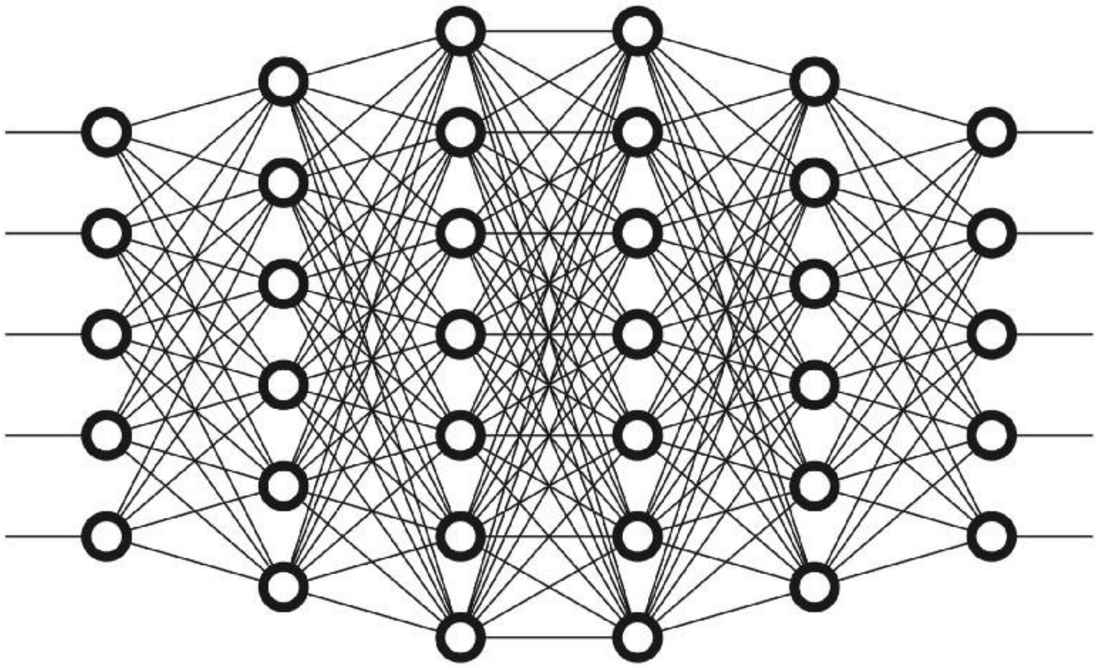
National Taiwan University of Science and Technology
Computer Science and Information Engineering

TEXT MINING TUTORIAL #8



Instructor: Prof. Hsing-Kuo Pao
TA: Zolnamar Dorjsembe (Zola)

Date: Apr 16, 2024

[RECAP] ONE HOT ENCODING & WORD EMBEDDING

(Source: www.dreamstime.com)

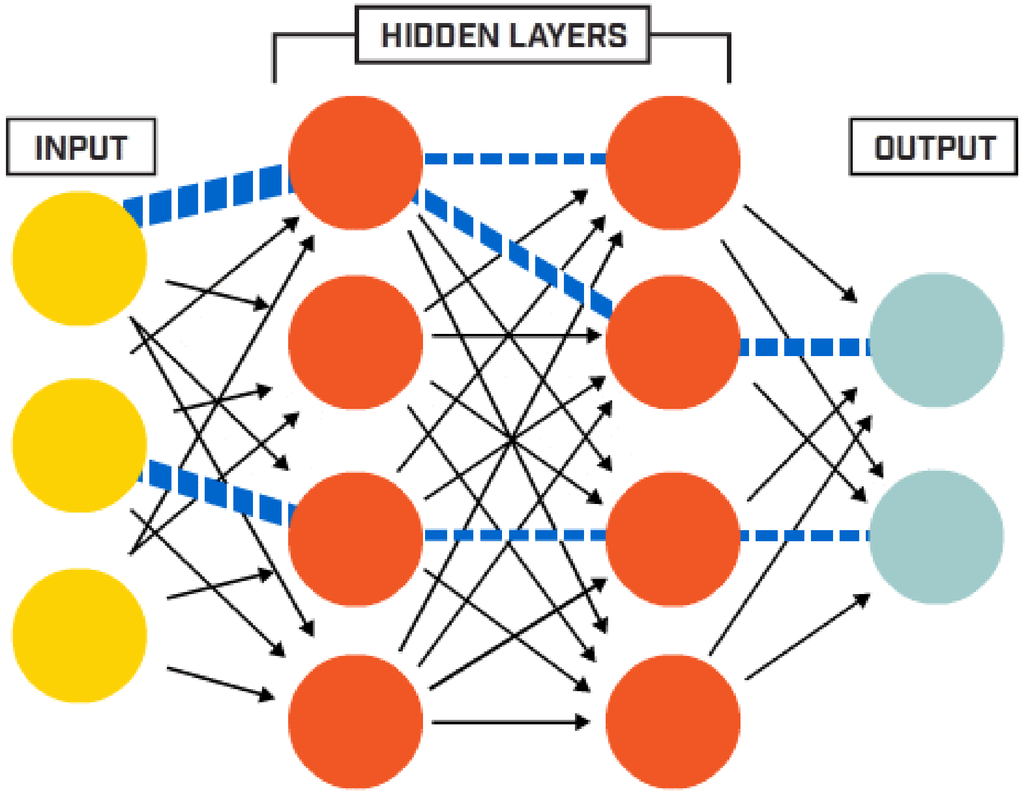

Join at www.kahoot.it
or with the Kahoot! app
Game PIN:
6710
Kahoot!
Waiting for players...

LEARN MORE: ONE-HOT ENCODING & WORD EMBEDDING

DeepLearning.AI: Dr. Andrew Ng’s lecture series

https://www.youtube.com/playlist?list=PLhWB2ZsrULv-wEM8JDKA1zk8_2Lc88I-s

ARTIFICIAL NEURAL NETWORK (ANN)



(Source: towardsai.net)

WHAT IS NEURAL NETWORK?
Try to define neural networks in your own words

WHAT IS NEURAL NETWORK?

A neural network is a mathematical model that takes in input data, applies a series of mathematical transformations (called "layers"), and produces an output. The model consists of multiple interconnected nodes (called "neurons") that perform mathematical operations on the input data. Each neuron takes in one or more inputs, multiplies them by a set of weights, and applies a non-linear activation function to the result. By adjusting the weights, the neural network can learn to recognize patterns and make predictions based on the input data.

WHAT ARE WEIGHTS AND BIASES IN A NEURAL NETWORK?

Try to define neural networks in your own words

WHAT ARE WEIGHTS AND BIASES IN A NEURAL NETWORK?

In a neural network, weights and biases are the parameters that are learned during the training process to make predictions based on the input data.

- Weights are the parameters that determine the strength of the connections between neurons in the network. They are multiplied by the input values at each layer to produce the output. In other words, the weights represent the importance or relevance of each input feature for the network's prediction.

- Biases are the parameters that are added to the weighted sum of inputs at each layer to help shift the output to a desired range. They represent the network's tendency to produce a certain output even when all inputs are zero.

What is the effect of not using a bias term in a neural network?

