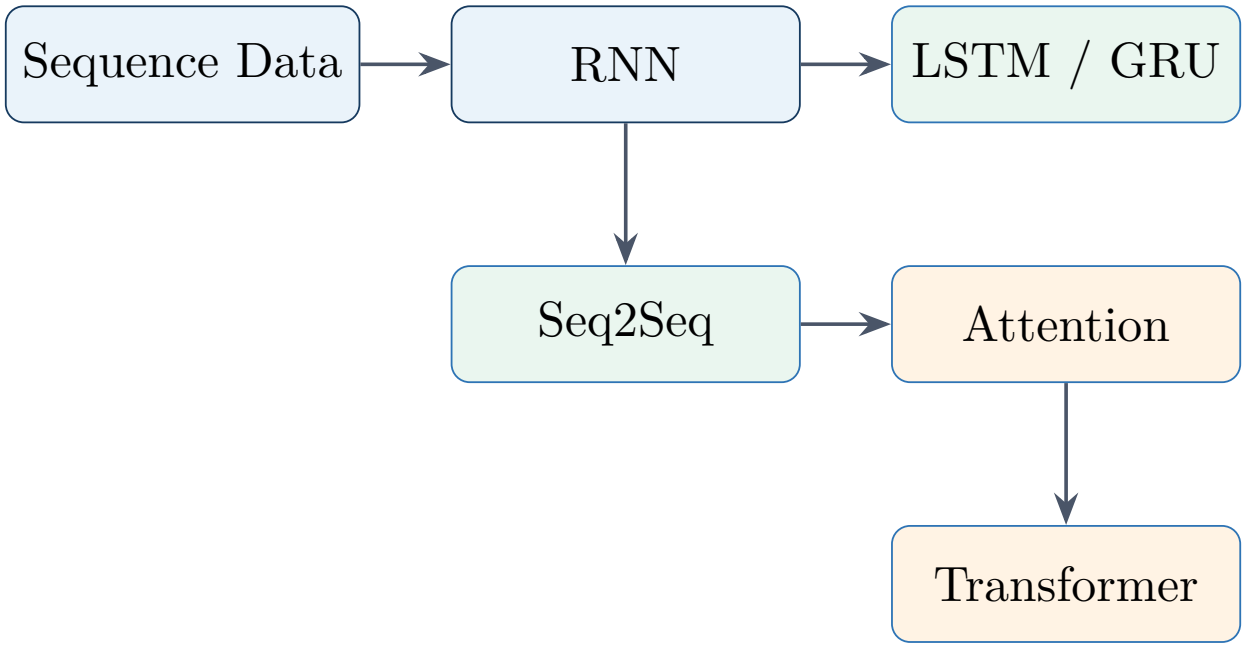
RNNs, Sequence-to-Sequence Learning, Attention Mechanism, and Transformers

Mathematical Notes

Recurrent States, BPTT, LSTM, GRU, Encoder-Decoder Learning, Teacher Forcing, Attention, Q-K-V, Self-Attention, Multi-Head Attention, Positional Encoding, and Applications

Central idea. Sequence models learn from ordered observations. RNNs carry information through recurrent states, gated models control memory, Seq2Seq models map one sequence to another, attention selects relevant states dynamically, and Transformers replace recurrence with parallel attention-based interactions.

Concept Map



I. Sequence Data and Recurrent Neural Networks

1. Why Sequence Models?

A sequence is an ordered collection

$$X = (x_1, x_2, \ldots, x_T).$$

The order often carries essential information. Examples include sentences, speech, sensor measurements, financial time series, weather observations, and user-click histories.

Worked example: order changes meaning.

Consider two token sequences:

$$S_1 = (\text{dog}, \text{bites}, \text{man}),$$

$$S_2 = (\text{man}, \text{bites}, \text{dog}).$$

Both have the same unigram count vector over

$$V = \{\text{dog}, \text{bites}, \text{man}\} :$$

$$x_{S_1} = x_{S_2} = (1, 1, 1).$$

However, their bigram sets differ:

$$B(S_1) = \{\text{dog bites}, \text{bites man}\},$$

$$B(S_2) = \{\text{man bites}, \text{bites dog}\}.$$

Thus a representation that ignores order cannot distinguish these meanings.

