MF20006: Introduction to Computer Science

Lecture 9: Artificial Intelligence II

Hui Xu

xuh@fudan.edu.cn



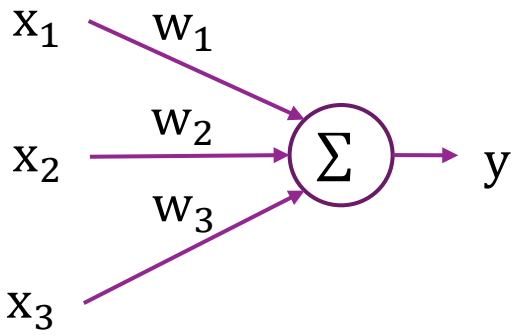
Outline

1. Neural Networks
2. Recurrent Neural Networks
3. Large Language Models

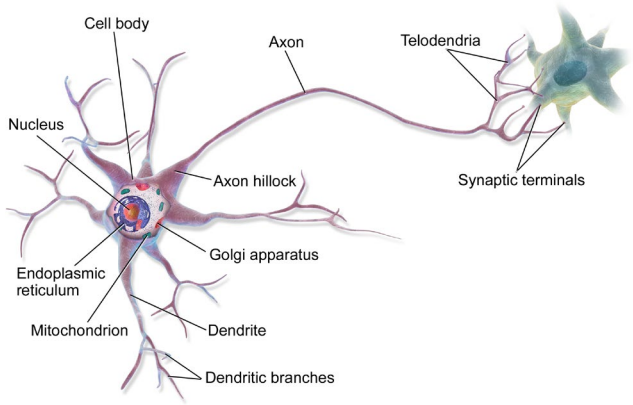
1. Neural Networks

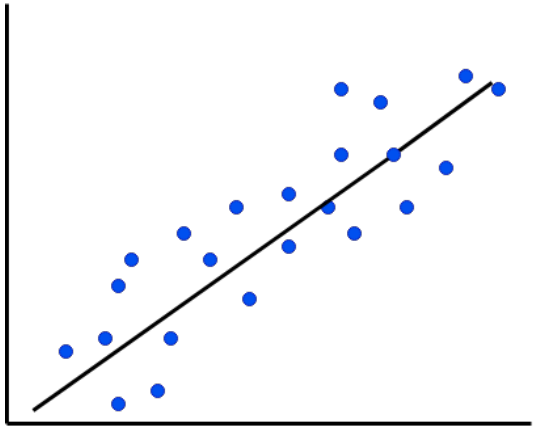
Perceptron

- One of the earliest models of a neuron in artificial intelligence.
- Introduced by Frank Rosenblatt (American psychologist) in 1958.



$$y = \sum_i w_i x_i + b$$

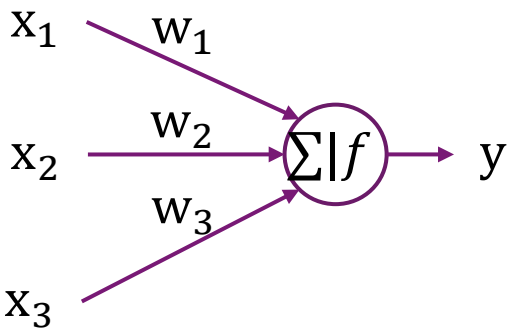




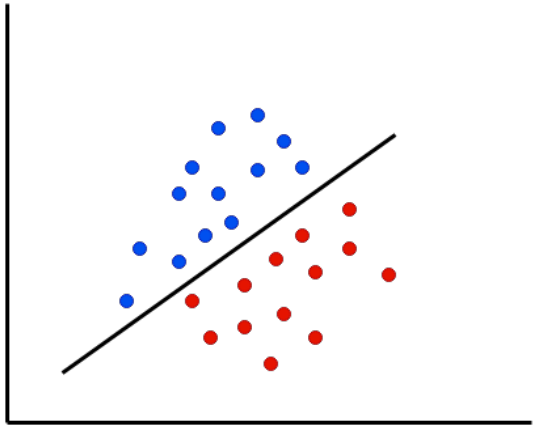
Linear Regression

General Form of Perceptron

- Introduce an activation function f into the neuron.



$$y = f\left(\sum_i w_i x_i + b\right)$$



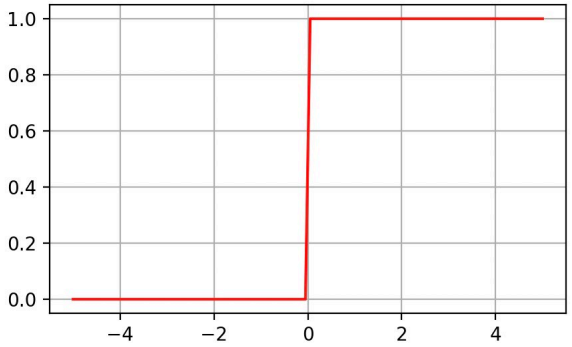
Linear Binary Classification

$$y = \begin{cases} 1, & \text{if } \left(\sum_i w_i x_i + b\right) > t \\ 0, & \text{otherwise} \end{cases}$$