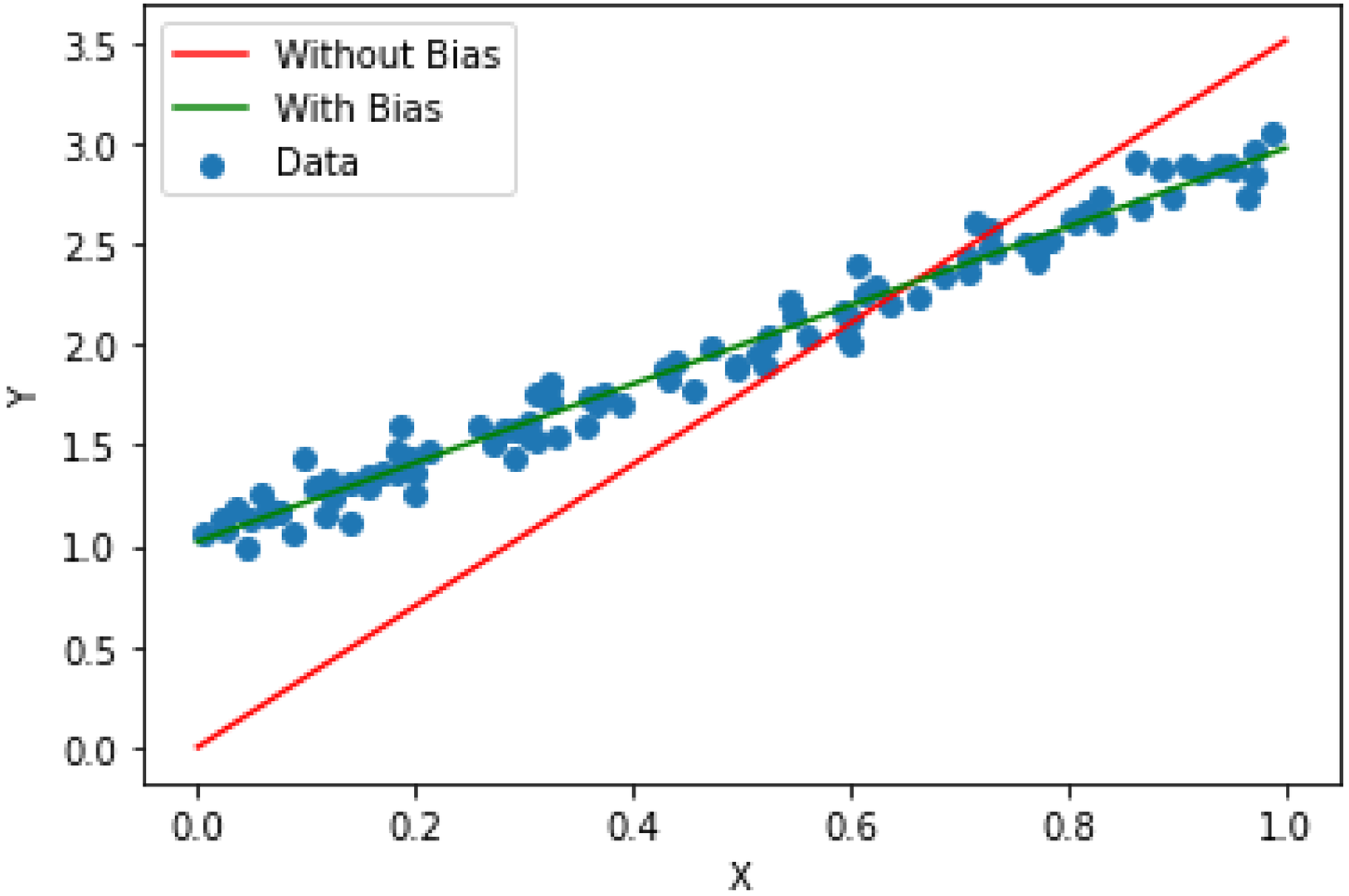
What is the effect of not using a bias term in a neural network?

If there is no bias in a neural network, the output of each neuron will be solely determined by the weighted sum of the inputs, which can limit the network's learning capacity and generalization ability. This is because the weights alone cannot account for all of the variations and biases in the input data. The bias term allows the network to shift the activation function and better fit the data.