2. Basic RNN State Update

An RNN updates its hidden state using

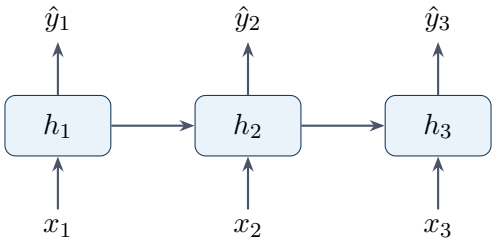
$$\boxed{h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)}.$$

The output may be computed by

$$\boxed{\hat{y}_t = g(W_{hy}h_t + b_y)},$$

where g is selected for the task, such as sigmoid for binary prediction or softmax for multiclass prediction.

The same parameters are reused at every time step.



Worked example: scalar RNN forward pass.

Let

$$W_{xh} = 0.5, \quad W_{hh} = 0.8, \quad b_h = 0, \quad h_0 = 0,$$

and

$$(x_1, x_2, x_3) = (1, 2, -1).$$

At $t = 1$,

$$a_1 = 0.5(1) + 0.8(0) = 0.5,$$

$$h_1 = \tanh(0.5) = 0.4621.$$

At $t = 2$,

$$a_2 = 0.5(2) + 0.8(0.4621) = 1.3697,$$

$$h_2 = \tanh(1.3697) = 0.8786.$$

At $t = 3$,

$$a_3 = 0.5(-1) + 0.8(0.8786) = 0.2029,$$

$$h_3 = \tanh(0.2029) = 0.2002.$$

t	x_t	Preactivation a_t	Hidden state h_t
1	1	0.5000	0.4621
2	2	1.3697	0.8786
3	−1	0.2029	0.2002

Let $W_{hy} = 1.2$ and $b_y = -0.1$. A final binary output is

$$\hat{y}_3 = \text{sigmoid}(1.2h_3 - 0.1) = \text{sigmoid}(0.1402) \approx 0.5350.$$

$$\boxed{h_3 \approx 0.2002, \qquad \hat{y}_3 \approx 0.5350.}$$

3. RNN Input-Output Structures

Structure	Example	Mapping
One-to-one	Image classification	one input to one output
One-to-many	Image captioning	one input to an output sequence
Many-to-one	Sentiment classification	input sequence to one label
Many-to-many aligned	POS tagging	one output per input position
Many-to-many unaligned	Translation	input and output lengths differ

Worked example: output-shape calculation.

Suppose a batch contains $B = 32$ sequences, each with $T = 20$ positions. The hidden dimension is $d_h = 64$ and the tag set has $d_y = 10$ classes.

For aligned many-to-many tagging,

$$H \in \mathbb{R}^{32\times 20\times 64},$$

and the output logits have shape

$$Y \in \mathbb{R}^{32\times 20\times 10}.$$

The number of output logits is

$$32 \times 20 \times 10 = 6400.$$

For many-to-one classification, only the final state is used, so the output shape becomes

$$Y_{\text{many-to-one}} \in \mathbb{R}^{32\times 10},$$

containing

$$32 \times 10 = 320$$

logits.

4. RNN Parameter Count

For input dimension d_x, hidden dimension d_h, and output dimension d_y, a basic RNN has

$$W_{xh} \in \mathbb{R}^{d_h\times d_x}, \quad W_{hh} \in \mathbb{R}^{d_h\times d_h}, \quad b_h \in \mathbb{R}^{d_h},$$

$$W_{hy} \in \mathbb{R}^{d_y\times d_h}, \quad b_y \in \mathbb{R}^{d_y}.$$

Therefore,

$$\boxed{P = d_h d_x + d_h^2 + d_h + d_y d_h + d_y.}$$

Worked example: parameter counting.
Let $d_x = 3$, $d_h = 4$, and $d_y = 2$.