Activation Functions

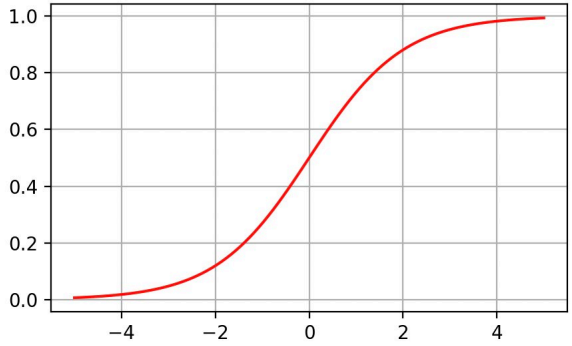
Step Function

$$y = \begin{cases} 1, & \text{if } x > t \\ 0, & \text{otherwise} \end{cases}$$



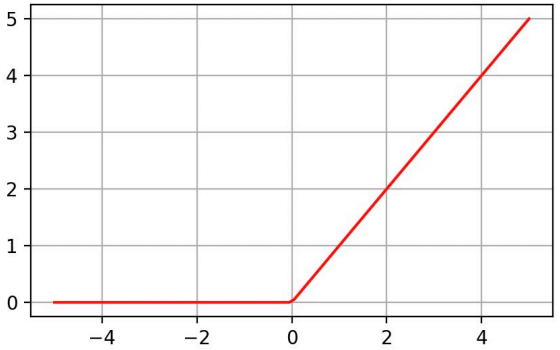
Sigmoid Function

$$y = \frac{1}{1 + e^{-x}}$$



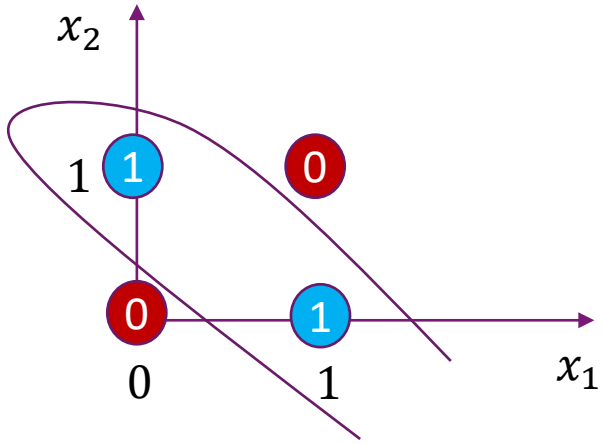
ReLU Function

$$y = \max(0, x)$$



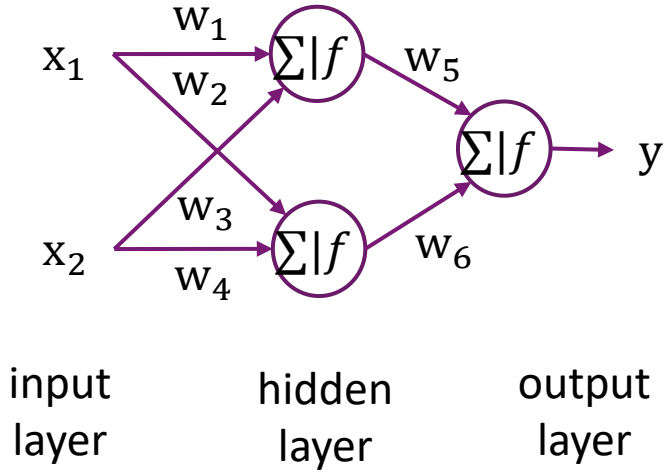
Limitation of Single-layer Perceptron

- A single-layer perceptron is essentially linear regression followed by a fixed nonlinearity.
- It cannot model general nonlinear functions.
- It can only classify linearly separable data.
  - For example, XOR is not linearly separable.
  - Even with common monotonic nonlinear activations (step, sigmoid, ReLU), the decision boundary remains a linear hyperplane.

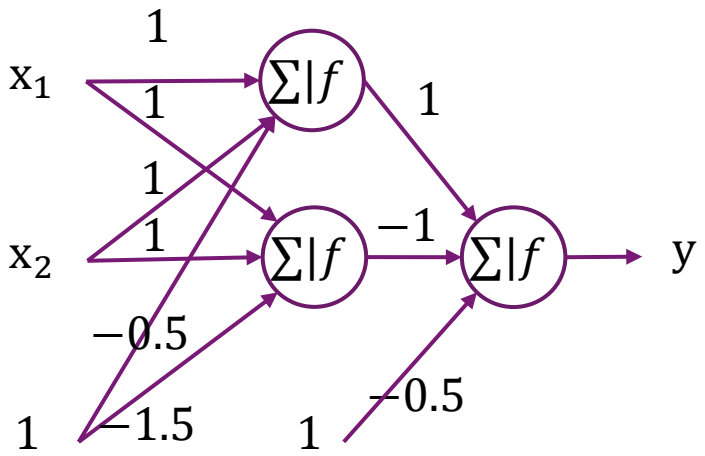


Multi-layer Perceptron

- **MLP stacks perceptron into layers:**
  - An input layer
  - One or multiple hidden layers
  - An output layer
- **With nonlinear activations in the hidden layers, the network composes nonlinearities and becomes capable of learning arbitrary nonlinear mappings.**



Example MLP for XOR



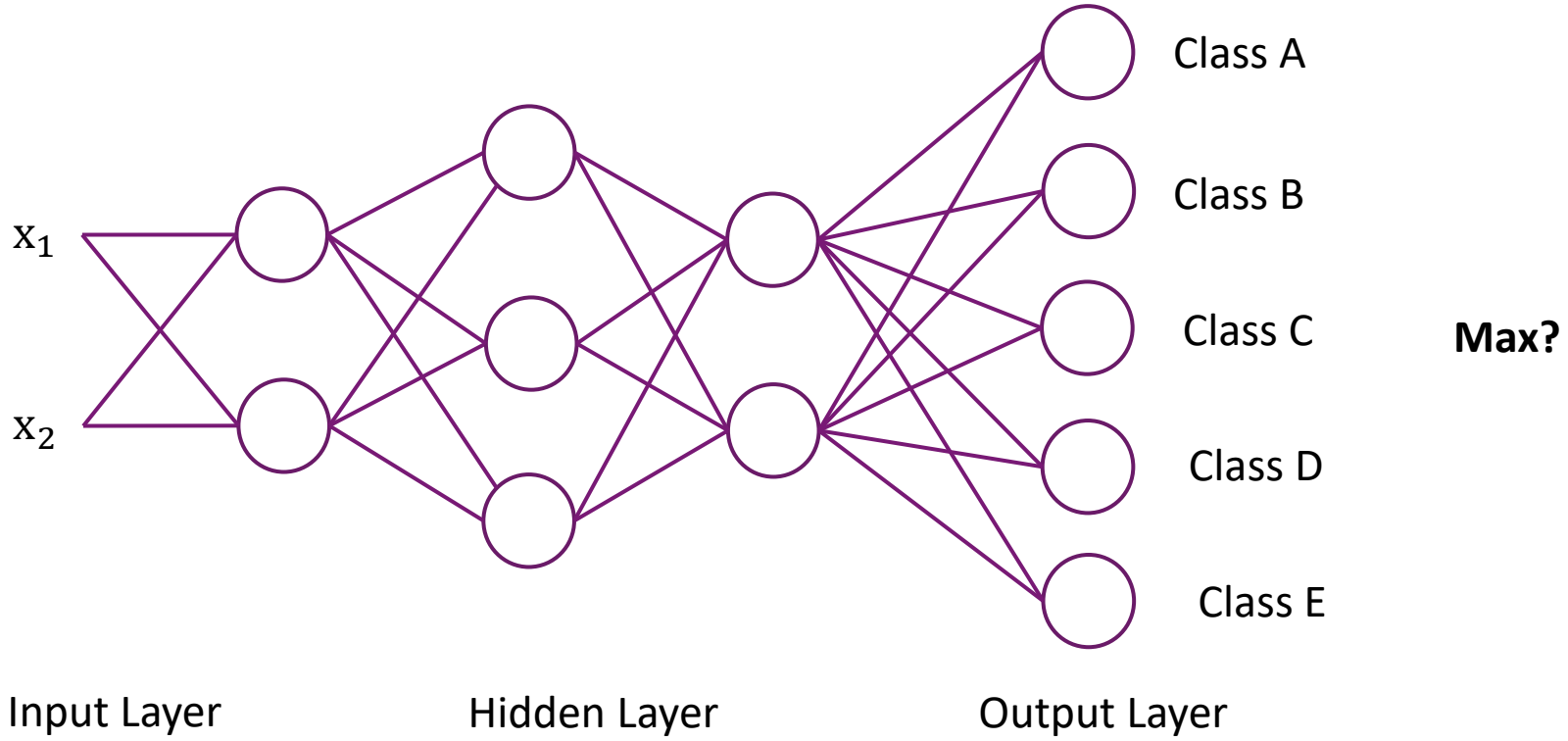
$$f(\mathrm{x}) = \begin{cases} 1, & \text{if } \left(\sum_{\mathrm{i}} \mathrm{w_i x_i} > 0\right) \\ 0, & \text{otherwise} \end{cases}$$

$y = f(f(x_1 * 1 + x_2 * 1 - 1 * 0.5) * 1 - f(x_1 * 1 + x_2 * 1 - 1 * 1.5) * 1 - 1 * 0.5)$

1) When $x_1 = 0, x_2 = 0$, $y = f(0 - 0 - 0.5) = 0$
2) When $x_1 = 1, x_2 = 0$, $y = f(1 - 0 - 0.5) = 1$
3) When $x_1 = 0, x_2 = 1$, $y = f(1 - 0 - 0.5) = 1$
4) When $x_1 = 1, x_2 = 1$, $y = f(1 - 1 - 0.5) = 0$

Multi-class Classification

- Each neuron in the output layer corresponds to a class.
- The network outputs a vector of logits (raw scores).
- The class with the highest logit is selected as the prediction.



