units can form a "memory" the persists through time.



Artificial Neural Networks Summary

- ANNs are justified by the inability of single-layer approximators (e.g., linear, logistic) to capture non-monotone functions.
- ANN weights can be trained for classification or regression by the error backpropagation algorithm ("backprop")
- A variety of tricks are used to make optimization more efficient
  - Initialization of weights to small random values
  - Good choice of input-output coding (including normalization)
  - Momentum, delta-bar-delta, or second-order methods to speed convergence

Artificial Neural Networks Summary (II)

- ANN structure is often chosen by the designer, but can be part of the fitting process.
  - One must watch out for overtraining (a special kind of overfitting)
  - Destructive methods (weight decay, optimal brain damage) try to simplify the network during or after learning
  - Constructive methods (dynamic node creation, cascade correlation) create new units during learning
- Special cases for special tasks
  - Autoencoders for dimensionality reduction
  - Time-delay neural nets for prediction based on recent history of a time-series
  - "Unrolled" networks and backprop through time for trajectory matching