Importance of Bias in Neural Networks

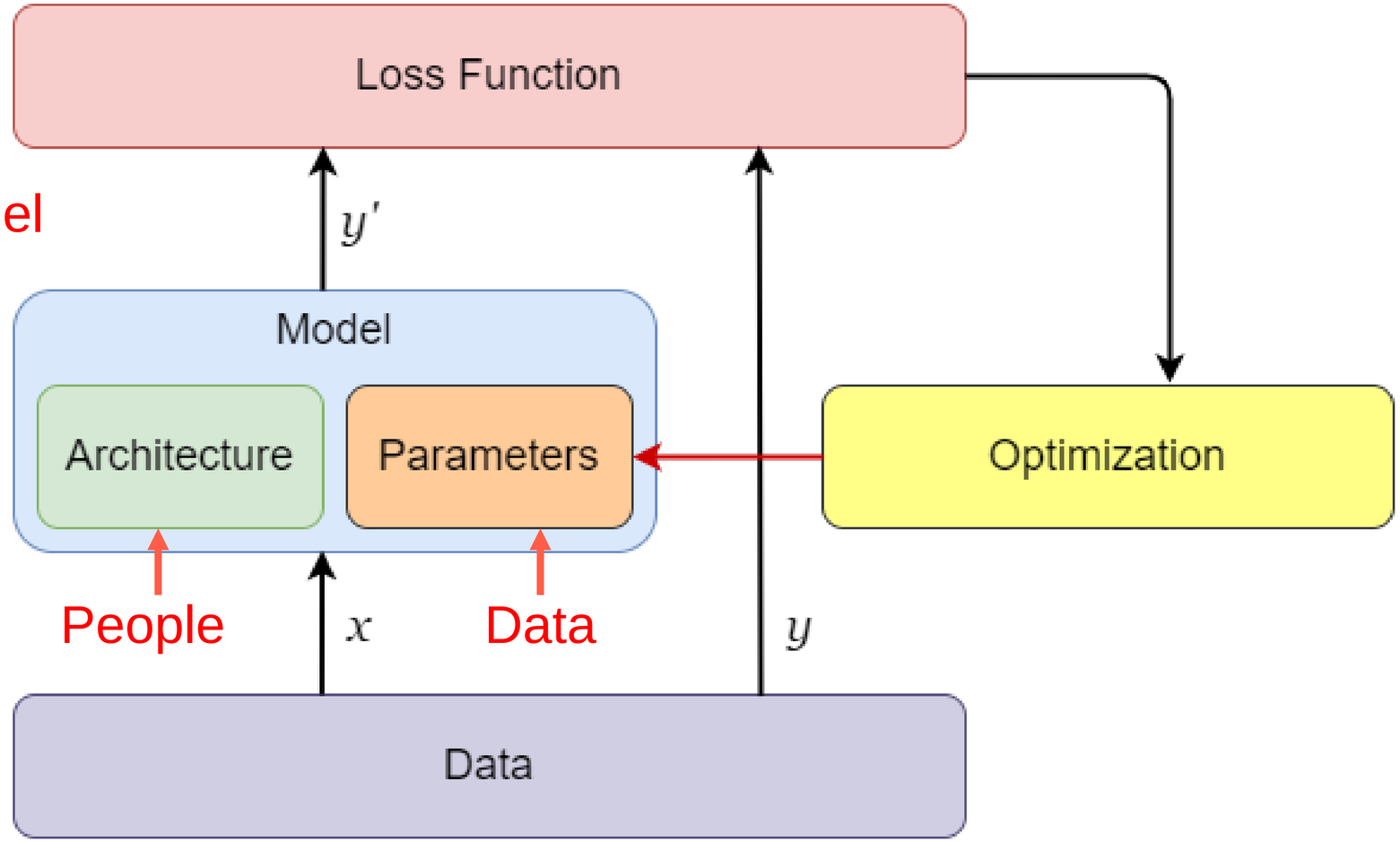
```python
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data
np.random.seed(42)
x_data = np.random.rand(100, 1)
y_data = 2 * x_data + 1 + 0.1 * np.random.randn(100, 1)

x_tensor = torch.tensor(x_data, dtype=torch.float32)
y_tensor = torch.tensor(y_data, dtype=torch.float32)

# Define the models
class LinearModelWithoutBias(nn.Module):
    def __init__(self):
        super(LinearModelWithoutBias, self).__init__()
        self.linear = nn.Linear(1, 1, bias=False)

    def forward(self, x):
        return self.linear(x)

class LinearModelWithBias(nn.Module):
    def __init__(self):
        super(LinearModelWithBias, self).__init__()
        self.linear = nn.Linear(1, 1, bias=True)

    def forward(self, x):
        return self.linear(x)

# Training function
def train(model, x, y, epochs, learning_rate):
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
    loss_fn = nn.MSELoss()

    for epoch in range(epochs):
        optimizer.zero_grad()
        output = model(x)
        loss = loss_fn(output, y)
        loss.backward()
        optimizer.step()

# Train models
model_without_bias = LinearModelWithoutBias()
model_with_bias = LinearModelWithBias()

train(model_without_bias, x_tensor, y_tensor, epochs=500, learning_rate=0.1)
train(model_with_bias, x_tensor, y_tensor, epochs=500, learning_rate=0.1)

# Plot the results
plt.scatter(x_data, y_data, label="Data")
x_test = np.linspace(0, 1, 100).reshape(-1, 1)
x_test_tensor = torch.tensor(x_test, dtype=torch.float32)
y_without_bias = model_without_bias(x_test_tensor).detach().numpy()
y_with_bias = model_with_bias(x_test_tensor).detach().numpy()

plt.plot(x_test, y_without_bias, 'r-', label="Without Bias")
plt.plot(x_test, y_with_bias, 'g-', label="With Bias")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```

Importance of Bias in Neural Networks



REMEMBER! NEURAL NETWORK OVERVIEW

Parametric model



SIMPLE ARTIFICIAL NEURAL NETWORK (ANN)

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Text preprocessing: Convert the entire string to lowercase and split it into individual words, removing
# This normalization helps ensure the model treats variations of the same word (e.g., with and without pu
text = "Tell me and I forget, teach me and I may remember, involve me and I learn"
words = text.lower().replace(',', '').split()

# Vocabulary creation: Map each unique word to a unique index. This numerical representation of words is
# because models can only process numbers, not text.
vocab = {word: i for i, word in enumerate(set(words))}
vocab_size = len(vocab)  # The size of the vocabulary

# Prepare the input and target pairs for training the model.
# The input will be sequences of words (context), and the target will be the word that follows the contex
inputs = []
targets = []
context_size = 3  # The number of words considered as context for predicting the next word.

# Generate sequences of contexts and their corresponding target words.
for i in range(len(words) - context_size):
    input_idx = [vocab[words[j]] for j in range(i, i + context_size)]  # Indices of the context words
    target_idx = vocab[words[i + context_size]]  # Index of the target word
    inputs.append(input_idx)
    targets.append(target_idx)
```

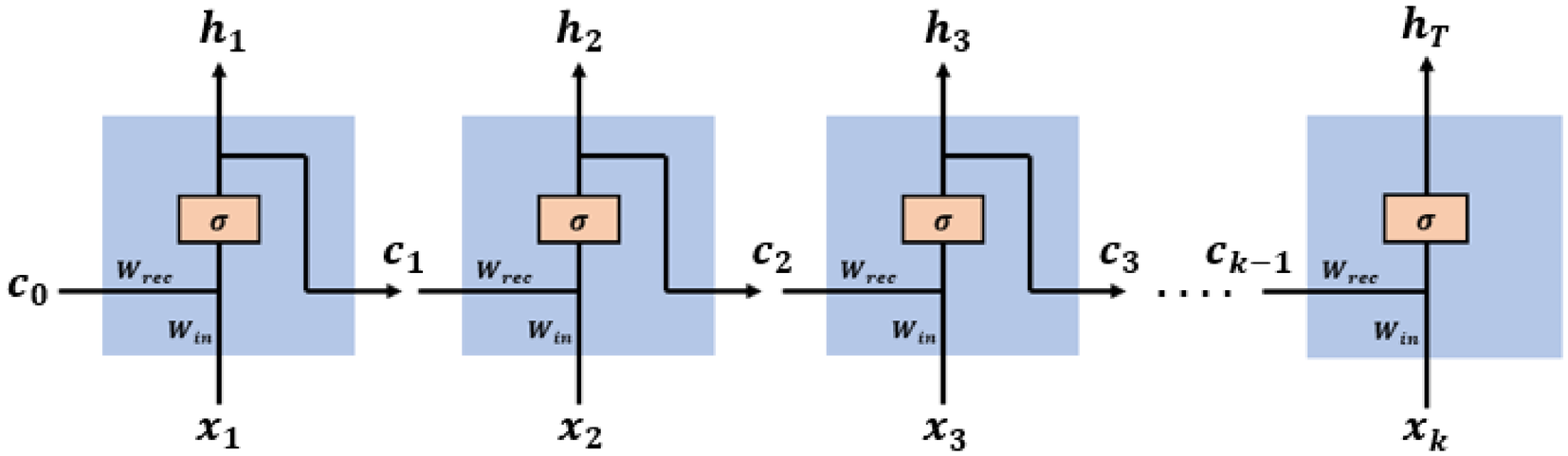
SIMPLE ARTIFICIAL NEURAL NETWORK (ANN)

```
29  # Convert the lists of indices into PyTorch tensors, which are optimized fo
30  inputs_tensor = torch.tensor(inputs, dtype=torch.long)
31  targets_tensor = torch.tensor(targets, dtype=torch.long)
32
33  # Packaging the tensors into a dataset and a DataLoader for efficient batch
34  dataset = TensorDataset(inputs_tensor, targets_tensor)
35  dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
36
37  # Define the neural network model structure.
38  class SimpleNN(nn.Module):
39      def __init__(self, vocab_size, embedding_dim):
40          super(SimpleNN, self).__init__()
41          self.embeddings = nn.Embedding(vocab_size, embedding_dim)  # Conver
42          self.fc1 = nn.Linear(embedding_dim * context_size, 50)     # First
43          self.fc2 = nn.Linear(50, 20)                               # Secona
44          self.fc3 = nn.Linear(20, vocab_size)                       # Output
45
46      def forward(self, inputs):
47          # Define the forward pass through the network.
48          embeds = self.embeddings(inputs).view((inputs.shape[0], -1))  # Fla
49          out = torch.relu(self.fc1(embeds))                          # Appl
50          out = torch.relu(self.fc2(out))                             # Anot
51          out = self.fc3(out)                                         # Outp
52          return out
53
```

