CST8506 – Advanced Machine Learning

Week 4: Recurrent Neural Networks (RNNs)

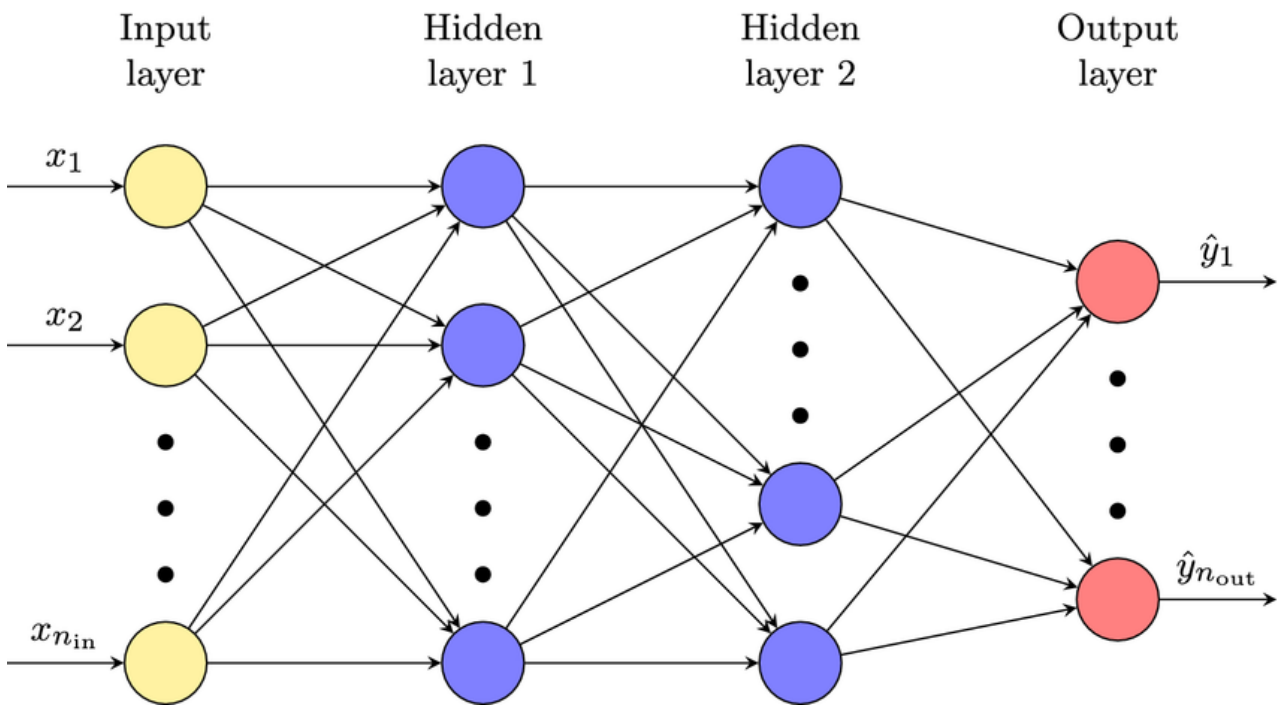
Dr. Abbas Akkasi

Winter 2025

- FNN– Review
- Motivation
- Usages of Sequential Data
- Time Series
- Time Series – Components
- Recurrent Neural Networks (RNNs)
- Backpropagation Refresher
- Backpropagation Through Time (BTT)
- Vanishing Gradient Problem
- Long-Short Term Memory

Review on Feed Forward Network

- Information flows only in the **forward direction**. No **cycles or Loops**.
- Decisions are based on current input, **no memory** about the past
- Doesn't know how to handle **sequential data**



Motivation

Questions:

- How Google's autocomplete feature predicts the next word when a user is typing?
- How Translators converting sentences from English to French?
- How Siri or Google Assistant converting spoken words into text?
- How AI composes melodies or generates background music?
- How it is possible to predict the future prices based on historical trends?
- Etc.

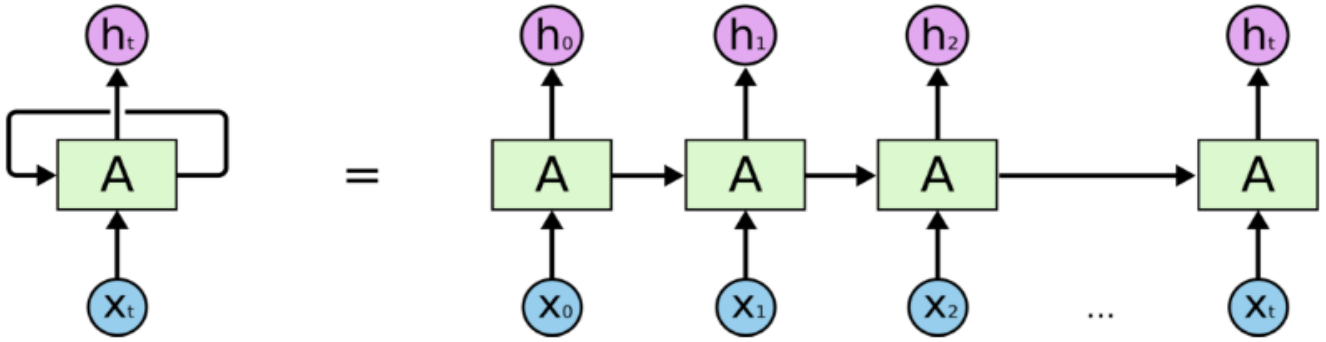
Motivation …

We need a model:

- To handle sequential data.
- Able to consider the **current input** also the **previously received** inputs.
- Able to memorize history in its internal memory.

FFNs cannot process the sequential data!

What is the solution?



Usages of Sequence Data - Examples

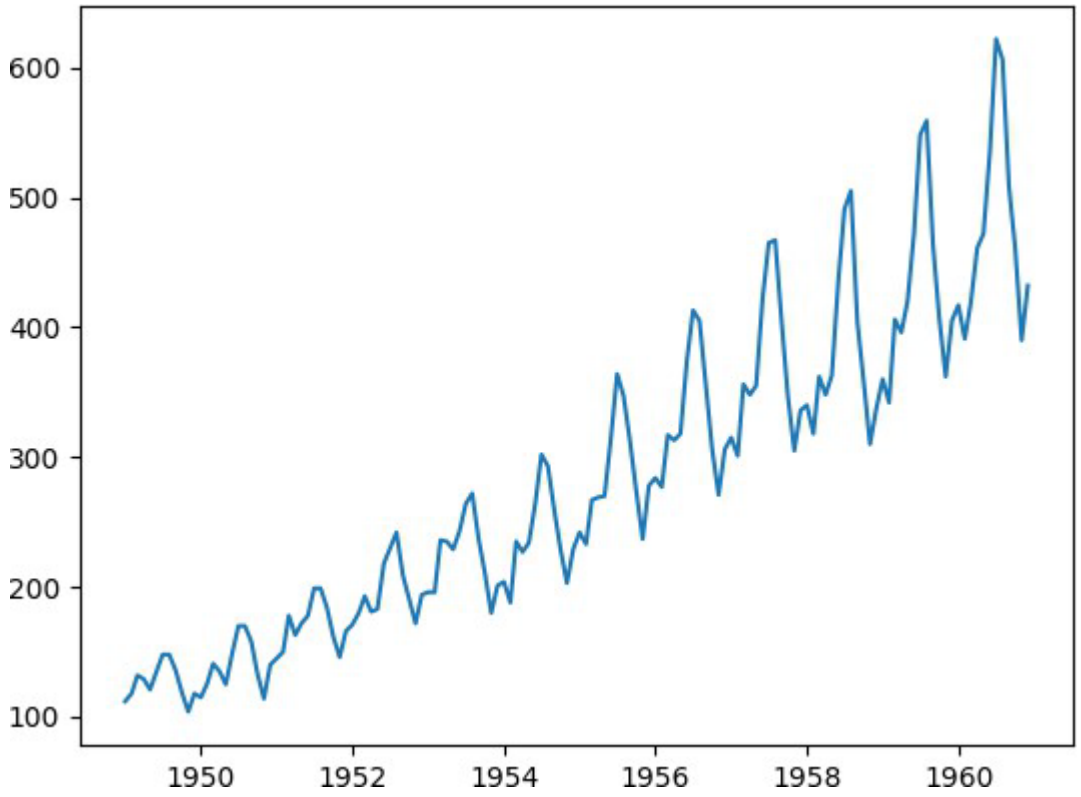
- Speech recognition (audio clip to text)
- Sentiment analysis (sequence of text to number of stars)
- DNA Sequence analysis
- Machine translation (sequence of text in one language translated to another)
- Video activity recognition (detect the activity from a sequence of video frames)
- **Time Series Forecasting**

