Forecasting

PACF: the partial autocorrelation function

- **Motivation:**

  For MA(q) models, the ACF will be zero for lags greater than q, and will not be zero at lag q.

  However, the ACF alone does not tell us much for ARMA or AR models

  For a causal AR(1) model: $\gamma(2) = \phi^2\gamma(0) \neq 0$ as x_{t-2} is dependent on x_t through x_{t-1}

  To remove the effect of x_{t-1}, we have $\text{cov}(x_t - \phi x_{t-1}, x_{t-2} - \phi x_{t-1}) = 0$

The partial autocorrelation function is defined as the conditional correlation between x_t and x_{t-k}, after controlling for the effects of the intermediate variables $x_{t-1}, x_{t-2}, \ldots, x_{t-k+1}$.

(Removing the linear dependence of both variables on the intermediate observations)

Review

PACF measures the correlation between a time series observation and a lagged observation, while **removing the effects** of any intermediate lags.

- **PACF definition:**

The partial autocorrelation function (PACF) of a stationary process, x_t, denoted ϕ_{hh}, for $h = 1, 2, \ldots,$ is

$$\phi_{11} = \text{corr}(x_{t+1}, x_t) = \rho(1),$$
$$\phi_{hh} = \text{corr}(x_{t+h} - \hat{x}_{t+h}, x_t - \hat{x}_t), \; h \geq 2$$

$\hat{x}_{t+h}$: regression of x_{t+h} on $\{x_{t+h-1}, x_{t+h-2}, \ldots, x_{t+1}\}$ as

$$\hat{x}_{t+h} = \beta_1 x_{t+h-1} + \beta_2 x_{t+h-2} + \cdots + \beta_{h-1} x_{t+1}$$

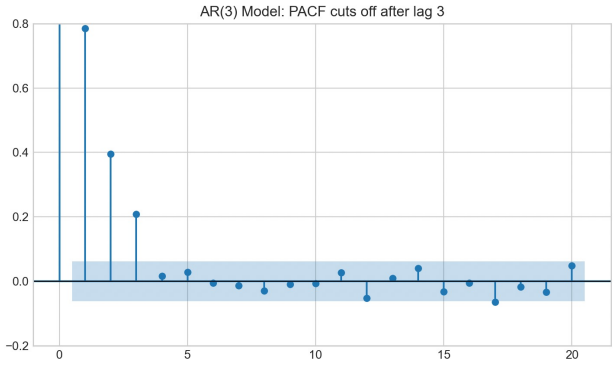
$\hat{x}_t$: regression of x_t on $\{x_{t+h-1}, x_{t+h-2}, \ldots, x_{t+1}\}$ as

$$\hat{x}_t = \beta_1 x_{t+1} + \beta_2 x_{t+2} + \cdots + \beta_{h-1} x_{t+h-1}$$

Review

Table 3.1. Behavior of the ACF and PACF for ARMA models

	AR(p)	MA(q)	ARMA(p, q)
ACF	Tails off	Cuts off after lag q	Tails off
PACF	Cuts off after lag p	Tails off	Tails off

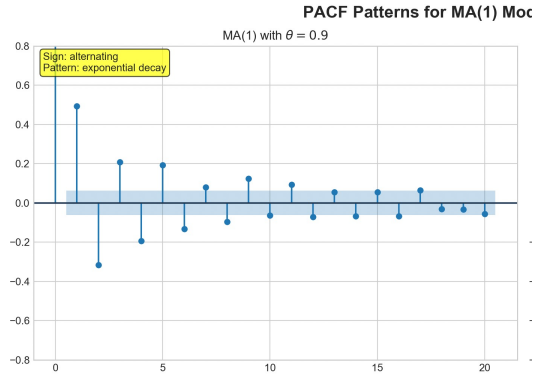


PACF Cut-Off Property:

• AR(1): Significant at lag 1 only
• AR(2): Significant at lags 1-2 only
• AR(3): Significant at lags 1-3 only

The PACF 'cuts off' after lag p for an AR(p) model, meaning: $\varphi_{hh} \approx 0$ for all $h > p$

This is the key identifying characteristic of AR processes.



Forecasting

- **Objective:**

  Predict future values of a time series, $x_{n+m},\ m = 1, 2, \ldots,$
  based on the data collected to present, $x_{1:n} = \{x_1, x_2, \ldots, x_n\}$

1. Linear Model Setup

Assume that the future value x_{n+m} of the time series can be predicted by a weighted sum of the existing data points $x_1, x_2, \ldots, x_n$. The linear prediction model is given by:

$$x_{n+m} = \alpha_0 + \alpha_1 x_n + \alpha_2 x_{n-1} + \cdots + \alpha_k x_{n-k+1}$$

where $\alpha_0, \alpha_1, \ldots, \alpha_k$ are the coefficients to be determined.

2. Optimal Linear Prediction

The **optimal linear predictor** is the one that minimizes the prediction error. This is typically achieved by minimizing the **Mean Squared Error (MSE)**. The goal of minimizing the MSE is to reduce the difference between the predicted values and the true values.

- **Mean square prediction error:**

  $$\mathrm{E}(x_{n+m} - x_{n+m}^{n})^2$$

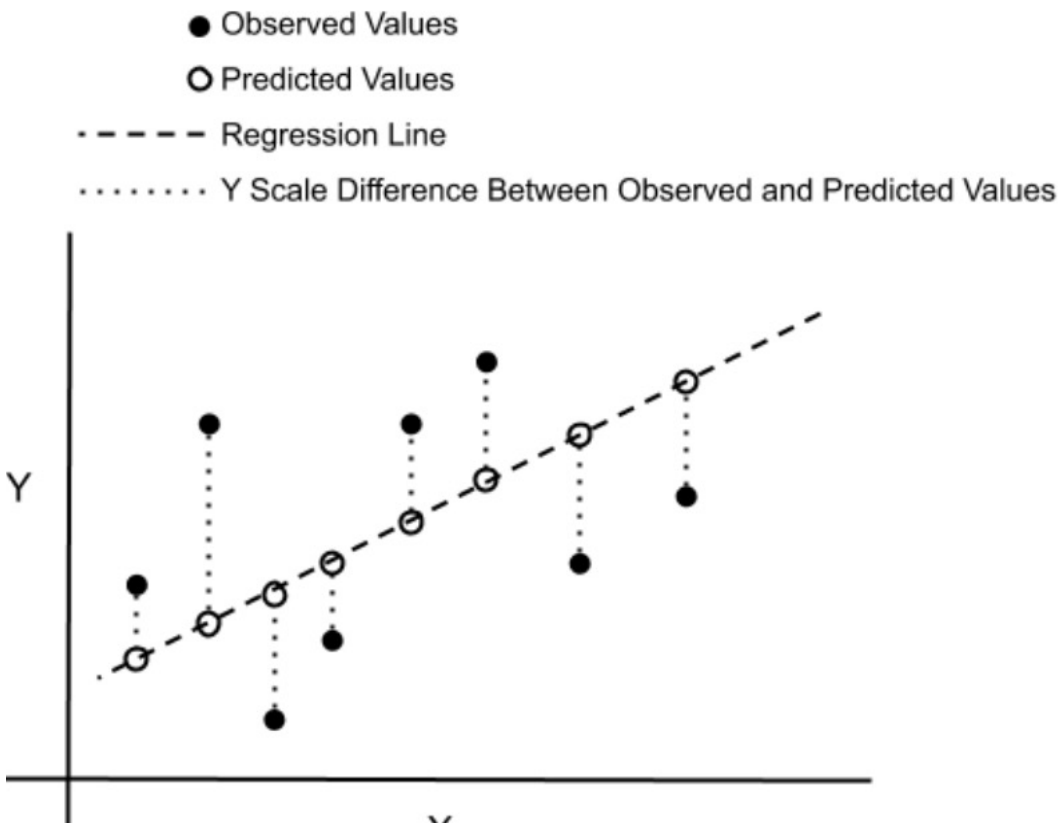
  the average of the squares of the errors
  the error is the amount by which the actual values differ from the predicted values.

- **Minimum mean square error predictor:**

  $$x_{n+m}^{n} = \mathrm{E}(x_{n+m} | x_{1:n})$$

- **Mean square prediction error:**

$$\mathrm{E}(x_{n+m} - x_{n+m}^{n})^2$$



- **Minimum mean square error predictor:**

$$x_{n+m}^{n} = \mathrm{E}(x_{n+m}|x_{1:n})$$

The conditional expectation has an important mathematical property: given the past data, the conditional expectation is the best predictor that minimizes the mean square error. In other words, given the past data $x_{1:n}$, the conditional expectation $E(x_{n+m}|x_{1:n})$ is the best estimate of the future value because it minimizes the mean square error (MSE).

Explanation:

- **Conditional Expectation**: Given certain information, the conditional expectation is the optimal predictor for future values. It is the best predictor because it minimizes the mean square error (MSE).
- **Mean Square Error (MSE)**: MSE is a common metric for measuring prediction errors. It is the expected value of the square of the difference between predicted and actual values.

Linear predictor

- **Predictors of the form:**

$$x_{n+m}^{n} = \alpha_0 + \sum_{k=1}^{n} \alpha_k x_k$$

- **Best linear predictors (BLPs):**

  Linear predictors that minimize the mean square prediction error

  For Gaussian process, minimum mean square error predictors and best linear predictors are the same

  Typically, this can be found by minimizing the mean square prediction error (MSE).

BLP for stationary processes

- **The BLP can be found by solving**

$$\mathrm{E}[(x_{n+m} - x_{n+m}^{n})x_k] = 0,\ k = 0, 1, 2, \ldots, n,$$

$$\text{where } x_0 = 1$$

x_{n+m}^{n} represents:

- **Subscript** $n + m$: The time point we want to predict (the m-th step into the future).
- **Superscript** n: All the data up to time n are used for the prediction.