Loss Function for Classification: Cross-Entropy

- Measure the distance between the true label distribution y and the predicted probability distribution p.

$$CrossEntropy(y, p) = -\sum_{i=1}^{C} y_i \log(p_i)$$

y is one-hot, $y_k = 1$, others = 0

$$CrossEntropy(y, p) = -\log(p_k)$$

Softmax before Cross-Entropy

- Convert a vector of logits (raw scores) into a probability distribution over classes.
- Use exponentiation to amplify differences between logits.
- Normalize the resulting values so that all probabilities sum to 1.

$$\text{Softmax}(z_k) = \frac{e^{z_k}}{\sum_{i=i}^{n} e^{z_i}}$$

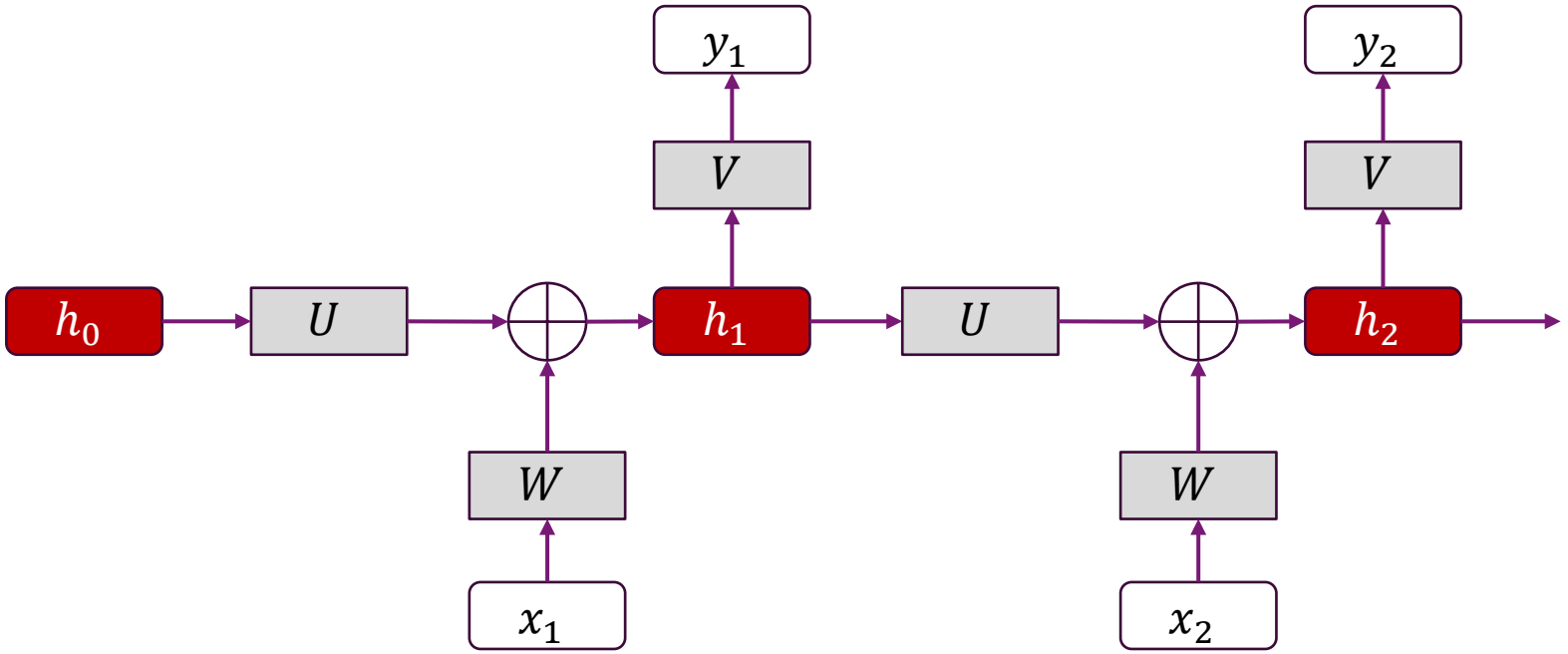
2. Recurrent Neural Networks

Recurrent Neural Networks (RNN)

- **Previous models process fixed-size inputs and produce fixed-size outputs.**
- **However, many real-world problems involve sequences of varying lengths:**
  - Finance: Stock prices over time.
  - Text: Sequences of words in sentences.
  - Speech/Music: Audio signals over time.
- **We need a model that can retain information from previous inputs when processing the current input.**

Recurrent Neural Networks

- RNN introduces the concept of a hidden state or memory h_i.
- x_i is the word embedding of input; y_i is the output.
- W, U, V are the weight matrices to be trained.