Time Series

- A **Time Series** is a sequence of data points collected or recorded at specific time intervals.
- Unlike standard "cross-sectional" data (where you look at a snapshot of many things at once), time series focuses on **one(more) thing over a duration.**
  - **The X-Axis:** Almost always represents time (seconds, days, years).
  - **The Y-Axis:** The variable you are measuring (Price, Temperature, Population).
  - **The Goal:** To understand the past and, ideally, peer into the future (Forecasting).

Time Series - example

- Air Passengers
- Non-stationary data
  - Mean & sd changes with time
- Seasonal data
- Data from Jan 1949-Dec 1960



Taken from: https://www.kaggle.com/datasets/rakannimer/air-passengers

Time Series – Components

1. **Trend:** The long-term "direction." Is it generally going up, down, or staying flat?
2. **Seasonal :** Patterns that repeat over a fixed period (e.g., retail sales spiking every December).
3. **Cycle:** A cycle is a long-term fluctuation in a time series that repeats, but NOT at a fixed, regular interval.
4. **Noise (Residuals):** The random "hiccups" in the data that can't be explained by the other three.

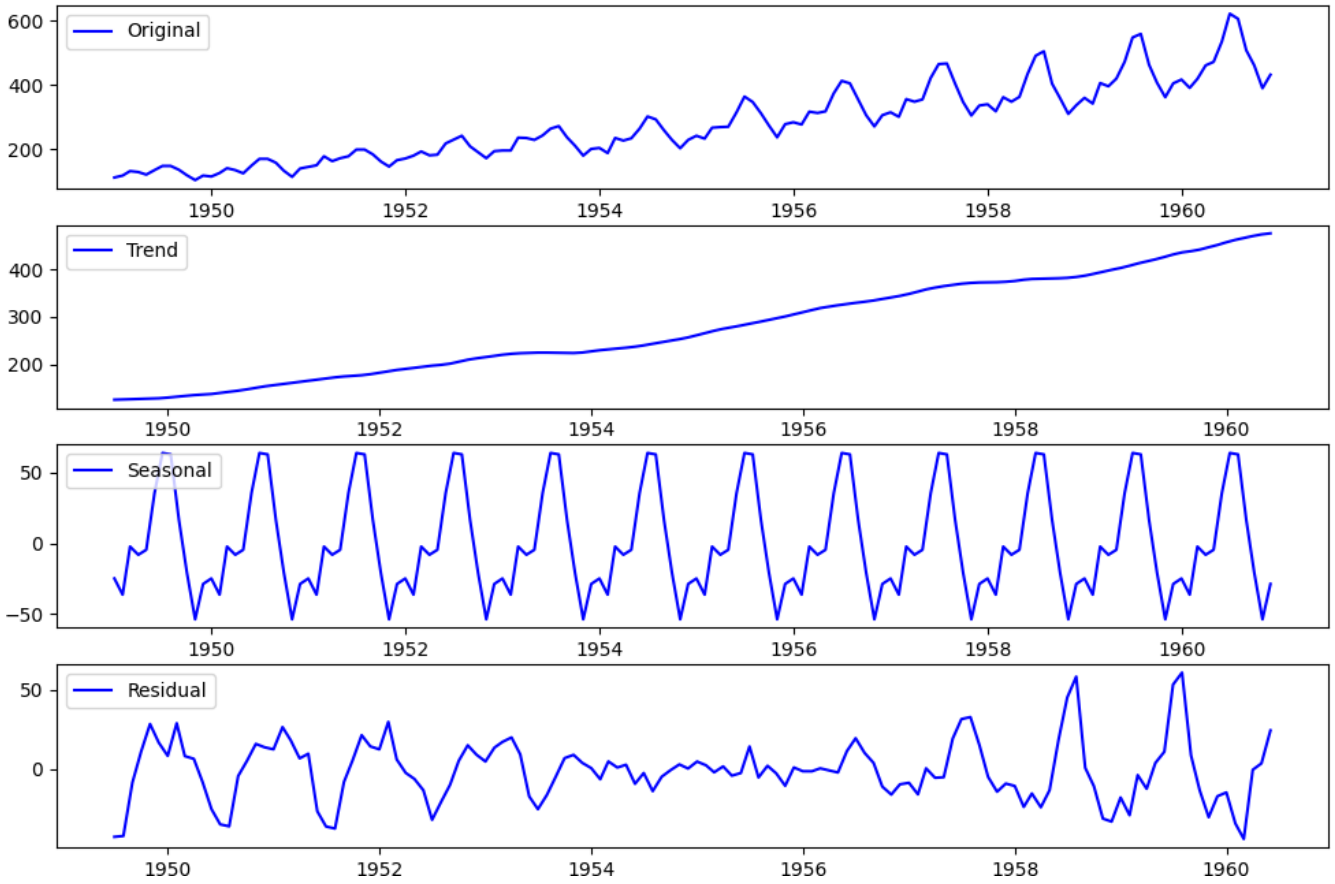
Time Series – Components

Original plot

Trend

Seasonal

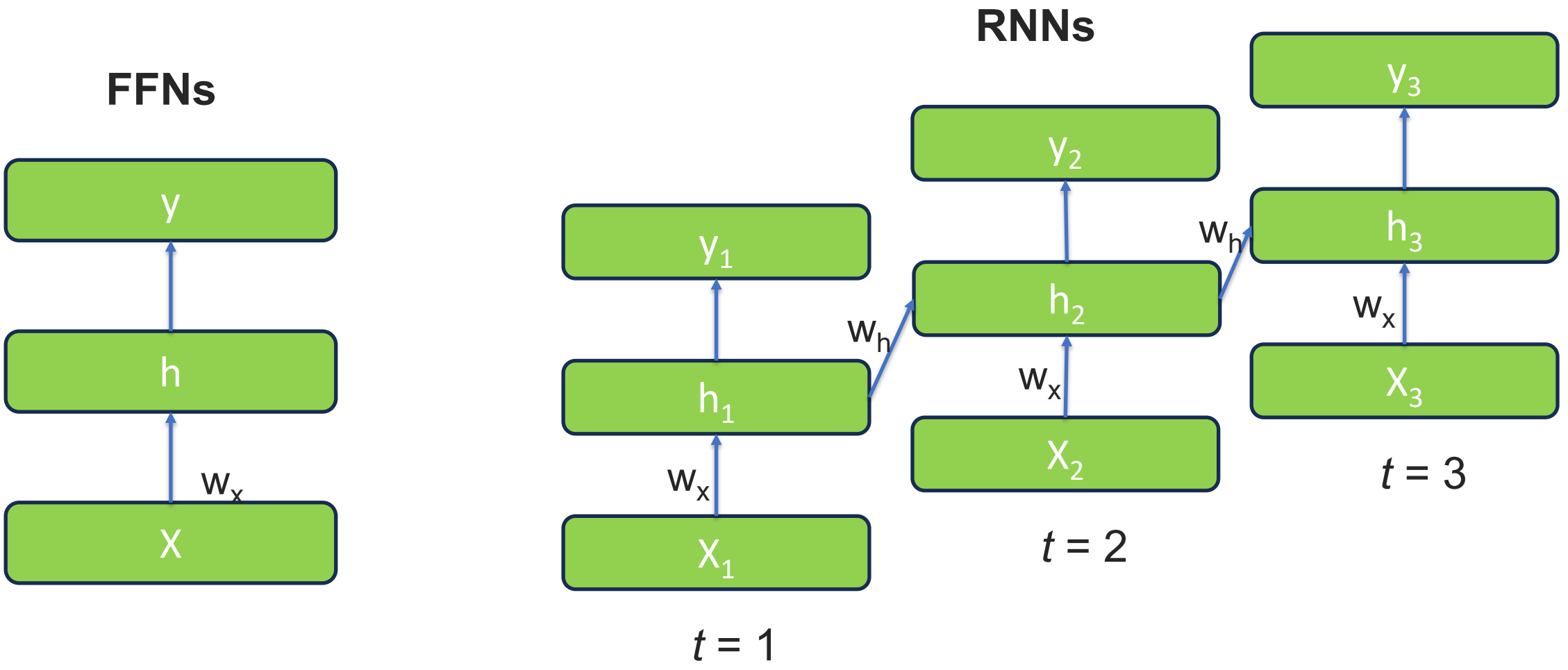
Residual



Recurrent Neural Networks (RNNs)

- RNNs are kind of DL models that takes the **previous output or hidden states** as inputs. i.e. the **composite input** at time *t* has some historical information about the happenings at time T < *t*.
- RNNs are useful as their **intermediate states** can store information about **past inputs** for a time that is not fixed.
- In RNNs, each input vector (e.g. word vector) is typically fed into the network **one at a time**, not all at once.

RNNs ...



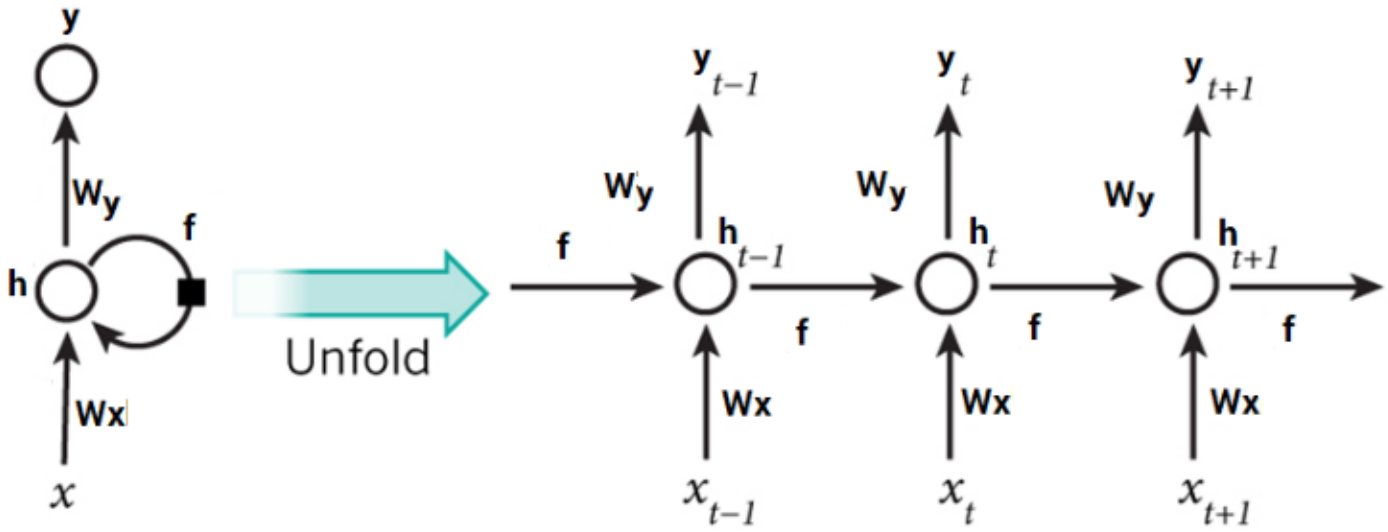
RNNs …

$h_t = f(W_x\, x_t + W_y\, h_{t-1})$

h_t : *hidden state at time step t*

x_t : *input at time step t*

W_x *and* W_y : *Weight matrices. Filters that determine the importance of the present input and past information.*



- Note that the weights are **shared over time**

  copies

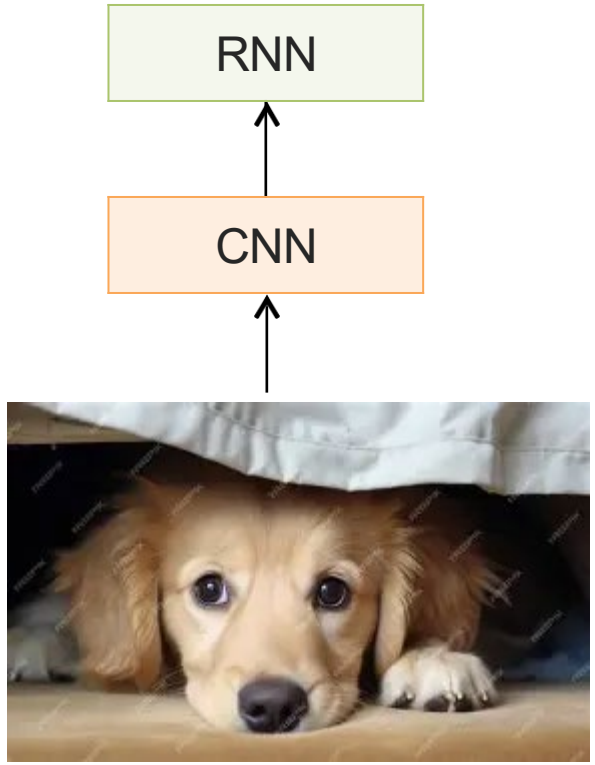
Example (Image captioning)