  If $n = 5, m = 3$:

  x_8^5 represents using $x_0, x_1, x_2, x_3, x_4, x_5$ to predict x_8.

  - The superscript 5 indicates that the data up to the 5th data point are used.
  - The subscript 8 indicates that the prediction is for the 8th data point.

BLP for stationary processes

- **The BLP can be found by solving**

$$\mathrm{E}[(x_{n+m} - x_{n+m}^{n})x_k] = 0,\ k = 0, 1, 2, \ldots, n,$$

$$\text{where } x_0 = 1$$

The mathematical principle of the **Best Linear Predictor (BLP)** for a **stationary process**

Orthogonality Principle – the prediction error is orthogonal (uncorrelated) to all variables used for prediction.

The prediction error $(x_{n+m} - \hat{x}_{n+m})$

uncorrelated (with zero covariance)

with each of the predictor variables x_k

Geometric Intuitive Understanding:

Imagine the random variables as vectors:

- $x_1, x_2, \ldots, x_n$ are the "basis vectors" used for prediction.
- x_{n+m} is the target vector we want to predict.
- x_{n+m}^n is the projection of the target vector onto the space spanned by the basis vectors.

Orthogonality means that the prediction error vector is perpendicular to the entire space spanned by the basis vectors.

Practical Example:

Suppose we use height (x_1) and weight (x_2) to predict blood pressure (y):

If we find that the prediction error is correlated with height, it means that taller individuals' blood pressure is systematically under- or overestimated.

This indicates that the height factor has not been properly utilized.

We should adjust the coefficient for height.

Only when the error is uncorrelated with all predictor variables does it indicate that we have fully utilized all available information.

BLP for stationary processes

- **The BLP can be found by solving**

$$\mathrm{E}[(x_{n+m} - x_{n+m}^{n})x_k] = 0,\ k = 0, 1, 2, \ldots, n,$$

where $x_0 = 1$

$\Rightarrow$ The prediction error is uncorrelated with each predictor variable.

$\Rightarrow$ All linear information has been extracted.

$\Rightarrow$ The best linear prediction has been achieved.

BLP for stationary processes

- **The BLP can be found by solving**

$$\mathrm{E}[(x_{n+m} - x_{n+m}^{n})x_k] = 0,\ k = 0, 1, 2, \ldots, n,$$

where $x_0 = 1$

$\Rightarrow$ The prediction error is uncorrelated with each predictor variable.

$\Rightarrow$ All linear information has been extracted.

$\Rightarrow$ The best linear prediction has been achieved.

To find the Best Linear Predictor (BLP), we need to solve a system of equations to find the optimal regression coefficients such that the error is uncorrelated with each predictor variable.

Let's consider a simple example:

Suppose we want to predict the value of x_{n+m} at a future time, using the past values $x_0, x_1, \ldots, x_n$ to build a linear model:

$$\hat{x}_{n+m} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

To find the best linear predictor, the error must be uncorrelated with all predictor variables (including $x_1, x_2, \ldots, x_n$ and the constant term β_0). We ensure this by solving the following equation:

$$E[(x_{n+m} - \hat{x}_{n+m})x_k] = 0$$

The steps to solve this are as follows:

1. **Calculate the covariance** between each variable $x_1, x_2, \ldots, x_n$ and the target value x_{n+m}.
2. **Form the system of linear equations** based on the equation $E[(x_{n+m} - \hat{x}_{n+m})x_k] = 0$, which will be used to solve for the regression coefficients.
3. The solution for the coefficients $\beta_0, \beta_1, \ldots, \beta_n$ will be the parameters of the best linear predictor model.

Suppose we have the following data:

Time t	Observed Value x_t
x_0	1
x_1	2
x_2	3
x_3	4
x_4	5

We want to use x_0, x_1, x_2 to predict x_4, i.e.,

$$\hat{x}_4 = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Calculate the expectation: We need to calculate the covariance between the error and each of the predictive variables, i.e.,

$$E[(x_4 - \hat{x}_4)x_k] \qquad\qquad E[(x_4 - \hat{x}_4)x_k] = 0$$

BLP for stationary processes

- **The BLP can be found by solving**

$$\mathrm{E}[(x_{n+m} - x_{n+m}^{n})x_k] = 0,\ k = 0, 1, 2, \ldots, n,$$

where $x_0 = 1$

- **Prove:**

$$\text{minimize } \mathrm{E}(x_{n+m} - x_{n+m}^{n})^2$$

$$\frac{\partial \mathrm{E}(x_{n+m} - x_{n+m}^{n})^2}{\partial \alpha_k} = \mathrm{E}[(x_{n+m} - x_{n+m}^{n})x_k] = 0$$

One-step ahead prediction

- **The BLP of x_{n+1}:** Predicting the value at the next time step

$$x_{n+1}^{n} = \phi_{n1} x_n + \phi_{n2} x_{n-1} + \cdots + \phi_{nn} x_1$$

$$\mathrm{E}\left(\left(x_{n+1} - \sum_{j=1}^{n} \phi_{nj} x_{n+1-j} \right) x_{n+1-k} \right) = 0, \; k = 1, 2, \ldots, n$$

$$E\left[x_{n+1} x_{n+1-k}\right] - \sum_{j=1}^{n} \phi_{nj} E\left[x_{n+1-j} x_{n+1-k}\right] = 0$$

$$\sum_{j=1}^{n} \phi_{nj} \gamma(k-j) = \gamma(k), \; k = 1, 2, \ldots, n$$

$$E\left[x_{n+1}x_{n+1-k}\right] - \sum_{j=1}^{n} \phi_{nj} E\left[x_{n+1-j}x_{n+1-k}\right] = 0$$

$$\sum_{j=1}^{n} \phi_{nj}\gamma(k-j) = \gamma(k),\ k = 1, 2, \ldots, n$$

For a zero-mean stationary process:

$$E[x_t x_s] = \mathrm{Cov}(x_t, x_s) = \gamma(|t-s|)$$

So:

$$E[x_{n+1}x_{n+1-k}] = \gamma(k) \quad \text{(because the time difference is } k\text{)}$$

$$E[x_{n+1-j}x_{n+1-k}] = \gamma(|j-k|) \quad \text{(because the time difference is } |j-k|\text{)}$$

One-step ahead prediction

- **Prediction equation:**

$$\sum_{j=1}^{n} \phi_{nj}\gamma(k-j) = \gamma(k),\ k = 1, 2, \ldots, n$$

$$\begin{bmatrix} \gamma(0) & \gamma(1) & \cdots & \gamma(n-1) \\ \gamma(1) & \gamma(0) & & \gamma(n-2) \\ \vdots & & \ddots & \vdots \\ \gamma(n-1) & \gamma(n-2) & \cdots & \gamma(0) \end{bmatrix} \begin{bmatrix} \phi_{n1} \\ \phi_{n2} \\ \vdots \\ \phi_{nn} \end{bmatrix} = \begin{bmatrix} \gamma(1) \\ \gamma(2) \\ \vdots \\ \gamma(n) \end{bmatrix}$$

Γ_n $\quad$ ϕ_n $\quad$ γ_n

One-step ahead prediction

- **Prediction equation:** $\Gamma_n \phi_n = \gamma_n$

$$\sum_{j=1}^{n} \phi_{nj} \gamma(k-j) = \gamma(k), \; k = 1, 2, \ldots, n$$

$$\begin{bmatrix} \gamma(0) & \gamma(1) & \cdots & \gamma(n-1) \\ \gamma(1) & \gamma(0) & & \gamma(n-2) \\ \vdots & & \ddots & \vdots \\ \gamma(n-1) & \gamma(n-2) & \cdots & \gamma(0) \end{bmatrix} \begin{bmatrix} \phi_{n1} \\ \phi_{n2} \\ \vdots \\ \phi_{nn} \end{bmatrix} = \begin{bmatrix} \gamma(1) \\ \gamma(2) \\ \vdots \\ \gamma(n) \end{bmatrix}$$

Γ_n ϕ_n γ_n

For a stationary process, "uncorrelated" translates into an equality involving the **autocovariance function**.

$$\Gamma_n \phi_n = \gamma_n$$

Γ_n is the $n \times n$ autocovariance matrix.

ϕ_n is the coefficient vector $(\phi_{n1}, \phi_{n2}, \ldots, \phi_{nn})'$.

γ_n is the autocovariance vector $(\gamma(1), \gamma(2), \ldots, \gamma(n))'$.

Only Autocovariance Function is Needed

To predict the future, we only need to know the autocovariance function $\gamma(h)$ of the process.

There is no need to know the entire probability distribution.

Special Structure of the Matrix

The matrix Γ_n is a **Toeplitz matrix** (its diagonal elements are equal).

This structure allows for efficient solution algorithms such as the **Levinson-Durbin recursion**.

Connection to AR Models

This is exactly the **Yule-Walker equation** for the **AR(p)** model!

