LECTURE 2: SUPERVISED ML AND RNN

CSC5991: Introduction to LLMs

OUTLINE

- Supervised learning.
- Deep neural networks.
- Loss functions.
- Training neural networks.
- Recurrent neural networks.

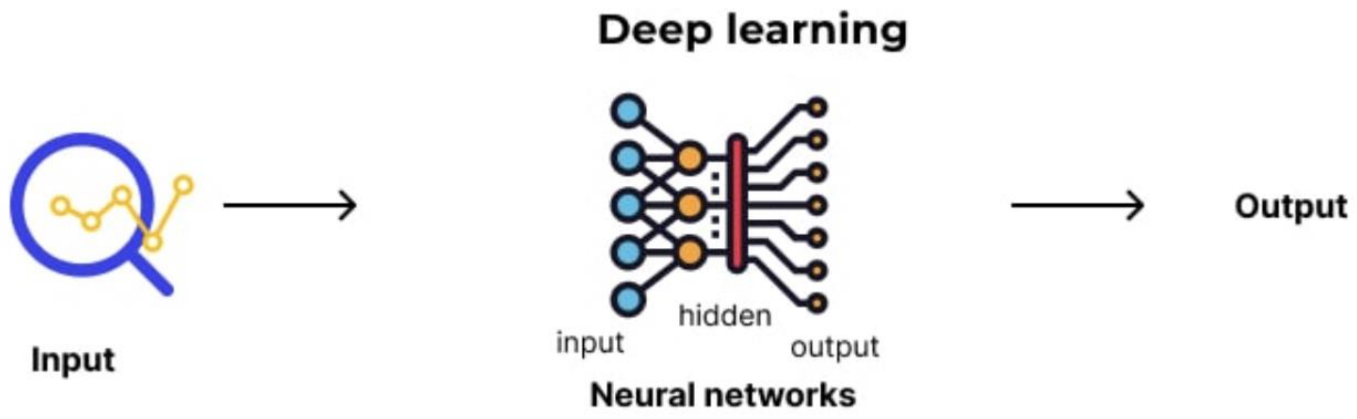
WHAT IS SUPERVISED LEARNING?

Definition

Machine learning (ML) paradigm where the model learns from labeled data.

- Data samples: (x_i, y_i) for $i = 1, \dots, n$.
- Input: x_i, Output: y_i for $i = 1, \dots, n$.

Goal of ML: Given an unseen x predict the label y.



EXAMPLES

Image Classification

Document Categorization

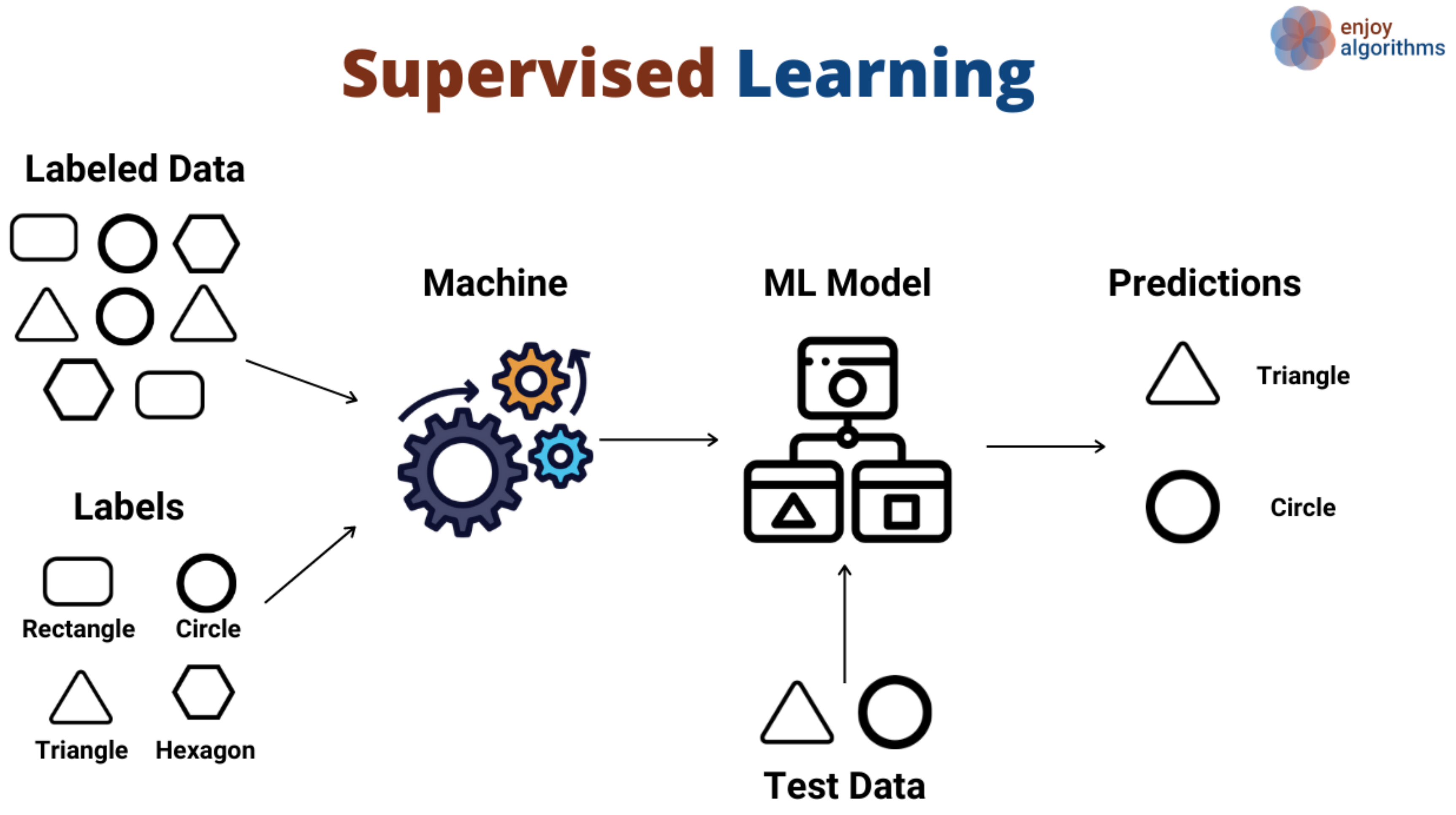
Speech Recognition Protein Classification Spam Detection

Branch Prediction Fraud Detection Natural Language Processing

Playing Games Computational Advertising

Taken from CMU 10-401, Spring 2018

SUPERVISED LEARNING WORKFLOW



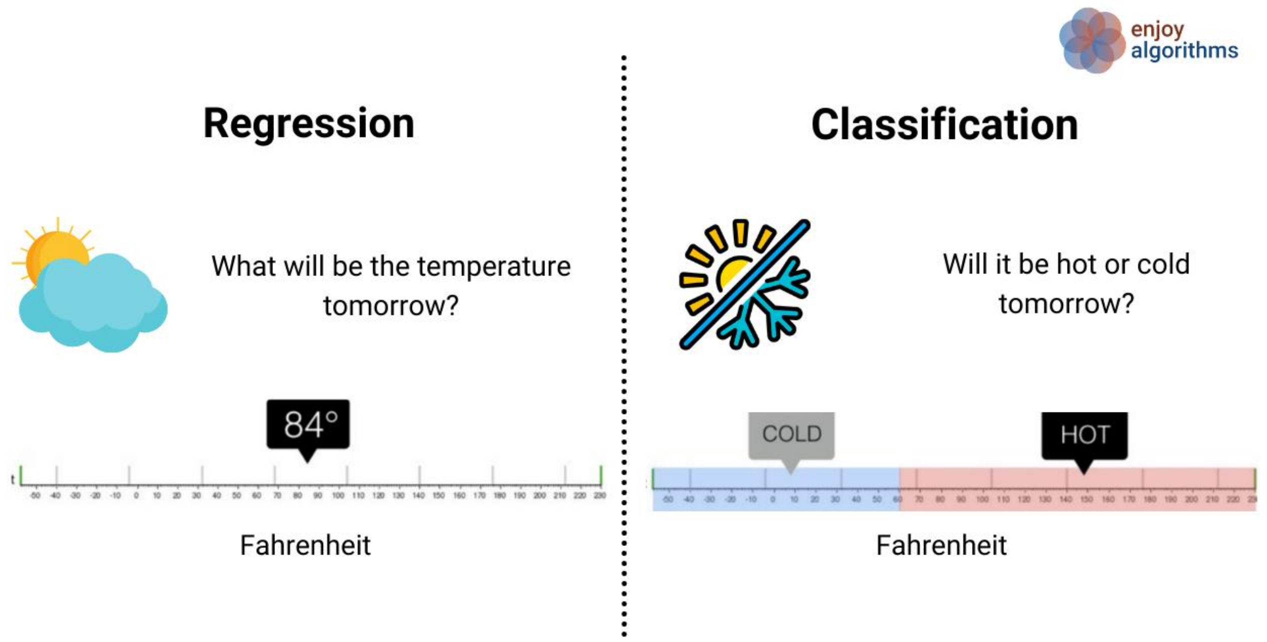
TYPES OF SUPERVISED LEARNING

Regression:

