Systematic Trading Strategies with Machine Learning Algorithms

Time Series Forecasting with Neural Networks

Imperial College Business School

June 12, 2025

Outline

Outline

Position of the Problem

- **Goal:** Predict multiple future time steps for a target variable (y_i^t) using:
    - Past observations of the target variable.
    - Additional features that provide context and improve forecasting accuracy.
- **Multiple time series:**
    - We consider several entities $i \in \{1, \dots, N\}$, each associated with its own time series $(y_i^t)_{t-T_b+1 \leq t \leq t+T_f}$.
    - Examples of entities:
        - **Finance:** Volatility for different stocks in financial markets.
        - **Energy:** Consumption or production across multiple regions.
        - **Traffic:** Flow rates at various locations.

- Let us consider an entity i at time t:
- We aim to predict the future values of the univariate time series $(y_i^t)_{t-T_b+1 \leq t \leq t+T_f}$:
  - The past values in a T_b sized window of the target time series: $(y_i^{t-T_b+1}, \ldots, y_i^t)$
  - The future values up to the horizon T_f: $(y_i^{t+1}, \ldots, y_i^{t+T_f})$
- There are 3 possible inputs:

Name	Notation
Static attributes	$s_i \in \mathbb{R}^{d_s}$
Time varying unknown	$(z_i^{t-T_b+1}, \ldots, z_i^t) \in \mathbb{R}^{T_b \times d_z}$
Time varying known	$(x_i^{t+1}, \ldots, x_i^{t+T_f}) \in \mathbb{R}^{T_f \times d_x}$

Table: Types of Inputs

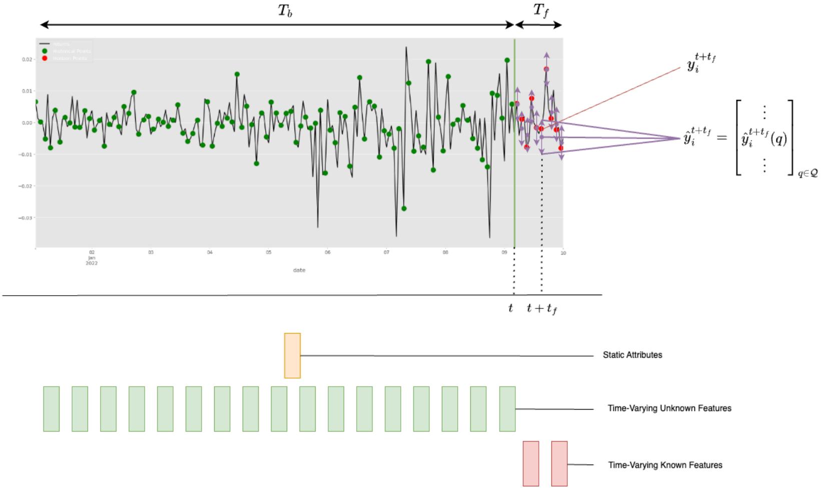
- **Features for Prediction:**
  - **Static Attributes:**
    - Fixed characteristics of each financial asset.
    - Example: Industry sector or market capitalization of a stock.
  - **Time-Varying Known Features:**
    - Features whose future values are available or predictable.
    - Example: Economic calendar events, such as interest rate decisions or earnings announcements.
  - **Time-Varying Unknown Features:**
    - Sequential features observed only up to the present time.
    - Example: Recent trends in stock price movements or realized volatility.

Position of the Problem

- Let $\mathcal{Q}$ be the set of quantiles that interest us. For this example, $\mathcal{Q} = \{0.1, 0.5, 0.9\}$.
- The model outputs for each time step $t + t_f$ (for $t_f \in \{1, \ldots, T_f\}$) the prediction associated with each quantile $q \in \mathcal{Q}$, denoted as $\hat{y}_i^{t+t_f}(q)$.
- Thus, for each $t_f \in \{1, \ldots, T_f\}$, the output vector at each time step $t + t_f$ is given by:

$$\hat{y}_i^{t+t_f} = \begin{bmatrix} \vdots \\ \hat{y}_i^{t+t_f}(q) \\ \vdots \end{bmatrix}_{q \in \mathcal{Q}}$$

Example: The following graph summarizes the previous notations:



The Learning Problem

- To train the model, we compare the predictions $\hat{y}_i^{t+t_f} \in \mathbb{R}^{|\mathcal{Q}|}$ to the true values $y_i^{t+t_f}$ for all $t_f \in \{1, \ldots, T_f\}$.
- The loss function is defined as:

$$\mathcal{L}\left(\mathcal{B}, \theta\right) = \sum_{i \in \mathcal{B}} \sum_{q \in \mathcal{Q}} \sum_{t_f=1}^{T_f} \frac{QL_q\left(y_i^{t+t_f}, \hat{y}_i^{t+t_f}(q)\right)}{|\mathcal{B}|\, T_f}$$

- Where:
  - $\mathcal{B}$ is the batch of training data.
  - $\forall y, \hat{y} \in \mathbb{R},\ QL_q(y, \hat{y}) = q(y - \hat{y})_+ + (1-q)(\hat{y} - y)_+$
  - Equivalently:

$$QL_q(y, \hat{y}) = \max\left((q-1)(y-\hat{y}), q(y-\hat{y})\right)$$

Click here to take the quiz

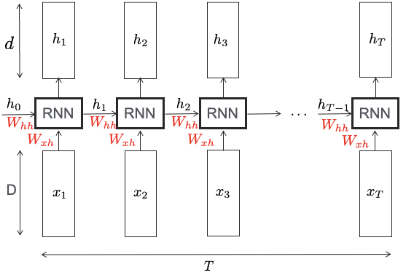
Outline

From Hidden Markov Models to RNNs

- **Hidden Markov Models (HMMs)**:
  - Popular in the 1980s for sequence modeling (e.g., speech recognition [10]).
  - Relied on the Markov assumption for hidden states, limiting their ability to model long-range dependencies.
- **Recurrent Neural Networks (RNNs)**:
  - Introduced to overcome HMM limitations.
  - Achieved state-of-the-art performance in tasks such as speech recognition [4].

Introduction to Vanilla RNNs

- Feed-forward neural networks assume data is independent and identically distributed (i.i.d).
- Recurrent Neural Networks (RNNs) [11] process data sequentially, making them suitable for time-series and other sequence-based tasks.



- Objective: Process an input sequence of D-dimensional vectors $x_1, \ldots, x_T$ to generate d-dimensional hidden states $h_1, \ldots, h_T$.
- Model Parameters:
  - $W_{xh} \in \mathbb{R}^{D \times d}$: Input-to-hidden weights.
  - $W_{hh} \in \mathbb{R}^{d \times d}$: Hidden-to-hidden weights.
- Hidden state at time t:

$$h_t = \tanh\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right)$$

Gradient Problems

- **Exploding Gradients:**
  - Occur when gradients become excessively large, destabilizing model training.
- **Vanishing Gradients:**
  - Occur when gradients diminish during backpropagation, preventing the model from learning long-term dependencies.
  - Often observed in deep or sequential networks when dealing with long input sequences.

Addressing Gradient Problems

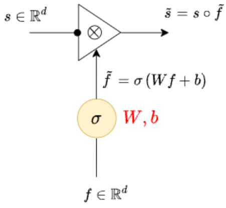
- **Solutions to Exploding Gradients:**
  - Gradient Clipping: Caps gradients to stabilize training [9].
- **Solutions to Vanishing Gradients:**
  - **Regularization:** Preserves norm consistency during training [9].
  - **Gated Architectures:**
    - Long Short-Term Memory (LSTM) [5]: Introduces gates to manage information flow.
    - Gated Recurrent Unit (GRU) [3]: Simplified alternative to LSTM.

- LSTMs were state-of-the-art for tasks like:
  - Machine Translation [13, 3, 1].
  - Language Modeling [12].
  - Time Series Prediction [6].
  - Robot Reinforcement Learning [2].
- Core Idea: Maintain long-term dependencies through a **cell state** regulated by **gates**.
- Gates are responsible for filtering information flow:
  - Input: New information to add.
  - Forget: Remove irrelevant information.
  - Output: Decide what to expose to the hidden state.

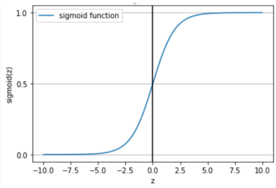
- Gates use a sigmoid function to scale values between 0 and 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Point-wise multiplication adjusts information based on gate values.



(a) Filtering a signal using a sigmoid function and a neural network



(b) The sigmoid function

The LSTM Architecture

- Each time step has:
  - **Cell State** C^t: Preserves long-term memory.
  - **Hidden State** h^t: Represents short-term output.
- Transition from (h^{t-1}, C^{t-1}) to (h^t, C^t) involves:
  1. Filtering with input and forget gates.
  2. Generating a memory candidate $\tilde{C}^t$.
  3. Updating the cell state and computing the hidden state.

The LSTM Architecture

- Each time step has:
  - **Cell State** C^t: Preserves long-term memory.
  - **Hidden State** h^t: Represents short-term output.
- Transition from (h^{t-1}, C^{t-1}) to (h^t, C^t) involves:
  1. Filtering with input and forget gates.
  2. Generating a memory candidate $\tilde{C}^t$:

  $$\tilde{C}^t = \tanh\left(W_C[h^{t-1}, x^t] + b_C\right)$$

  3. Updating the cell state and computing the hidden state.

LSTM Gate Operations

- **Forget Gate:** Filters irrelevant past memory.

$$f^t = \sigma\left(W_f[h^{t-1}, x^t] + b_f\right)$$

- **Input Gate:** Filters new memory candidate.

$$i^t = \sigma\left(W_i[h^{t-1}, x^t] + b_i\right)$$

- **Output Gate:** Determines visible parts of the cell state.

$$o^t = \sigma\left(W_o[h^{t-1}, x^t] + b_o\right)$$

Memory Update and Hidden State

- **Cell State Update:**

$$C^t = f^t \circ C^{t-1} + i^t \circ \tilde{C}^t$$

- **Hidden State Update:**

$$h^t = o^t \circ \tanh(C^t)$$

- Result: LSTMs Handle long-term dependencies better than vanilla RNNs. LSTMs can:
  - **Write:** Add new information via the input gate.
  - **Erase:** Remove irrelevant information via the forget gate.
  - **Read:** Expose relevant memory via the output gate.

The LSTM Architecture

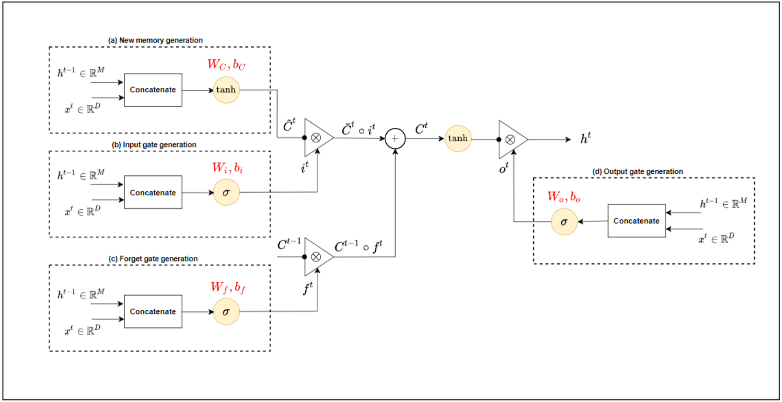
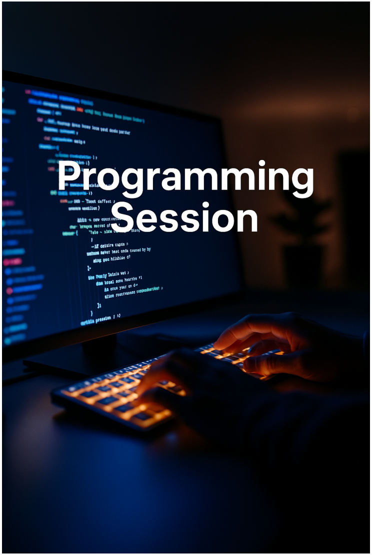


Figure: LSTM architecture.

Quiz Time!

Click here to take the quiz



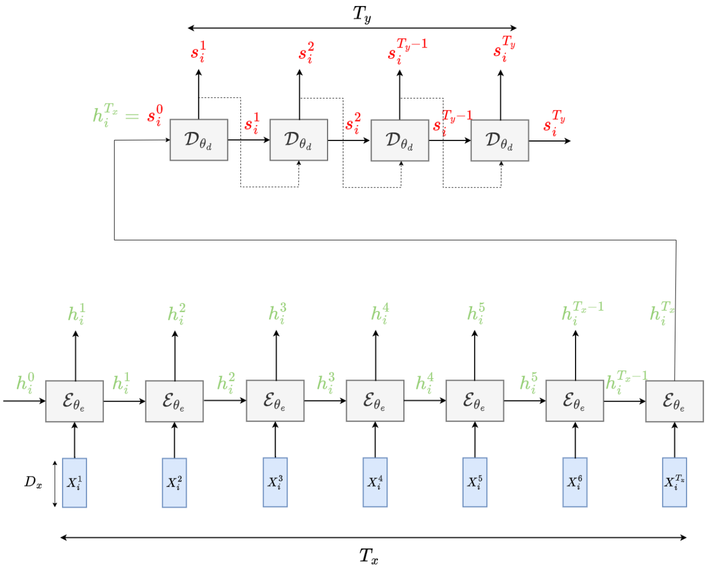
Programming Session 7: Section 1

- Section 1: Temporal Processing using RNNs
- *Click here to access the programming session*

Solution will be posted tonight on the GitHub page.

- *Click here to access ccess the GitHub Page*

- LSTM/GRU models in Many-to-Many settings require input and output sequences of the same length (e.g., POS tagging [15]).
- For applications where $T_x \neq T_y$ (e.g., machine translation), we need the Sequence to Sequence (Seq2Seq) framework.
- Seq2Seq maps an input sequence of length T_x to an output sequence of length T_y using two components:
    1. **Encoder:** Encodes the input sequence into a fixed-length representation.
    2. **Decoder:** Generates the output sequence from the encoded representation.

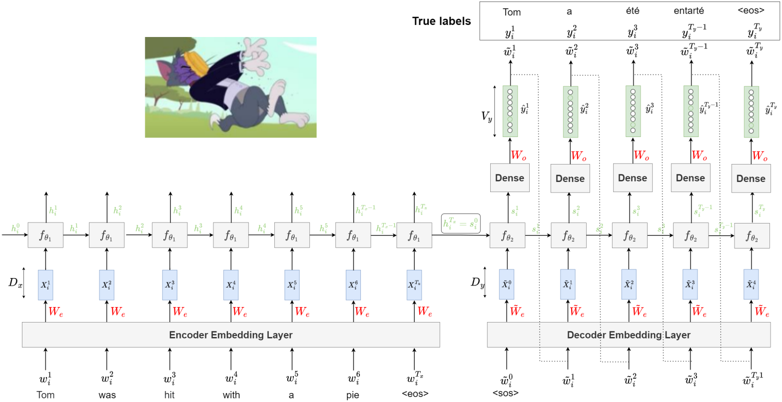


T_y
s_i^1
s_i^2
$s_i^{T_y-1}$
$s_i^{T_y}$
$h_i^{T_x} = s_i^0$
$\mathcal{D}_{\theta_d}$
s_i^1
$\mathcal{D}_{\theta_d}$
s_i^2
$\mathcal{D}_{\theta_d}$
$s_i^{T_y-1}$
$\mathcal{D}_{\theta_d}$
$s_i^{T_y}$
h_i^1 h_i^2 h_i^3 h_i^4 h_i^5 $h_i^{T_x-1}$ $h_i^{T_x}$
h_i^0 $\mathcal{E}_{\theta_e}$ h_i^1 $\mathcal{E}_{\theta_e}$ h_i^2 $\mathcal{E}_{\theta_e}$ h_i^3 $\mathcal{E}_{\theta_e}$ h_i^4 $\mathcal{E}_{\theta_e}$ h_i^5 $\mathcal{E}_{\theta_e}$ $h_i^{T_x-1}$ $\mathcal{E}_{\theta_e}$
D_x
X_i^1 X_i^2 X_i^3 X_i^4 X_i^5 X_i^6 $X_i^{T_x}$
T_x

Encoder and Decoder Components

- **Encoder:**
  - Maps the input sequence $(X_i^1, \ldots, X_i^{T_x}) \in \mathbb{R}^{T_x \times D_x}$ into hidden states $h_i^1, \ldots, h_i^{T_x}$.
  - Final hidden state $h_i^{T_x}$ summarizes the input sequence.
- **Decoder:**
  - Takes the encoder's last hidden state $h_i^{T_x}$ as its initial hidden state s_i^0.
  - Generates the output sequence $s_i^1, \ldots, s_i^{T_y}$.

Example: Seq2seq for machine translation



Limitations of Classical Models

- **Challenges with Seq2Seq Framework:**
  - Encoder compresses all input information into a fixed-length vector, leading to information loss.
  - Performance degrades for long input sequences.
  - No mechanism for aligning input and output sequences.
- **Alignment Intuition:**
  - For each output Y_i^t, the model should selectively focus on relevant parts of the input sequence $X_i^{t'}$.
  - Alignment helps determine how much of each $X_i^{t'}$ contributes to generating Y_i^t.

- The following figure shows the desired alignment matrix, where scores indicate the relevance of each input vector to a specific output.

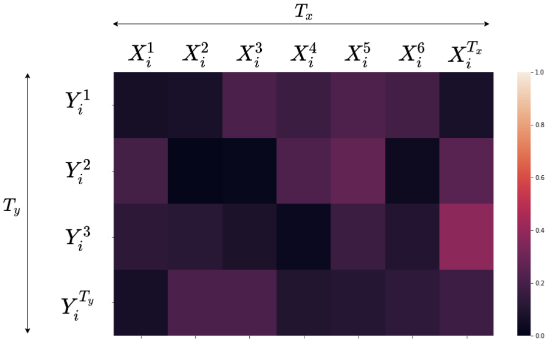
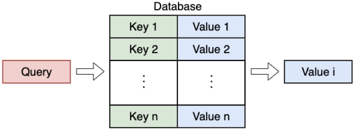


Figure: Matrix of alignment scores.

Addressing the Challenges with Attention

- **Key Challenges of Seq2Seq:**
  - No explicit mechanism to focus on relevant parts of the input.
- **Why Attention?**
  - Allows models to dynamically focus on relevant input parts.
  - Combines perception with a selective memory mechanism for reasoning.
- **Applications:**
  - Machine translation.
  - Time series prediction.
  - Speech-to-text.

- Attention mechanisms are inspired by database Query-Retrieval Problems:
  - A query is matched against keys to retrieve values.
  - The following figure shows a classic hard query retrieval system.



- In attention mechanisms:
  - Multiple keys can match a query (**soft query retrieval**).
  - The result is a weighted sum of values, called the **attention vector**.

Soft Query Retrieval: Steps

- Given: a query $q \in \mathbb{R}^{d_q}$, keys $(k_i)_{1 \leq i \leq n} \in \mathbb{R}^{n \times d_k}$, and values $(v_i)_{1 \leq i \leq n} \in \mathbb{R}^{n \times d_v}$.
- Steps:
  1. Compute alignment scores a_i between the query and each key:
  $$a_i = a(q, k_i) \quad \forall i \in \{1, \dots, n\}.$$
  2. Normalize scores to get attention weights α_i using a distribution function (e.g., softmax):
  $$\alpha_i = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}}.$$
  3. Compute the attention vector as a weighted sum of values:
  $$A(q, K, V) = \sum_{i=1}^{n} \alpha_i v_i.$$

- Alignment functions compute similarity between query q and keys k_i:

Function	**Equation**
Dot Product	$a(q, k_i) = q^T k_i$
Scaled Dot Product	$a(q, k_i) = \frac{q^T k_i}{\sqrt{d_k}}$
Luong's Multiplicative	$a(q, k_i) = q^T W k_i$
Bahdanau's Additive	$a(q, k_i) = v_a^T \tanh(W_1 q + W_2 k_i)$
Feature-based	$a(q, k_i) = W_{imp}^T \mathrm{act}(W_1 \phi_1(k_i) + W_2 \phi_2(q) + b)$
Kernel Method	$a(q, k_i) = \phi(q)^T \phi(k_i)$

Table: Common Alignment Functions.

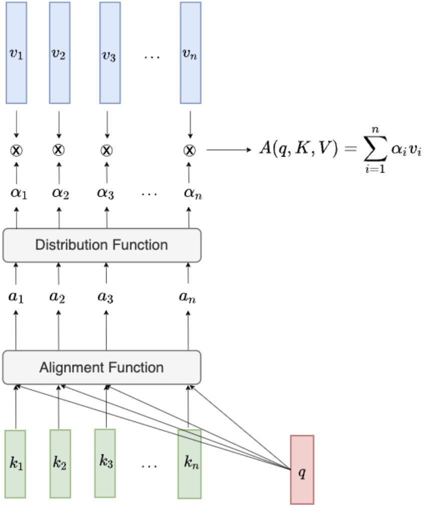
- **Soft Attention:**
  - Uses dense alignments with a softmax function:

$$\alpha_i = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}}.$$

- **Sparse Attention:**
  - Assigns non-zero probabilities to only a few values.
  - Examples:
    - Sparsemax [7].
    - Sparse Entmax [8].
- The attention vector combines weighted values:

$$A(q, K, V) = \sum_{i=1}^{n} \alpha_i v_i.$$

Soft Query Retrieval Summary



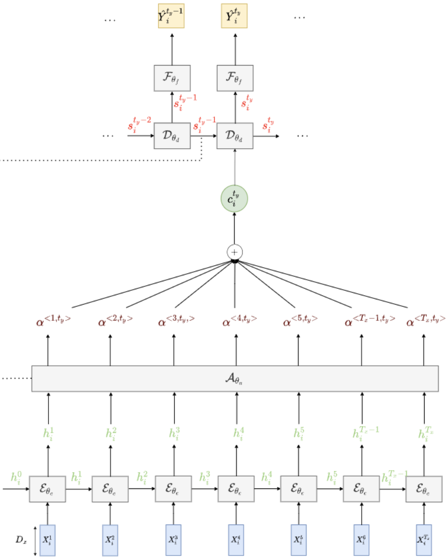
Imperial College Business School

- Objective: Learn a mapping function Φ_θ from input sequences to output sequences.

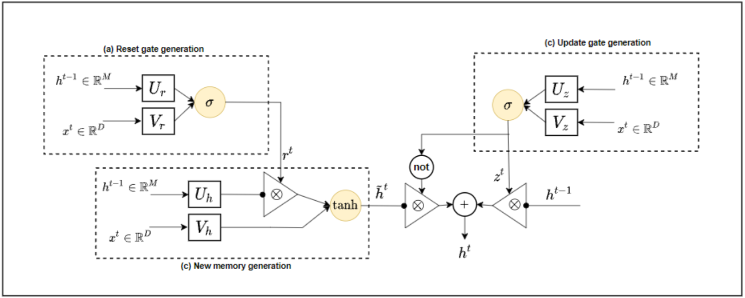
$$(\hat{Y}_i^1, \dots, \hat{Y}_i^{T_y}) = \Phi_\theta(X_i^1, \dots, X_i^{T_x}).$$

- Components of Φ_θ:
  1. **Encoder:** Maps input sequence to hidden states $h_i^1, \dots, h_i^{T_x}$.
  2. **Attention Layer:** Computes context vector $c_i^{t_y}$ for each output step.
  3. **Decoder:** Generates output sequence using attention and decoder states.

Sequence to Sequence with Attention



- The encoder can be a GRU model or an LSTM model that transforms input sequence $(X_i^1, \dots, X_i^{T_x})$ into hidden states $(h_i^1, \dots, h_i^{T_x})$.
- The GRU Model



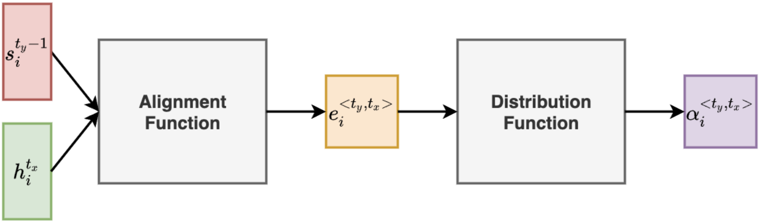
- Assigns weights to encoder hidden states to compute a **context vector** $c_i^{t_y}$:

$$c_i^{t_y} = \sum_{t_x=1}^{T_x} \alpha_i^{<t_y,t_x>} h_i^{t_x}.$$

- Steps:
  1. Compute alignment scores $e_i^{<t_y,t_x>}$ between decoder hidden state $s_i^{t_y-1}$ and encoder hidden states $h_i^{t_x}$.
  2. Normalize scores into attention weights $\alpha_i^{<t_y,t_x>}$ using a distribution function (e.g., softmax).
  3. Calculate context vector $c_i^{t_y}$ as a weighted sum of encoder hidden states.

The Attention Layer

- **Calculating the weights**: $\alpha_i^{<t_y,t_x>}$ for all $t_x \in \{1, \ldots, T_x\}$:



The Decoder and Application-Specific Final Layer

- **Decoder:** Combines:
  - Previous hidden state $s_i^{t_y-1}$,
  - Context vector $c_i^{t_y}$ (from the attention mechanism),
  - To generate the decoder hidden state $s_i^{t_y}$.
- **Final Layer:** Maps $s_i^{t_y}$ to the output prediction $\hat{Y}_i^{t_y}$.
  - The nature of the final layer depends on the application:
    - **Machine Translation:** Dense layer with a softmax activation to predict the next word in a target language.
    - **Text Generation:** Softmax-based layer for generating characters or tokens.
    - **Time Series Forecasting:** Regression output layer for predicting continuous values, such as stock prices or energy consumption.

Click here to take the quiz

Outline

Introduction to the Transformer

- The paper **"Attention is All You Need"** [14] introduced a groundbreaking model called the **Transformer**.
- Key contributions:
  - Eliminates the need for recurrent units (e.g., RNNs, LSTMs) in sequence-to-sequence tasks.
  - Fully relies on self-attention mechanisms for capturing dependencies.
- The Transformer model has revolutionized sequence modeling tasks such as machine translation, text summarization, and more.

Introduction to the Transformer

- The following figure illustrates the full Transformer architecture.

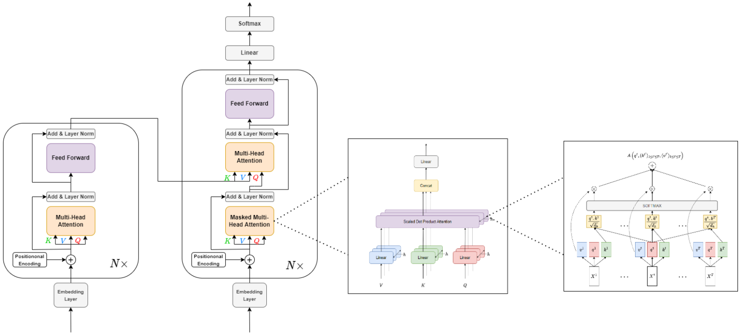
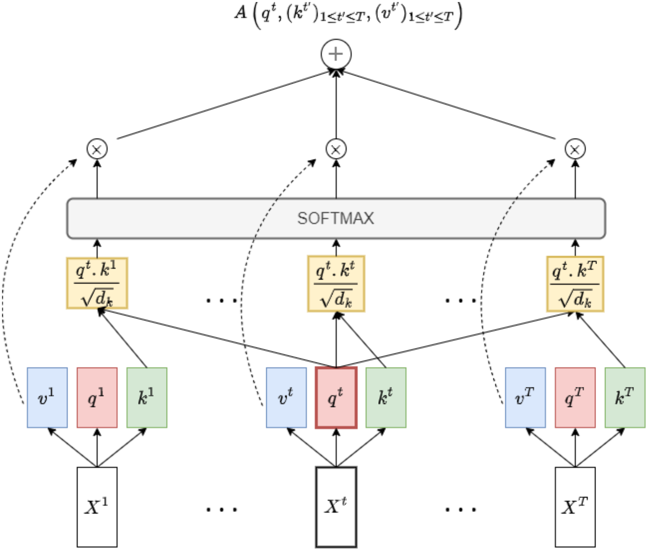


Figure: The Transformer Architecture [14].

- Given a sequence of D-dimensional input vectors $(X^t)_{1 \leq t \leq T}$, we project each vector X^t into:
  - Query space: $q^t = W_Q^T X^t,\ W_Q \in \mathbb{R}^{D \times d_q}$,
  - Key space: $k^t = W_K^T X^t,\ W_K \in \mathbb{R}^{D \times d_k}$,
  - Value space: $v^t = W_V^T X^t,\ W_V \in \mathbb{R}^{D \times d_v}$.
- **Objective**: Create a **contextual embedding** for each query q^t, leveraging all keys $(k^{t'})_{1 \leq t' \leq T}$ and values $(v^{t'})_{1 \leq t' \leq T}$.
- **Intuition**: Compute the attention weights $\alpha^{<t,t'>}$ to determine how much each value $v^{t'}$ contributes to the embedding $A\left(q^t, (k^{t'})_{1 \leq t' \leq T}, (v^{t'})_{1 \leq t' \leq T}\right)$.

Creating a Contextual Embedding with Self-Attention



- Use the **scaled dot product alignment function** [14] to compute similarity scores:

$$e^{<t,t'>} = \frac{q^t \cdot k^{t'}}{\sqrt{d_k}}$$

- Convert similarity scores to attention weights using the softmax distribution:

$$\alpha^{<t,t'>} = \frac{e^{<t,t'>}}{\sum_{s=1}^{T} e^{<t,s>}}$$

- Compute the **contextual embedding**:

$$A\left(q^t, (k^{t'})_{1 \leq t' \leq T}, (v^{t'})_{1 \leq t' \leq T}\right) = \sum_{t'=1}^{T} \alpha^{<t,t'>} v^{t'}$$

- To compute contextual embeddings for all input vectors $(X^t)_{1 \leq t \leq T}$, we define:

$$Q = \begin{bmatrix} q^1 \\ \vdots \\ q^T \end{bmatrix} \in \mathbb{R}^{T \times d_q}, \; K = \begin{bmatrix} k^1 \\ \vdots \\ k^T \end{bmatrix} \in \mathbb{R}^{T \times d_k}, \; V = \begin{bmatrix} v^1 \\ \vdots \\ v^T \end{bmatrix} \in \mathbb{R}^{T \times d_v}.$$

- Each q^t, k^t, v^t is computed using projection matrices:

$$q^t = W_Q^T X^t, \quad k^t = W_K^T X^t, \quad v^t = W_V^T X^t.$$

- Q, K, V represent the query, key, and value matrices, respectively.

Scaled Dot Product Attention Matrix

- Definition:

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

- Explanation:
  - $\frac{QK^T}{\sqrt{d_k}}$: Computes pairwise similarities between queries and keys.
  - Softmax: Converts similarities into attention weights.
  - Multiplication with V: Aggregates values using attention weights.
- Each row of $A(Q,K,V)$ corresponds to:

$$A(q^t,K,V) = \sum_{t'=1}^{T} \alpha^{<t,t'>} v^{t'}, \quad \forall t \in \{1,\ldots,T\}.$$

- **Objective:** Extend the attention mechanism to multiple heads to capture diverse notions of similarity.
- Attention mechanism is applied h times:

$$A\left(QW_Q^{h'}, KW_K^{h'}, VW_V^{h'}\right) \quad \text{for } h' \in \{1, \ldots, h\}.$$

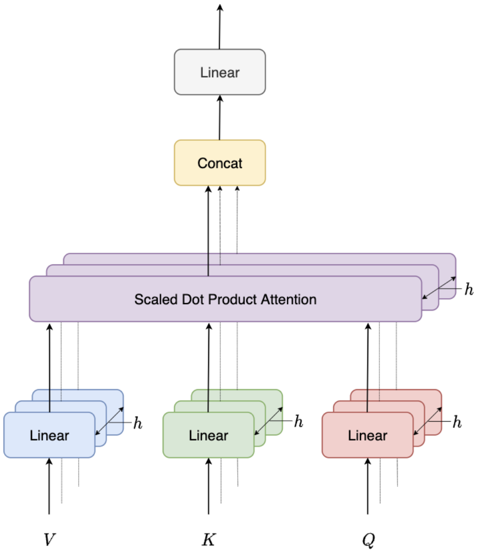
- Projection matrices for each head h':

$$W_Q^{h'} \in \mathbb{R}^{d_q \times p_q}, \quad W_K^{h'} \in \mathbb{R}^{d_k \times p_k}, \quad W_V^{h'} \in \mathbb{R}^{d_v \times p_v}.$$

- Outputs are concatenated and projected:

$$P = \text{concat}\left(A_1, \ldots, A_h\right) W_o \in \mathbb{R}^{T_q \times p_o}.$$

- The following figure illustrates MHA with h attention heads



Positional Encoding: Intuition and Key Idea

- **Objective:** Incorporate positional information into the permutation-invariant attention mechanism to reflect the order of sequence elements.
- **Intuition:**
  - Positions in a sequence need a unique representation to differentiate elements based on their location.
  - Shifting a positional encoding by k steps results in a consistent transformation that preserves relative distances.
- **Key Idea:**
  - Add positional encoding vectors $p^1, \ldots, p^T \in \mathbb{R}^D$ to input embeddings $X^1, \ldots, X^T$.
  - Use periodic functions (sine and cosine) to define the positional encodings in a way that captures relative positions effectively.

- **Method:**
  - Positional encoding at step t:

$$p_d^t = \begin{cases} \sin(w_d t), & \text{if } d \text{ is odd,} \\ \cos(w_d t), & \text{if } d \text{ is even.} \end{cases}$$

    Where $w_d = \frac{1}{10000^{\frac{2d}{D}}}$ ensures unique frequencies for different dimensions.
  - Positional encodings are added to input embeddings:

$$\tilde{X}^t = X^t + p^t.$$

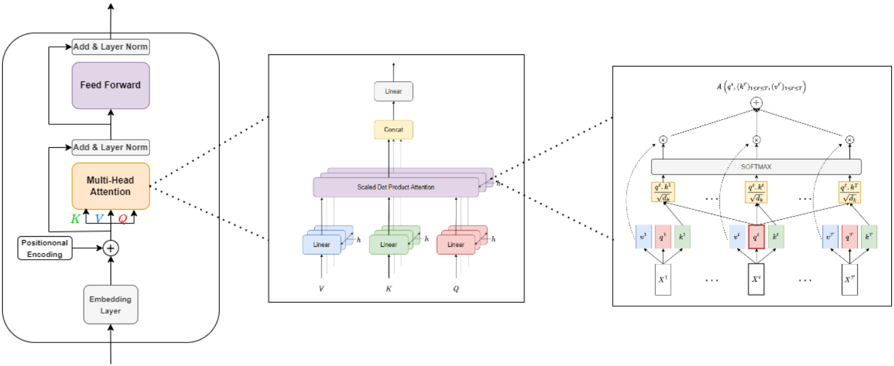
- **Properties:**
  - **Shift Consistency:** Shifting p^t by k steps aligns with p^{t+k}.
  - **Relative Distance Encoding:** The sine and cosine functions ensure relative positional information is preserved across sequences.

The Transformer Architecture

- **Sequence-to-Sequence Model:** Built entirely on attention mechanisms, eliminating recurrent units.
- **Core Components:**
  - **Multi-Head Attention Layer:** Captures different notions of similarity.
  - **Feed-Forward Layer:** Applies pointwise transformations.
  - **Normalization Layer:** Ensures stability and accelerates convergence.
- **Architecture Overview:** Combines stacked encoder and decoder layers to process and generate sequences efficiently.

The Encoder Layer

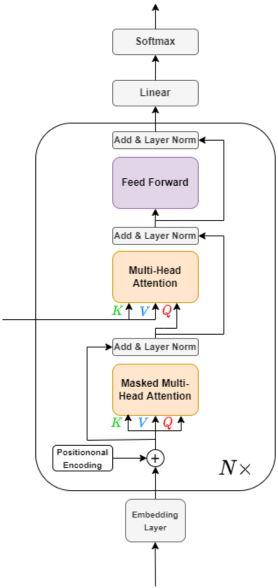
- **Objective:** Generate attention-based contextual embeddings that focus on relevant parts of the input sequence.
- **Structure:**
  - Stack of $N = 6$ identical layers.
  - Each layer consists of:
    - **Multi-Head Self-Attention:** Re-averages value vectors for contextual embeddings.
    - **Feed-Forward Layer:** Fully connected, applied pointwise.
    - **Residual Connections and Normalization:** Added after each sub-layer.
- **Output Dimension:** $d_{\text{model}} = 512$ for all layers.



Add & Layer Norm
Feed Forward
Add & Layer Norm
Multi-Head Attention
K
V
Q
Positiononal Encoding
Embedding Layer
Linear
Concat
Scaled Dot Product Attention
h
Linear
Linear
Linear
V
K
Q
$A\left(q^t, (k^{t'})_{1\le t'\le T}, (v^{t'})_{1\le t'\le T}\right)$
SOFTMAX
$\frac{q^t.k^1}{\sqrt{d_k}}$
$\frac{q^t.k^t}{\sqrt{d_k}}$
$\frac{q^t.k^T}{\sqrt{d_k}}$
v^1 q^1 k^1
v^t q^t k^t
v^T q^T k^T
X^1
X^t
X^T

- **Objective:** Retrieve and use information from encoder outputs to generate target sequences.
- **Structure:**
  - Stack of $N = 6$ identical layers.
  - Each layer includes:
    - **Masked Multi-Head Self-Attention:** Prevents information leakage (look-ahead masking).
    - **Multi-Head Attention:** Queries the encoder outputs.
    - **Feed-Forward Layer:** Applies pointwise transformations.
    - **Residual Connections and Normalization:** Enhance gradient flow and stability.

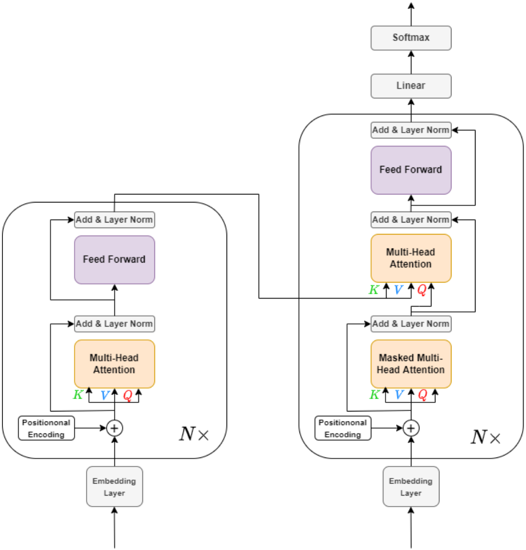
The Decoder Layer in the Transformer

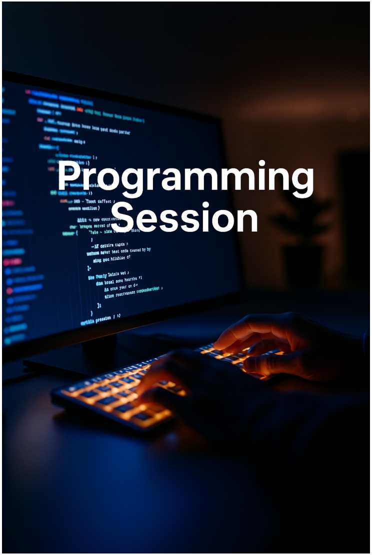


The Complete Transformer Architecture

- **Input Processing:**
  - Input sequence $X = (X^1, \dots, X^{T_x})$ embedded and combined with positional encodings.
  - Encoder outputs context-aware representations $H = (h^1, \dots, h^{T_x})$.
- **Decoding Process:**
  - Decoder uses:
    - **Self-Attention:** Processes previously generated tokens with masked attention.
    - **Encoder-Decoder Attention:** Focuses on encoder outputs H to generate context for predictions.
  - Outputs generated step-by-step using linear and softmax layers.

The Transformer Architecture





Programming Session 7: Section 2

- Section 2: Coding the Transformer Architecture
- *Click here to access the programming session*

Solution will be posted tonight on the GitHub page.

- *Click here to access ccess the GitHub Page*

Outline

Position of the Problem

Temporal Processing using RNNs

The Transformer Architecture

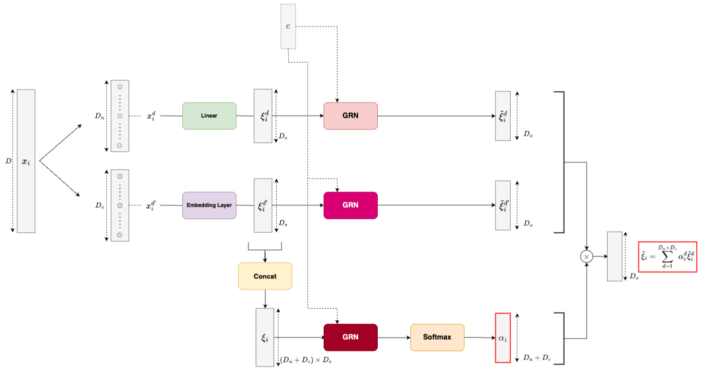
The Variable Selection Network

The TFT Architecture

Programming Session: Forecasting daily realized volatility of 31 stock indices

Imperial College Business School

- **Explore the Variable Selection Network (VSN):** Click here for the detailed implementation

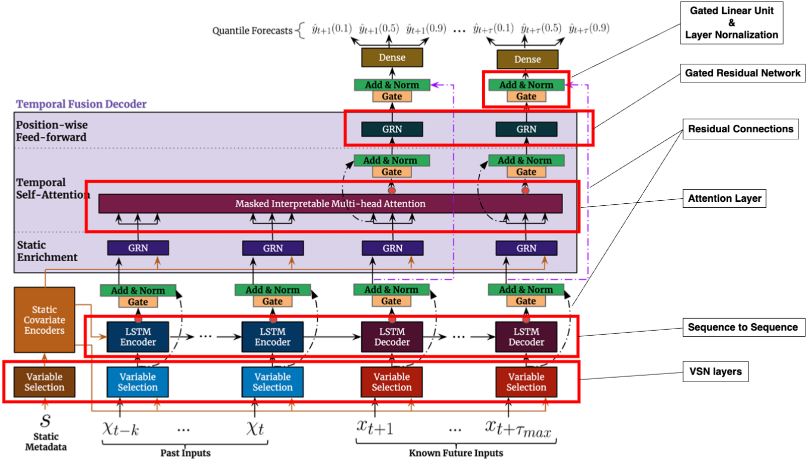


- **Overview:** A specialized deep learning model for time series forecasting with the following components:
  - **Variable Selection Networks (VSN):**
    - Dynamically select the most relevant features from static, time-varying known features, and time-varying unknown features.
    - Employ Gated Residual Networks for feature transformation and importance estimation.
  - **Sequence-to-Sequence Framework:**
    - Encoder-decoder architecture for multi-step forecasting.
    - Encoder processes historical data, while the decoder generates future predictions.

Temporal Fusion Transformer Architecture

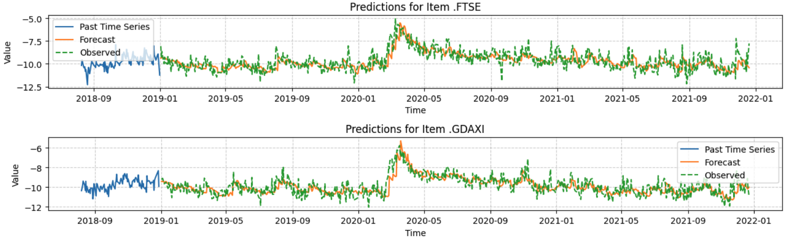
- **Masked Multi-Head Attention:**
  - Enables context-aware forecasting by focusing on relevant time steps in the past.
  - Prevents information leakage by masking future time steps during decoding.
- **Gated Residual Networks (GRN):**
  - Adds non-linear transformations and flexible gating mechanisms.
  - Regularizes and improves robustness across diverse datasets.

TFT Architecture: High-Level View



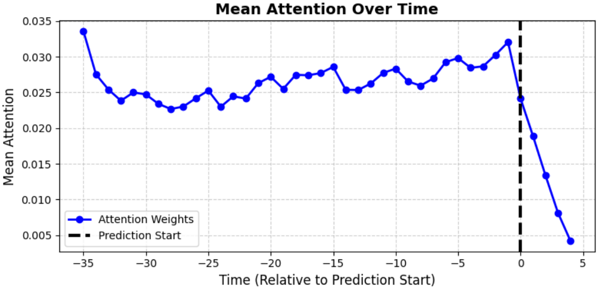
- **Objective:** Implement and experiment with the Temporal Fusion Transformer (TFT) for forecasting realized volatility.
- **Dataset:** Realized volatility data from 31 financial indices.
- **Goals:**
  - Build the TFT model architecture.
  - Train the model on time series data with static, time-varying known, and time-varying unknown features.
  - Evaluate predictions and interpret model outputs.

- **Task:** Forecast realized volatility for 31 indices.
- **Outcome**: Example of predicted vs. actual realized volatility for two indices.



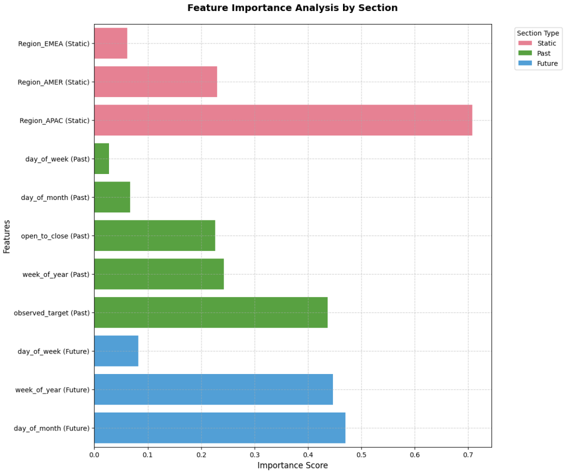
Results: Attention Weights

- **Objective:** Understand how the model uses historical data for forecasting by highlighting the most influential historical time steps.



Results: Feature Importance

- **Objective:** Quantify the impact of input features on predictions.



Quiz Time!

Click here to take the quiz

Feedback Poll

Click here to participate in the poll

Thank you for your attention

Imperial College Business School

Module Excellence Survey (MODES)

We are going to ask you to spend the next 10 minutes providing feedback on your module. Your feedback is very helpful in enhancing student experiences here at Imperial.

The instructor will leave the room whilst you provide feedback and your feedback is anonymous and instructors will only see your feedback once grading of assessment is complete.

- Please check your email now for the survey link (hosting@evasys.co.uk) or use the QR code above.
- There is a survey for each instructor who taught more than 3 hours on the module (including Teaching Assistants, where applicable)
- Please:
  - Ensure your feedback is constructive
  - Identify areas of good practice, as well as any suggestions for improvement.
  - Note that the scale runs from negative on the left (strongly disagree) to positive on the right (strongly agree)
- Your feedback will be passed to the instructors, Programme Team and the School's senior management for consideration.
- Programme Teams will provide a summary response to the feedback received from your cohort.

Thank you.

[1] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *3rd International Conference on Learning Representations, ICLR 2015*. 2015.

[2] Bram Bakker et al. "Reinforcement Learning with Long Short-Term Memory.". In: *NIPS*. 2001, pp. 1475–1482.

[3] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. 2014.

[4] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee. 2013, pp. 6645–6649.

[5] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[6] YiFei Li and Han Cao. "Prediction for tourism flow based on LSTM neural network". In: *Procedia Computer Science* 129 (2018), pp. 277–283.

[7] Andre Martins and Ramon Astudillo. "From softmax to sparsemax: A sparse model of attention and multi-label classification". In: *International conference on machine learning*. PMLR. 2016, pp. 1614–1623.

[8] André Martins et al. "Sparse and continuous attention mechanisms". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 20989–21001.

References III

[9] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318.

[10] Lawrence Rabiner and Biinghwang Juang. "An introduction to hidden Markov models". In: *ieee assp magazine* 3.1 (1986), pp. 4–16.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[12] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling". In: *Thirteenth annual conference of the international speech communication association*. 2012.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[14] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[15] Peilu Wang et al. "Part-of-speech tagging with bidirectional long short-term memory recurrent neural network". In: *arXiv preprint arXiv:1510.06168* (2015).