Practical Application

Suppose we have a stationary time series, and we know its autocovariances:

$$\gamma(0) = 1.0 \quad (\text{Variance})$$

$$\gamma(1) = 0.8$$

$$\gamma(2) = 0.6$$

$$\gamma(3) = 0.4$$

We want to use the first three values to predict the fourth value (n = 3). We solve the following system of equations:

$$\begin{bmatrix} 1.0 & 0.8 & 0.6 \\ 0.8 & 1.0 & 0.8 \\ 0.6 & 0.8 & 1.0 \end{bmatrix} \begin{bmatrix} \phi_{31} \\ \phi_{32} \\ \phi_{33} \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.6 \\ 0.4 \end{bmatrix}$$

Solving for $\phi_{31}, \phi_{32}, \phi_{33}$, we get the optimal prediction coefficients!

$$\begin{bmatrix} \phi_{n1} \\ \phi_{n2} \\ \vdots \\ \phi_{nn} \end{bmatrix} = \begin{bmatrix} \gamma(0) & \gamma(1) & \cdots & \gamma(n-1) \\ \gamma(1) & \gamma(0) & \cdots & \gamma(n-2) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma(n-1) & \gamma(n-2) & \cdots & \gamma(0) \end{bmatrix}^{-1} \begin{bmatrix} \gamma(1) \\ \gamma(2) \\ \vdots \\ \gamma(n) \end{bmatrix}$$

Suppose we have the following autocovariances:

$$\gamma(0) = 1.0 \quad (\text{Variance})$$

$$\gamma(1) = 0.7$$

$$\gamma(2) = 0.4$$

We want to use the first two values to predict the third value (n = 2).

$$\begin{bmatrix} \gamma(0) & \gamma(1) \\ \gamma(1) & \gamma(0) \end{bmatrix} \begin{bmatrix} \phi_{21} \\ \phi_{22} \end{bmatrix} = \begin{bmatrix} \gamma(1) \\ \gamma(2) \end{bmatrix} \qquad \begin{bmatrix} \phi_{21} \\ \phi_{22} \end{bmatrix} = \begin{bmatrix} 1.0 & 0.7 \\ 0.7 & 1.0 \end{bmatrix}^{-1} \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.82 & -0.57 \\ -0.57 & 0.82 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.52 \\ -0.12 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 0.7 \\ 0.7 & 1.0 \end{bmatrix} \begin{bmatrix} \phi_{21} \\ \phi_{22} \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}$$

$$x_3^2 = 0.52x_2 - 0.12x_1$$

One-step ahead prediction

- **Prediction equation:**

  $\Gamma_n \phi_n = \gamma_n$

- **One-step ahead forecast:** $x_{n+1}^n = \phi_{n1} x_n + \phi_{n2} x_{n-1} + \cdots + \phi_{nn} x_1$

  $x_{n+1}^n = \phi_n'(x_n, x_{n-1}, \ldots, x_1)'$

- **Mean square one-step ahead prediction error:**

  $P_{n+1}^n = \gamma(0) - \gamma_n' \Gamma_n^{-1} \gamma_n$

- **Prediction interval:**

  $x_{n+1}^n \pm c_{\alpha/2} \sqrt{P_{n+1}^n}$

- **Mean square one-step ahead prediction error:**

$$P_{n+1}^{n} = \gamma(0) - \gamma_n' \Gamma_n^{-1} \gamma_n$$

- $\gamma(0)$: The variance of the series (uncertainty).
- $\gamma_n' \Gamma_n^{-1} \gamma_n$: The reduction in uncertainty due to the prediction.

The difference between these terms represents the **remaining uncertainty** (the variance of the prediction error).

represents the uncertainty that is **reduced** after using the past observations to make the prediction. This is the amount of variance that is explained by the model.

- **Prediction interval:**

$$x_{n+1}^{n} \pm c_{\alpha/2}\sqrt{P_{n+1}^{n}}$$

is the **prediction standard error**.

is the **quantile** of the normal distribution (e.g., 1.96 corresponds to the 95% confidence interval).

One-step ahead prediction

- **Example: prediction for AR(p)**

$$x_{n+1}^{n} = \phi_1 x_n + \phi_2 x_{n-1} + \cdots + \phi_p x_{n+1-p}$$

$$\begin{bmatrix} \gamma(0) & \gamma(1) & \dots & \gamma(p-1) \\ \gamma(1) & \gamma(0) & \dots & \gamma(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma(p-1) & \gamma(p-2) & \dots & \gamma(0) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_p \end{bmatrix} = \begin{bmatrix} \gamma(1) \\ \gamma(2) \\ \vdots \\ \gamma(p) \end{bmatrix}$$

m-step ahead prediction

Predicting the value after m future time steps.

- **The BLP of x_{n+m}:**

$$x_{n+m}^{n} = \phi_{n1}^{(m)} x_n + \phi_{n2}^{(m)} x_{n-1} + \cdots + \phi_{nn}^{(m)} x_1$$

$$\mathrm{E}\left(\left(x_{n+m} - \sum_{j=1}^{n} \phi_{nj}^{(m)} x_{n+1-j}\right) x_{n+1-k}\right) = 0,\ k = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} \phi_{nj}^{(m)} \gamma(k-j) = \gamma(k+m-1),\ k = 1, 2, \ldots, n$$

m-step ahead prediction

Predicting the value after mmm future time steps.

- **The BLP of x_{n+m}:**

$$x_{n+m}^{n} = \phi_{n1}^{(m)} x_n + \phi_{n2}^{(m)} x_{n-1} + \cdots + \phi_{nn}^{(m)} x_1$$

$$\mathrm{E}\left(\left(x_{n+m} - \sum_{j=1}^{n} \phi_{nj}^{(m)} x_{n+1-j}\right) x_{n+1-k}\right) = 0,\ k = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} \phi_{nj}^{(m)} \gamma(k-j) = \gamma(k+m-1),\ k = 1, 2, \ldots, n$$

One-Step-Ahead Prediction

The formula for one-step prediction is:

$$\hat{x}_{n+1} = \phi_1 x_n + \phi_2 x_{n-1} + \cdots + \phi_p x_{n-p+1}$$

This is the predicted value for the next time point x_{n+1}.

m-Step-Ahead Prediction

1. **Calculate one-step prediction** (already provided above).
2. **Calculate two-step prediction**: Based on the one-step prediction result, continue the prediction as follows:

$$\hat{x}_{n+2} = \phi_1 \hat{x}_{n+1} + \phi_2 x_n + \cdots + \phi_p x_{n-p+2}$$

Here, $\hat{x}_{n+1}$ is the previously calculated predicted value.

3. **Recursively perform m-step prediction**: Continue using the recursive method to predict further future time points. For example, if $m = 3$, you would continue calculating:

$$\hat{x}_{n+3} = \phi_1 \hat{x}_{n+2} + \phi_2 \hat{x}_{n+1} + \cdots + \phi_p x_{n-p+3}$$

Continue in this manner until the m-step prediction is completed.

Backcasting

- **Predicting m steps in the past** Given data $x_1, \ldots, x_n$, predict the past value x_{1-m}.

  Given $x_1, \ldots, x_n$, we wish to predict x_{1-m} for $m > 0$

- **The BLP can be found by solving**

$$\mathrm{E}[(x^n_{1-m} - x_{1-m})x_k] = 0, \; k = 1, 2, \ldots, n$$

$$x^n_{1-m} = \sum_{j=1}^{n} \alpha_j x_j$$

$$\sum_{j=1}^{n} \alpha_j \gamma(k-j) = \gamma(k+m-1), \; k = 1, 2, \ldots, n$$

same as $\phi^{(m)}_{nj}$ in m-step ahead prediction

Backcasting

- **The BLP of $x_{1\text{-}m}$:**

$$\sum_{j=1}^{n} \alpha_j \gamma(k-j) = \gamma(k+m-1),\ k = 1, 2, \ldots, n$$

$$m = 1 : \sum_{j=1}^{n} \alpha_j \gamma(k-j) = \gamma(k),\ k = 1, 2, \ldots, n$$

same as ϕ_{nj} in 1-step ahead prediction

also consistent with PACF definition

Example of Backcasting

Given the following autocovariances:

$$\gamma(0) = 1.0, \quad \gamma(1) = 0.5, \quad \gamma(2) = 0.2, \quad \gamma(3) = 0$$

We want to backcast x_0 using x_1, x_2, x_3. The equation for backcasting is equivalent to the one-step ahead prediction. Let's start by solving for the coefficients $\alpha_1, \alpha_2, \alpha_3$.

The equation:

The equation for backcasting is:

$$\sum_{j=1}^{n} \alpha_j \gamma(k-j) = \gamma(k), \quad k = 1, 2, \ldots, n$$

For this example, $n = 3$ (since we are using three values x_1, x_2, x_3 to predict x_0) and $m = 1$ (one-step backcasting).

Matrix form:

We can represent these equations in matrix form as:

$$\begin{bmatrix} 1.0 & 0.5 & 0.2 \\ 0.5 & 1.0 & 0.5 \\ 0.2 & 0.5 & 1.0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.2 \\ 0 \end{bmatrix}$$

Now, solving this system of linear equations will give us the values of $\alpha_1, \alpha_2, \alpha_3$.

Solution:

Solving this matrix equation will provide the coefficients $\alpha_1, \alpha_2, \alpha_3$ that we use to backcast x_0.

The core problem in time series forecasting can be reduced to solving a system of linear equations:

$$\Gamma_n \phi_n = \gamma_n$$

Where:

- Γ_n: A matrix formed from the autocovariances of the sequence (size $n \times n$).
- γ_n: A vector formed from the autocovariances of the sequence.
- ϕ_n: The vector of prediction coefficients that we need to solve for.

The solution ϕ_n represents the "optimal weights" used to predict the next value based on the past n values.

Core Method: Utilizing the "Recursive Structure"

Directly solving the system of equations requires a large amount of computation.

Durbin-Levinson algorithm with a key recursive structure.

The Durbin–Levinson algorithm

- **Computes x_{n+1}^n and P_{n+1}^n recursively as**

$$\phi_{00} = 0, \quad P_1^0 = \gamma(0)$$

Initialization

initialize the prediction error variance and coefficient

The Durbin–Levinson algorithm

- **Computes x_{n+1}^n and P_{n+1}^n recursively as**

$$\phi_{00} = 0, \quad P_1^0 = \gamma(0)$$

To update the prediction error variance and prediction coefficients using recursive relationships. This process allows us to avoid directly solving large-scale linear equations.