Parameter	Shape	Count
W_{xh}	4×3	12
W_{hh}	4×4	16
b_h	4	4
W_{hy}	2×4	8
b_y	2	2
Total		42

$$\boxed{P = 12 + 16 + 4 + 8 + 2 = 42.}$$

This count does not grow with sequence length because the parameters are shared across time.

II. Backpropagation Through Time

An unrolled RNN is trained by Backpropagation Through Time (BPTT). A gradient reaching an earlier state contains a product of recurrent Jacobians:

$$\boxed{\frac{\partial L}{\partial h_k} = \frac{\partial L}{\partial h_t} \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}}.$$

Repeated multiplication can cause vanishing or exploding gradients.

1. Vanishing and Exploding Gradients

Worked example: repeated gradient factors.

Assume the magnitude of the local recurrent derivative is constant.

For a derivative factor 0.5 across ten steps,

$$g_{\text{early}} = (0.5)^{10} = 0.0009766.$$

The gradient becomes almost zero.

For a derivative factor 1.5 across ten steps,

$$g_{\text{early}} = (1.5)^{10} = 57.665.$$

The gradient becomes very large.

Local factor	Ten-step product	Effect
0.5	0.0009766	Vanishing
1.0	1.0000	Stable magnitude
1.5	57.665	Exploding

2. Gradient Clipping

If the gradient norm exceeds threshold τ, norm clipping uses

$$\boxed{g_{\text{clip}} = g\frac{\tau}{\|g\|_2} \quad \text{when } \|g\|_2 > \tau.}$$

Worked example: clipping an exploding gradient.

Let

$$g = \begin{bmatrix} 6 \\ 8 \end{bmatrix}, \qquad \|g\|_2 = \sqrt{6^2 + 8^2} = 10,$$

and let $\tau = 5$. Then

$$g_{\text{clip}} = \begin{bmatrix} 6 \\ 8 \end{bmatrix} \frac{5}{10} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

The clipped norm is

$$\|g_{\text{clip}}\|_2 = \sqrt{3^2 + 4^2} = 5.$$

$$\boxed{g_{\text{clip}} = (3, 4)^\top.}$$

III. Gated Recurrent Networks

1. Long Short-Term Memory

An LSTM introduces a persistent cell state c_t and gates that control information flow:

$$f_t = \text{sigmoid}(W_f[h_{t-1}, x_t] + b_f),$$

$$i_t = \text{sigmoid}(W_i[h_{t-1}, x_t] + b_i),$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c),$$

$$o_t = \text{sigmoid}(W_o[h_{t-1}, x_t] + b_o),$$

$$\boxed{c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t},$$

$$\boxed{h_t = o_t \odot \tanh(c_t)}.$$

Worked example: one scalar LSTM step.
Suppose

$$h_{t-1} = 0.2, \qquad c_{t-1} = 0.4, \qquad x_t = 0.7,$$

and the learned gate preactivations are

$$a_f = 1.0, \quad a_i = 0.2, \quad a_c = 0.5, \quad a_o = 0.8.$$

Then

$$f_t = \text{sigmoid}(1.0) = 0.7311,$$

$$i_t = \text{sigmoid}(0.2) = 0.5498,$$

$$\tilde{c}_t = \tanh(0.5) = 0.4621,$$

$$o_t = \text{sigmoid}(0.8) = 0.6900.$$

The new cell state is

$$\begin{aligned} c_t &= 0.7311(0.4) + 0.5498(0.4621) \\ &= 0.2924 + 0.2541 \\ &= 0.5465. \end{aligned}$$