- Predicting continuous values

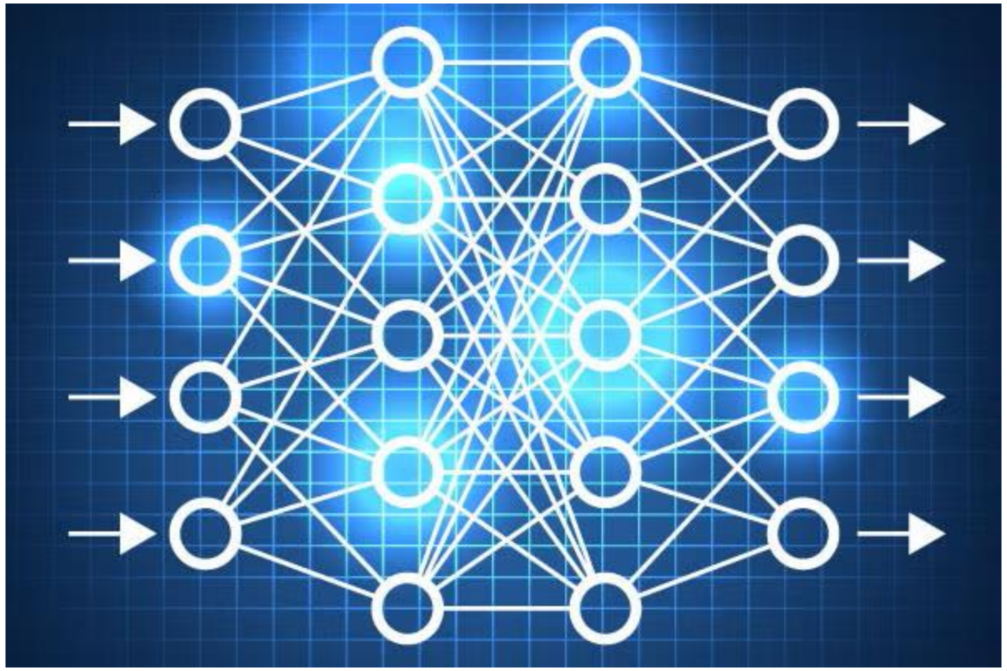
Classification:

- Predicting discrete labels

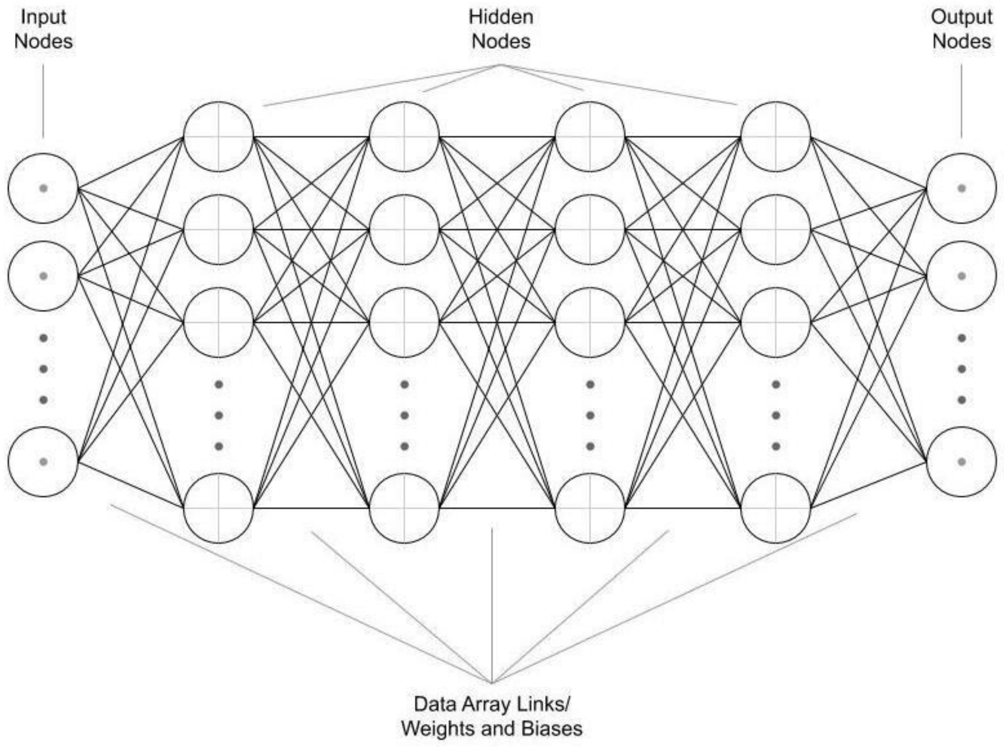

enjoy algorithms
Regression
What will be the temperature tomorrow?
84°
Fahrenheit
Classification
Will it be hot or cold tomorrow?
COLD
HOT
Fahrenheit

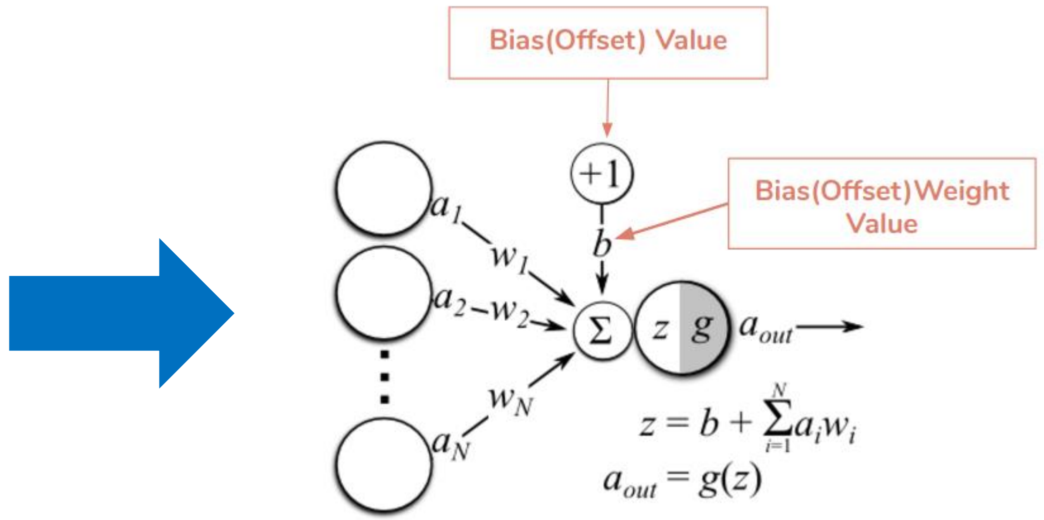
DEEP NEURAL NETWORKS

- **Definition:** Neural networks are computational models inspired by the human brain, designed to recognize patterns.
- **Key Components:**
  - Neurons.
  - Layers.
  - Weights and biases.



NEURAL NETWORK ARCHITECTURE





$$z = b + \sum_{i=1}^{N} a_i w_i$$

$$a_{out} = g(z)$$

LOSS FUNCTION

Recall:

- Data samples: (x_i, y_i) for $i = 1, \dots, n$.
- Input: x_i, Output: y_i for $i = 1, \dots, n$.
- **Loss function:** $L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i;\theta), y_i)$.

Goal: $\min_\theta L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i;\theta), y_i)$

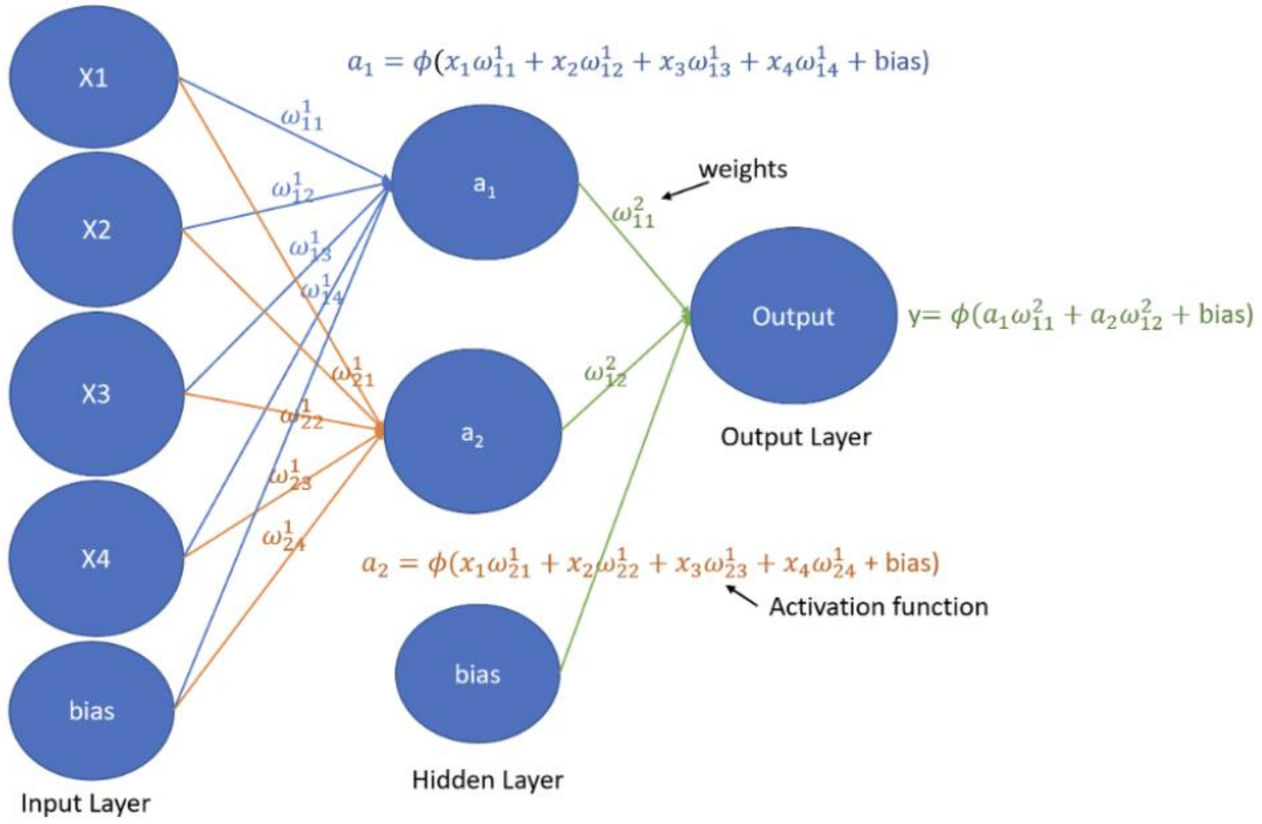
where $f(x_i;\theta)$ is the prediction of true label y_i and θ are the neural network parameters to be learned.