SIMPLE ARTIFICIAL NEURAL NETWORK (ANN)

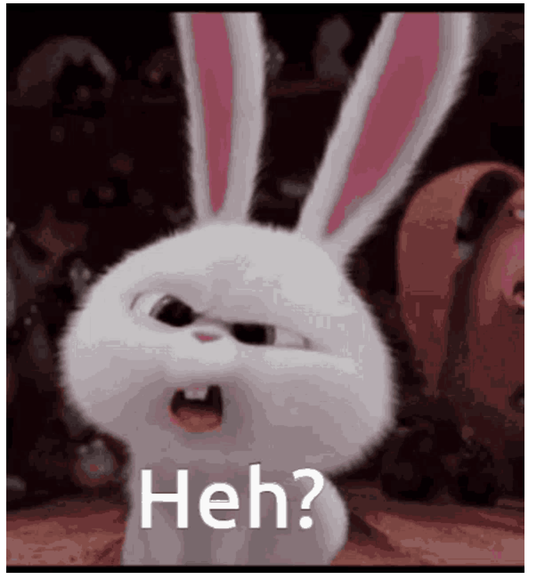
```
embedding_dim = 10  # Size of the embedding vectors. Smaller for simplicity in this exa

# Initialize the model, loss function, and optimizer.
model = SimpleNN(vocab_size, embedding_dim)
loss_function = nn.CrossEntropyLoss()  # Suitable for classification tasks with multipl
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam optimizer with a learning

# Training loop: Iterate over the data multiple times (epochs) to optimize the model pa
epochs = 300
for epoch in range(epochs):
    total_loss = 0
    for context, target in dataloader:
        model.zero_grad()            # Clear old gradients from the last step.
        log_probs = model(context)   # Calculate the log probabilities of the next word.
        loss = loss_function(log_probs, target)  # Compute the cross-entropy loss.
        loss.backward()              # Perform backpropagation to calculate gradients.
        optimizer.step()             # Update the weights.
        total_loss += loss.item()    # Accumulate the loss for monitoring.
    if epoch % 50 == 0:
        print(f'Epoch {epoch}, Loss: {total_loss / len(dataloader)}')  # Print loss ev

# Function to predict the next word given a string of text
def predict(text):
    """
    Predicts the next word based on the last few words of the input text using the trai
    This function preprocesses the text to fit the model's training setup, then perform
    """
    words = text.lower().replace(',', '').split()
    input_idx = [vocab.get(word, 0) for word in words[-context_size:]]  # Convert last
    input_tensor = torch.tensor([input_idx], dtype=torch.long)
    with torch.no_grad():
        log_probs = model(input_tensor)
    return max(zip(log_probs[0].exp(), vocab.keys()), key=lambda p: p[0])[1]  # Return

# Test the prediction function
print(predict("teach me and"))  # Expected to output 'I', given the training context.
```

RECURRENT NEURAL NETWORK



LET'S PLAY

CHINESE WHISPER GAME



LET’S PLAY

CHINESE WHISPER GAME

The objective of our game is for the last person to correctly predict the missing word in the sentence

LET'S PLAY CHINESE WHISPER GAME

The objective of our game is for the last person to correctly predict the missing word in the sentence

(Source: medium.com)

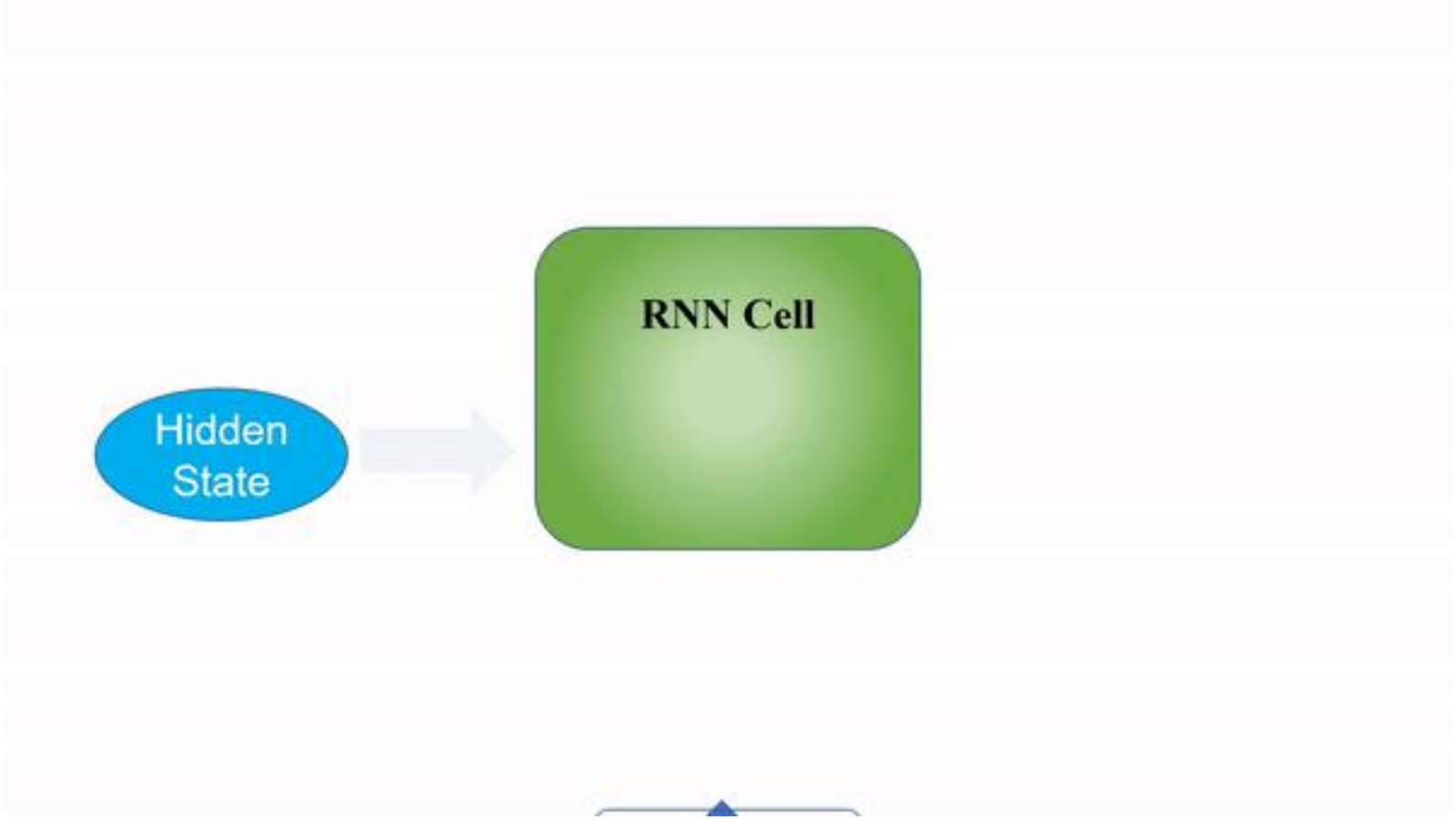
"Tell me and I forget, teach me and I may remember, involve me and I learn"

~ Benjamin Franklin

(Source: google.com)