Problem: Given an image, produce a sentence describing its contents

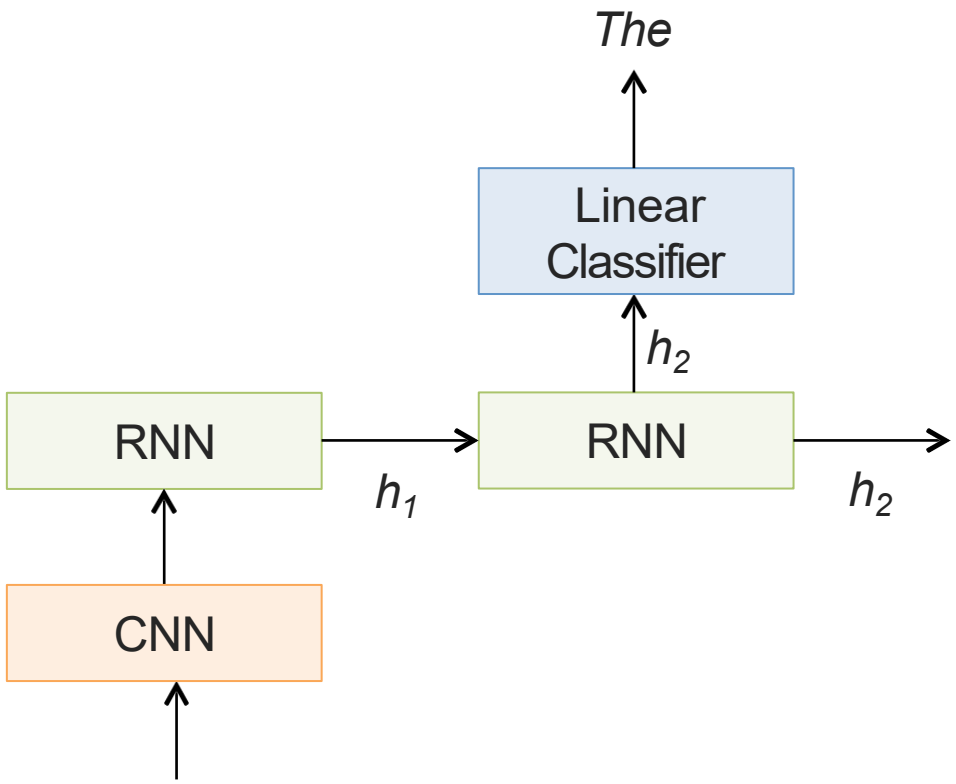
- **Inputs:** Image feature (from a CNN)
- **Outputs:** Multiple words

The dog is hiding

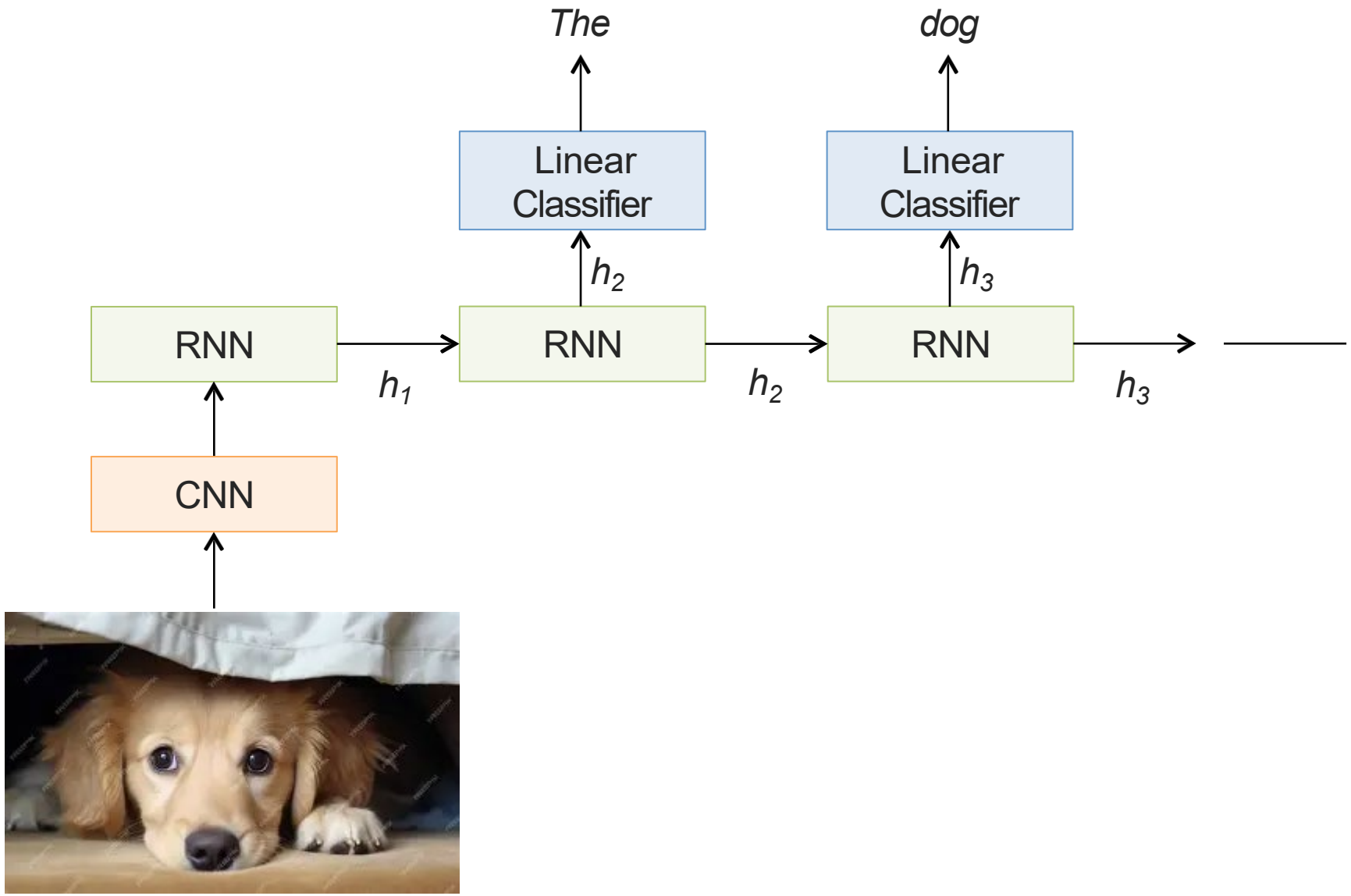
Example (Image Captioning)



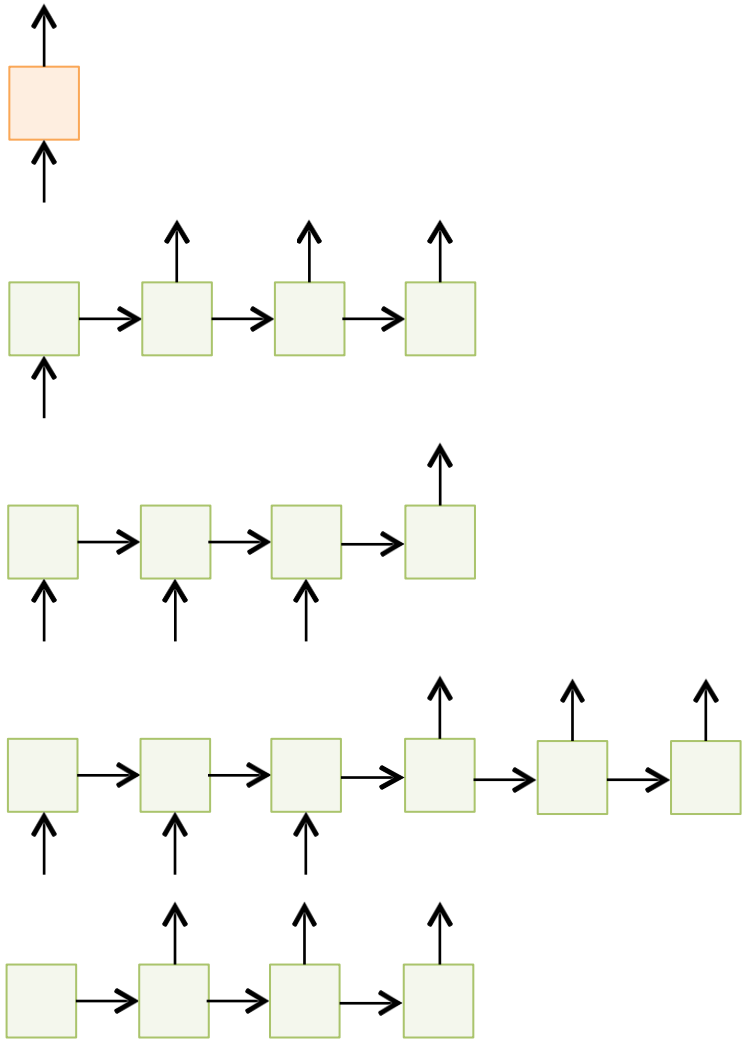
Example (Image Captioning)