Q: Can you give examples of some common loss functions?

FORWARD PROPAGATION

- **Computes the loss function** by doing a forward pass.
  - Measures the difference between predicted outputs and actual labels.

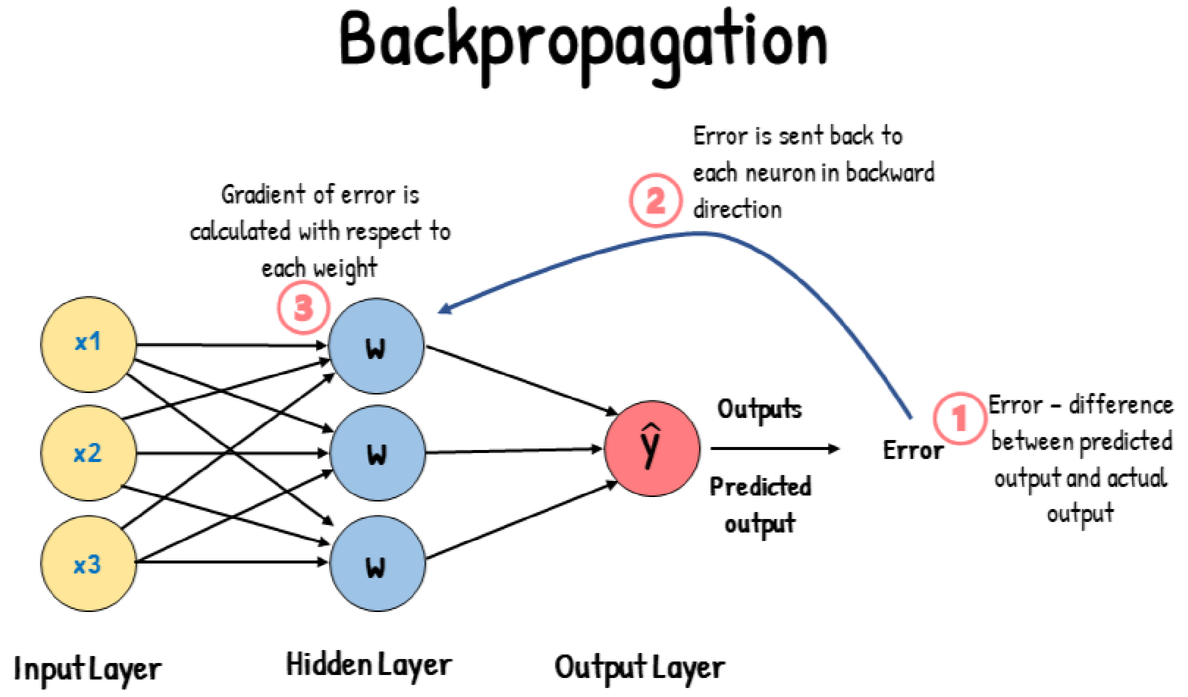
- **Examples of loss functions:**
  - Mean Squared Error (MSE)
  - Cross-Entropy Loss



$a_1 = \phi(x_1\omega_{11}^1 + x_2\omega_{12}^1 + x_3\omega_{13}^1 + x_4\omega_{14}^1 + \text{bias})$
X1
X2
X3
X4
bias
Input Layer
ω_{11}^1
ω_{12}^1
ω_{13}^1
ω_{14}^1
ω_{21}^1
ω_{22}^1
ω_{23}^1
ω_{24}^1
a_1
a_2
bias
Hidden Layer
weights
ω_{11}^2
ω_{12}^2
Output
Output Layer
$y = \phi(a_1\omega_{11}^2 + a_2\omega_{12}^2 + \text{bias})$
$a_2 = \phi(x_1\omega_{21}^1 + x_2\omega_{22}^1 + x_3\omega_{23}^1 + x_4\omega_{24}^1 + \text{bias})$
Activation function

BACKPROPAGATION

- **Definition:** Algorithm to compute gradients for updating weights.
- **Steps:**
  - Calculate the loss
  - Compute gradients
  - Update weights using gradients
- **Resource:** https://www.youtube.com/watch?v=Ilg3gGewQ5U&ab_channel=3Blue1Brown


Backpropagation
Error is sent back to each neuron in backward direction
2
Gradient of error is calculated with respect to each weight
3
x1
x2
x3
w
w
w
ŷ
Outputs
Predicted output
Error
1
Error – difference between predicted output and actual output
Input Layer
Hidden Layer
Output Layer

OPTIMIZATION ALGORITHMS

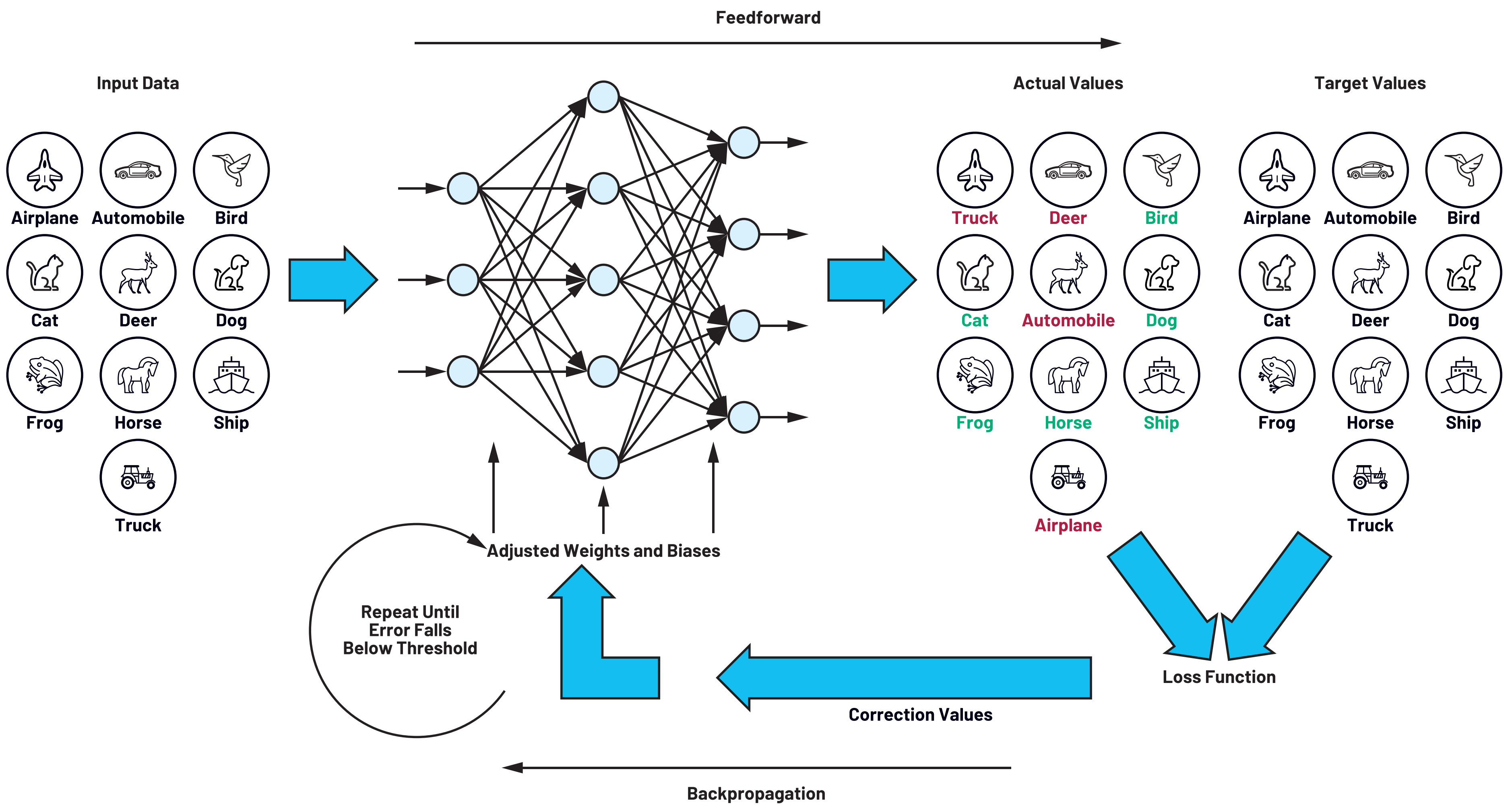
- Gradient Descent
- Stochastic Gradient Descent (SGD)
- Adam Optimizer

$$\theta_{t+1} = \theta_t - \gamma\, g_t$$

where g_t is

- Gradient of the loss function $g_t = \nabla L(\theta) = \nabla\left(\frac{1}{n}\sum_{i=1}^{n} l(f(x_i;\theta), y_i)\right)$ for gradient descent
- Batch gradient computed using samples chosen at iteration t $g_t = \nabla\left(\frac{1}{b}\sum_{i=1}^{b} l(f(x_i;\theta_t), y_i)\right)$ for stochastic gradient descent
- For Adam g_t is adaptive