Benjamin Franklin (January 17, 1706 – April 17, 1790) was an American polymath who was active as a writer, scientist, inventor, statesman, diplomat, printer, publisher, forger and political philosopher. Among the leading intellectuals of his time, Franklin was one of the Founding Fathers of the United States, a drafter and signer of the Declaration of Independence, and the first Postmaster General. (Source: Wikipedia)

SAME IDEA: RECURRENT MODEL



(Source: blog.floydhub.com)

SEQUENCE MODELING: RECURRENT & RECURSIVE NETS

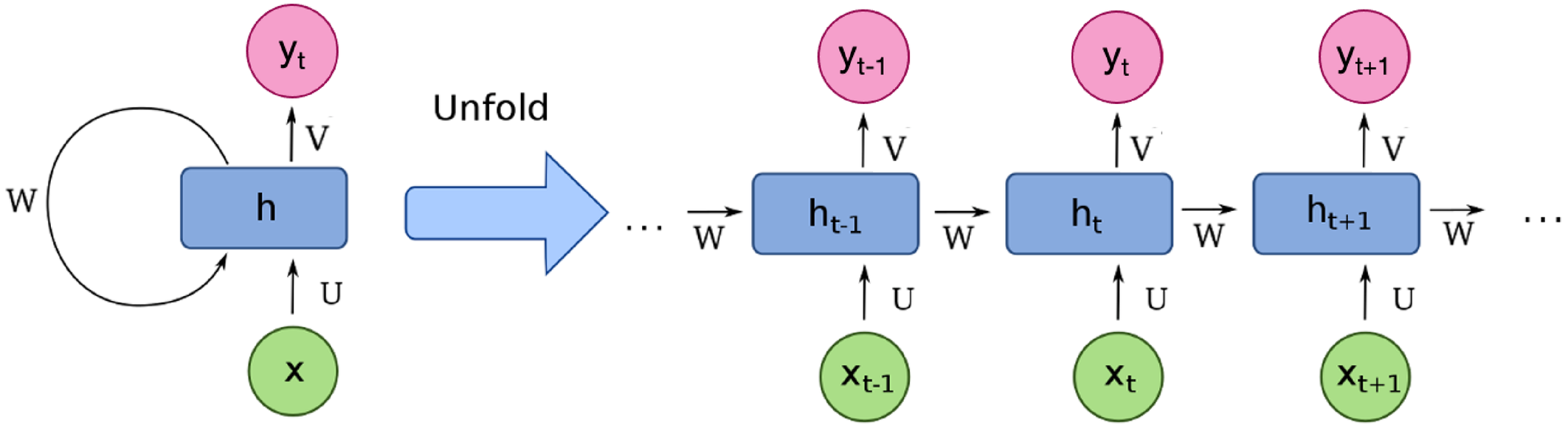
- Recurrent neural networks: (RNN; Rumelhart et al. 1986)
- A family of NNs for processing sequential data -- specialized to handle a sequence of inputs

$$\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$$

(can scale to much longer sequence & handle sequences with variable lengths)

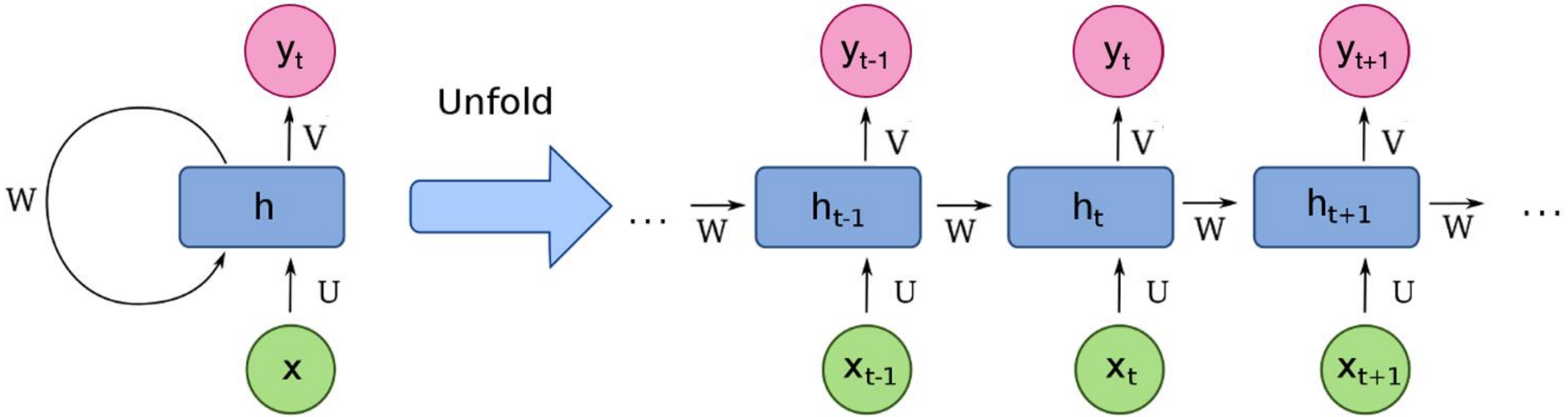
RECURRENT NEURAL NETWORK (RNN)

- Sequence data
- Feedforward vs Recurrent
- Memory

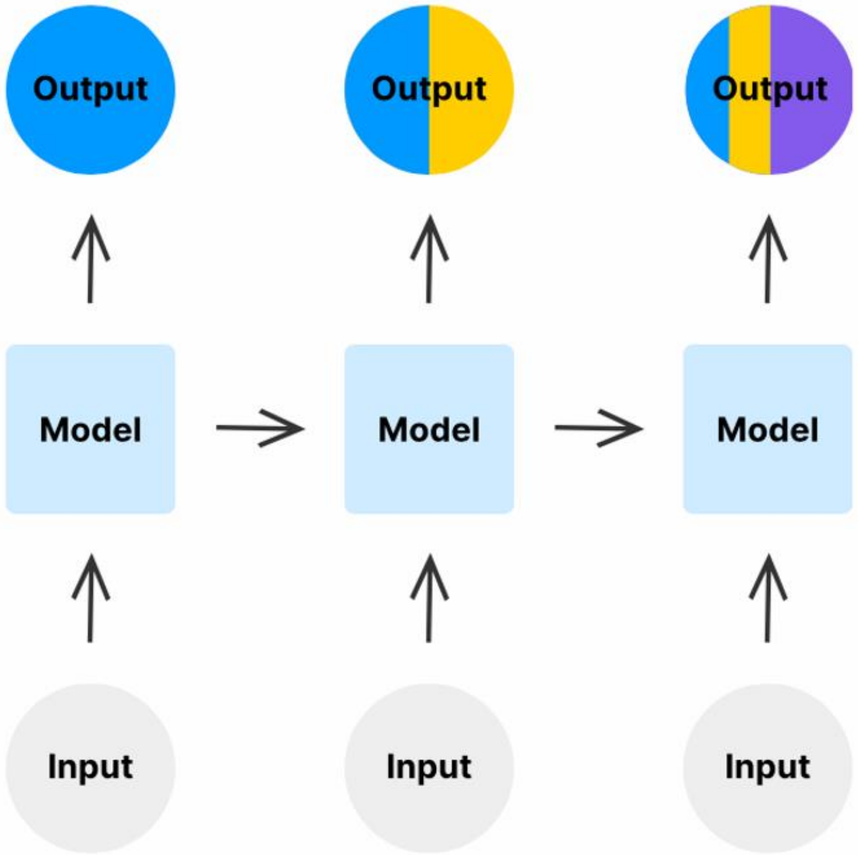


RECURRENT NEURAL NETWORK (RNN)