For $n \geq 1$:

$$\phi_{nn} = \frac{\rho(n) - \sum_{k=1}^{n-1} \phi_{n-1,k}\rho(n-k)}{1 - \sum_{k=1}^{n-1} \phi_{n-1,k}\rho(k)}, \; P_{n+1}^n = P_n^{n-1}(1 - \phi_{nn}^2)$$

For $n \geq 2$:

$$\phi_{nk} = \phi_{n-1,k} - \phi_{nn}\phi_{n-1,n-k}, \; k = 1, 2, \ldots, n-1$$

Objective: Use the first 2 observations (x_1, x_2) to predict the 3rd value x_3.

Given:

- $\gamma(0) = 2, \quad \gamma(1) = 1, \quad \gamma(2) = 0.5$

Autocorrelation coefficients:

- $\rho(0) = 1, \quad \rho(1) = 0.5, \quad \rho(2) = 0.25$

Formulate the Matrix Equation

Optimal linear prediction:

$$x_3^2 = \phi_{21} x_2 + \phi_{22} x_1$$

According to the orthogonality principle, the coefficients satisfy the Yule-Walker equation:

$$\Gamma_2 \phi_2 = \gamma_2$$

Where:

- Γ_2 is the covariance matrix of x_1, x_2
- $\phi_2 = (\phi_{21}, \phi_{22})^T$ is the vector of prediction coefficients
- γ_2 is the covariance vector between x_3 and x_1, x_2

Written explicitly:

$$\Gamma_2 = \begin{pmatrix} \gamma(0) & \gamma(1) \\ \gamma(1) & \gamma(0) \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\gamma_2 = \begin{pmatrix} \gamma(1) \\ \gamma(2) \end{pmatrix} = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

The equation becomes:

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \phi_{21} \\ \phi_{22} \end{pmatrix} = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$$

The Durbin–Levinson algorithm

Step 1: Initialization (n=0)

$$\phi_{00} = 0, \quad P_1^0 = \gamma(0) = 2$$

Step 2: Case n=1

Calculate ϕ_{11}:

$$\phi_{11} = \frac{\rho(1) - \sum_{k=1}^{0} \phi_{0,k}\rho(1-k)}{1 - \sum_{k=1}^{0} \phi_{0,k}\rho(k)} = \frac{\rho(1)}{1} = 0.5$$

Update prediction error:

$$P_2^1 = P_1^0(1 - \phi_{11}^2) = 2 \times (1 - 0.5^2) = 2 \times 0.75 = 1.5$$

The Durbin–Levinson algorithm

At this point we have:

- Prediction coefficient based on 1 observation: $\phi_{11} = 0.5$
- Prediction error variance: $P_2^1 = 1.5$

Step 3: Case n=2

Calculate ϕ_{22}:

$$\phi_{22} = \frac{\rho(2) - \sum_{k=1}^{1} \phi_{1,k}\rho(2-k)}{1 - \sum_{k=1}^{1} \phi_{1,k}\rho(k)}$$

Where $\phi_{1,1} = 0.5$, so:

- Numerator: $\rho(2) - \phi_{1,1}\rho(1) = 0.25 - 0.5 \times 0.5 = 0.25 - 0.25 = 0$
- Denominator: $1 - \phi_{1,1}\rho(1) = 1 - 0.5 \times 0.5 = 0.75$

$$\phi_{22} = \frac{0}{0.75} = 0$$

The Durbin–Levinson algorithm

Update other coefficients (ϕ_{21}):

$$\phi_{21} = \phi_{1,1} - \phi_{22} \cdot \phi_{1,1} = 0.5 - 0 \times 0.5 = 0.5$$

Update prediction error:

$$P_3^2 = P_2^1(1 - \phi_{22}^2) = 1.5 \times (1 - 0^2) = 1.5$$

Final Result

The optimal linear prediction for the 3rd value based on the first 2 observations is:

$$x_3^2 = \phi_{21}x_2 + \phi_{22}x_1 = 0.5x_2 + 0 \times x_1 = 0.5x_2$$

The prediction error variance is:

$$P_3^2 = 1.5$$

Result Analysis

1. $\phi_{22} = 0$ means that given x_2, x_1 provides no additional information for predicting x_3
2. The prediction depends only on the most recent observation x_2
3. The lack of improvement from $P_2^1 = 1.5$ to $P_3^2 = 1.5$ confirms that adding x_1 does not enhance prediction accuracy

Data: Temperature of the past 10 days [15, 16, 14, 17, 18, 20, 19, 21, 22, 23]
What the Durbin-Levinson algorithm does:
•Try AR(1): Tomorrow's temperature = 0.8 × today's temperature
•Try AR(2): Tomorrow's temperature = 0.6 × today's temperature + 0.3 × yesterday's temperature
•Try AR(3): Tomorrow's temperature = 0.5 × today's temperature + 0.3 × yesterday's temperature + 0.1 × the temperature from two days ago
•...
After testing, it is found that the AR(2) model gives the smallest prediction error → Choose the 2nd-order linear model.

$$P_{n+1}^n = P_n^{n-1} \times (1 - \phi_{nn}^2)$$

The Durbin–Levinson algorithm

- **Verify:**

Define $\tilde{\phi}_n = (\phi_{nn}, \ldots, \phi_{n1})'$ and $\tilde{\gamma}_n = (\gamma(n), \ldots, \gamma 1)'$

$$\begin{aligned}
\Gamma_n \phi_n &= \begin{pmatrix} \Gamma_{n-1} & \tilde{\gamma}_{n-1} \\ \tilde{\gamma}'_{n-1} & \gamma(0) \end{pmatrix} \begin{pmatrix} \phi_{n-1} - \phi_{nn}\tilde{\phi}_{n-1} \\ \phi_{nn} \end{pmatrix} \\
&= \begin{pmatrix} \gamma_{n-1} \\ \tilde{\gamma}'_{n-1}\phi_{n-1} + \phi_{nn}(\gamma(0) - \gamma'_{n-1}\phi_{n-1}) \end{pmatrix} \\
&= \gamma_n
\end{aligned}$$

The Durbin–Levinson algorithm

- **Verify:**

$$
\begin{aligned}
P_{n+1}^{n} &= \gamma(0) - \phi_n' \gamma_n \\
&= \gamma(0) - \begin{pmatrix} \phi_{n-1} - \phi_{nn}\tilde{\phi}_{n-1} \\ \phi_{nn} \end{pmatrix}' \begin{pmatrix} \gamma_{n-1} \\ \gamma(n) \end{pmatrix} \\
&= P_n^{n-1} - \phi_{nn}(\gamma(n) - \tilde{\phi}_{n-1}' \gamma_{n-1}) \\
&= P_n^{n-1} - \phi_{nn}^2(\gamma(0) - \phi_{n-1}' \gamma_{n-1}) \\
&= P_n^{n-1}(1 - \phi_{nn}^2)
\end{aligned}
$$

$$
P_{n+1}^{n} = \gamma(0)\Pi_{j=1}^{n}[1 - \phi_{jj}^2]
$$

Direct prediction using raw data:

$$x_{n+1}^{(n)} = \phi_{n1}x_n + \phi_{n2}x_{n-1} + \cdots + \phi_{nn}x_1$$

Problem: Data points may be correlated, which increases computation complexity.

Innovation method:

Use prediction errors to predict:

$$x_{n+1}^{(n)} = \theta_{n1}(x_n - x_n^{(n-1)}) + \theta_{n2}(x_{n-1} - x_{n-1}^{(n-2)}) + \cdots + \theta_{nn}(x_1 - x_1^{(0)})$$

The innovations algorithm

- **Instead of writing the BLP as**

$$x_{n+1}^{n} = \phi_{n1} x_n + \phi_{n2} x_{n-1} + \cdots + \phi_{nn} x_1$$

- **we can write it in terms of the innovations as**

$$x_{n+1}^{n} = \theta_{n1}(x_n - x_n^{n-1}) + \theta_{n2}(x_{n-1} - x_{n-1}^{n-2}) + \cdots + \theta_{nn}(x_1 - x_1^{0})$$

innovation

Still linear in $x_1, \ldots, x_n$

- **The innovations are uncorrelated as**

$$\mathrm{Cov}(x_j - x_j^{j-1}, x_i - x_i^{i-1}) = 0 \text{ for } i \neq j$$

Uncorrelation: This means that each innovation contains entirely new and independent information.

The core idea of traditional methods

Goal: To find a set of optimal weights (φ, θ, α) that minimize the prediction error.

Method: Determine these weights by solving equations (utilizing the covariance γ).

Can we just browse historical data (read the time series)

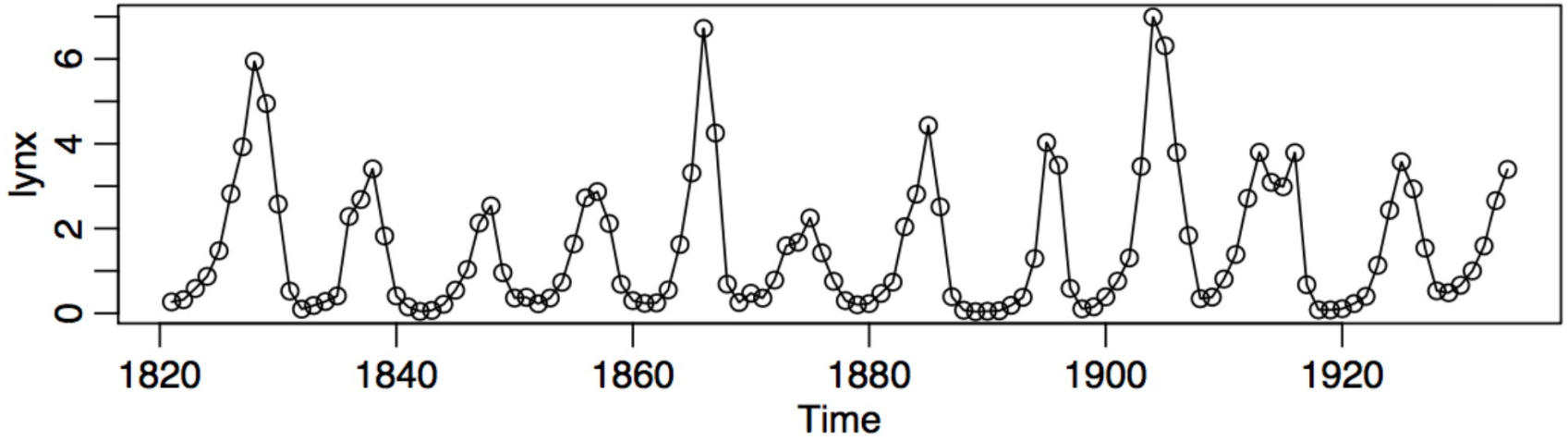
Form its own understanding and memory (internal hidden state)

Make predictions based on the complete understanding and experience (output the predicted value)

Or solving non-linear problems.