TRAINING NEURAL NETWORKS



OUR FOCUS

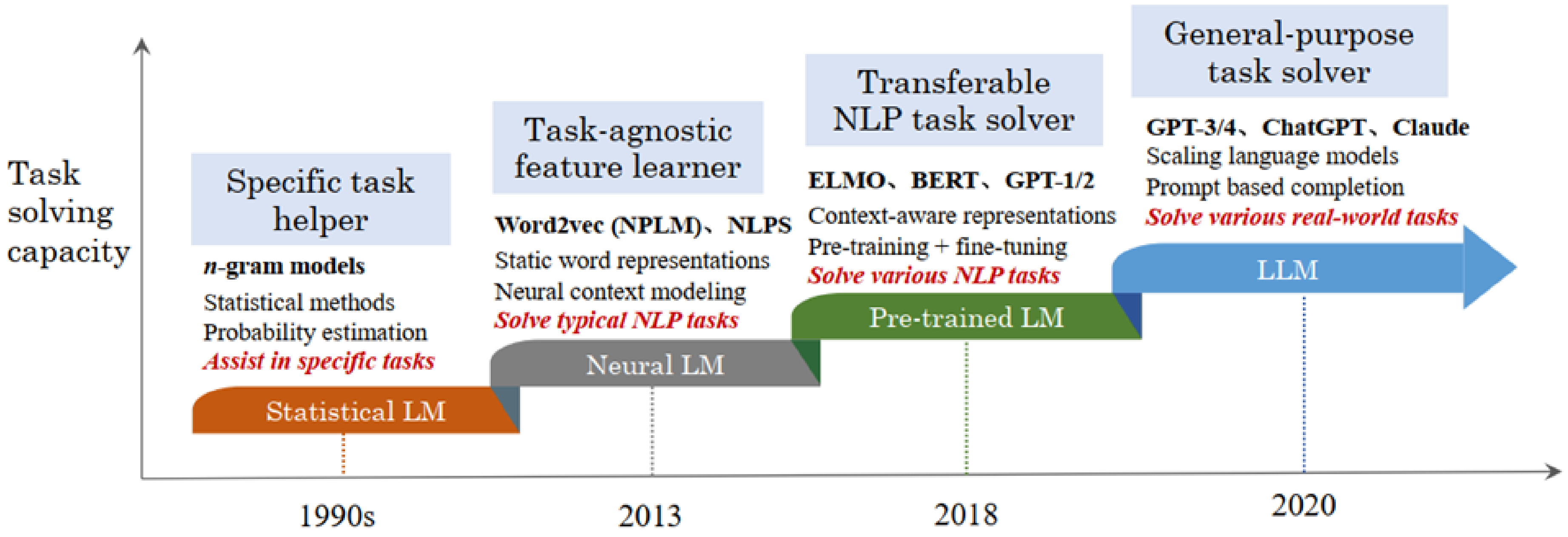
- **Sequential tasks:** NLP tasks like language modelling, machine translation, and text generation: Given $x_1, x_2, \dots, x_{t-1}$ predict x_t.

- **Loss function:** Maximizing the log-likelihood of a model given a set of training sequences (theta is a set of model parameters)

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log p\left(x_t^n \mid x_1^n, \ldots, x_{t-1}^n; \boldsymbol{\theta}\right),$$

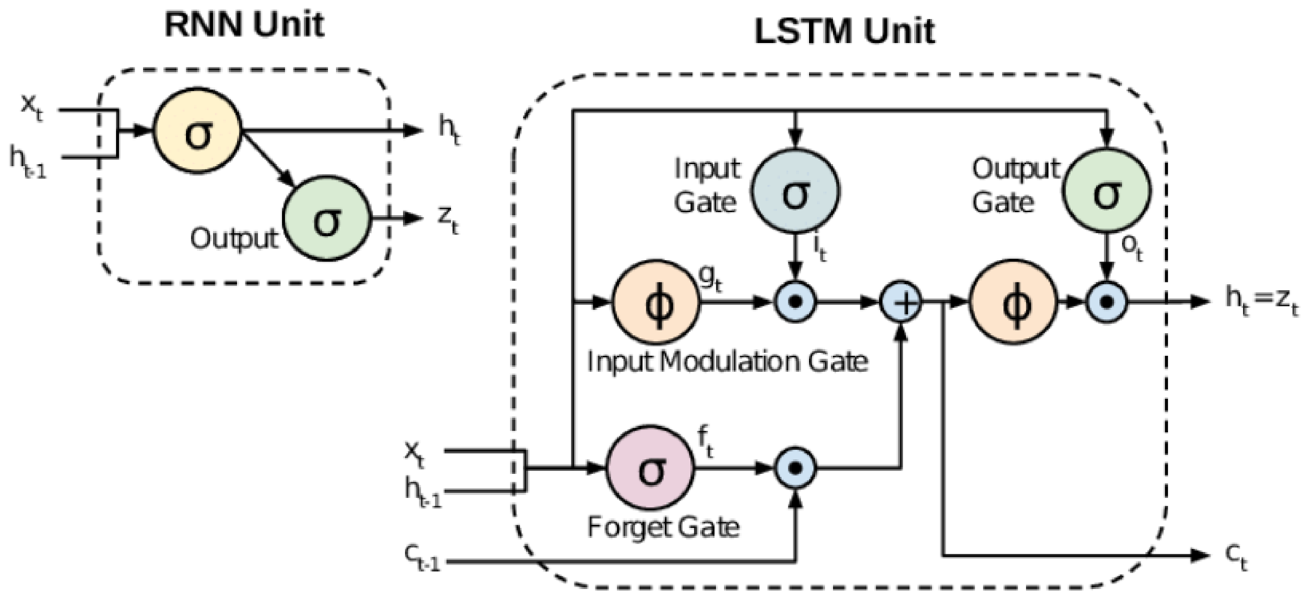
A BRIEF HISTORY

- Early language models: Markov Chains, n-grams.
- Neural networks for sequences: RNNs and LSTMs.
- The rise of **transformers** and the shift towards **self-attention**.

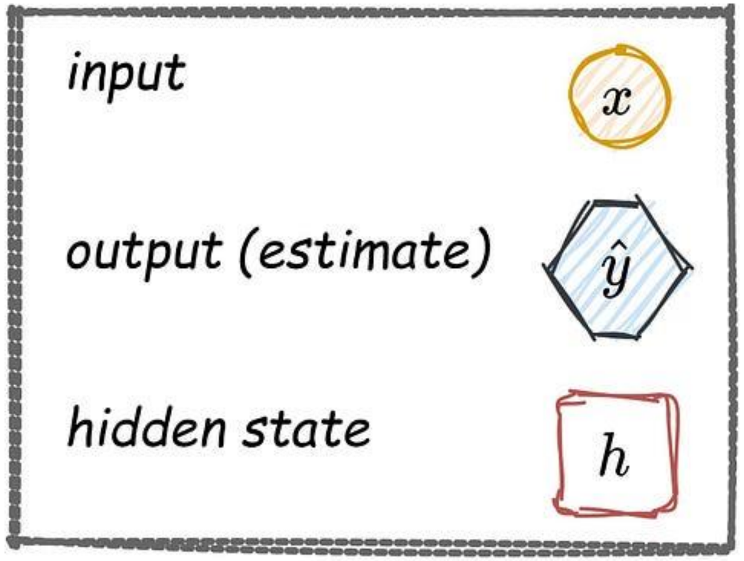


INTRODUCTION TO RNN

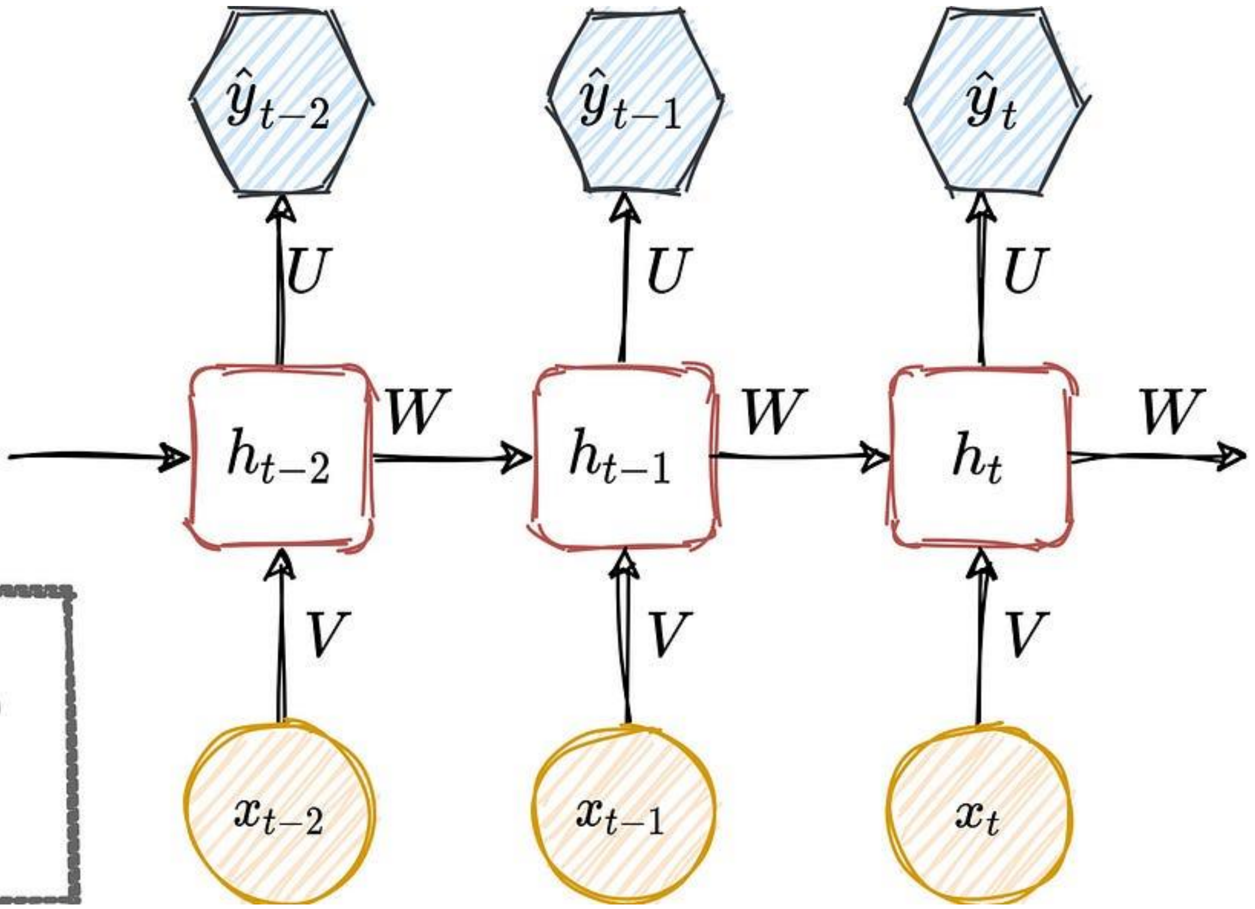
- Basics of recurrent neural networks.
- LSTM, GRU and other variants of RNN.
- How the RNNs have been used in different downstream applications.