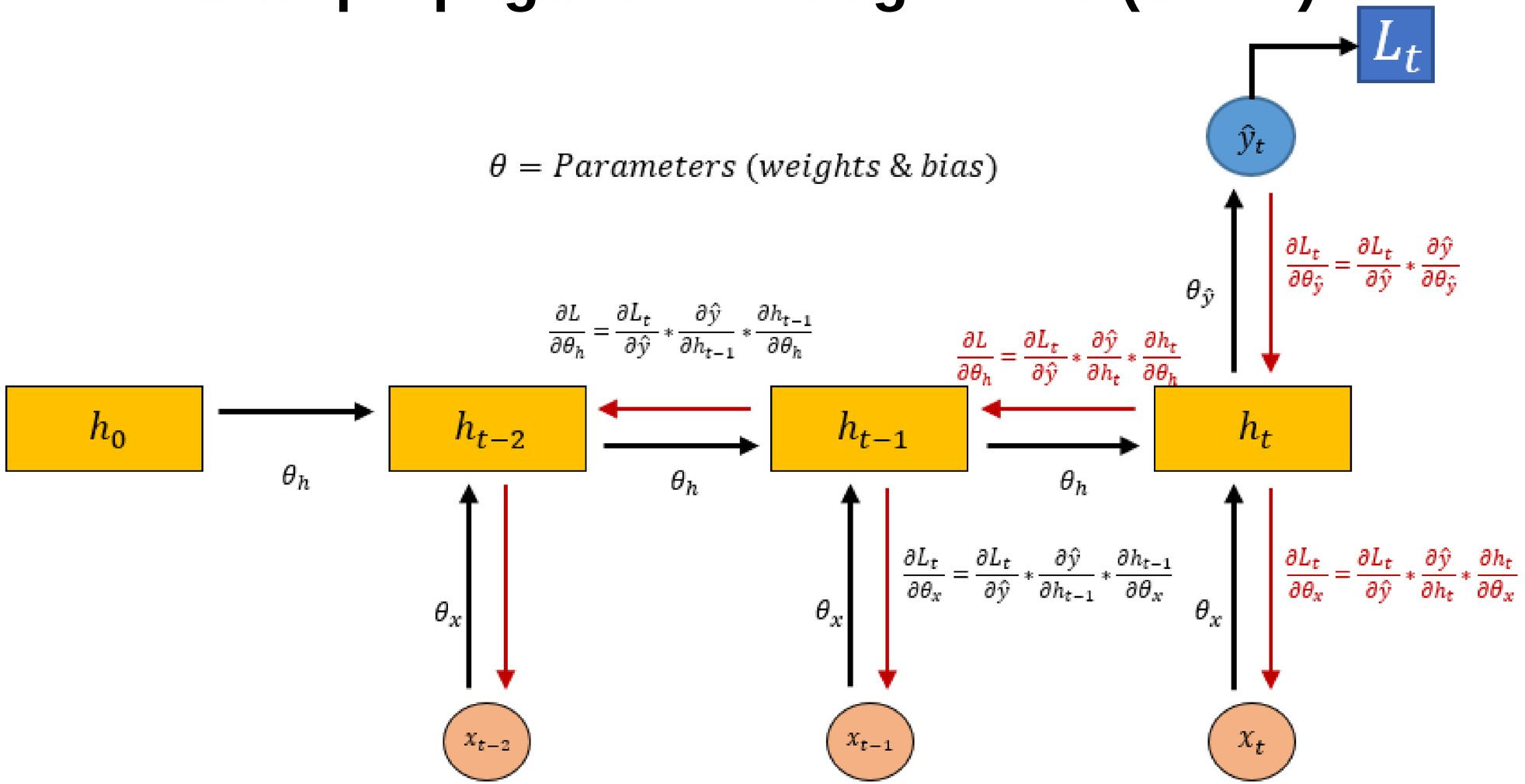
- Hidden layer: $h_t = \sigma(W h_{t-1} + U x_t + b)$
- Output: $y_t = V h_t$



RECURRENT NEURAL NETWORK (RNN)



RECURRENT NEURAL NETWORK (RNN)
Backpropagation Through Time (BPTT)

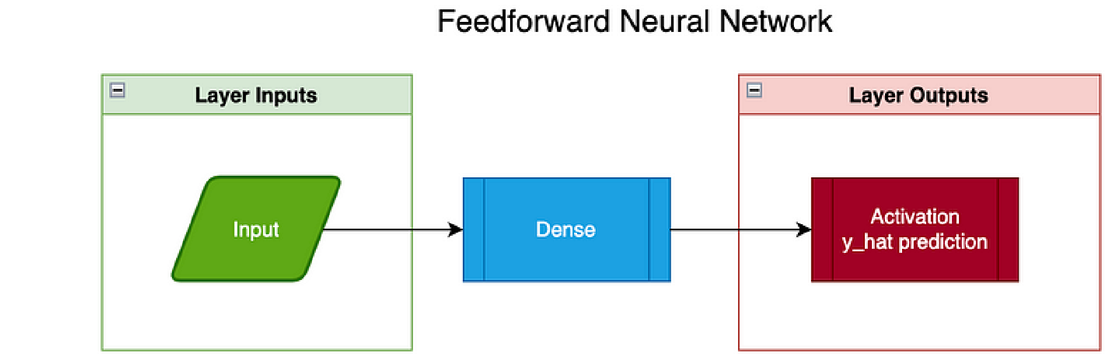


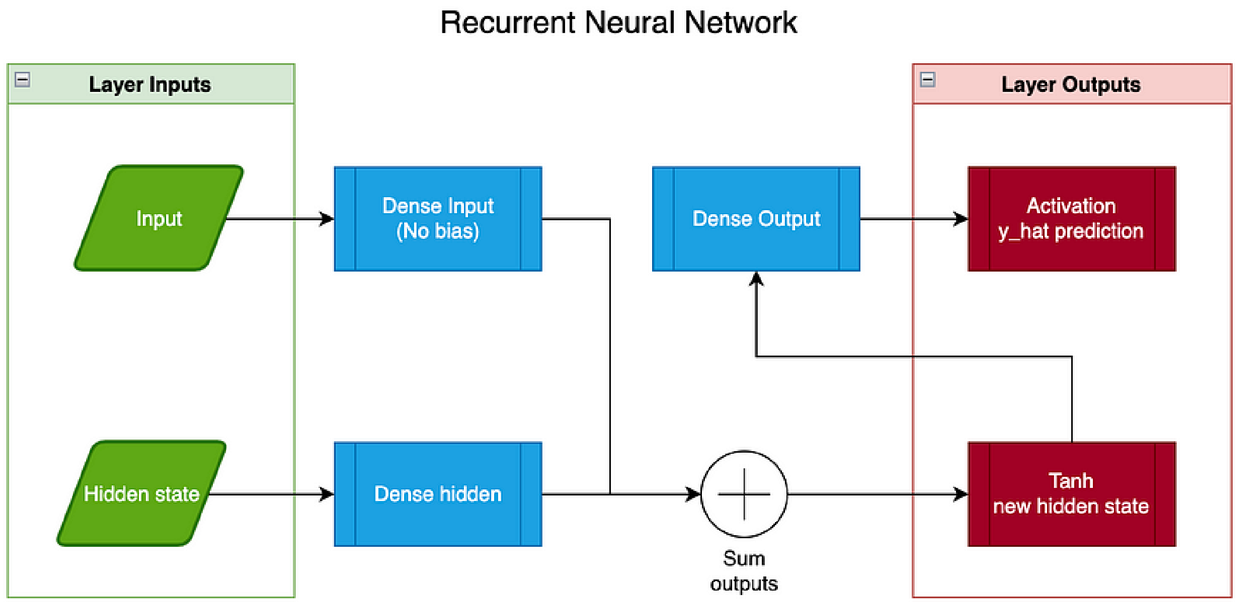
(More: https://www.youtube.com/watch?v=6EXP2-d_xQA&t=320s)