- **Example:**

  Annual numbers (÷1000) of lynx trappings for 1821–1934 in Canada

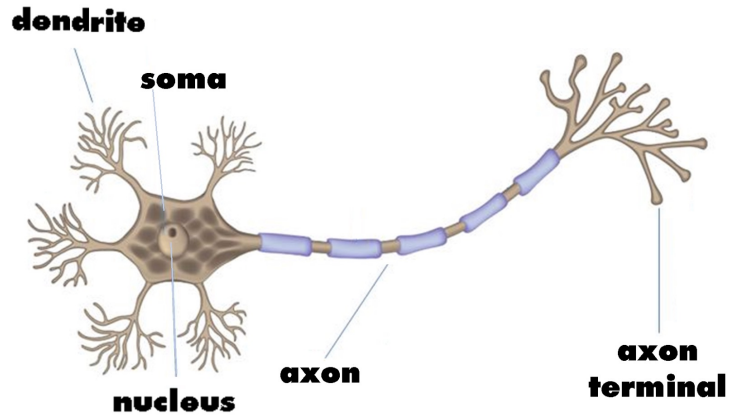


  - **A family of neural networks for handling sequential data**

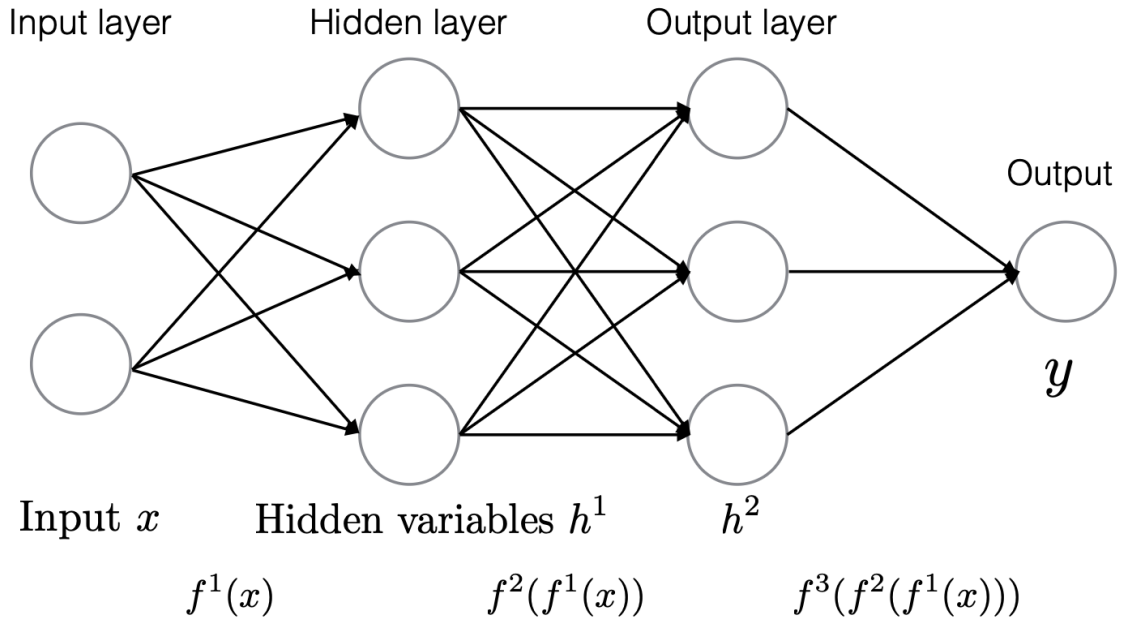
https://www.geeksforgeeks.org/machine-learning/time-series-forecasting-using-recurrent-neural-networks-rnn-in-tensorflow/

Motivation

- **Inspired by the way the human brain works**



- **A three-layer neural network**



Feedforward neural networks

- **Information flows from the input x through some intermediate steps, all the way to the output y**

- **There is no feedback connections.**

Input

- **Represent a data sample as a vector**

  Convert word or image to a vector

- **Preprocessing are required sometimes**

  For finance or econometric data, sometimes need to pre-calculate the returns

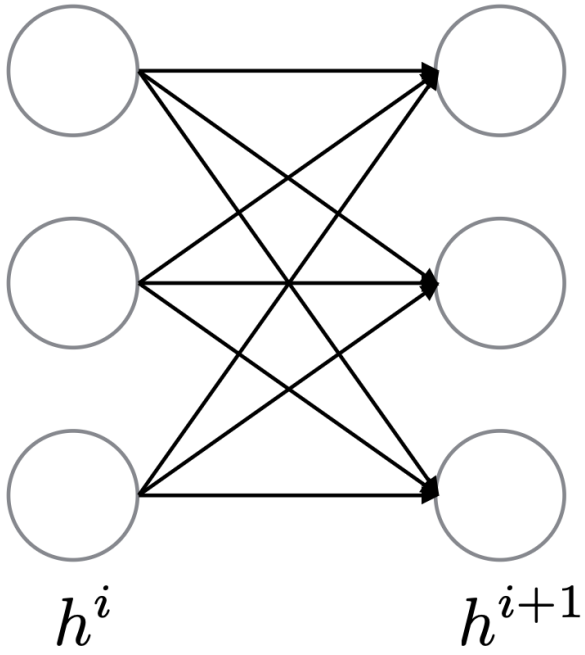
Hidden units

- **Hidden units are what makes deep learning unique**
- **Often start with a linear transformation**

$$z = W^T h + b$$

- **Followed by an element wise, nonlinear function** $g(z)$

$g(z)$: activation function



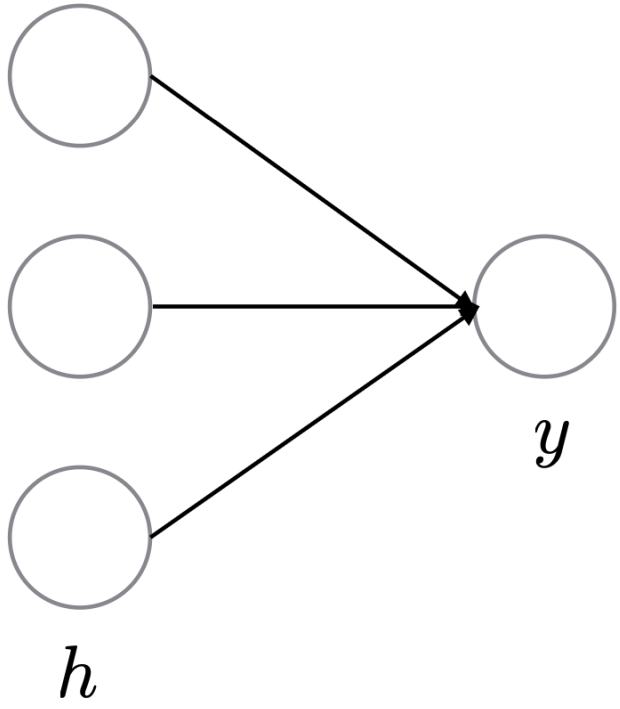
Output layer

- **Regression**

$$y = W^T h + b$$

y : 1-dimensional
or multi-dimensional

Linear units: no nonlinearity



Output layer

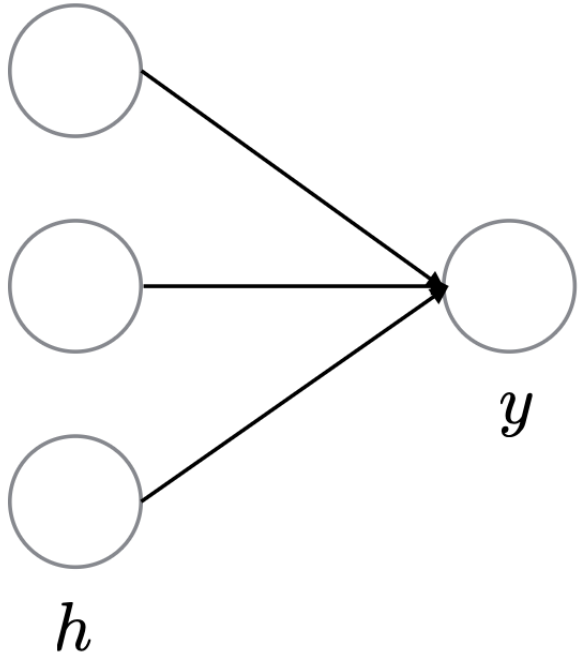
- **Binary classification**

$$y = \sigma(W^T h + b)$$

σ : logistic sigmoind function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid output unit