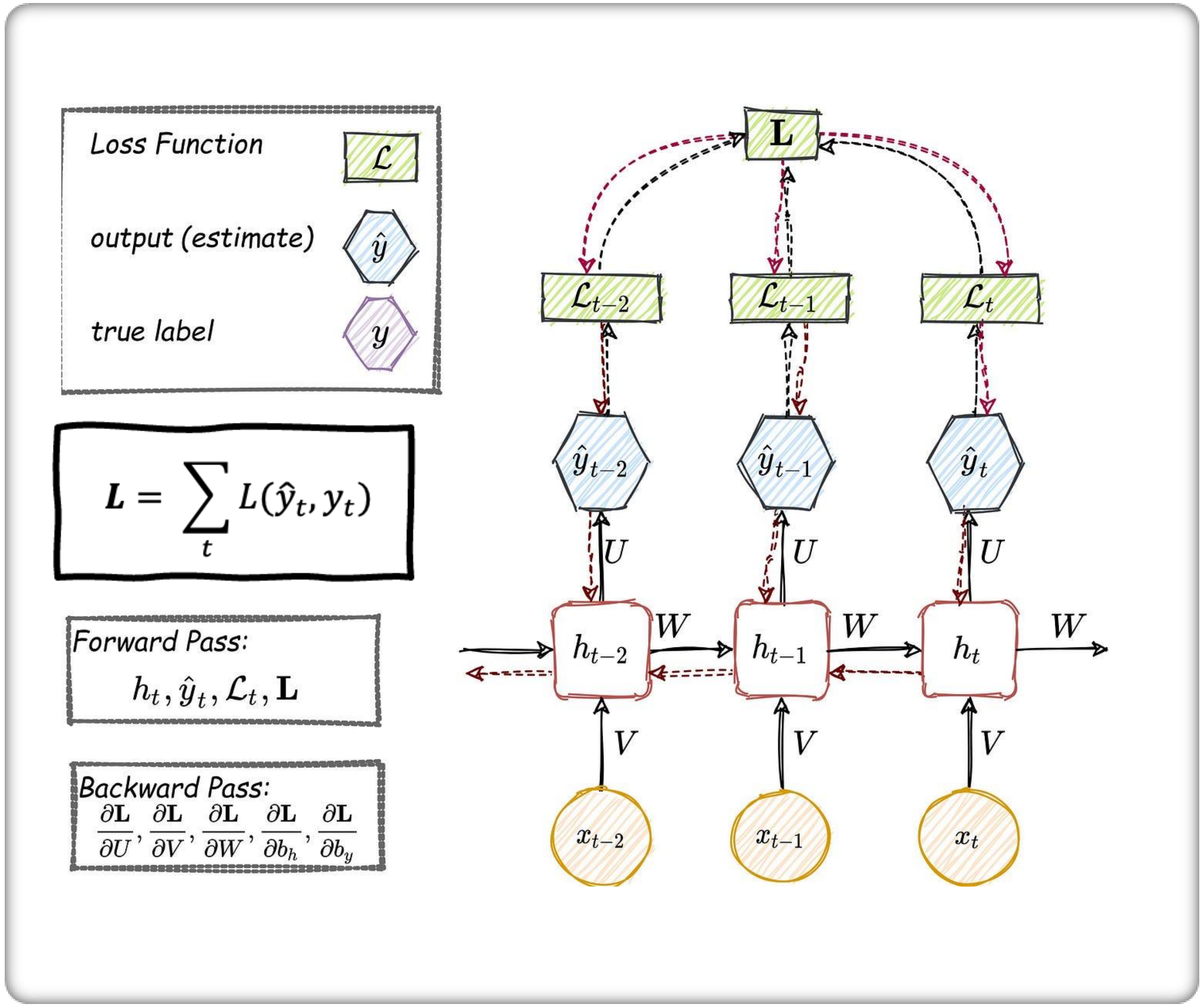


VANILLA RNN



$$h_t = f_h\left(Vx_t + Wh_{t-1} + b_h\right)$$
$$\hat{y}_t = f_y\left(Uh_t + b_y\right)$$





$$\boldsymbol{L} = \sum_t L(\hat{y}_t, y_t)$$

Forward Pass:

$h_t, \hat{y}_t, \mathcal{L}_t, \mathbf{L}$

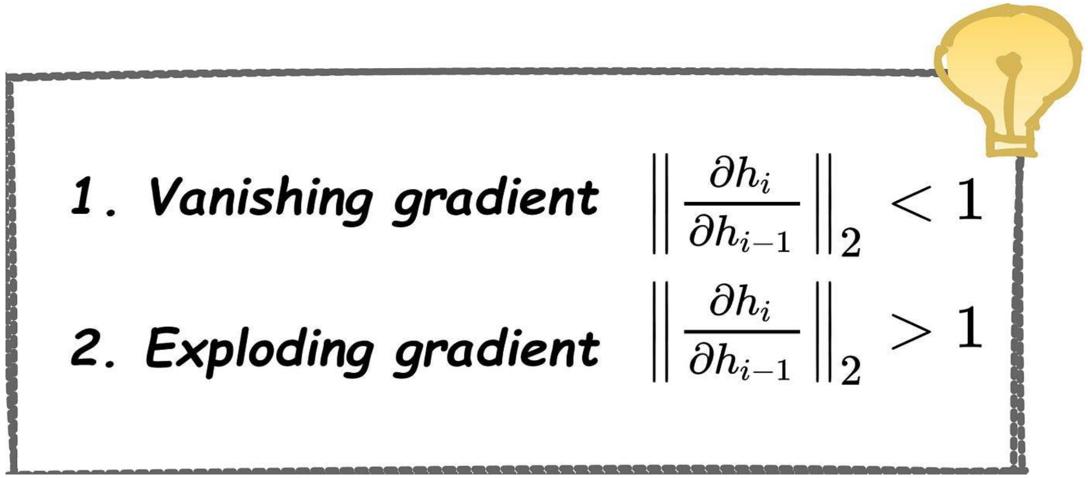
Backward Pass:

$\frac{\partial \mathbf{L}}{\partial U}, \frac{\partial \mathbf{L}}{\partial V}, \frac{\partial \mathbf{L}}{\partial W}, \frac{\partial \mathbf{L}}{\partial b_h}, \frac{\partial \mathbf{L}}{\partial b_y}$

BACKPROP. THROUGH TIME

GRADIENT VANISHING/EXPLOSION PROBLEM

$$\frac{\partial L}{\partial W} \propto \sum_{i=0}^{T-1} \left(\prod_{j=i+1}^{T} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_i}{\partial W}$$



1. *Vanishing gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$
2. *Exploding gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$

- **Vanishing gradient:** The term goes to zero exponentially fast, which makes it difficult to learn long-period dependencies.
- **Exploding gradient:** The term goes to infinity exponentially fast, and its value becomes NaN due to the unstable process.

PROBLEMS OF VANILLA RNNS

- **Gradient Vanishing and Explosion:** Due to the multiplication of gradients through the many layers during backpropagation.
  - Vanishing gradients make it **hard for the model to learn**, as updates to the weights become insignificantly small.
  - Exploding gradients can cause the weights to **oscillate or diverge wildly**.

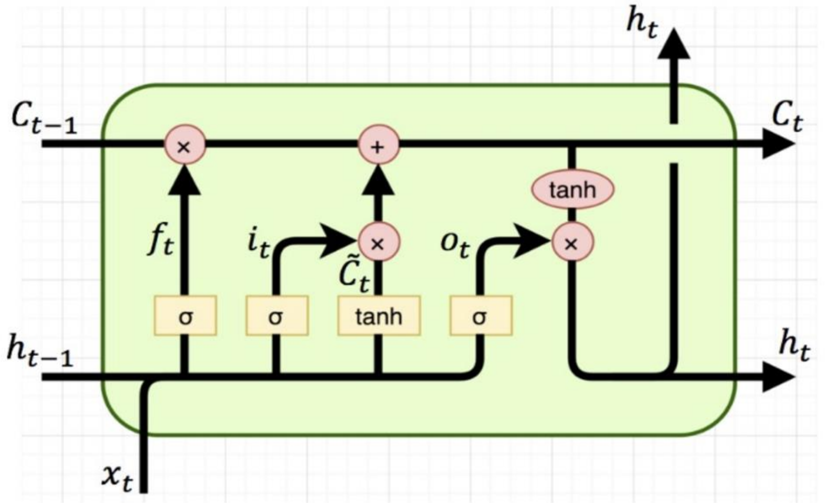
- **Long Range Memory:** RNNs ideally should remember information from early in the sequence to use much later, which is crucial for tasks like text translation or sentiment analysis.