The new hidden state is

$$h_t = 0.6900 \tanh(0.5465) \approx 0.6900(0.4979) \approx 0.3436.$$

Quantity	Activation	Value
Forget gate	sigmoid(1.0)	0.7311
Input gate	sigmoid(0.2)	0.5498
Candidate	tanh(0.5)	0.4621
Output gate	sigmoid(0.8)	0.6900
Cell state		0.5465
Hidden state		0.3436

$$\boxed{c_t \approx 0.5465, \qquad h_t \approx 0.3436.}$$

2. Gated Recurrent Unit

A GRU uses update and reset gates:

$$z_t = \text{sigmoid}(W_z[h_{t-1}, x_t]),$$

$$r_t = \text{sigmoid}(W_r[h_{t-1}, x_t]),$$

$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t]),$$

$$\boxed{h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t}.$$

Worked example: one scalar GRU step.
Let

$$h_{t-1} = 0.4, \quad x_t = 0.6, \quad z_t = 0.6, \quad r_t = 0.25.$$

Assume the candidate calculation is

$$\tilde{h}_t = \tanh\left(0.8(r_t h_{t-1}) + 0.5x_t\right).$$

First,

$$r_t h_{t-1} = 0.25(0.4) = 0.1.$$

Thus,

$$\tilde{h}_t = \tanh(0.8(0.1) + 0.5(0.6)) = \tanh(0.38) \approx 0.3627.$$

The state update is

$$\begin{aligned} h_t &= (1 - 0.6)(0.4) + 0.6(0.3627) \\ &= 0.16 + 0.2176 \\ &= 0.3776. \end{aligned}$$

$$\boxed{h_t \approx 0.3776.}$$

3. Parameter Comparison

Ignoring output layers, with input size d_x and hidden size d_h:

$$P_{\text{RNN}} = d_h d_x + d_h^2 + d_h,$$

$$P_{\text{GRU}} = 3P_{\text{RNN}},$$

$$P_{\text{LSTM}} = 4P_{\text{RNN}}.$$

Worked example: gated-model parameter count.
Let $d_x = 10$ and $d_h = 20$. Then

$$P_{\text{RNN}} = 20(10) + 20^2 + 20 = 620,$$

$$P_{\text{GRU}} = 3(620) = 1860,$$

$$P_{\text{LSTM}} = 4(620) = 2480.$$

Model	Number of gate/state transformations	Parameters
Simple RNN	1	620
GRU	3	1860
LSTM	4	2480

The additional parameters give gated models greater control over memory.

IV. Sequence-to-Sequence Learning

Seq2Seq maps a source sequence

$$x = (x_1, \ldots, x_{T_x})$$

to a target sequence

$$y = (y_1, \ldots, y_{T_y}),$$

where T_x and T_y may differ.

1. Encoder and Fixed Context Vector

The encoder computes

$$h_t^{\text{enc}} = f_{\text{enc}}(x_t, h_{t-1}^{\text{enc}}),$$

and a basic Seq2Seq model uses

$$\boxed{c = h_{T_x}^{\text{enc}}}$$

as a fixed context vector.

Worked example: scalar encoder.

Let

$$h_t = \tanh(0.7x_t + 0.6h_{t-1}), \qquad h_0 = 0,$$

and

$$(x_1, x_2, x_3) = (1, 0.5, -0.5).$$

Then

$$h_1 = \tanh(0.7) = 0.6044,$$

$$h_2 = \tanh(0.7(0.5) + 0.6(0.6044)) = \tanh(0.7126) = 0.6123,$$

$$h_3 = \tanh(0.7(-0.5) + 0.6(0.6123)) = \tanh(0.0174) = 0.0174.$$

Therefore,

$$\boxed{c = h_3 = 0.0174.}$$

The final scalar must summarize the complete three-token source, illustrating the fixed-vector bottleneck.

2. Conditional Sequence Probability and Loss

The decoder factorizes the target probability as

$$\boxed{P(y \mid x) = \prod_{t=1}^{T_y} P(y_t \mid y_{<t}, c)}.$$

The sequence negative log-likelihood is

$$\boxed{L = -\sum_{t=1}^{T_y} \log P(y_t \mid y_{<t}, c)}.$$

Worked example: target-sequence probability.