RNN VS FEEDFORWARD ARCHITECTURE

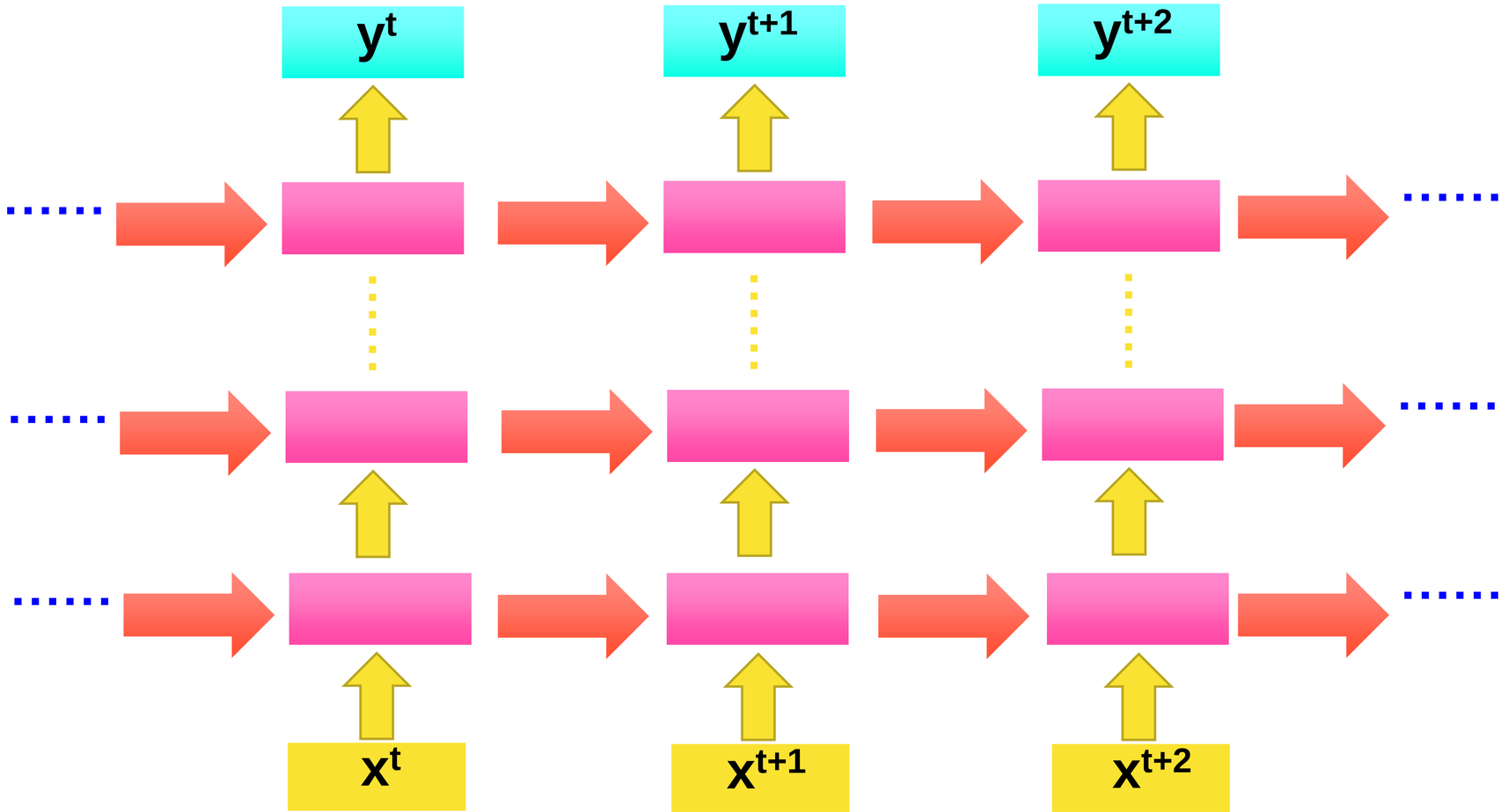
	Feedforward	RNN
Inputs	1	2
Weights	1	3
Outputs	1	2

(Source: Coding a Recurrent Neural Network (RNN) from scratch using Pytorch | by Diego Velez | Medium





RECURRENT NEURAL NETWORK (RNN)
Of course it can be deep



[Lee, H. (2020). *Recurrent Neural Network* [PowerPoint slides]. https://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/RNN.pptx]

TYPES OF RNN

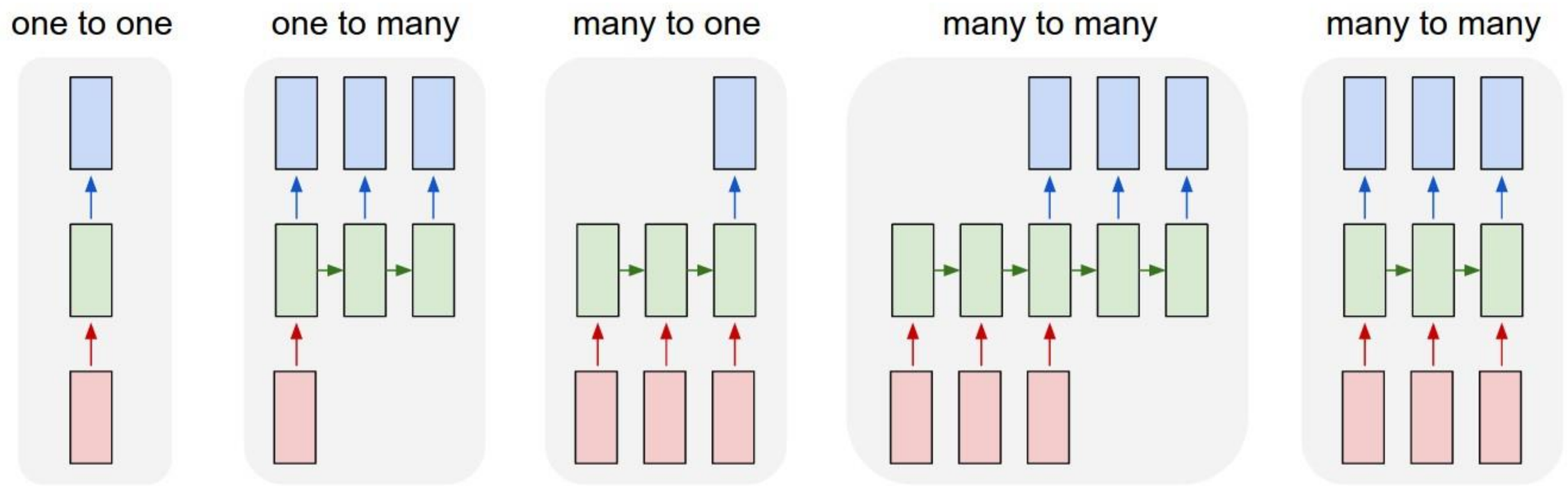


Image classification

Image captioning

Sentiment analysis

Language translation

Labeling each frame of video

(Source: google.com)