LONG SHORT-TERM MEMORY (LSTM)

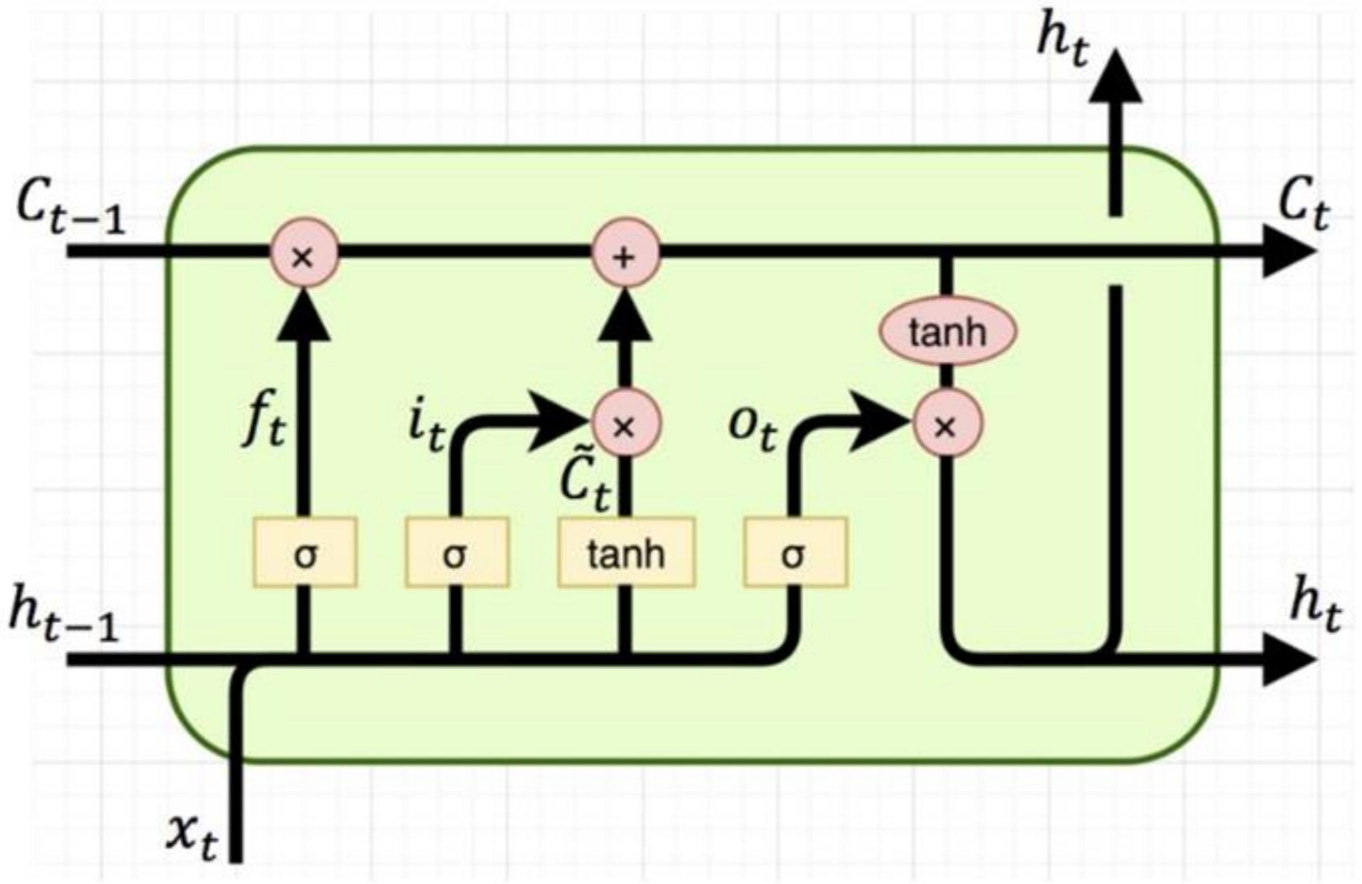
- Special **gated structure** to control memorization and forgetting
- Mitigate **gradient vanishing**
- Facilitate **long term memory**

❑These gates are essentially neural networks with sigmoid activation functions (outputting values between 0 and 1).

❑They are trained to selectively allow information to pass through, which helps in retaining the important information over longer periods.

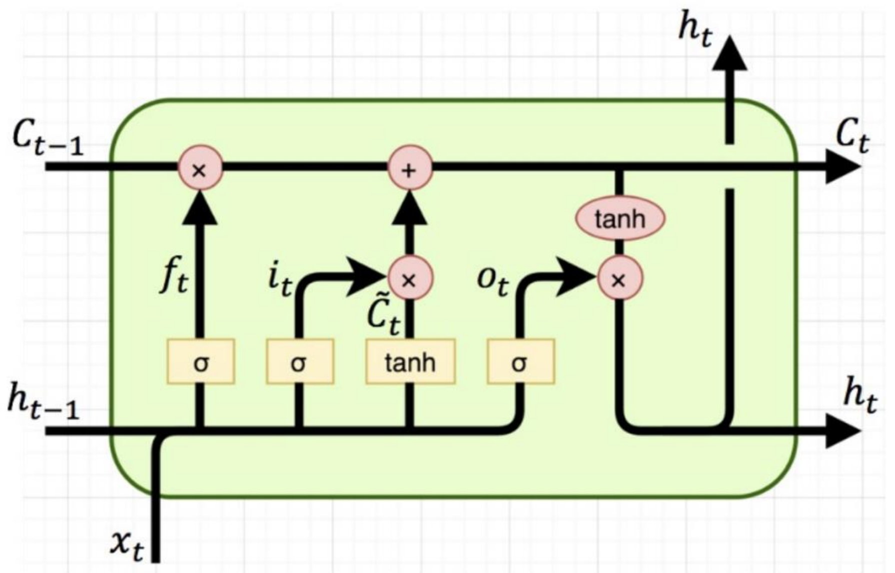


IMPLEMENTATION



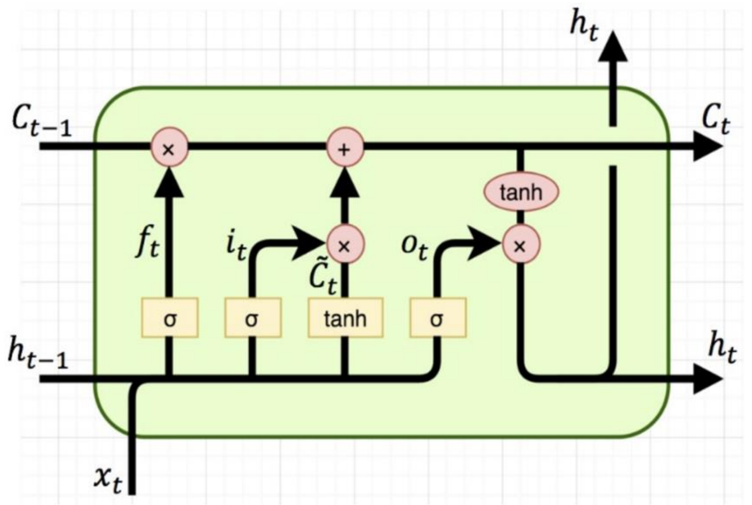
UPDATE EQUATIONS

- **Input Gate:** $i_t = \sigma(W_i[x_t, h_{t-1}] + b_i)$
- **Forget Gate:** $f_t = \sigma(W_f[x_t, h_{t-1}] + b_f)$
- **Output Gate:** $o_t = \sigma(W_o[x_t, h_{t-1}] + b_o)$
- **Process Input:** $\widetilde{C}_t = \tanh(W_C[x_t, h_{t-1}] + b_C)$
- **Cell Update:** $C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t$
- **Output:** $h_t = o_t \odot \tanh(C_t)$



ROLES OF DIFFERENT GATES IN LSTM

- **Input Gate $\boldsymbol{i_t}$:**Controls the extent to which a new value flows into the cell state.
- **Forget Gate $\boldsymbol{f_t}$:** Determines the information to be discarded from the cell state.
- **Output Gate $\boldsymbol{o_t}$:** Regulates the amount of information to output from the cell state.
- **Process Input $\boldsymbol{\widetilde{C_t}}$:** Creates a vector of new candidate values that could be added to the cell state.

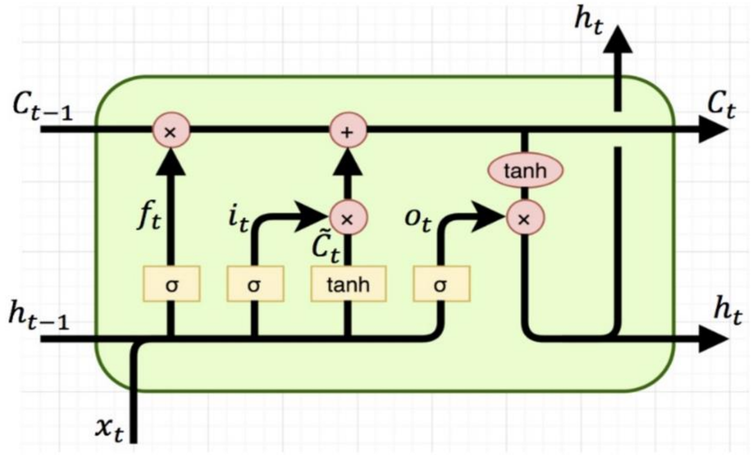


ROLES OF DIFFERENT GATES IN LSTM

- **Cell Update C_t :**Updates the cell state by combining the old state (influenced by the forget gate) and the new candidate values (modulated by the input gate).

- **Output h_t :**Determines the next hidden state based on the cell state and output gate.