Example (Image Captioning)



Input-output Scenarios



Loss Functions

- Method to evaluate how well an algorithm models the given data
- Quantifies the error between the output and the target
- Also known as cost function or error function
  - Regression Losses
  - Probabilistic Losses
  - Hinge Losses for maximum-margin classification
- https://keras.io/api/losses/

Regression Loss Functions

- Mean Square Error (MSE) / Quadratic Loss / L2 Loss
  - Average of the sum of the squared differences between actual value and the predicted value
- Mean Absolute Error (MAE) / L1 Loss
  - Average of the sum of the absolute differences between actual value and the predicted value
  - Robust to outliers since it does not make use of square
- Mean Bias Error
  - Average of the sum of the differences between actual value and the predicted value
  - Positive and negative values may cancel out – less accurate in practice
  - Can be used to see whether model has positive or negative bias

MSE & MAE

$w = -266.5344 + 6.1376h$

		x	y_i	y_i'	$y_i - y_i'$	$(y_i - y_i')^2$
-266.53	6.1376	63	127	120.1388	6.8612	47.076065
-266.53	6.1376	64	121	126.2764	-5.2764	27.840397
-266.53	6.1376	66	142	138.5516	3.4484	11.891463
-266.53	6.1376	69	157	156.9644	0.0356	0.0012674
-266.53	6.1376	69	162	156.9644	5.0356	25.357267
-266.53	6.1376	71	156	169.2396	-13.2396	175.28701
-266.53	6.1376	71	169	169.2396	-0.2396	0.0574082
-266.53	6.1376	72	165	175.3772	-10.3772	107.68628
-266.53	6.1376	73	181	181.5148	-0.5148	0.265019
-266.53	6.1376	75	208	193.79	14.21	201.9241
					Total	597.38627

Prediction Error	Absolute Error	Squared Error	
6.8612	6.8612	47.07607	
-5.2764	5.2764	27.8404	
3.4484	3.4484	11.89146	
0.0356	0.0356	0.001267	
5.0356	5.0356	25.35727	
-13.2396	13.2396	175.287	
-0.2396	0.2396	0.057408	
-10.3772	10.3772	107.6863	
-0.5148	0.5148	0.265019	
14.21	14.21	201.9241	
Mean Absolute Error	5.92384	7.729077	**Root Mean Squared Error**
		59.73863	**Mean Square Error (MSE)**

Probabilistic Loss Functions

Used when a model predicts probabilities for different classes instead of class labels

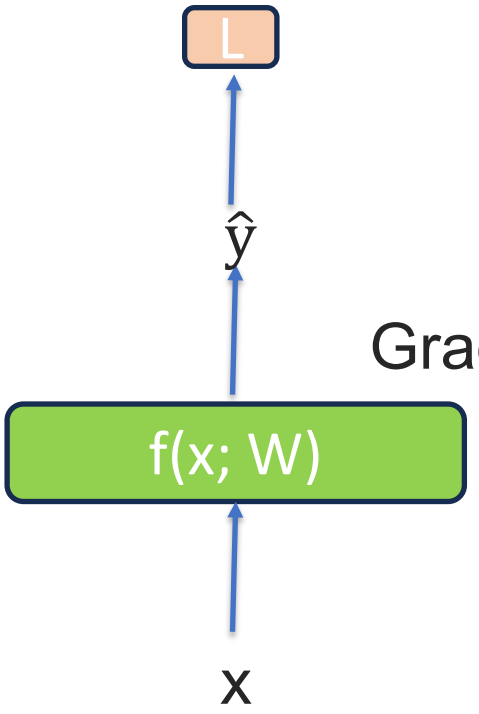
- Cross Entropy (also known as log loss): measure of the difference between two probability distributions (predicted vs actual).
  - Binary Cross Entropy (two classes – 0 and 1 as class labels)
  - Categorical Cross Entropy (one-hot encoded class labels)
  - Sparse Categorical Cross Entropy (integers as class labels)

https://www.youtube.com/watch?v=Pwgpl9mKars&ab_channel=AdianLiusie
https://www.youtube.com/watch?v=Md4b67HvmRo&ab_channel=DigitalSreeni
https://www.youtube.com/watch?v=6ArSys5qHAU&ab_channel=StatQuestwithJoshStarmer

Hinge Loss

- Primarily for classification tasks, especially with SVMs
- Helps maximizes the margin between different classes
- Loss is 0 when the correct class is confidently predicted, but penalizes predictions that are too close to the decision boundary
- Requires labels to be -1 and +1 (instead of 0 and 1)
- For multi-class classification: Categorical Hinge Loss
- Can be used in NN

Backpropagation Refresher

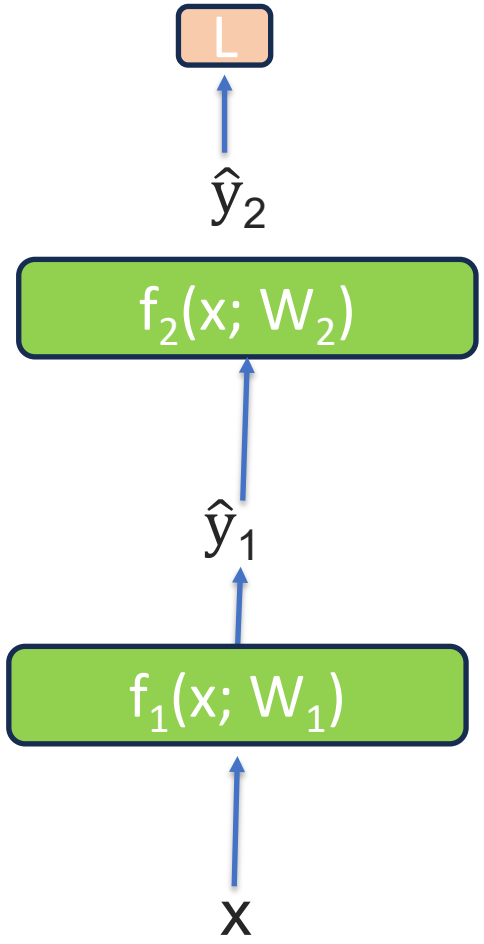


$$\hat{y} = f(x; W)$$
$$L = Loss(y, \hat{y})$$

Gradient Descent:

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W}$$



$$\hat{y}_1 = f_1(x; W_1)$$
$$\hat{y}_2 = f_2(\hat{y}_1; W_2)$$
$$L = Loss(y, \hat{y}_2)$$