Suppose the correct target has three tokens with conditional probabilities

Step	Correct token	Conditional probability
1	y_1	0.80
2	y_2	0.70
3	$\langle \text{EOS} \rangle$	0.90

Then

$$P(y \mid x) = 0.80(0.70)(0.90) = 0.504.$$

The loss is

$$L = -\log(0.504) \approx 0.685.$$

The average token loss is

$$\bar{L} = \frac{0.685}{3} \approx 0.228.$$

$$\boxed{P(y \mid x) = 0.504, \qquad L \approx 0.685.}$$

3. Teacher Forcing

During training, teacher forcing uses the correct previous token rather than the model's previous prediction:

$$y_{t-1}^{\text{true}} \rightarrow \text{Decoder} \rightarrow \hat{y}_t.$$

Worked example: expected number of teacher-forced steps.

Suppose a target contains $T_y = 20$ decoding steps and teacher forcing is applied independently with probability $p_{TF} = 0.75$.

The expected number of teacher-forced steps is

$$\mathbb{E}[N_{TF}] = T_y p_{TF} = 20(0.75) = 15.$$

The expected number of steps using model predictions is

$$20 - 15 = 5.$$

$$\boxed{\mathbb{E}[N_{TF}] = 15, \qquad \mathbb{E}[N_{\text{model}}] = 5.}$$

Teacher forcing stabilizes training, but the difference between training and inference creates exposure bias.

4. Greedy and Beam-Search Decoding

Greedy decoding chooses the highest-probability token at each step. Beam search retains several partial sequences.

Worked example: greedy versus beam search.

At the first step,

$$P(A) = 0.55, \qquad P(B) = 0.45.$$

If A is chosen, the next probability of $\langle \text{EOS} \rangle$ is 0.50:

$$P(A, \text{EOS}) = 0.55(0.50) = 0.275.$$

If B is retained by a beam, the next token C has probability 0.90 and then EOS has probability 0.80:

$$P(B, C, \text{EOS}) = 0.45(0.90)(0.80) = 0.324.$$

Although greedy decoding initially prefers A,

$$0.324 > 0.275.$$

Therefore, a beam can recover the higher-probability complete sequence

$$\boxed{B \to C \to \text{EOS}.}$$

V. Attention Mechanism

Attention removes the need to compress all source information into one fixed context vector. At decoder step t,

$$e_{t,i} = \text{Score}(h_{t-1}^{\text{dec}}, h_i^{\text{enc}}),$$

$$\boxed{\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}},$$

$$\boxed{c_t = \sum_i \alpha_{t,i} h_i^{\text{enc}}}.$$

1. Alignment Weights and Context

Worked example: softmax attention.

Suppose the alignment scores are

$$e = (1, 2, 0).$$

The exponentials are

$$(e^1, e^2, e^0) = (2.718, 7.389, 1).$$

Their sum is

$$2.718 + 7.389 + 1 = 11.107.$$

Thus,

$$\alpha = \left(\frac{2.718}{11.107}, \frac{7.389}{11.107}, \frac{1}{11.107}\right) = (0.245, 0.665, 0.090).$$

Let the scalar values be

$$v_1 = 2, \qquad v_2 = 4, \qquad v_3 = 1.$$

Then

$$\begin{aligned} c &= 0.245(2) + 0.665(4) + 0.090(1) \\ &= 0.490 + 2.660 + 0.090 \\ &= 3.240. \end{aligned}$$

$$\boxed{\alpha = (0.245, 0.665, 0.090), \qquad c = 3.240.}$$

The second source position receives the greatest attention.

2. Query, Key, and Value

The query describes what the current position needs, a key describes what a candidate position contains, and a value contains the information returned from that position.

For dot-product attention,

$$e_i = q^\top k_i.$$

Worked example: Q-K-V attention.