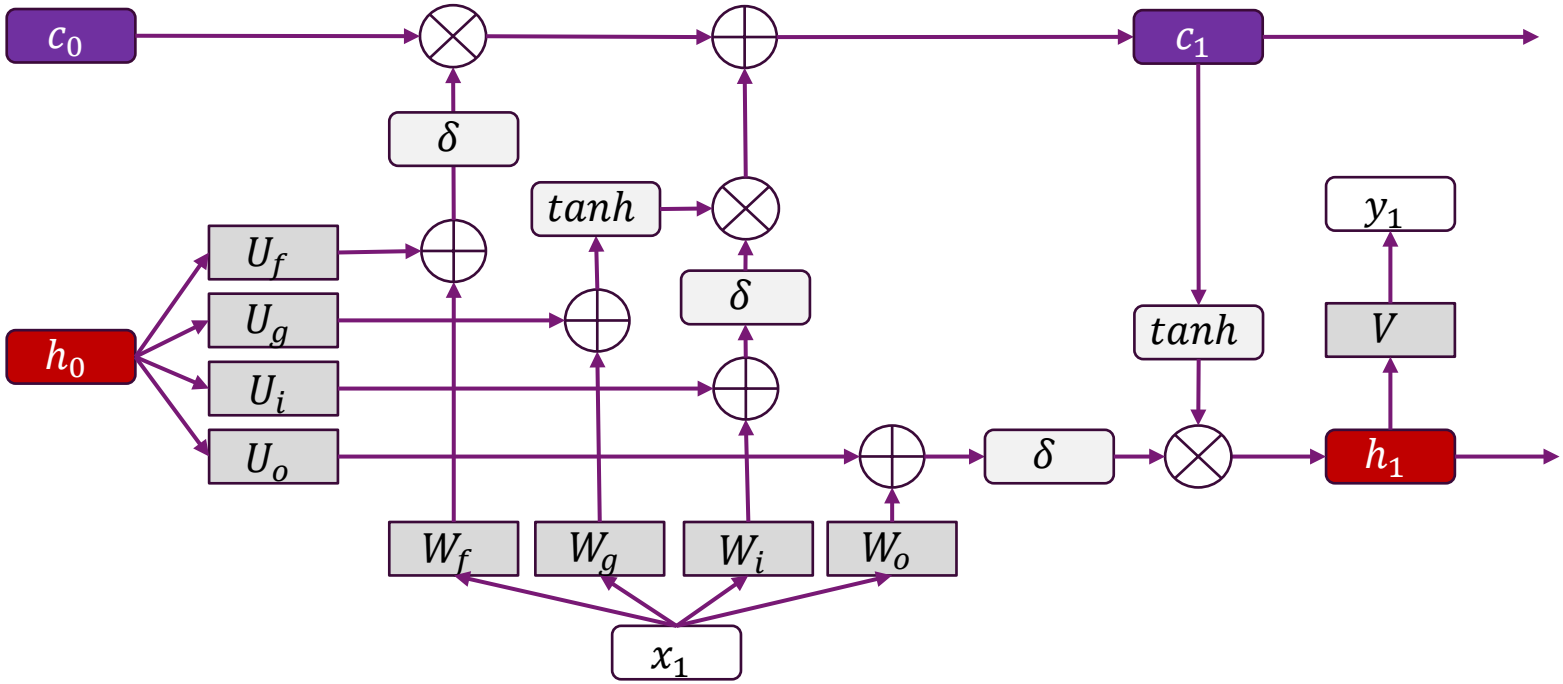
$$h_t = g(Uh_{t-1} + Wx_t) \qquad y_t = f(Vh_t)$$

Problem of RNN

- **Vanilla RNNs struggle to learn long-term dependencies.**
  - Due to the vanishing gradient problem.
  - This limitation motivated the development of LSTMs.
- **RNNs are slow to train on long sequences.**
  - Each hidden state depends on the previous one, limiting parallelization.
  - This limitation motivated the development of Transformers.

Long Short-Term Memory Network (LSTM)

- **Hidden state**: For short-term memory
- Cell state: For long-term memory



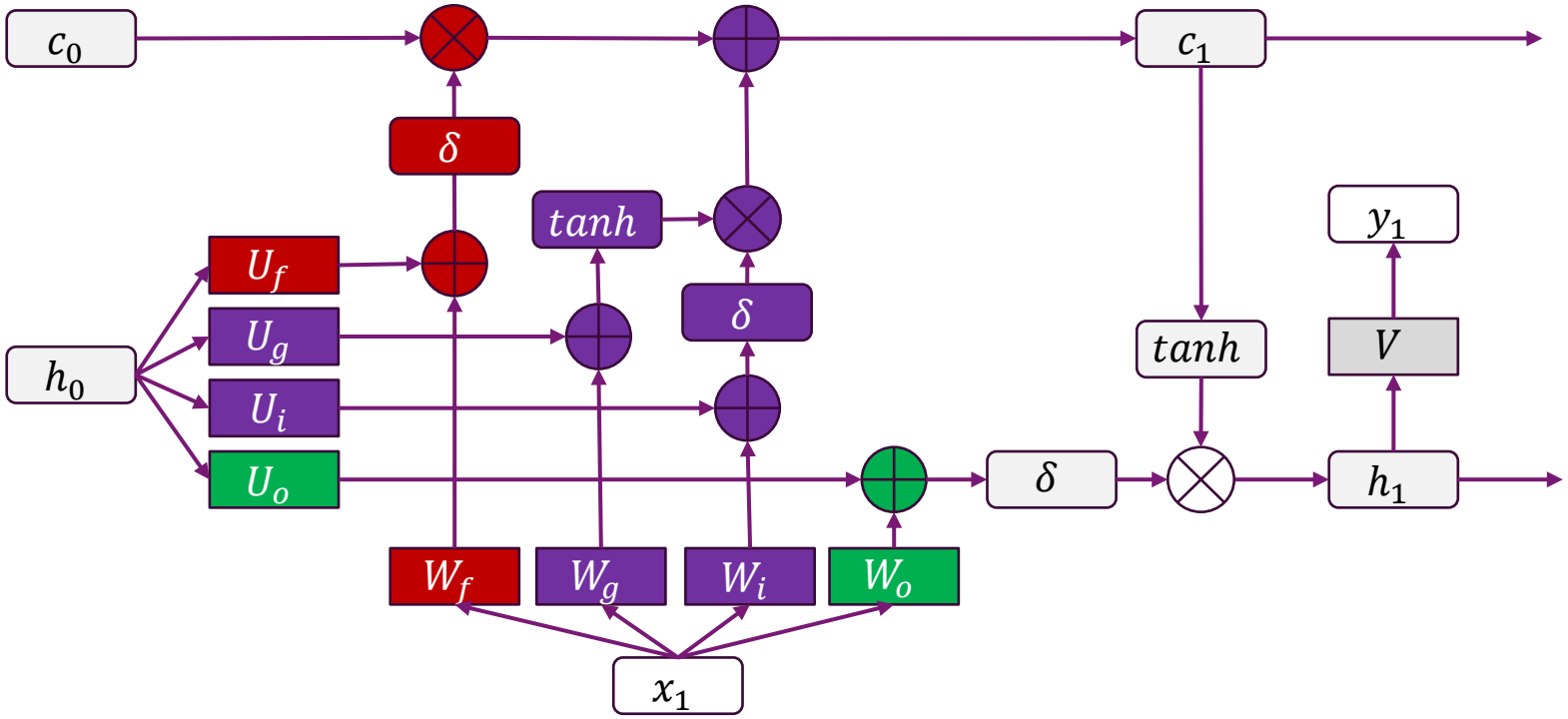
δ : sigmoid

$\otimes$: elementwise multiplication

$\oplus$: elementwise addition

LSTM

- **Forget Gate:** Control what old information to erase.
- **Input gate:** Control what new information to add.
- **Output gate:** Control what to send to hidden state.