Gradient Descent:

$$W_1 = W_1 - \alpha \frac{\partial L}{\partial W_1}$$

$$W_2 = W_2 - \alpha \frac{\partial L}{\partial W_2}$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial W_2}$$

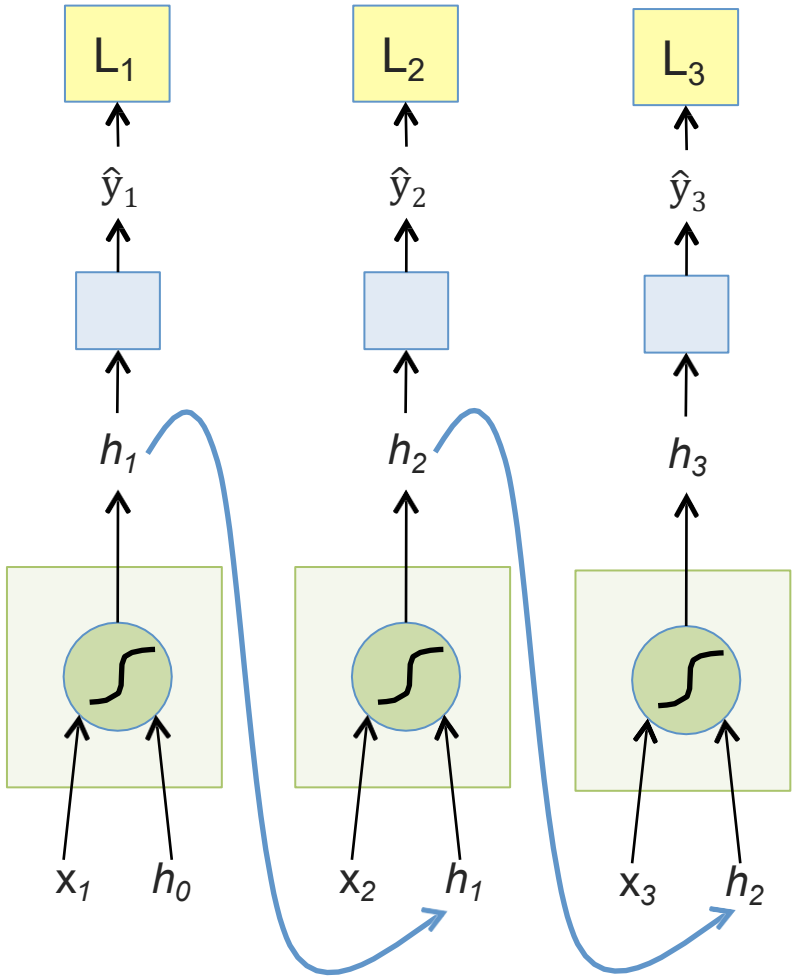
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial W_1}$$

$$= \frac{\partial L}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial W_1}$$

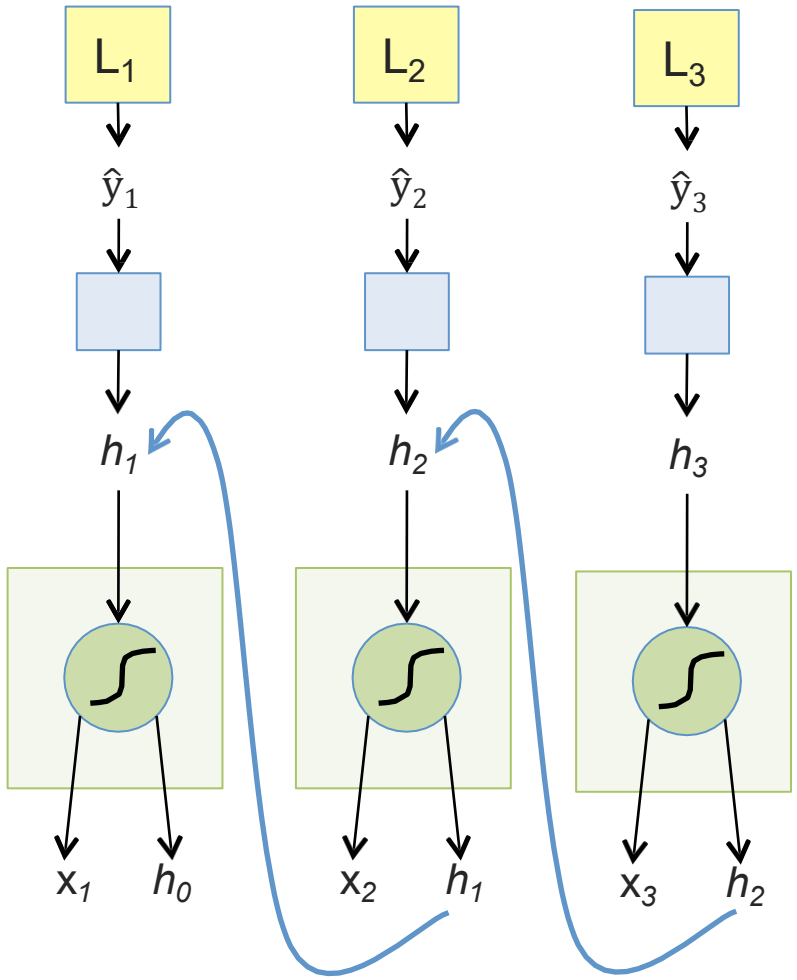
Backpropagation Through Time (BPTT)

- In a normal neural network, we use **backpropagation** to update weights by calculating gradients layer by layer.
- In an RNN, the same weights are used at every time step, and the network is "unrolled" across time steps.
- **BPTT means we compute gradients across all these time steps and update the shared weights.**
- The **weight updates** are computed for **each copy** in the **unfolded network**, then **summed** (or averaged) and then applied to the RNN weights.

BPTT – Unfolded RNN - Forward



BPTT – Unfolded RNN - Backward



$$\frac{\partial Lt}{\partial h_1} = \frac{\partial Lt}{\partial \hat{y}_t} * \frac{\partial \hat{y}_t}{\partial h_1}$$

$$= \frac{\partial Lt}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_{t\_1}} \cdots \frac{\partial h_2}{\partial h_1}$$

Problems with the Vanilla RNN

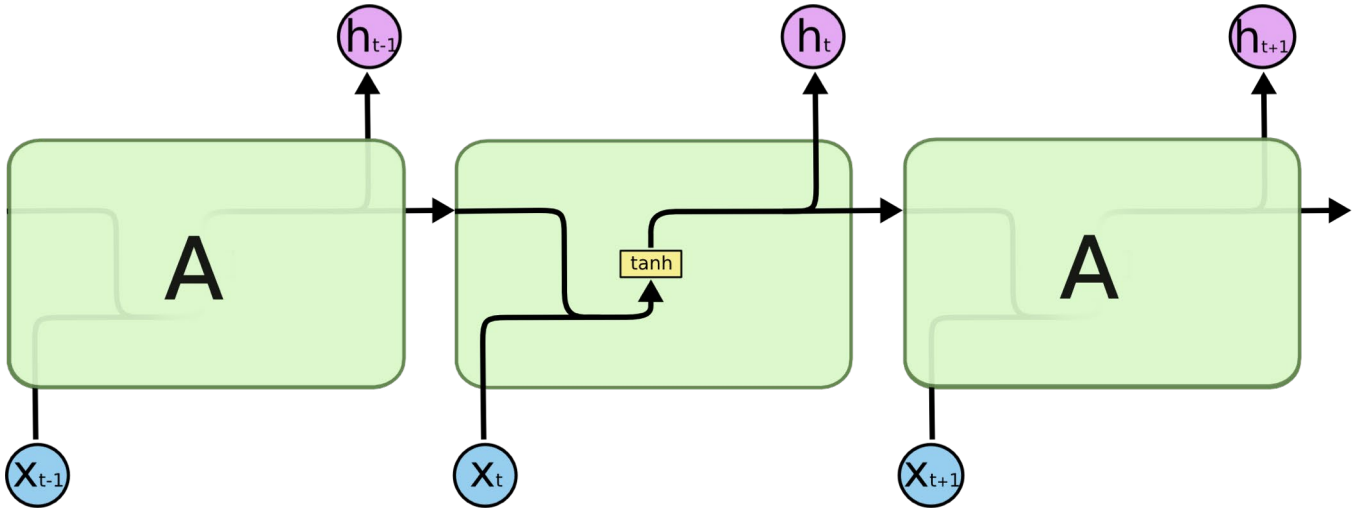
- In the same way a product of k real numbers can shrink to zero or explode to infinity, so can a product of matrices
- Vanishing gradient causes:
  - Gradients become extremely small as they propagate backward.
  - The first layers (or earliest time steps in RNN) receive almost no updates.
  - The network fails to learn long-term dependencies.