LET’S SEE AN EXAMPLE

RECURRENT NEURAL NETWORK (RNN)

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Text preprocessing
text = "Tell me and I forget, teach me and I may remember, involve me and I learn"
words = text.lower().replace(',', '').split()
vocab = {word: i for i, word in enumerate(set(words))}
vocab_size = len(vocab)

# Prepare data for the model
inputs = []
targets = []
context_size = 3

for i in range(len(words) - context_size):
    input_idx = [vocab[words[j]] for j in range(i, i + context_size)]
    target_idx = vocab[words[i + context_size]]
    inputs.append(input_idx)
    targets.append(target_idx)

inputs_tensor = torch.tensor(inputs, dtype=torch.long)
targets_tensor = torch.tensor(targets, dtype=torch.long)

# Create dataset and DataLoader
dataset = TensorDataset(inputs_tensor, targets_tensor)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
```

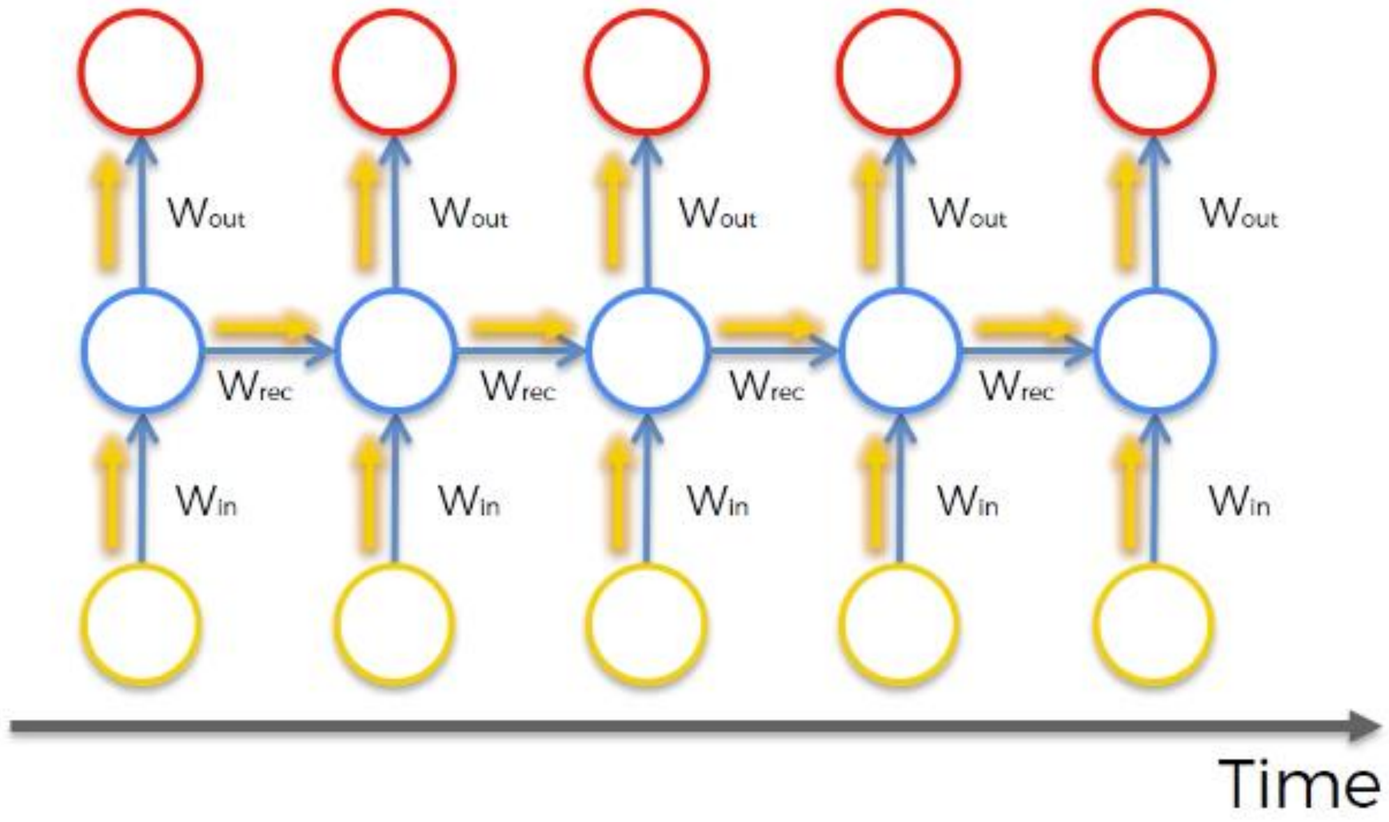
RECURRENT NEURAL NETWORK (RNN)

```python
# Define an RNN model
class SimpleRNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(SimpleRNN, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        # RNN layer, unidirectional and one layer
        self.rnn = nn.RNN(embedding_dim, hidden_dim, batch_first=True)
        # Output layer
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, inputs):
        # Input shape: (batch_size, sequence_length)
        embeds = self.embeddings(inputs)  # (batch_size, sequence_length, embedding_dim)
        rnn_out, _ = self.rnn(embeds)     # (batch_size, sequence_length, hidden_dim)
        # We use the last RNN output as the representation of the sequence
        final_output = rnn_out[:, -1, :] # (batch_size, hidden_dim)
        out = self.fc(final_output)       # (batch_size, vocab_size)
        return out

# Model parameters
embedding_dim = 10
hidden_dim = 20

# Initialize the RNN model, loss function, and optimizer
model = SimpleRNN(vocab_size, embedding_dim, hidden_dim)
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

RECURRENT NEURAL NETWORK (RNN)

```
58  # Training loop
59  epochs = 300
60  for epoch in range(epochs):
61      total_loss = 0
62      for context, target in dataloader:
63          model.zero_grad()
64          log_probs = model(context)
65          loss = loss_function(log_probs, target)
66          loss.backward()
67          optimizer.step()
68          total_loss += loss.item()
69      if epoch % 50 == 0:
70          print(f'Epoch {epoch}, Loss: {total_loss / len(dataloader)}')
71
72  # Prediction function
73  def predict(text):
74      words = text.lower().replace(',', '').split()
75      input_idx = [vocab.get(word, 0) for word in words[-context_size:]]
76      input_tensor = torch.tensor([input_idx], dtype=torch.long)
77      with torch.no_grad():
78          log_probs = model(input_tensor)
79      return max(zip(log_probs[0].exp(), vocab.keys()), key=lambda p: p[0])[1]
80
81  # Test the prediction function
82  print(predict("teach me and"))  # Expected to output 'I', given the training context.
83
```

Are there any problems with RNN? What do you think?



(Source: nickmccullum.com)

DISCUSSION