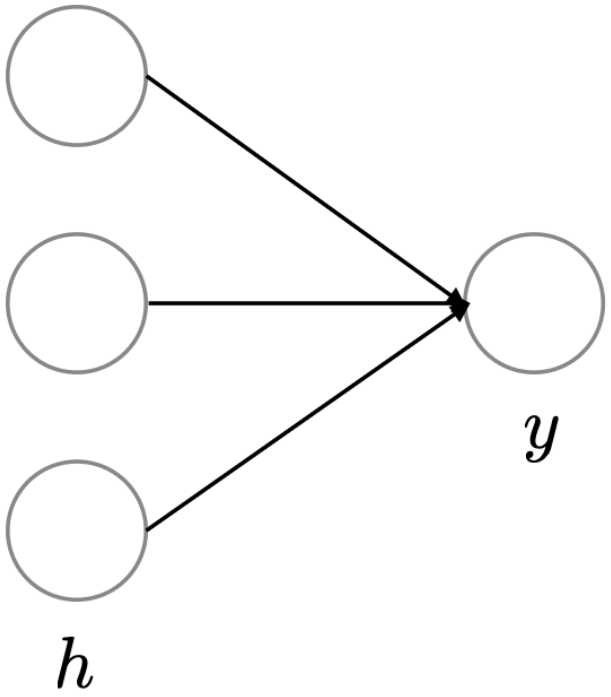


Output layer

- **Multi-class classification**

$$y = \text{softmax}(W^T h + b)$$

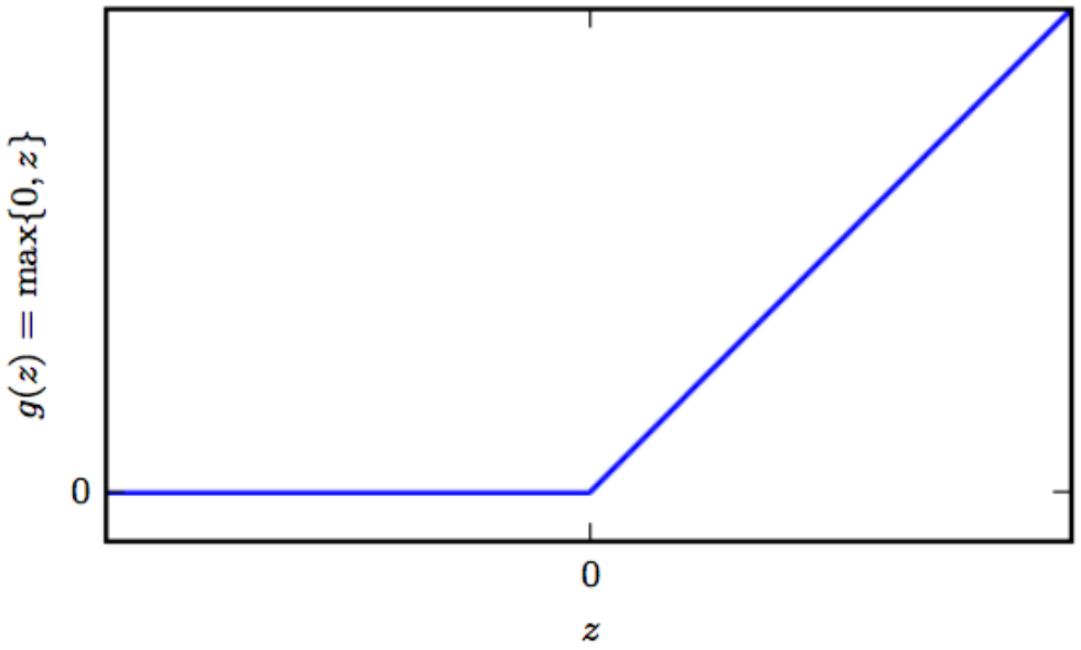
$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