Solutions to Avoid Vanishing Gradient Problem

1) **Use Gated Architectures (LSTM / GRU)**
2) **Gradient Clipping** - Prevents gradients from becoming too small or too large.
3) **Use Activation Functions Carefully** - functions like **ReLU** (instead of tanh or sigmoid) do not squash values as much
4) **Layer Normalization / Batch Normalization** - Normalizes activations to keep values in a stable range.
5) **Use Shorter Sequences** - Backpropagating through fewer time steps reduces gradient decay.

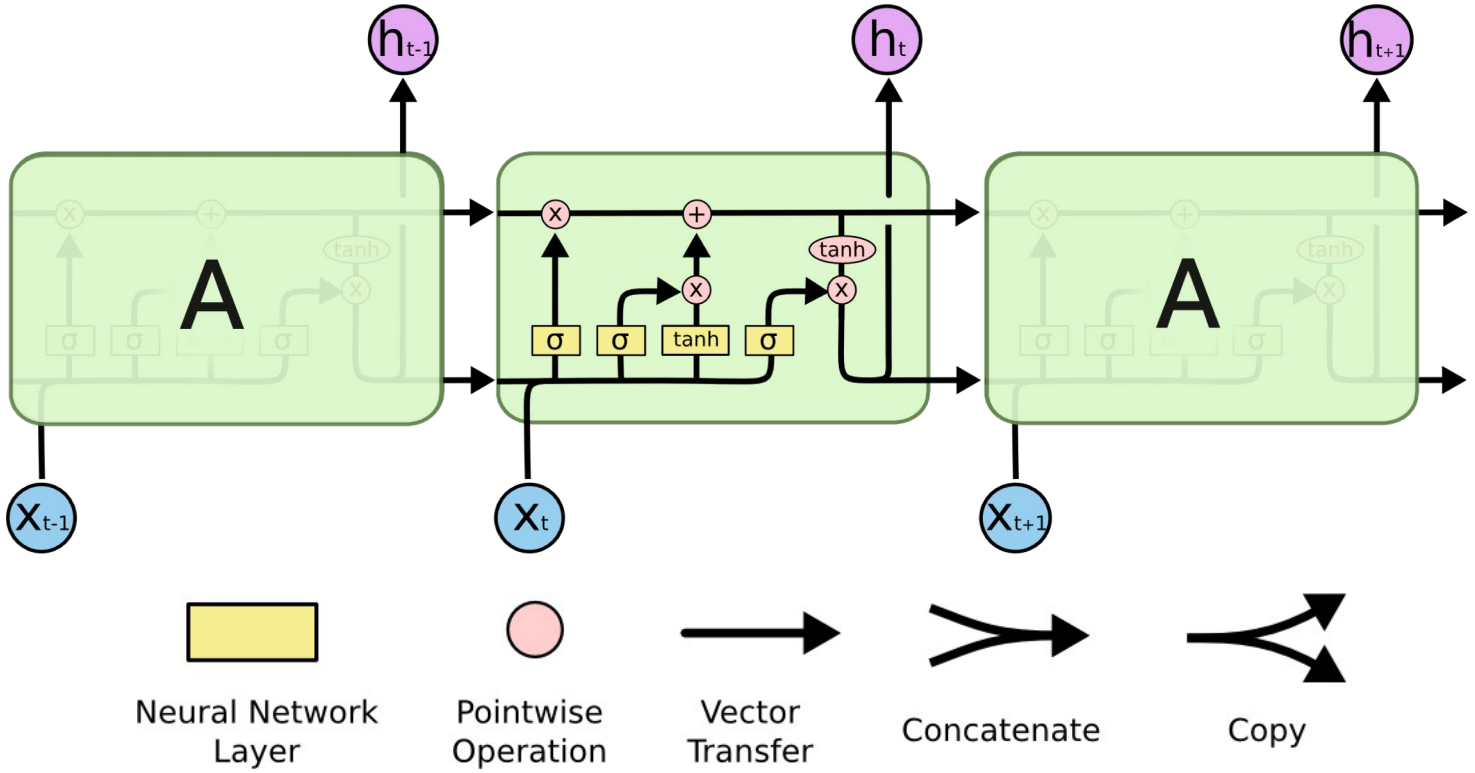
Long Short Term Memory (LSTM)

- **Long Short Term Memory networks** – usually just called "**LSTMs**" – are a special kind of RNN, capable of learning **long-term dependencies**. *Hochreiter & Schmidhuber (1997)*



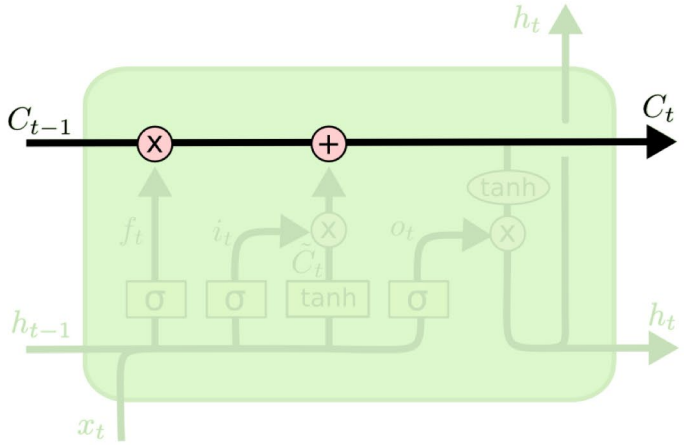
Long Short-Term Memory (LSTM)

The repeating module in a standard LSTM contains a single layer

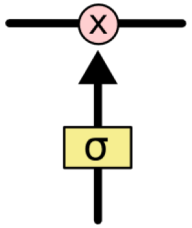


Long Short-Term Memory (LSTM)

- The core idea behind LSTMs is the **cell state**.