- **The network can selectively remember or forget information, aiding in preserving long-term dependencies.**

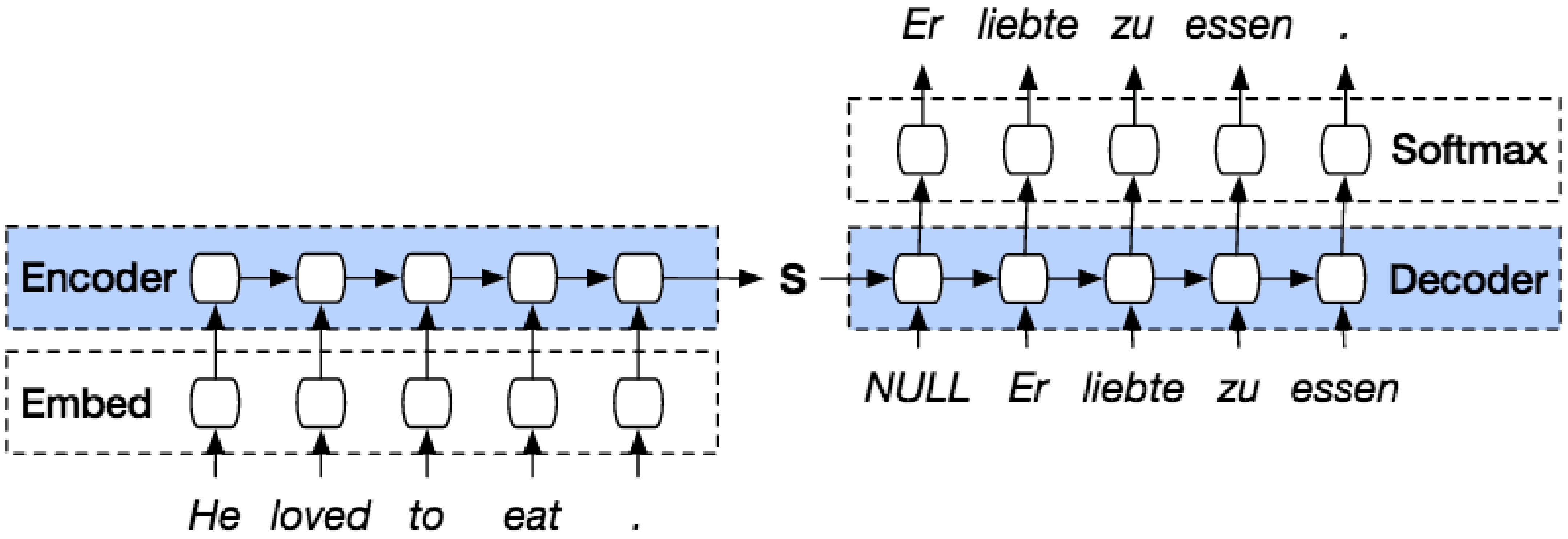


RESOURCES

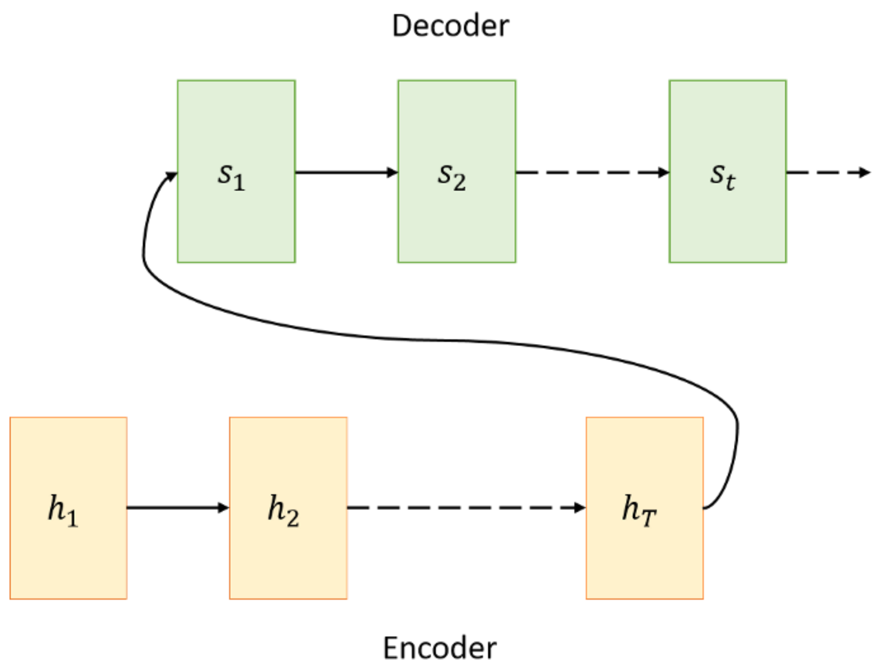
- **LSTM implementation using PyTorch**
  - https://www.geeksforgeeks.org/long-short-term-memory-networks-using-pytorch/

- **LSTM PyTorch library**
  - https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

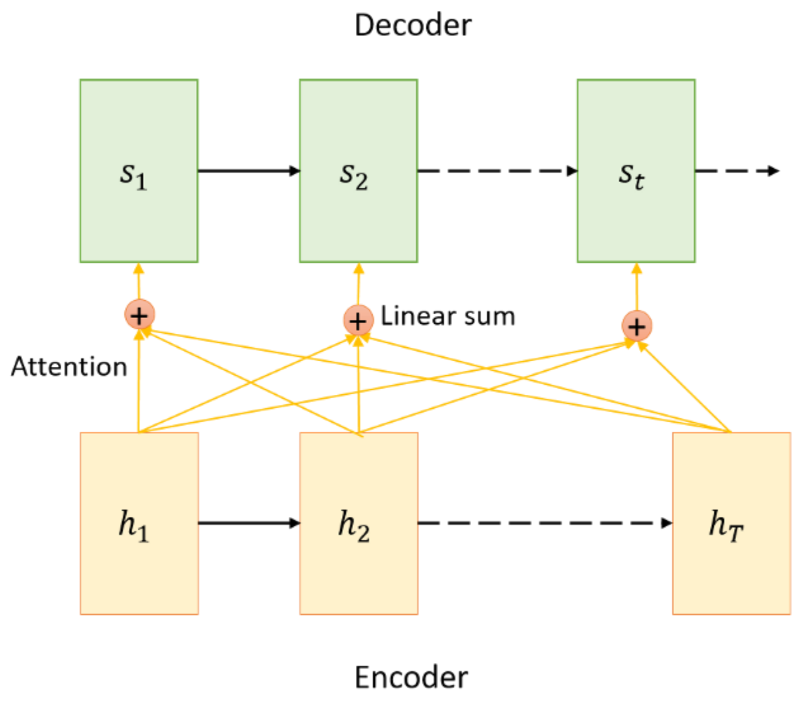
SEQUENCE TO SEQUENCE (SEQ2SEQ)