The Rectified Linear Units: ReLU

- **Activation function**

$$g(z) = \max\{0, z\}$$



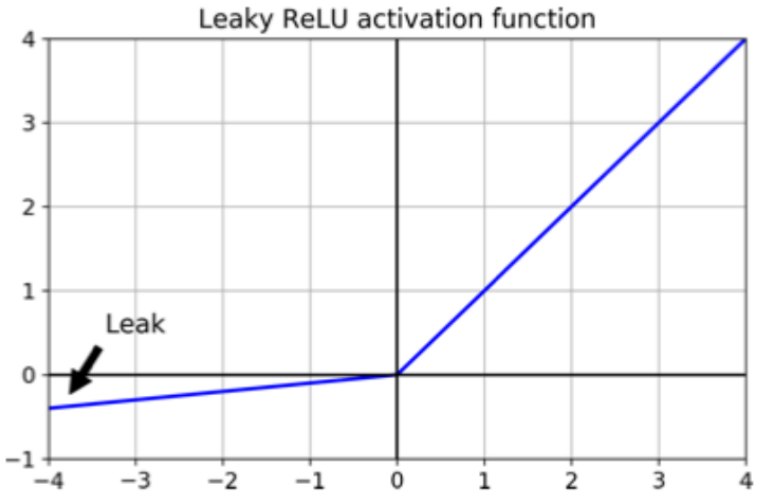
Generalizations of ReLU

- **Activation function**

$$\mathrm{gReLU}(z) = \max\{0, z\} + \alpha \min\{0, z\}$$

Leaky ReLU: $\alpha = 0.01$

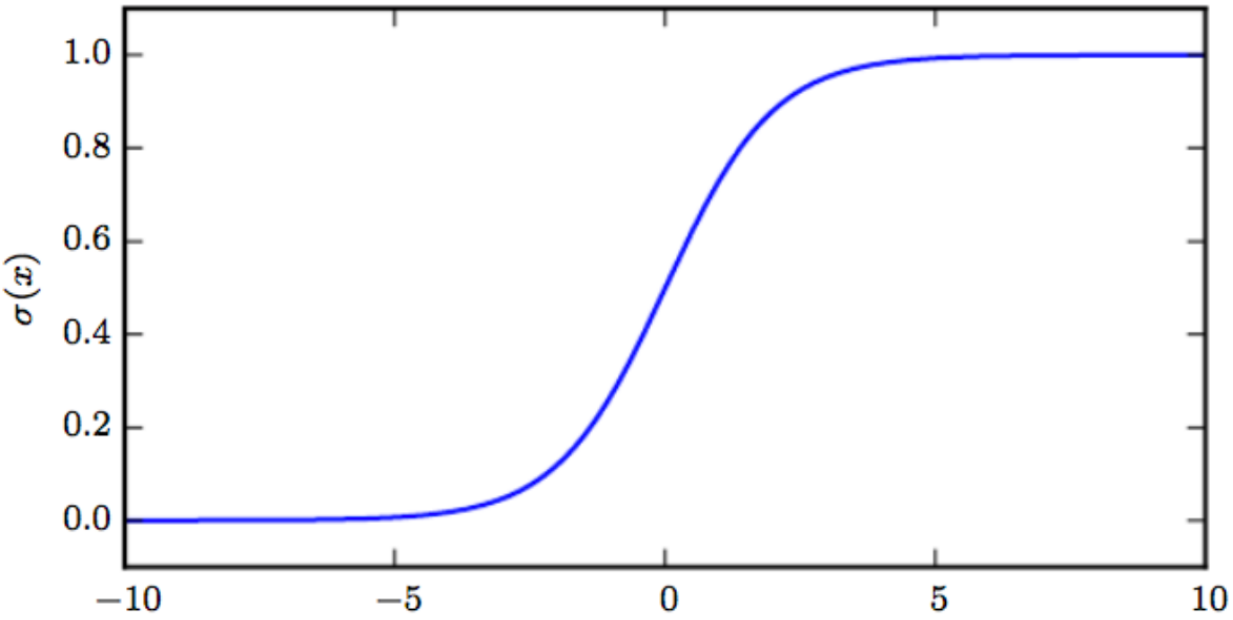
Parametric ReLU: α learnable



Sigmoid Units

- **Activation function**

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

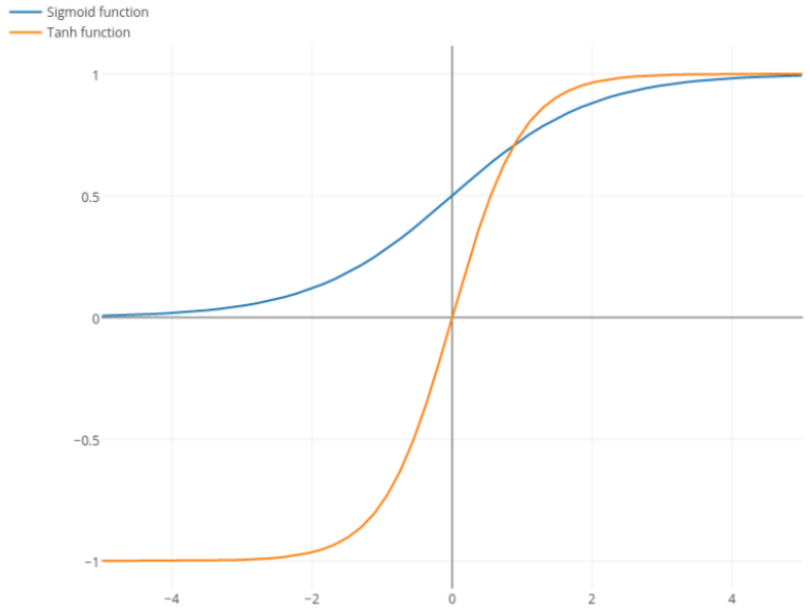


Tanh Units

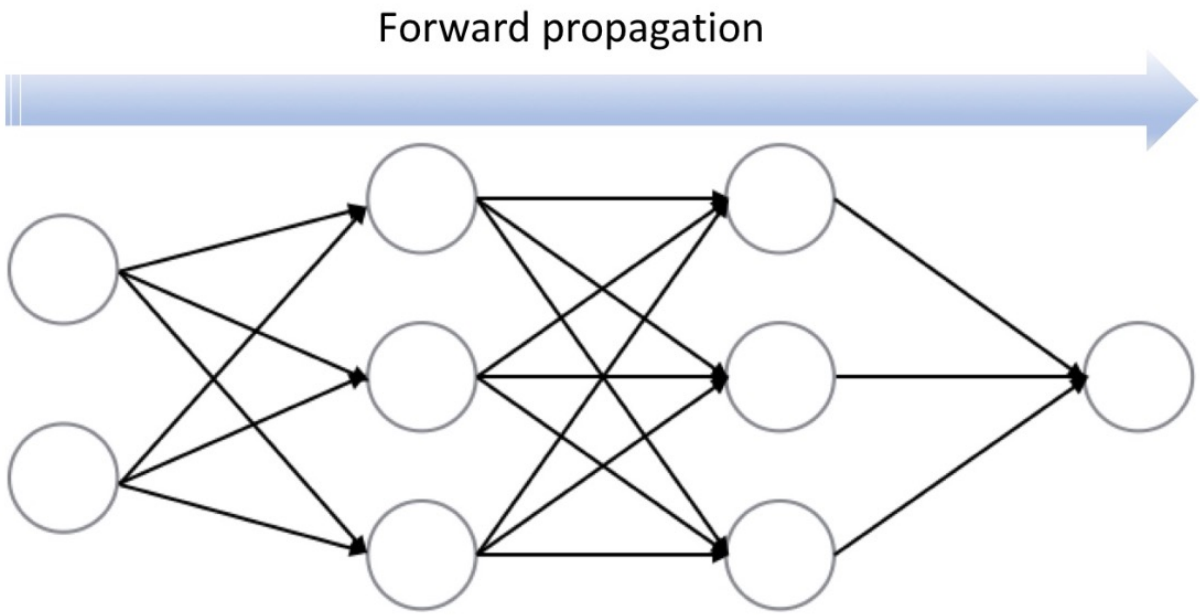
- **Activation function**

$$g(z) = \tanh(z) = 2\sigma(2z) - 1$$

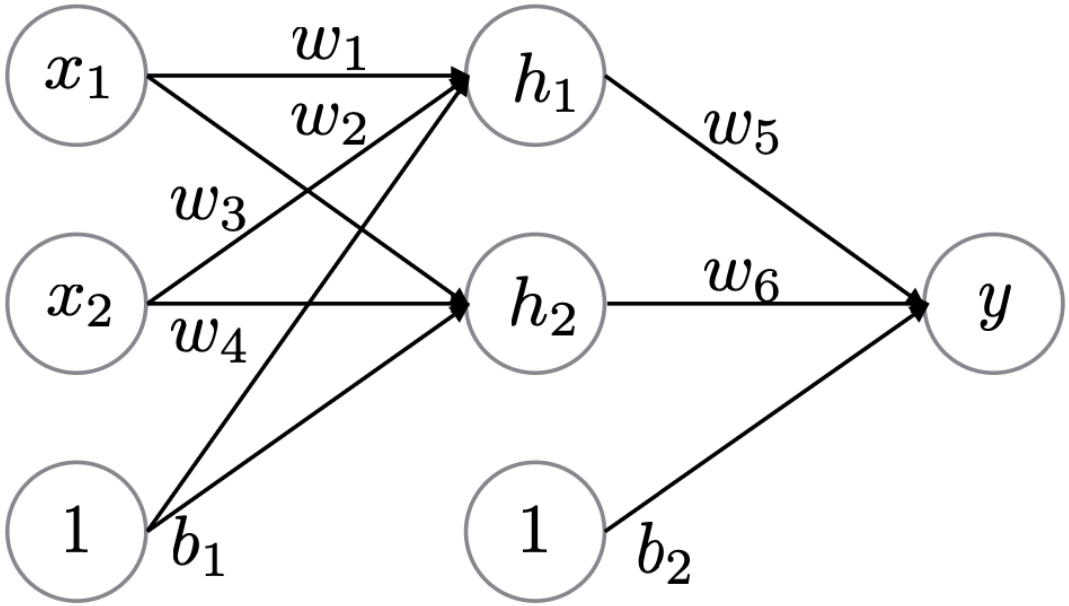
- **Zero-centered, but still saturate**



- **When we use the a feedforward neural network to accept an input and produce an output, information flows from the input to the cost function.**



- **Example:**



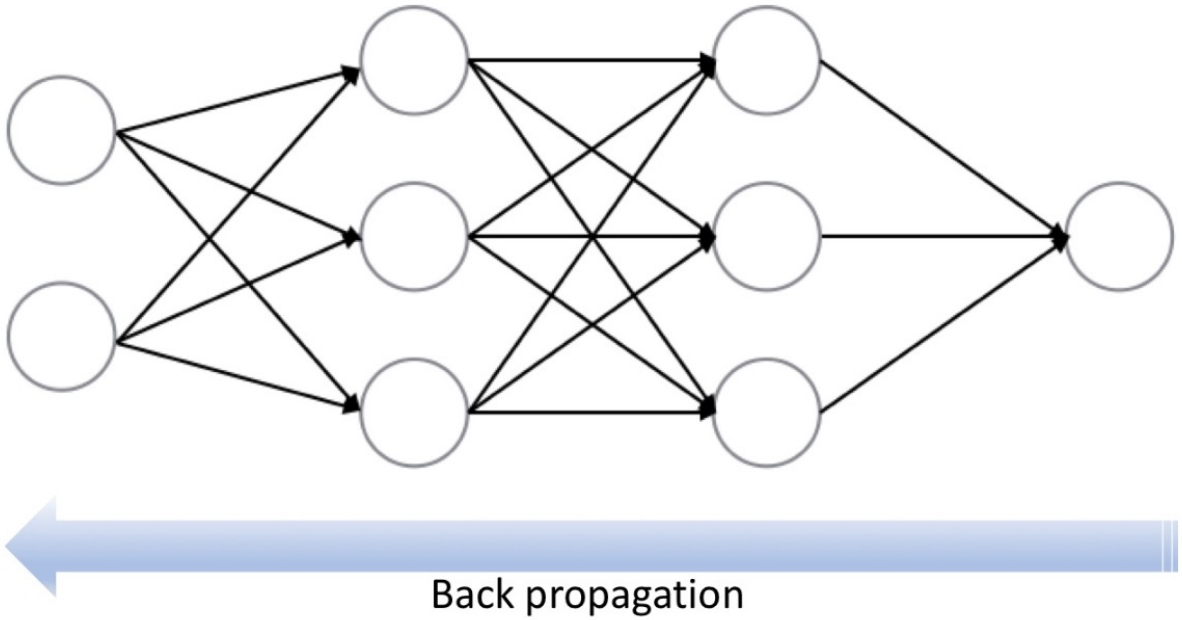
$$\delta_1 = x_1 \cdot w_1 + x_2 \cdot w_2 + b_1$$
$$h_1 = \sigma(\delta_1)$$
$$\delta_2 = x_1 \cdot w_3 + x_2 \cdot w_4 + b_1$$
$$h_2 = \sigma(\delta_2)$$

$$\delta_3 = h_1 \cdot w_5 + h_2 \cdot w_6 + b_2$$
$$y = \sigma(\delta_3)$$

$$J = \frac{1}{2}(y_{true} - y)^2$$

Back propagation

- **Back Propagation allows us to propagate information from the cost function through the parameters, which can be used to compute the gradient.**



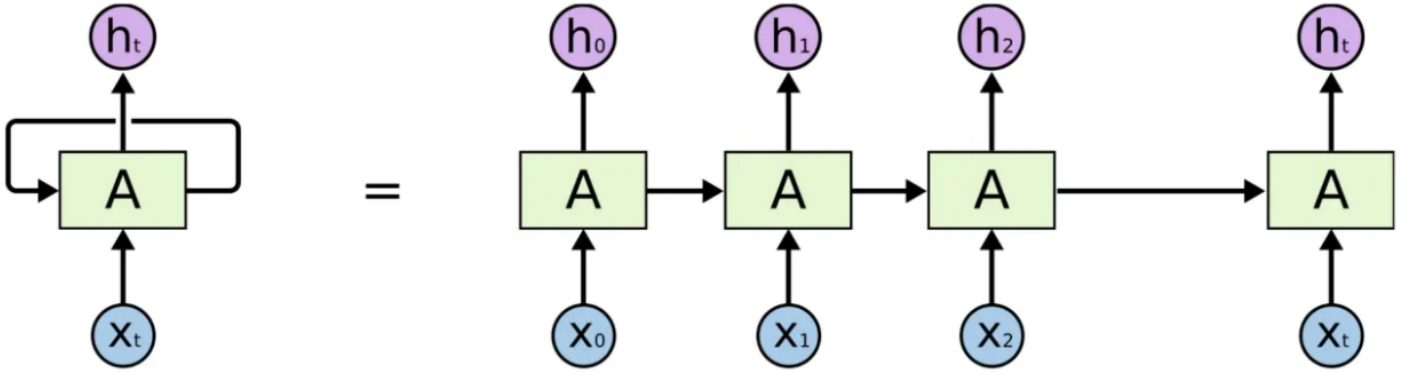
Cost function

- **The cost function is a measure of how well the algorithm performs.**
- **Training is performed through minimizing the cost function.**
- **The cost functions for neural networks are more or less the same as those for other parametric models, such as linear models.**

- **A family of neural networks for handling sequential data**
- **Especially for natural language processing**
- **RNNs have the concept of ‘memory’ that helps store the states or information of previous inputs to generate the next output of the sequence.**