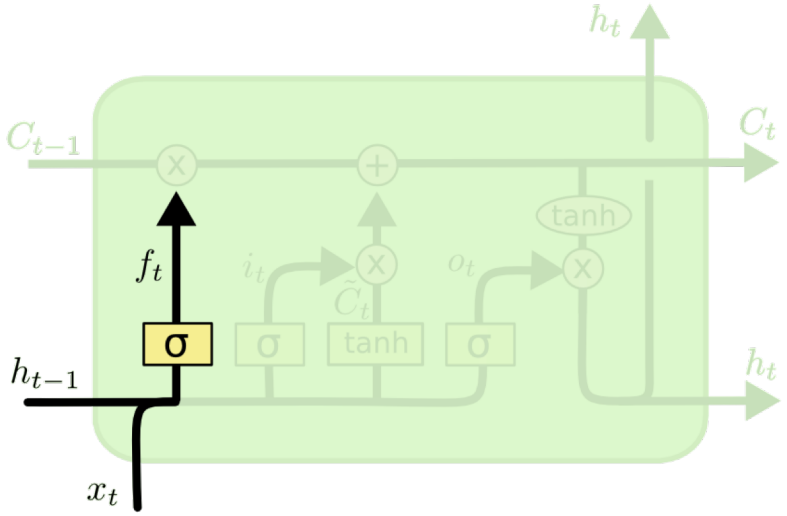
- The LSTM has the ability to **remove** or **add** information to the cell state : thanks to **gates**



- Gates are composed out of a sigmoid neural net layer and a pointwise multiplication operation

Long Short-Term Memory (LSTM)

- Step-by-Step LSTM Walk Through
  - **Step 1:** Decide what information to **throw away** from the cell state, **forget layer.**

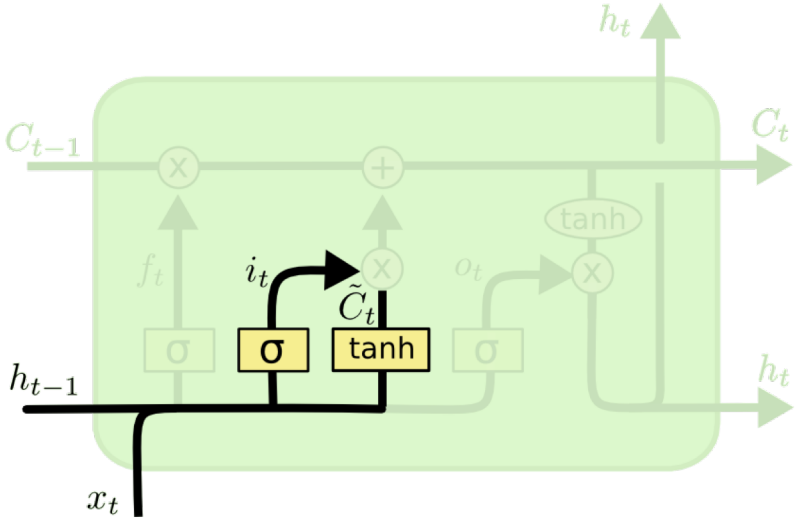


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

  - **1** represents “completely keep this”
  - **0** represents “completely get rid of this.”

Long Short-Term Memory (LSTM)

- Step-by-Step LSTM Walk Through
  - **Step 2**: Decide what new information we're going to store in the cell state



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

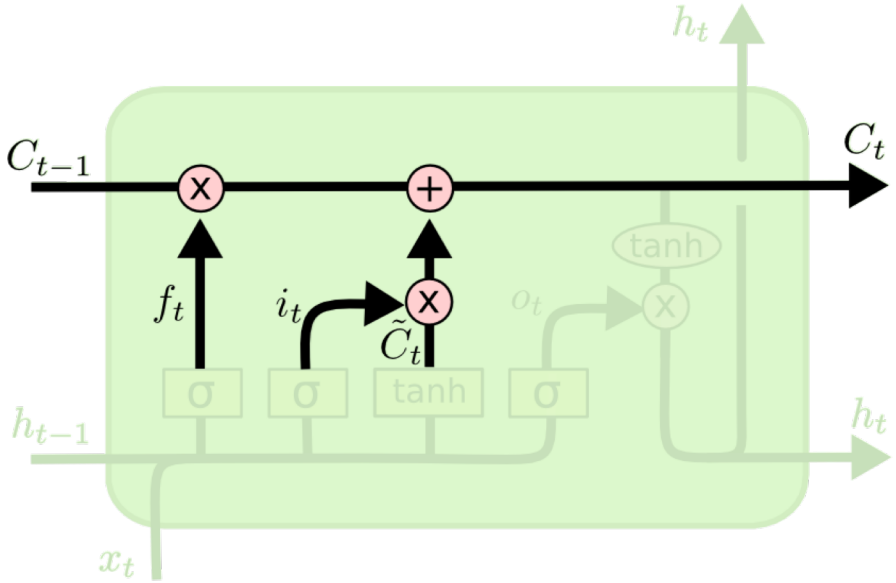
  - **Input gate layer** : decides which values we will update
  - **Tanh layer** : creates a vector of new candidate values

- **Example** : "I grew up in France… I speak fluent *French*."

Long Short-Term Memory (LSTM)

- Step-by-Step LSTM Walk Through

- **Step 3**: Update the cell state

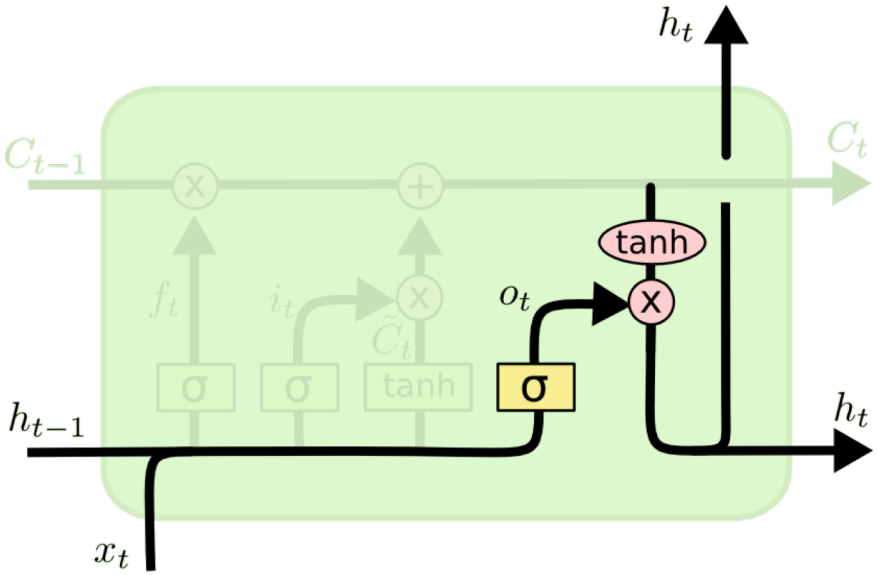


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **Example** : "I grew up in France… I speak fluent *French*."

Long Short-Term Memory (LSTM)

- Step-by-Step LSTM Walk Through
  - **Step 4**: Decide what is the output



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

- **Example** : "I grew up in France… I speak fluent *French*."

Summary

- FNN– Review
- Motivation
- Usages of Sequential Data
- Time Series
- Time Series – Components
- Recurrent Neural Networks (RNNs)
- Backpropagation Refresher
- Backpropagation Through Time (BTT)
- Vanishing Gradient Problem
- Long-Short Term Memory


WHY?
WHAT?
WHERE?
WHEN?
WHO?
HOW?
WHICH?
WHEN? WHERE?
WHAT? HOW? WHY?
WHY? HOW?
WHO? WHERE? WHAT? HOW?
WHY? HOW? WHERE?
WHO? WHOSE?
WHERE?
HOW? WHO?
WHO? WHERE? WHAT? HOW?
WHAT? HOW? WHY?
HOW?