δ : sigmoid

$\otimes$: elementwise multiplication

$\oplus$: elementwise addition

Problems that Can Be Modeled with RNN/LSTM

❑Time Series & Forecasting

- ➢Stock Price Prediction (financial time series forecasting)
- ➢Weather Forecasting (temperature, rainfall, wind patterns)

❑Natural Language Processing (before LMM)

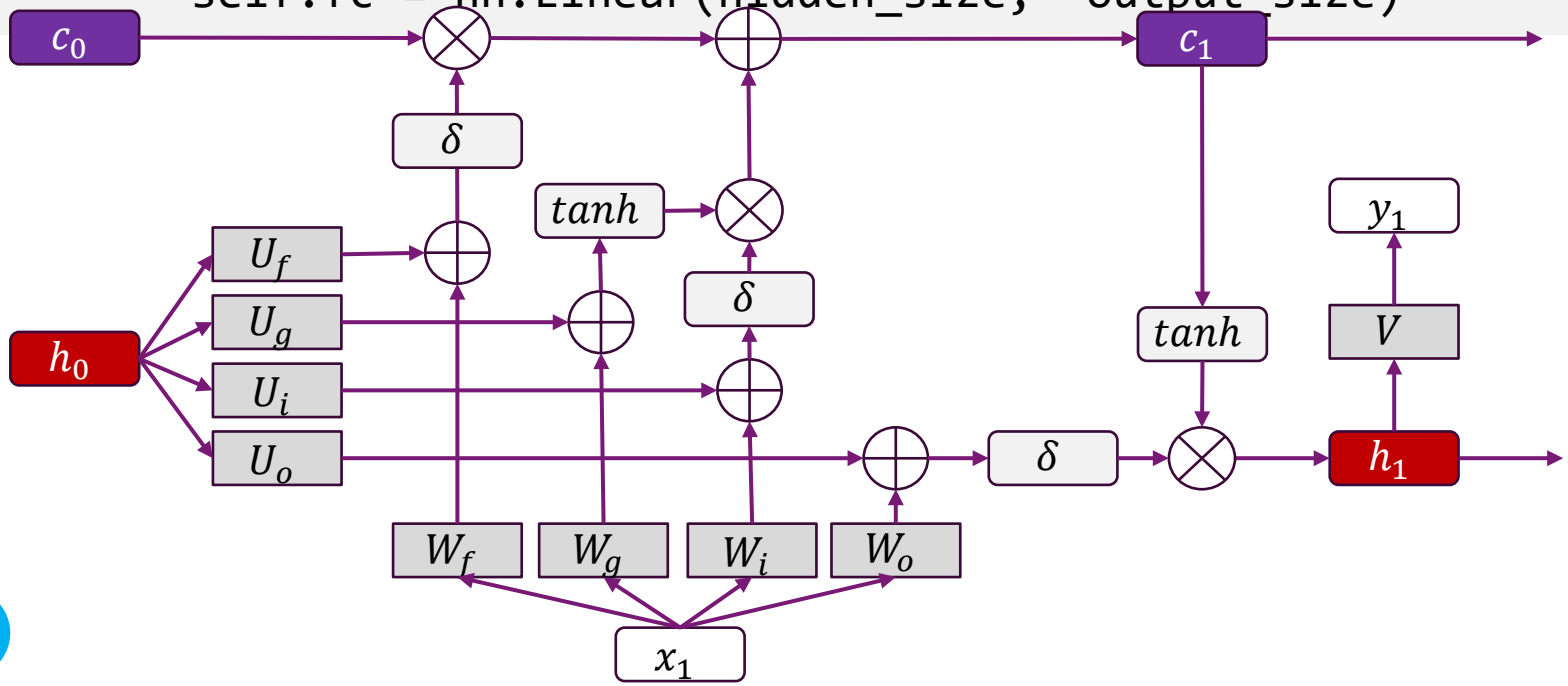
- ➢Text Classification: sentiment analysis, topic classification
- ➢Chatbots
- ➢Translation

Autoregressive Generation

- **Should the model consider previously generated outputs when making predictions?**
- **Scenarios: Long-term forecasting/chatbots.**
  - The model predicts the next output based on previous inputs and outputs.
  - Each predicted output is fed back as input for the next prediction.

LSTM Tutorial: Define LSTM Model

```
class LSTMModel(nn.Module):
    def __init__(self,
                 input_size=1,  # dimension of input
                 hidden_size=64,  # feature size
                 num_layers=1,
                 output_size=1):   # dimension of output
        super(LSTMModel,  self).__init__()
        self.lstm = nn.LSTM(input_size,  hidden_size,  num_layers,
True)
        self.fc = nn.Linear(hidden_size,  output_size)
```



LSTM Tutorial: Define LSTM Model

```
class LSTMModel(nn.Module):
    ...
    def forward(self,  x):
        # shape of x: [batch,  seq_len,  hidden_size]
        # shape of out: [batch,  seq_len,  hidden_size]
        out,  _ = self.lstm(x)
        out = out[:,  -1,  :] # The last item: [batch,
hidden_size]
        out = self.fc(out) # fully connected
        return out
```

LSTM Tutorial: Data Preparation

❑Data in CSV format.

```
Date, Open, High, Low, Close, Volume
2015-01-02, 24.26, 24.73, 23.82, 24.72, 212818400
2015-01-05, 23.58, 24.11, 23.39, 24.03, 257142000
2015-01-06, 23.58, 23.84, 23.22, 23.64, 263188400
2015-01-07, 23.91, 24.01, 23.68, 23.79, 160423600
2015-01-08, 24.83, 24.89, 24.12, 24.24, 237458000
2015-01-09, 24.86, 25.13, 24.46, 25.0, 214798000
...
```

LSTM Tutorial: Data Preprocessing

```
df = pd.read_csv("AAPL.csv",  index_col="Date",  parse_dates=True)

prices = df[['Close']].values

# Normalize to [0, 1]
scaler = MinMaxScaler()
prices_scaled = scaler.fit_transform(prices)

# Crate sequence: predict the 21st item based on the past 20 items
def create_sequences(data,  seq_len=20):
    xs,  ys = [],  []
    for i in range(len(data) - seq_len):
        xs.append(data[i:i+seq_len])
        ys.append(data[i+seq_len])
    return np.array(xs),  np.array(ys)

X,  y = create_sequences(prices_scaled,  20)
```

LSTM Tutorial: Prepare Training Data and Testing Data

```
# Convert to torch Tensor
X = torch.tensor(X,  dtype=torch.float32)
y = torch.tensor(y,  dtype=torch.float32)

# Split the dataset into two parts:
# 80% for training and 20% for testing
train_size = int(len(X) * 0.8)
X_train,  X_test = X[:train_size],  X[train_size:]
y_train,  y_test = y[:train_size],  y[train_size:]

print("Training data:",  X_train.shape,  y_train.shape)
print("Testing data:",  X_test.shape,  y_test.shape)
```

LSTM Tutorial: Training

```
model = LSTMModel()
optimizer = torch.optim.Adam(model.parameters(),  lr=0.001)

# DataLoader
train_loader = DataLoader(list(zip(X_train,  y_train)),  batch_size=64,
shuffle=True)

EPOCHS = 20
for epoch in range(EPOCHS):
    model.train()
    for xb,  yb in train_loader:
        optimizer.zero_grad()
        preds = model(xb)
        loss = nn.MSELoss()(preds,  yb)
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch+1}/{EPOCHS},  Loss: {loss.item():.6f}")
```