Issues in the feed forward neural network : -

1. Can’t handle sequential data.
2. Consider only current input.
3. Can’t memorize the previous input.



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

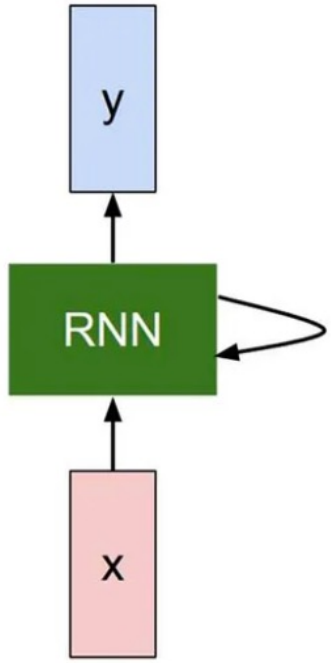
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

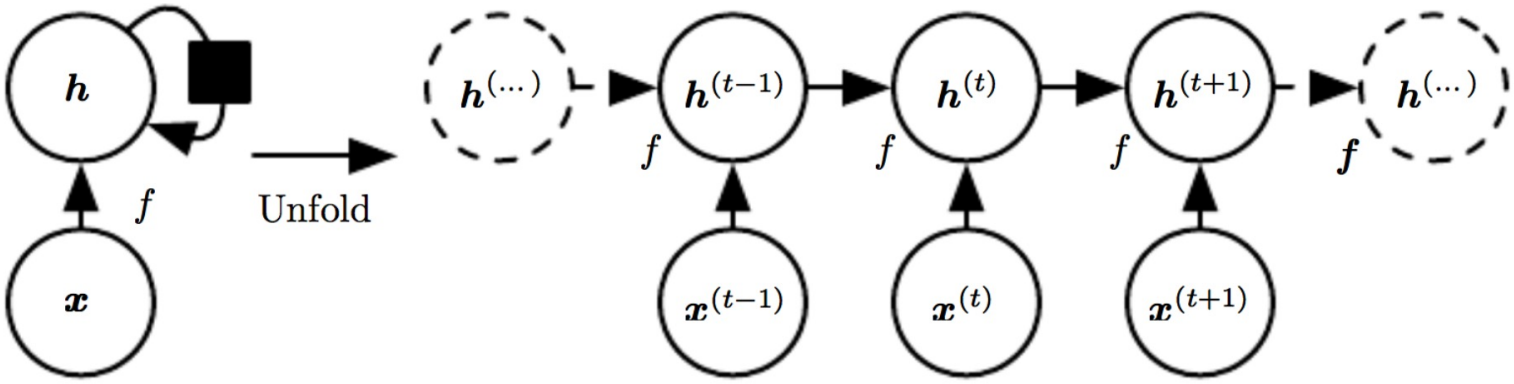
input vector at some time step



Recurrent neural networks

- **Use the same function and parameters across different time steps of the sequence**
- **At each time step, use the input and the previous hidden state to calculate the hidden state and the output**

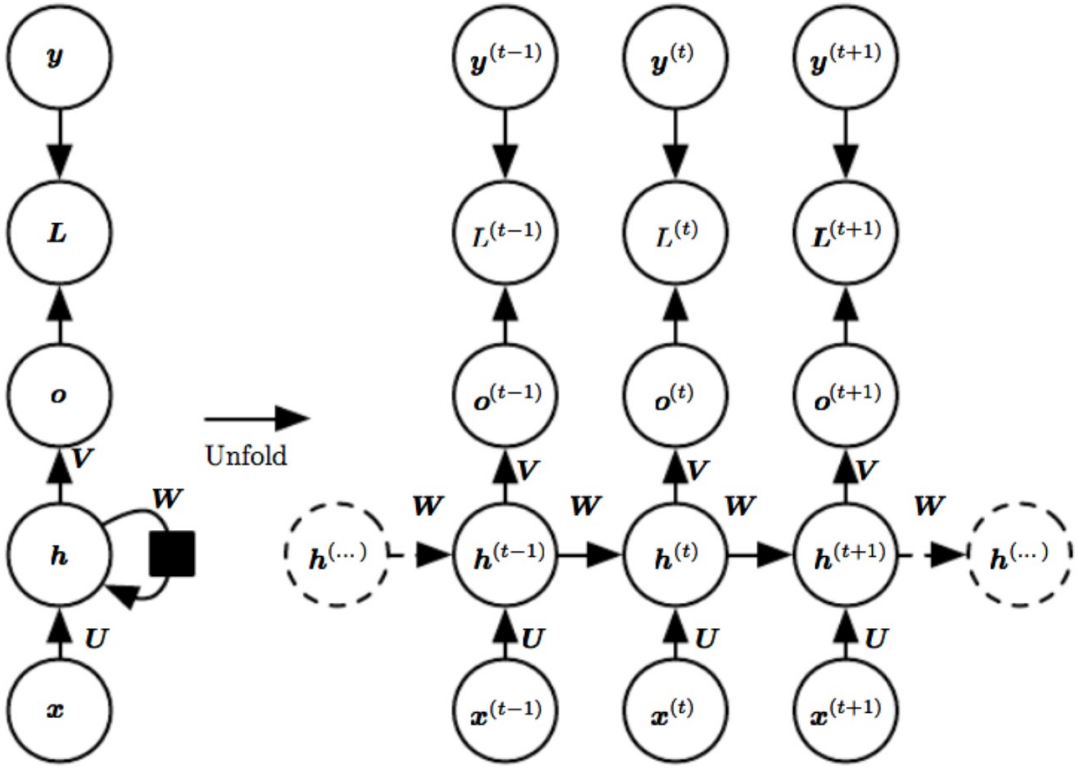
$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$



Recurrent neural networks

- **Computational graph**

  Maps an input sequence of x values to a corresponding sequence of output o values



Recurrent neural networks

- **Training**

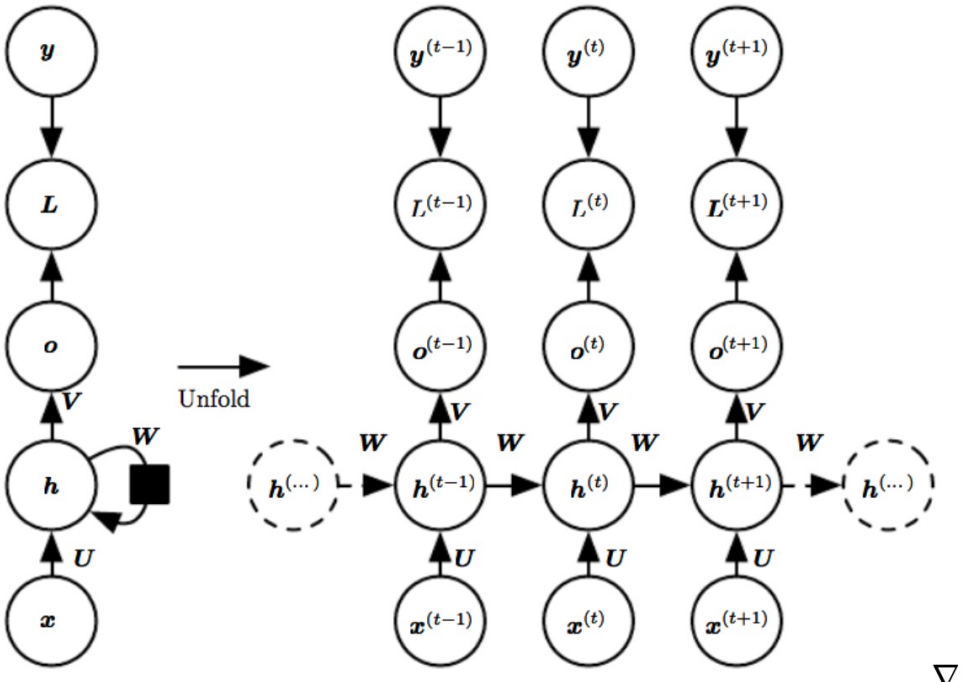
  Unfold the computational graph

  Use back propagation through time to compute the gradient

  Apply gradient-based optimization techniques

Recurrent neural networks

- **Back propagation through time**



$$\frac{\partial L}{\partial L^{(t)}}$$

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{o}^{(t)}}$$

$$\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \qquad \frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}$$

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^T \nabla_{\boldsymbol{o}^{(t)}} + \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^T \nabla_{\boldsymbol{h}^{(t+1)}} L$$

Recurrent neural networks

- **Vanishing gradients**

  Multiply many small numbers together

  Errors due to further back time steps have smaller and smaller gradients

  The gradient of a long-term interaction has exponentially smaller magnitude than the gradient of a short-term interaction

  Bias parameters to capture short-term dependencies

Deal with long-term dependencies

- **Long short-term memory networks (LSTMs)**

  Useful for analyzing time series, since there can be lags of unknown duration between important events in a time series.

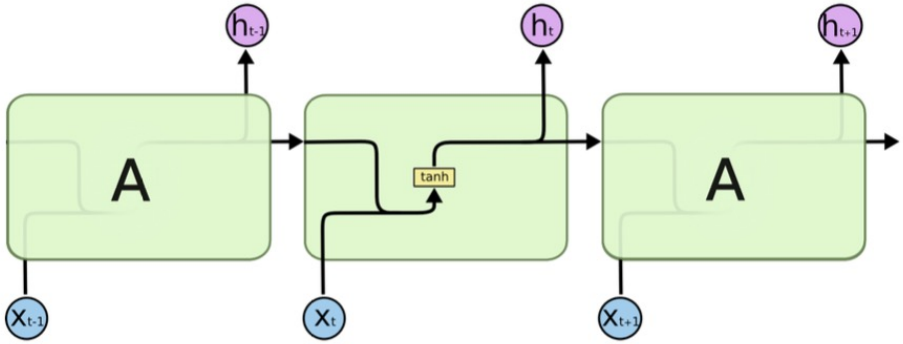
  In addition to the hidden states, the LSTMs introduce cell states to give the model longer memory of the past events

  LSTMs use gates to control the through flow of the information

LSTMs

- **Repeating recurrent unit**

RNN:



LSTM: