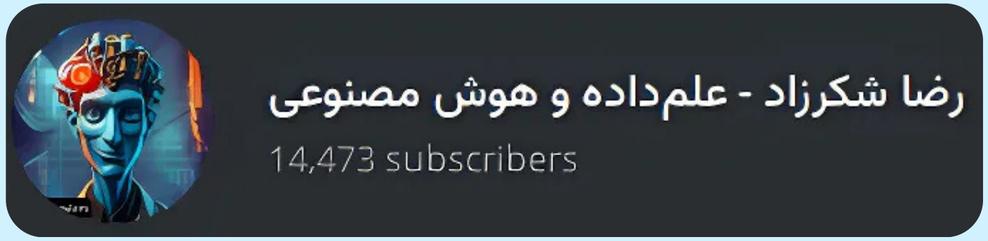
Reza Shokrzad

April 2025

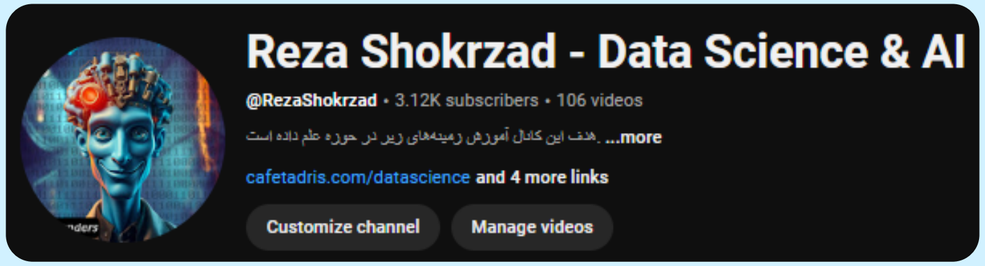
NLP Workshop

Session 4 - RNNs, LSTM, GRUs

Communities



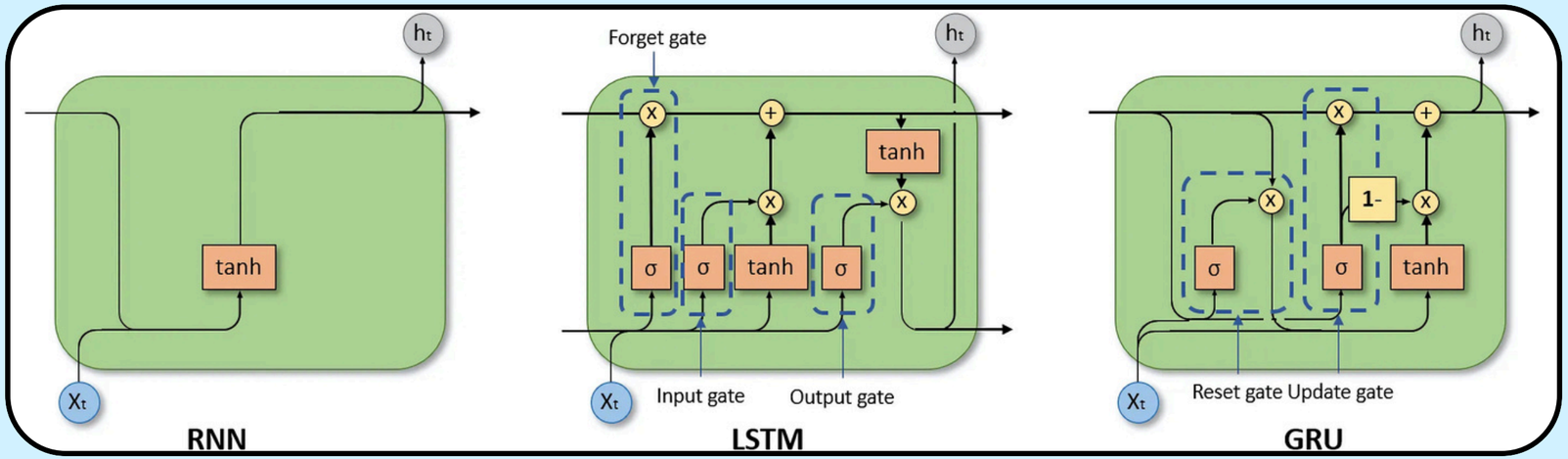
@DSLanders



@RezaShokrzad

Content

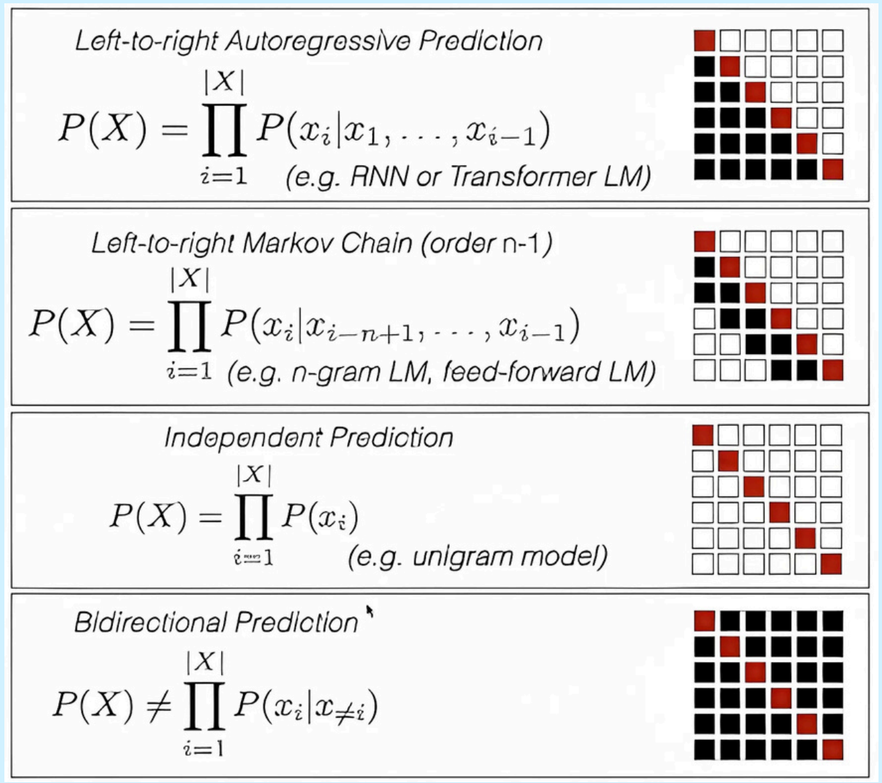
- Language Models
- RNNs
- LSTM: Long Short-Term Memory
- GRU: Gated Recurrent Unit
- Comparative Summary



Language Models

- **N-gram Models:** Predict next word based on previous word frequencies.
- **Hidden Markov Models (HMMs):** Model word sequences using hidden state transitions.
- **Maximum Entropy Models:** Balance feature constraints with maximum uncertainty.

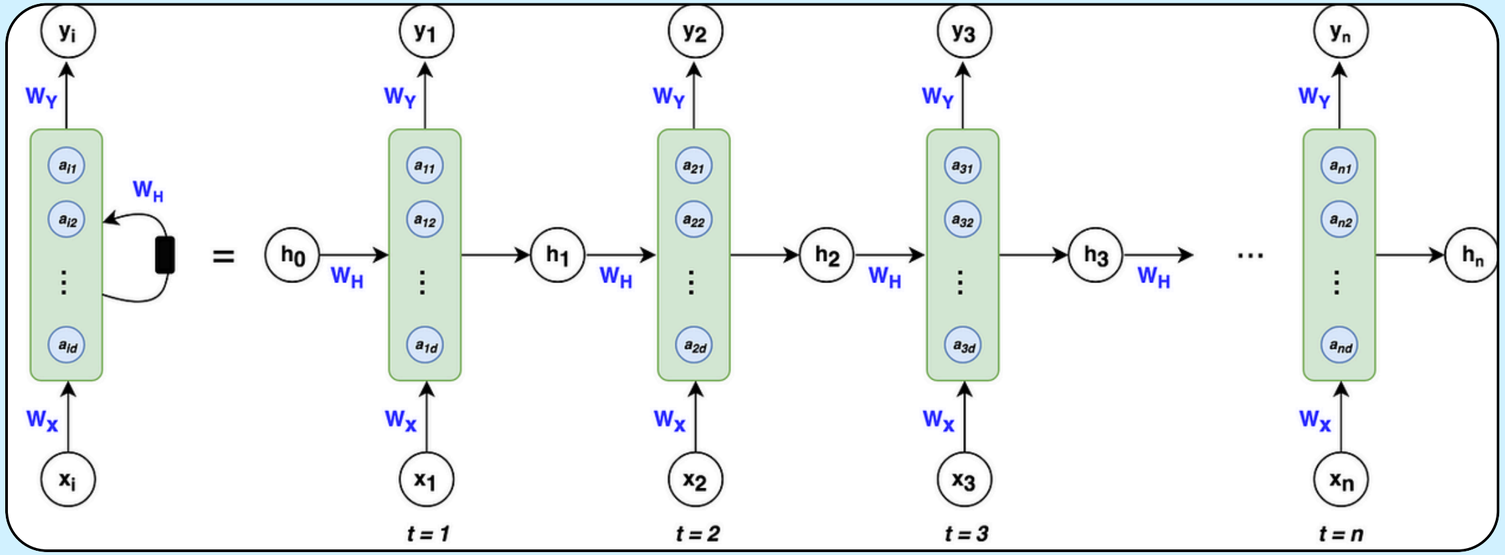
Limited context window;
data sparsity issues;
no semantic understanding.

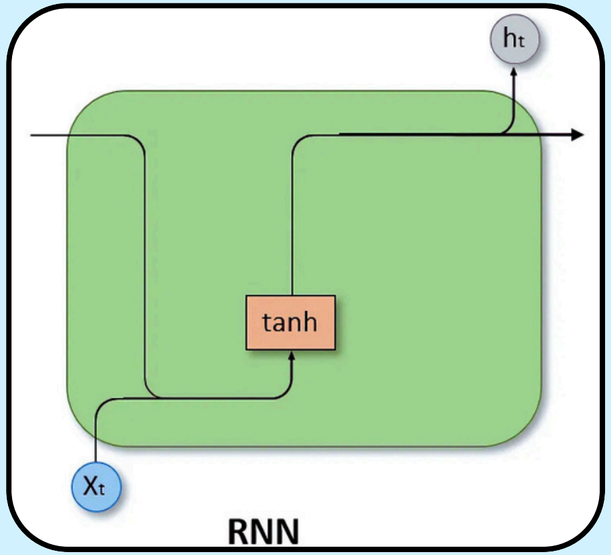


https://en.wikipedia.org/wiki/Language_model

RNNs: Recurrent neural networks

- Neural networks with feedback loops processing sequential data.
- Handle variable-length sequences; share parameters across time steps; maintain hidden state as memory.
- Struggle with long-range dependencies; suffer from vanishing/exploding gradients; sequential processing prevents parallelization.

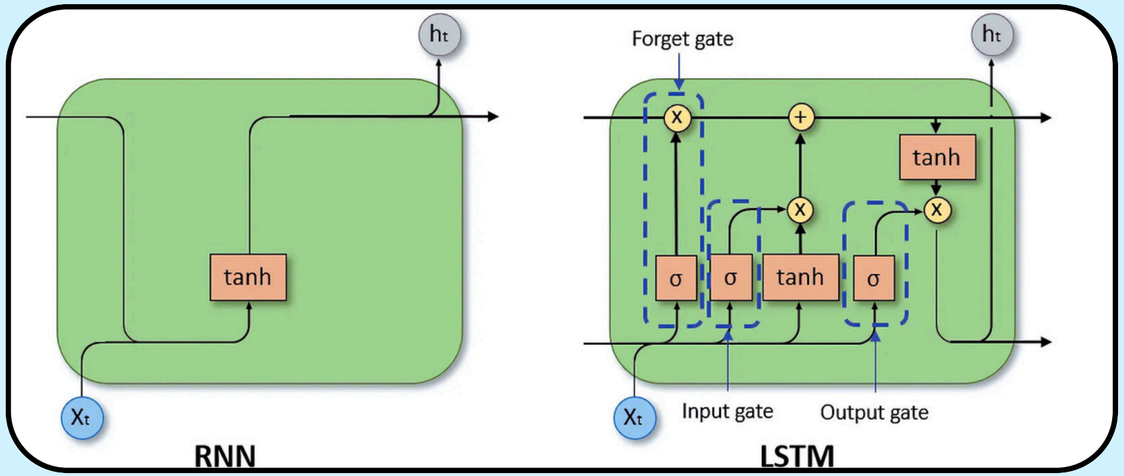




https://en.wikipedia.org/wiki/Recurrent_neural_network

LSTM: Long Short-Term Memory

- RNN variant with gated cells that control information flow through the network.
- Solve vanishing gradient problem; maintain long-term dependencies; use gates (input, forget, output) to regulate memory cell state.
- Computationally expensive; complex architecture with many parameters; still challenged by very long sequences despite improvements over RNNs.

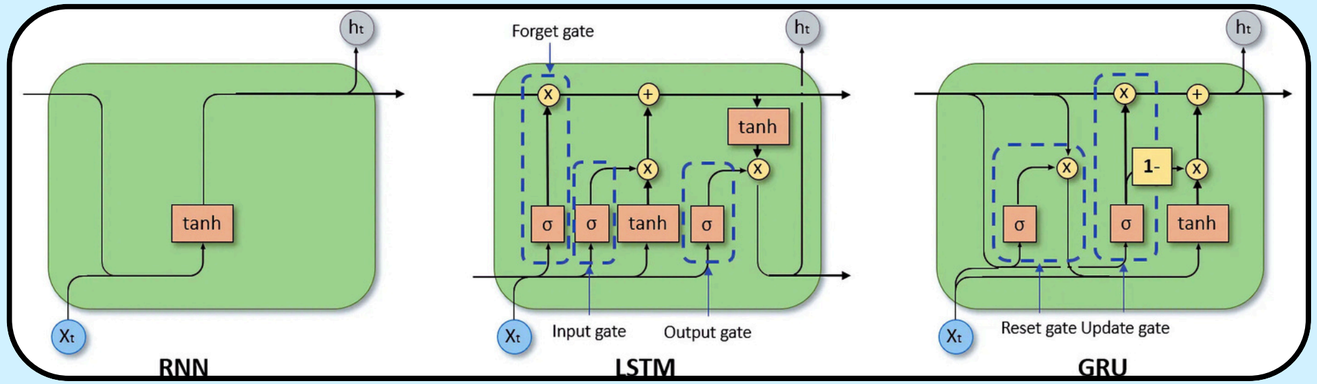


https://www.researchgate.net/publication/13853244_Long_Short-Term_Memory

https://en.wikipedia.org/wiki/Long_short-term_memory

GRU: Gated Recurrent Unit

- Simplified RNN variant with gating mechanisms that regulate information flow.
- Fewer parameters than LSTMs; combine forget/input gates into update gate; use reset gate to control past information influence.
- May underperform LSTMs on certain complex sequence tasks; less memory capacity than LSTMs; still susceptible to long-range dependency challenges.



https://arxiv.org/abs/1406.1078

https://en.wikipedia.org/wiki/Gated_recurrent_unit

Comparative Summary

- **Architecture:**
  - RNNs use simple feedback loops;
  - LSTMs add cell state and three gates;
  - GRUs simplify with two gates (reset and update).
- **Memory capability:**
  - RNNs struggle with long sequences;
  - LSTMs excel at long-term dependencies;
  - GRUs perform comparably to LSTMs with simpler structure.
- **Computational efficiency:**
  - RNNs fastest but least capable;
  - GRUs balance efficiency and performance;
  - LSTMs most powerful but computationally expensive.
- **Training complexity:**
  - RNNs prone to vanishing gradients;
  - LSTMs and GRUs designed to mitigate this;
  - GRUs typically converge faster than LSTMs.
- **Practical applications:**
  - RNNs for simple sequences;
  - LSTMs preferred for complex tasks requiring precise memory;
  - GRUs often sufficient for many applications with better speed.

Variants

RNNs

- **Simple/Vanilla RNN**
  - Basic feedback connections allowing networks to maintain memory of previous inputs.
- **Bidirectional RNN**
  - Processes sequences in both forward and backward directions for better context.
- **Deep RNN**
  - Multiple recurrent layers stacked for capturing hierarchical temporal patterns.
- **Recursive Neural Network**
  - Applies same weights recursively over hierarchical tree-like structures.

LSTM

- **Standard LSTM**
  - Solves vanishing gradient problem using gates to control information flow.
- **Peephole LSTM**
  - Adds connections from cell state to gates for finer timing control.
- **Convolutional LSTM**
  - Combines convolutional operations with LSTM for spatial-temporal data processing.
- **Bidirectional LSTM**
  - Processes sequence data in both directions for comprehensive context understanding.
- **MultiLayer/Stacked LSTM**
  - Multiple LSTM layers capturing different temporal abstraction levels.

GRUs

- **Standard GRU**
  - Simplified LSTM with fewer parameters using reset and update gates.
- **Minimal Gated Unit**
  - Further simplifies GRU by combining reset and update gates.
- **Bidirectional GRU**
  - Processes sequences forward and backward for richer contextual representation.

Others

- **Neural Turing Machine**
  - RNN with external memory and attention for complex algorithmic tasks.
- **Transformer (Self-Attention)**
  - Replaced RNNs with attention mechanisms for better parallel processing.
- **Quasi-Recurrent Neural Network**
  - Combines convolution and pooling for faster parallel processing.
- **Multiplicative RNN/LSTM**
  - Uses multiplicative connections for enhanced expressivity in sequence modeling.

PyTorch

```python
# Simple RNN
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleRNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        out = self.fc(out[:, -1, :])  # Take final hidden state
        return out
```

```python
# LSTM
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, (hidden, _) = self.lstm(x)
        out = self.fc(out[:, -1, :])  # Take final hidden state
        return out
```

```python
# GRU
class GRUModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(GRUModel, self).__init__()
        self.gru = nn.GRU(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, hidden = self.gru(x)
        out = self.fc(out[:, -1, :])  # Take final hidden state
        return out
```

https://pytorch.org

Keras

```python
# Simple RNN
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

rnn_model = Sequential([
    SimpleRNN(64, input_shape=(sequence_length, input_size), return_sequences=False),
    Dense(output_size, activation='softmax')
])
```

```python
# LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

lstm_model = Sequential([
    LSTM(64, input_shape=(sequence_length, input_size), return_sequences=False),
    Dense(output_size, activation='softmax')
])
```

```python
# GRU
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense

gru_model = Sequential([
    GRU(64, input_shape=(sequence_length, input_size), return_sequences=False),
    Dense(output_size, activation='softmax')
])
```

https://keras.io

مبحث چهارم در یک نگاه: NLP

شبکه‌های عصبی بازگشتی

* RNNs
* LSTM
* GRUs

→ داده‌های ترتیبی (Sequential)
- Text
- داده‌های سری زمانی (Time_Series)
- ..., Anomaly detection, financial (Stock Market)

▷ Language Model

1950s → Noam Chamsky → Language Model

(مدل زبان)

1980 → Statistical LM

Probabilistic

<cls> I am a teacher. <eos>

↑ توکن شروع جمله

↑ توکن پایان جمله

$P(\overbrace{w_1}^{I} \mid <cls>) = 0.02$

$P(\overbrace{w_2}^{am} \mid <cls>, \overbrace{w_1}^{I}) = 0.05$

$P(w_3 \mid <cls>, w_1, w_2) = 0.04$

$Pr(\overbrace{w_4}^{teacher} \mid <cls>, w_1, w_2, w_3) = 0.001$

محدودیت ← میل به سمت صفر

$$P(S) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1,w_2) \cdots P(w_n|w_1 \cdots w_{n-1})$$

---

n-grams: unigram / bigram / trigram / 4-gram

* *

bigram ↓

$$P(S) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_2) \cdots P(w_n|w_{n-1})$$

↳ میل به صفر کمتر ← محدودیت ← Context_free

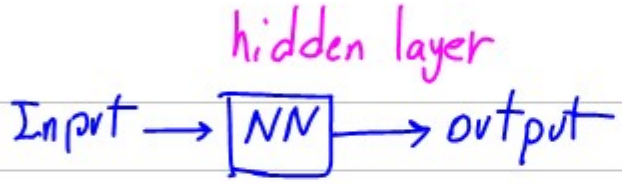
Iran is a large country. ....

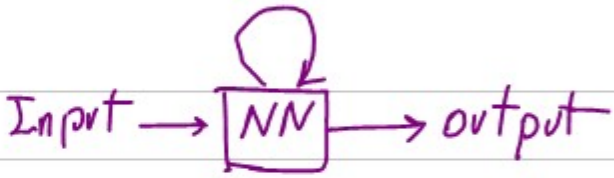
. . .

. . .

In this country

→ RNNs





CNNs

(Image Processing)

RNNs

(Text Processing)

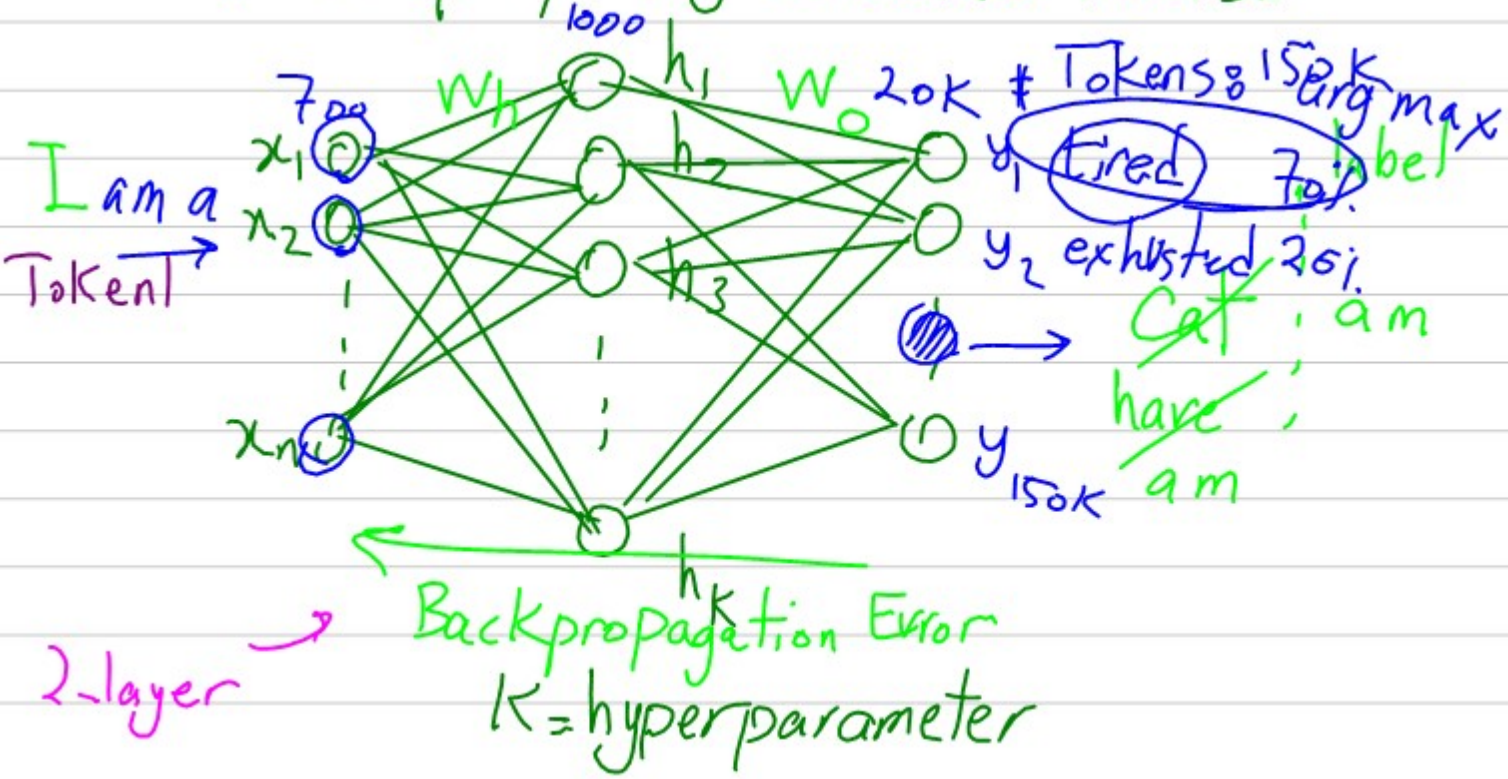
Input → embedding

	x_1 ..... x_n (700)
Token1	20K × 700
T_2 (20K)	
T_3	
⋮	

Parameter (W) ماشین یادگیری

hyperparameter تعیین ML تنظیم

Multi-layer Perceptron / Fully Connected / dense



بزرگترین مقدار ویژه W

$\lambda_{max} = 1 \longrightarrow$ Context window (infinit)

$\lambda_{max} < 1 \longrightarrow$ Vanishing Gradient (کوچک شدن گرادیان)

$\lambda_{max} > 1 \longrightarrow$ Exploding Gradient (انفجار گرادیان)

RNN limitations: memory

LSTM: Long Short-term Memory (حافظه کوتاه مدت طولانی)

(Skip Connection) Res_Net (این ایده)

رفع محوشدگی → Skip Connection

رفع مشکل انفجار → clipping & Min(1, $\frac{\nabla}{\|\nabla\|}$)

Attention is all you need 2017

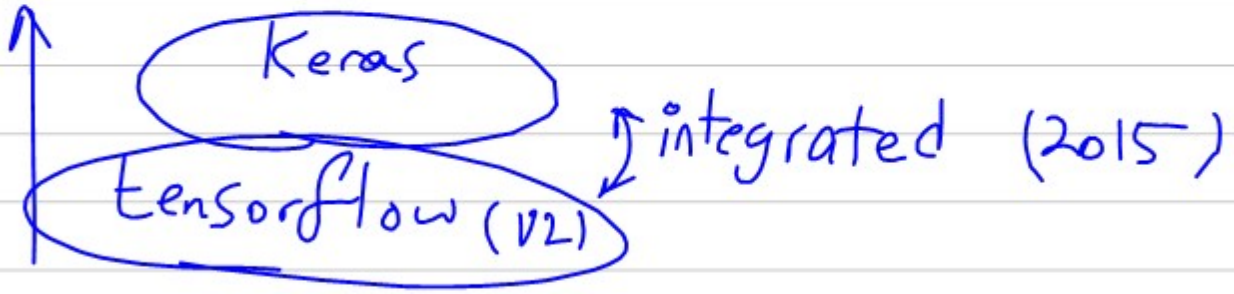
were RNNs all we need 2024

Deep learning Python Packages

1. PyTorch (Facebook) Academic — 2008

2. Keras (Chollet) 2014/2015 Industry
   - Google / Tensorflow (V1)

*



Text Generation:

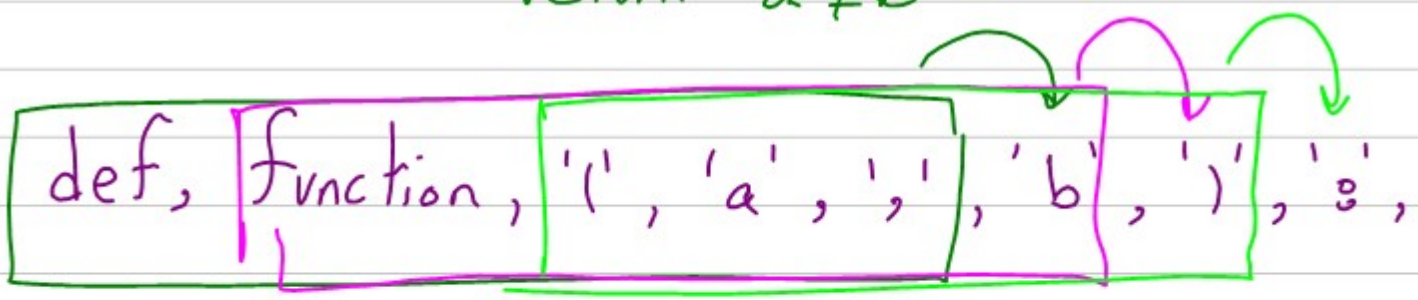
- Summarization
- Code generation
- Topic Modeling
- ⋮

Task: Code Generation (Python)

Dataset: Code Corpus

Replace

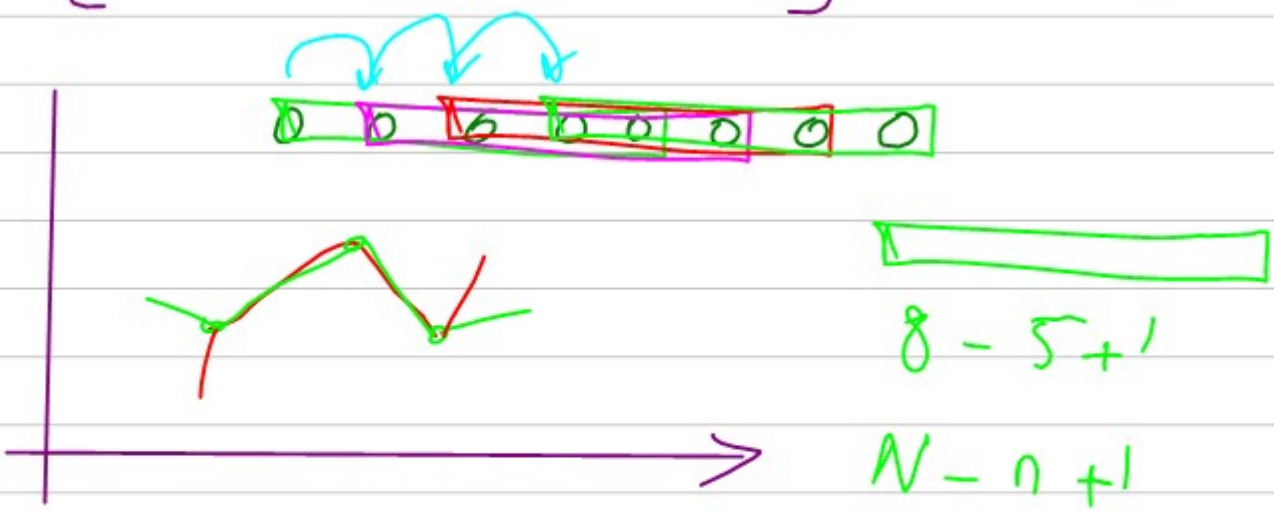
```
def function(a, b):\n
    return a + b
```



context_window = 5

S1: [def, function, (, a, ,] → T1: [b]

S2: [function, (, a, ,, b] → T2: [)]



free dataset :

- Github
- Kaggle
- HuggingFace (datasets)
- .
- .
- .

TPU : سرورهای GPU گوگل → (NNs)

GPU : Graphical Process Unit

واحد پردازش گرافیکی

↳ بازی ⟶ DL (NNs)

$\{a, a, b\} = \{a, b\}$

```
from sklearn.decomposition import PCA

pca = PCA(n_component=3)  #instance/object
pca.fit_transform(X_train)
pca.transform(X_test)

pca.singular_values

pca_5 = PCA(n_component=5)
```

class

object Oriented Programming (oop)