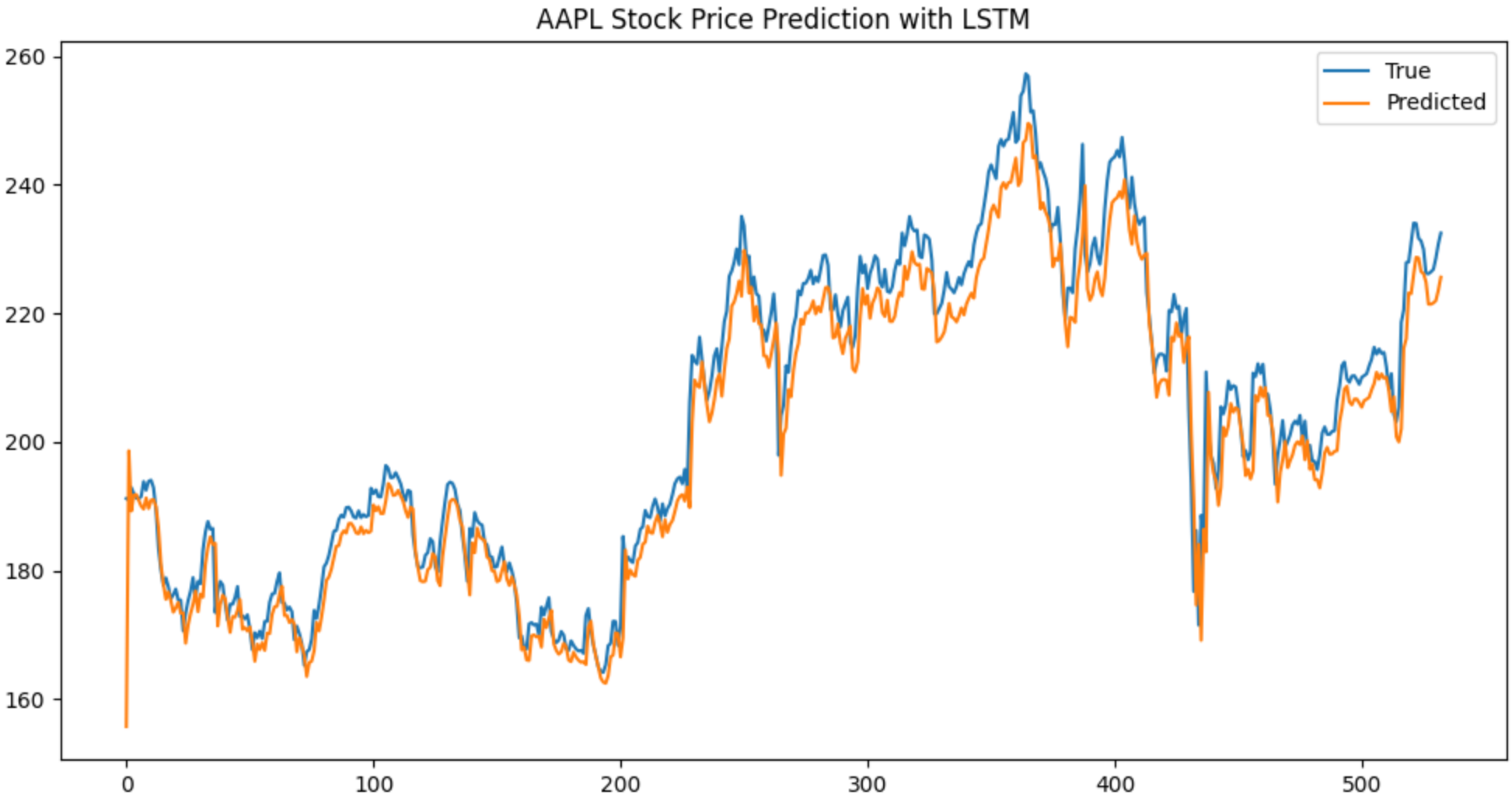
LSTM Tutorial: Evaluation

```
# Switch the model to evaluation mode
model.eval()
with torch.no_grad():
    preds = model(X_test)

# Convert the data back to the original price scale
preds_rescaled = scaler.inverse_transform(preds.numpy())
y_test_rescaled = scaler.inverse_transform(y_test.numpy())

# Plot the true prices vs. predicted prices
plt.figure(figsize=(12, 6))
plt.plot(y_test_rescaled,  label="True")
plt.plot(preds_rescaled,  label="Predicted")
plt.legend()
plt.title("AAPL Stock Price Prediction with LSTM")
plt.show()
```

Example Result



The predictions are good in trend, but they respond one day late.

3. Large Language Models

Textual Data: How to Represent Text for Computation?

Related information

Seeking Alpha • 23 hours ago

Nvidia Stock: 5 Reasons I Bought More Ahead of Earnings and So Should You (NASDAQ: NVDA)



Yahoo Finance • 1 year ago

Nvidia's China Trouble Could Slash Billions From Revenue, Analysts Warn



MSN • 1 year ago

Cantor Fitzgerald Raises PT on NVIDIA Corporation (NVDA) Amid Growing Demand for AI Compute

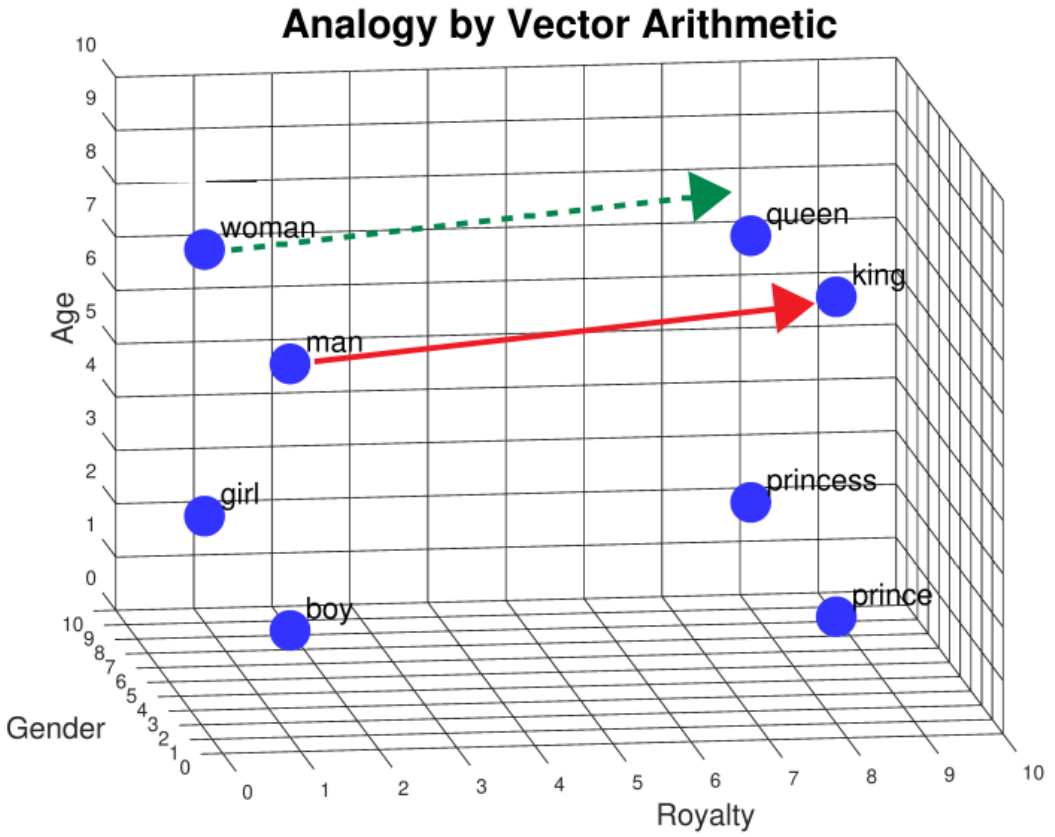


How to Represent Each Word of a Sentence?

- **Use plain numbers: Each number represents a word, *e.g.*,**
  - "cat" → 1
  - "dog" → 2
- **One-Hot Encoding: Each bit represents a word, *e.g.*,**
  - "cat" → 10000000
  - "dog" → 01000000
- **Embedding: Each vector represents a word, *e.g.*,**
  - "cat" → [0.21, -0.56, 0.91, …]
  - "dog" → [0.20, -0.53, 0.88, …]

Intuition of Embeddings

- Similar words have similar embeddings.
- Relationships can be captured.




```
                [age, Gender, Royalty]
    woman: [5, 0, 1]
      man: [5, 1, 1]
     girl: [1, 0, 1]
      boy: [1, 1, 1]
    queen: [5, 0, 8]
     king: [5, 1, 8]
 princess: [1, 0, 8]
   prince: [1, 1, 8]
```

Relationships:

woman - girl = man – boy

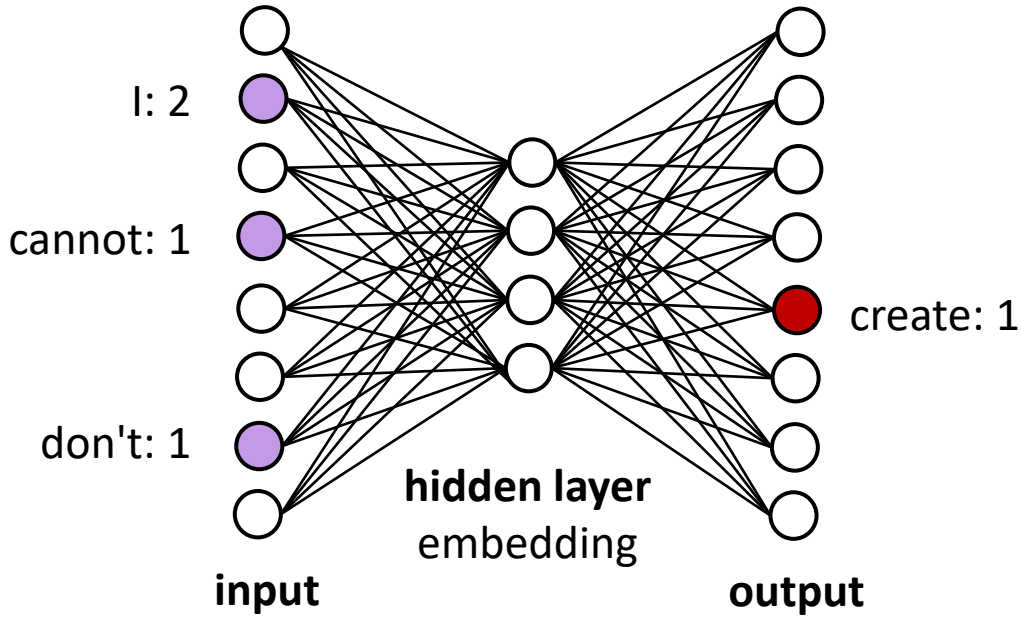
boy + princess – girl = prince

Embedding

- **Designing a semantic space manually is challenging.**
- **Use neural networks to learn vector representations automatically.**
- **Common approaches:**
  - Word2Vec: CBOW and Skip-gram.
  - LLMs: End-to-end embedding learning.

Word2Vec: Continuous Bag of Words

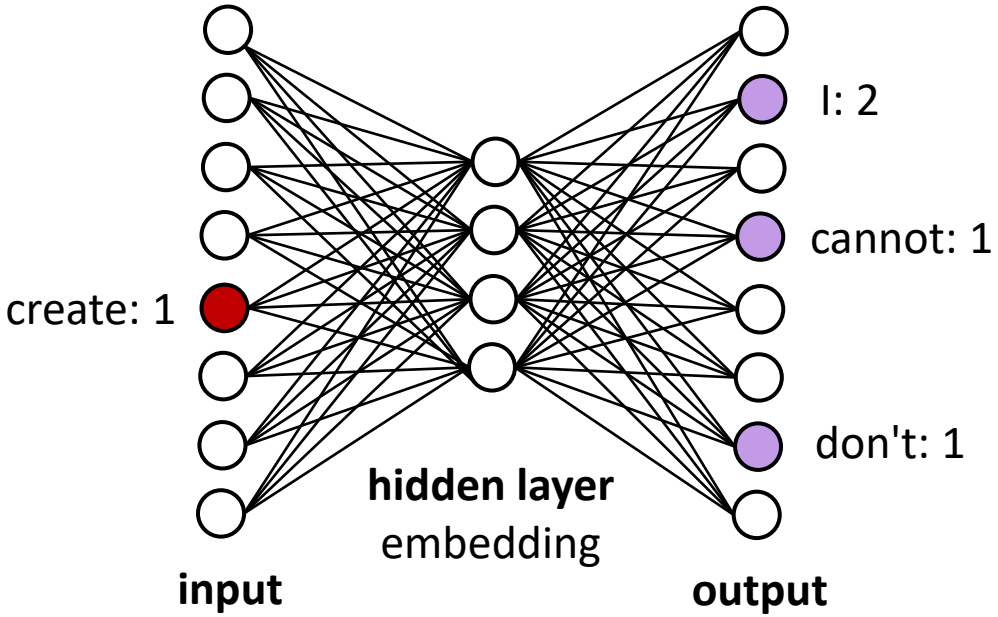
- Predict a **target word** given its surrounding **context words.**
- Use one-hot encoding for the input and output.
- Each input neuron corresponds to a word
  - The value is the frequency.



What I cannot create, I don't understand

Word2Vec: Skip-gram

❑Predict the context words surrounding a given target word.



What I cannot create, I don't understand

LLM based on Transformer

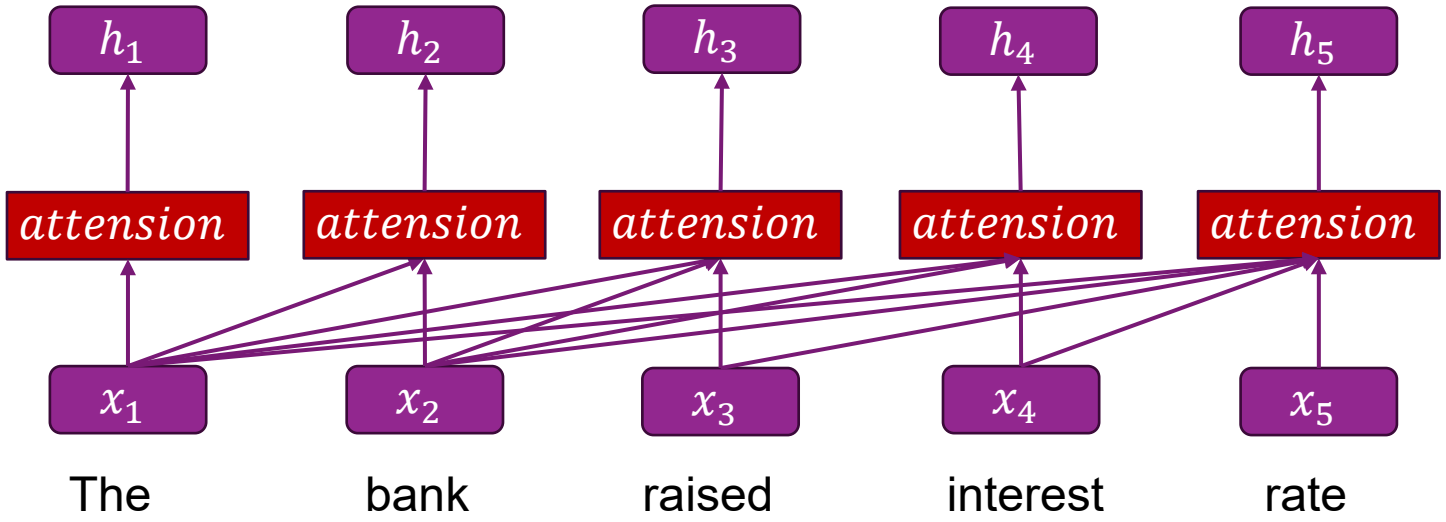
- **RNN/LSTM is slow and the capability is limited.**
- **We need a model that:**
  - Handles long sequences well;
  - Works in parallel (fast training);
  - Captures relationships between tokens, no matter how far apart.
- **That's the Transformer**
  - Vaswani et al., 2017, "Attention Is All You Need"

Core Idea: Attention

- **At the heart of a Transformer is self-attention.**
- **When reading a sentence, a word's meaning depends on other words in context.**
  - "The bank raised interest rates". Here, "bank" refers to a financial institution.
  - But in "He sat on the bank of the river", "bank" means riverside.
- **We need a mechanism where each word looks at other words and decides how much attention to give them.**

Attention: Simplified Version

- **The context of previous tokens are captured by the attention matrix.**
  - $\{h_1, \dots, h_n\}$ are similar to the hidden states of LSTM.
- **The calculation of $\{h_1, \dots, h_n\}$ are parallel.**



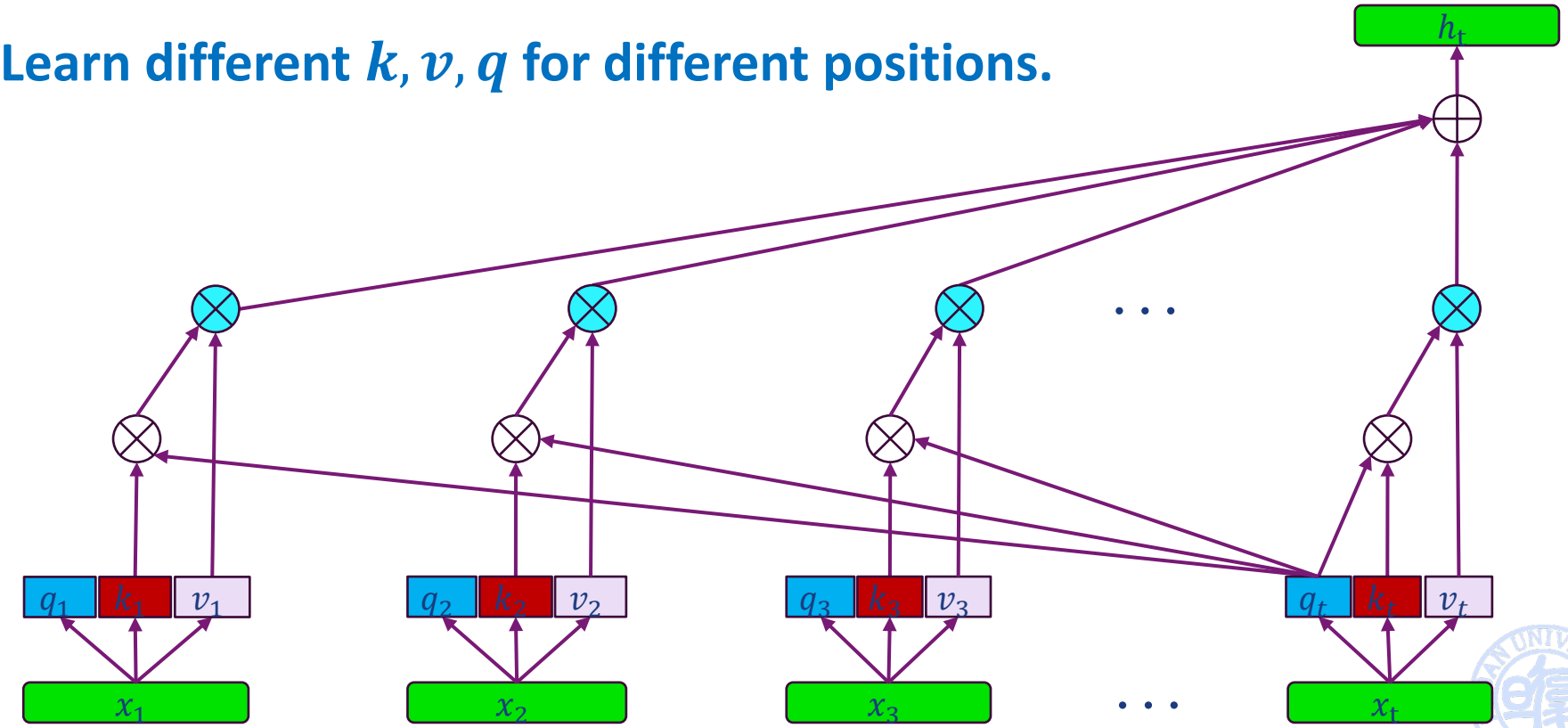
$$h_t = \sum_{i \le t} w_{it} \times x_i$$

Detailed Structure of Attention: Q, K, V

- **Think of each token embedding going into three linear projections:**
  - **Q (Query)** → "What am I looking for?"
  - **K (Key)** → "What do I contain?"
  - **V (Value)** → "What information do I provide?"

$$Attention(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- **Learn different k, v, q for different positions.**



Masking Mechanism

- The scaled attention scores are computed as: $\frac{QK^T}{\sqrt{d_k}}$
- To mask future positions during training, set their attention scores to $-\infty$.

$$Score_{i,j} = \begin{cases} \dfrac{QK^T}{\sqrt{d_k}}, & \text{if } j < i \\ -\infty, & otherwise \end{cases}$$

$$\text{mask matrix} = \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{bmatrix}$$

Token 1 can see it self.
Token 2 can see both token 1 and 2.
Token 3 can see token 1, 2, and 3.

LLM: Decoder-only Transformer