BREAKING THE BOTTLENECK

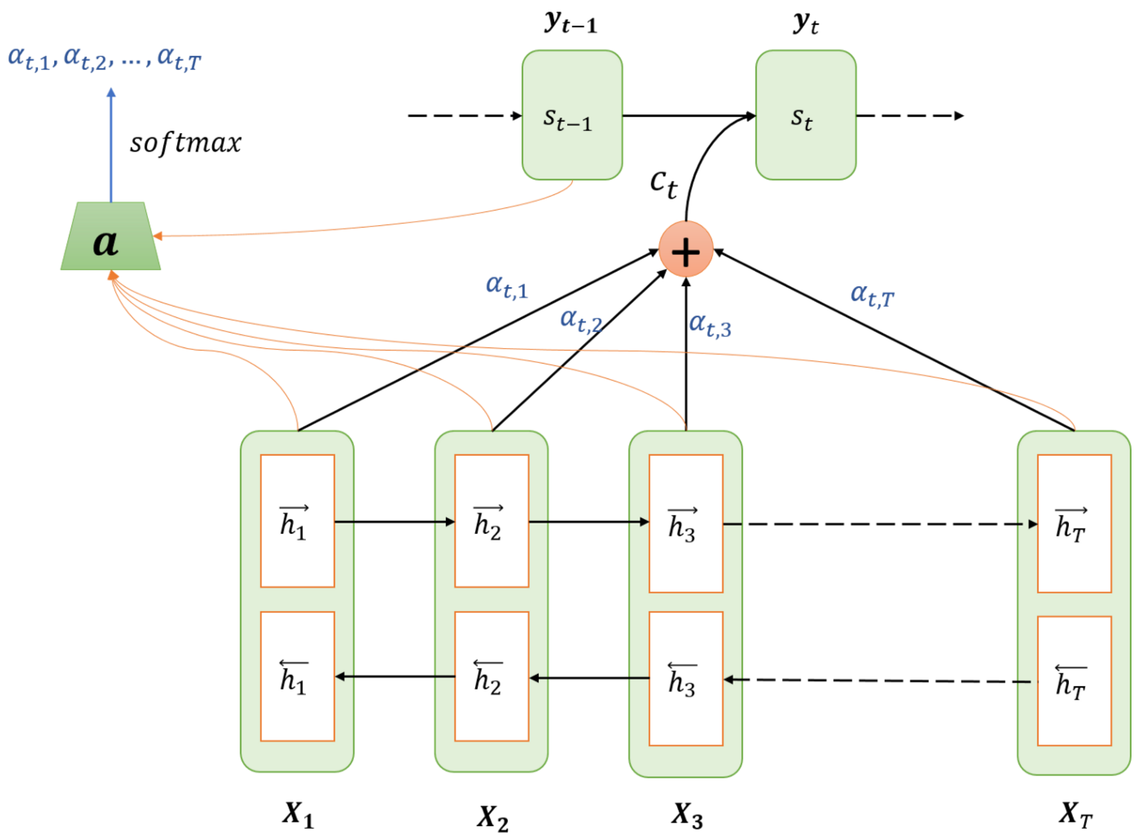


(a) Vanilla Encoder Decoder Architecture

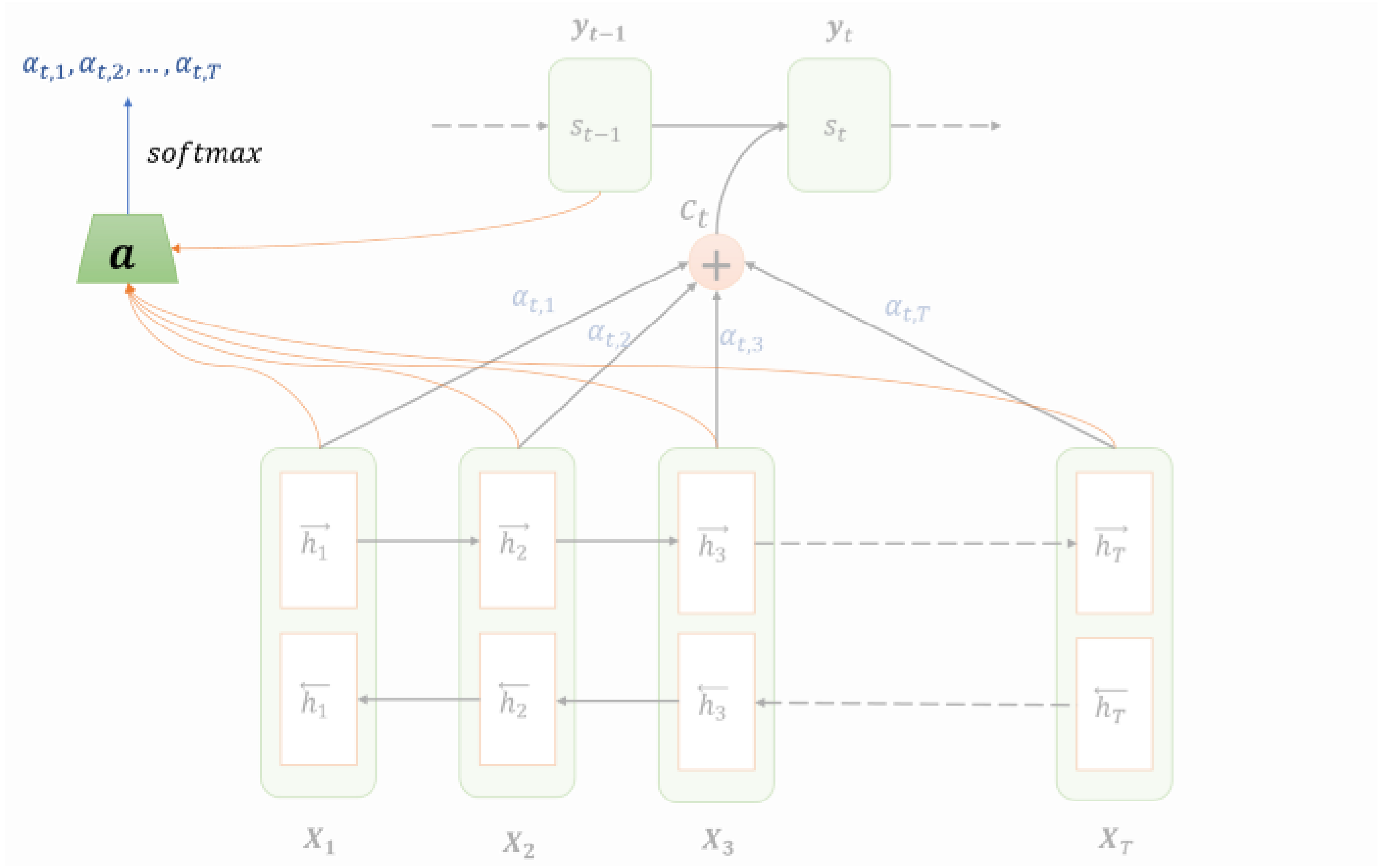


(b) Attention Mechanism

SEQ2SEQ WITH ATTENTION



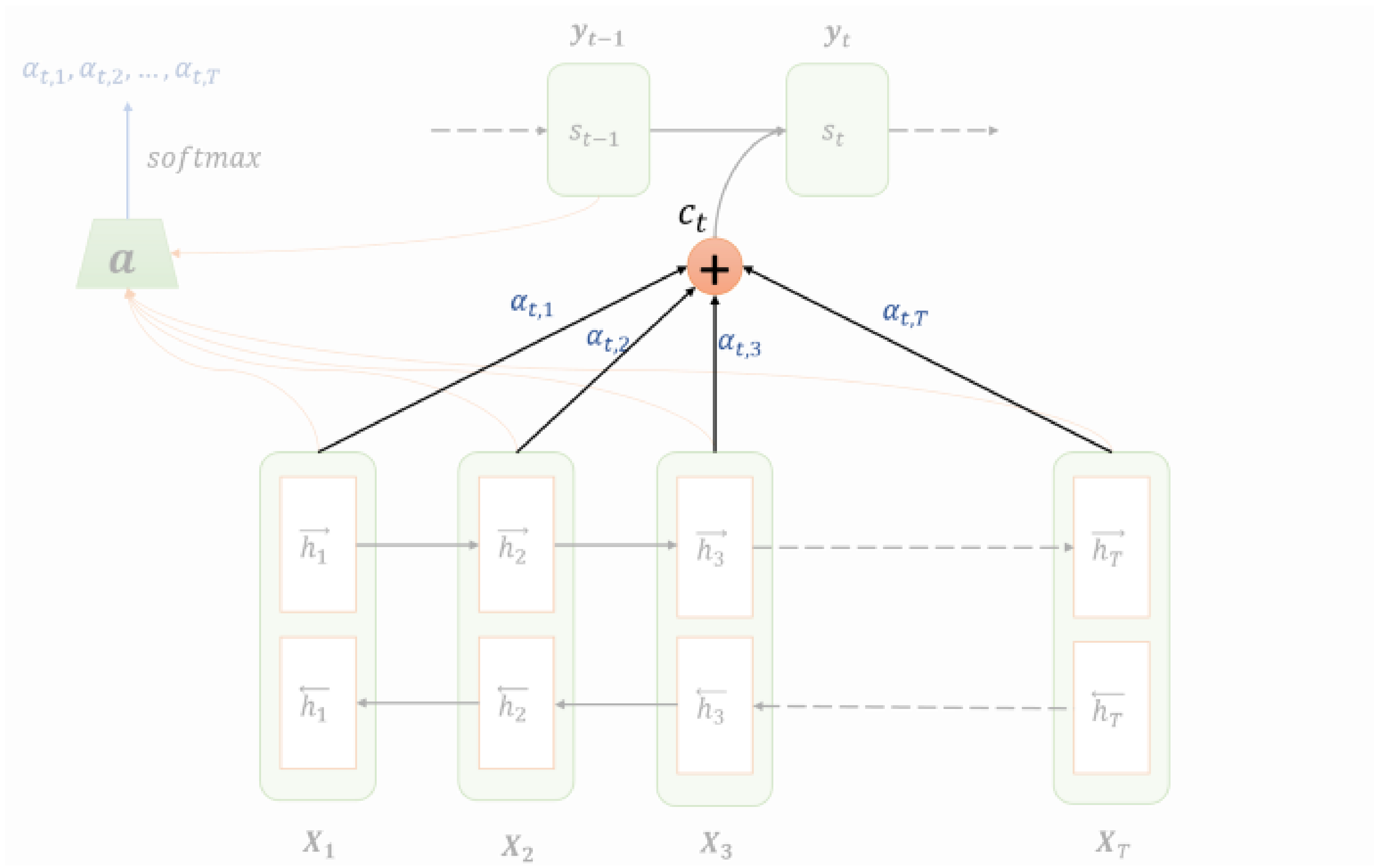
COMPUTING THE ATTENTION



$$e_{tj} = \boldsymbol{a}(h_j, s_{t-1}), \forall j \in [1, T]$$

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})}$$

CREATING THE CONTEXT VECTOR



$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$$

MACHINE TRANSLATION WITH ATTENTION

Bahdanau, Cho, Bengio (ICLR-2015)

Bleu: BiLingual Evaluation Understudy -> Percentage of translated words that appear in ground truth

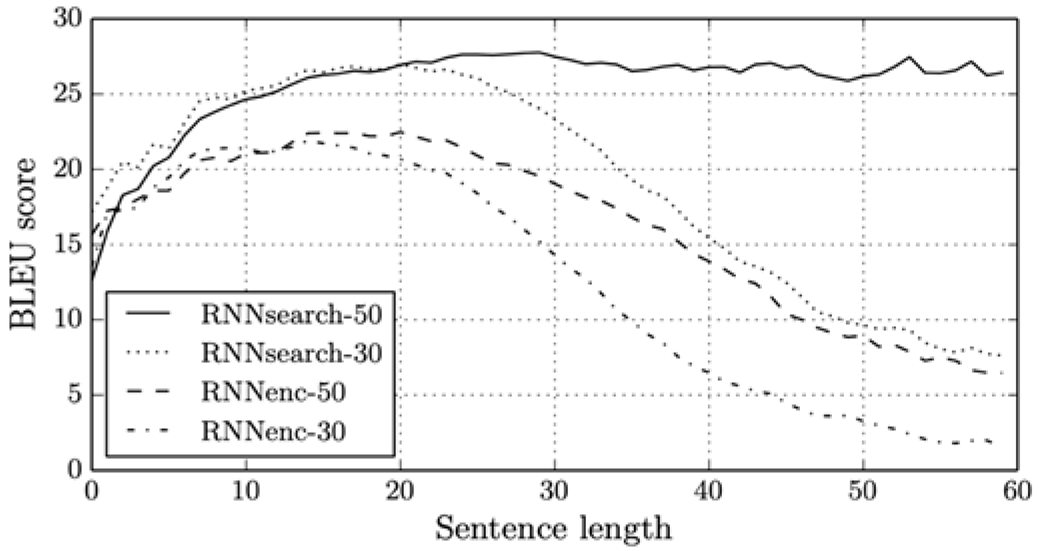


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

RESOURCES

- **Sequence to Sequence (seq2seq) and Attention**
  - https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
- **NLP From Scratch: Translation with a Seq2Seq Network and Attention**
  - https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html