Let

$$q = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

$$k_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad k_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad k_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The dot-product scores are

$$q^\top k_1 = 1, \quad q^\top k_2 = 2, \quad q^\top k_3 = 3.$$

For $d_k = 2$, the scaled scores are

$$s = \frac{(1,2,3)}{\sqrt{2}} = (0.707, 1.414, 2.121).$$

Softmax gives approximately

$$\alpha = (0.140, 0.284, 0.576).$$

Let

$$v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \quad v_3 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

The output is

$$\begin{aligned} z &= 0.140v_1 + 0.284v_2 + 0.576v_3 \\ &= \begin{bmatrix} 0.140 + 0 + 1.728 \\ 0 + 0.568 + 0.576 \end{bmatrix} \\ &= \begin{bmatrix} 1.868 \\ 1.144 \end{bmatrix}. \end{aligned}$$

$$\boxed{z \approx (1.868, 1.144)^\top.}$$

VI. Transformer Attention

Transformers replace recurrence with attention-based token interactions.

1. Scaled Dot-Product Attention

The fundamental operation is

$$\boxed{\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V.}$$

Scaling prevents large dot products from making softmax excessively sharp. The optional mask M can prevent attention to forbidden positions.

Worked example: matrix self-attention.

Let

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad W^Q = W^K = W^V = I_2.$$

Therefore,

$$Q = K = V = X.$$

The score matrix is

$$QK^\top = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}.$$

Because $d_k = 2$,

$$S = \frac{QK^\top}{\sqrt{2}} = \begin{bmatrix} 0.707 & 0 & 0.707 \\ 0 & 0.707 & 0.707 \\ 0.707 & 0.707 & 1.414 \end{bmatrix}.$$

Applying row-wise softmax gives

$$A \approx \begin{bmatrix} 0.401 & 0.198 & 0.401 \\ 0.198 & 0.401 & 0.401 \\ 0.248 & 0.248 & 0.503 \end{bmatrix}.$$

Finally,

$$Z = AV \approx \begin{bmatrix} 0.802 & 0.599 \\ 0.599 & 0.802 \\ 0.751 & 0.751 \end{bmatrix}.$$

$$\boxed{Z \approx \begin{bmatrix} 0.802 & 0.599 \\ 0.599 & 0.802 \\ 0.751 & 0.751 \end{bmatrix}.}$$

Each output row mixes information from all three input positions.

2. Causal Masked Self-Attention

In a decoder, position t must not access future target positions. A causal mask is

$$M_{ij} = \begin{cases} 0, & j \le i, \\ -\infty, & j > i. \end{cases}$$

Worked example: three-token causal mask.

For three tokens,

$$M = \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{bmatrix}.$$

Using the previous score matrix, the masked attention weights become approximately

$$A_{\text{mask}} = \begin{bmatrix} 1 & 0 & 0 \\ 0.330 & 0.670 & 0 \\ 0.248 & 0.248 & 0.503 \end{bmatrix}.$$

Thus,

$$Z_{\text{mask}} = A_{\text{mask}}V \approx \begin{bmatrix} 1 & 0 \\ 0.330 & 0.670 \\ 0.751 & 0.751 \end{bmatrix}.$$

The first position uses only itself, the second uses positions one and two, and the third can use all positions.

3. Cross-Attention

In encoder-decoder attention, the decoder provides queries, while encoder states provide keys and values.

Worked example: decoder query attending to encoder states.

Let

$$q = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

and encoder keys and values be

$$K = V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

The scaled scores are

$$\frac{q^\top K^\top}{\sqrt{2}} = (0.707, 0.707, 1.414).$$

Softmax gives

$$\alpha = (0.248, 0.248, 0.503).$$

Therefore,

$$c = \alpha V = 0.248(1,0) + 0.248(0,1) + 0.503(1,1),$$

$$\boxed{c = (0.751, 0.751).}$$

The third encoder position contributes the most.

VII. Multi-Head Attention and Position

1. Multi-Head Attention

For head i,

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).$$

The heads are concatenated and projected:

$$\boxed{\mathrm{MHA}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1,\ldots,\mathrm{head}_H)W^O}.$$

Worked example: combining two heads.

Suppose two attention heads return

$$\mathrm{head}_1 = (0.8, 0.6),$$

$$\mathrm{head}_2 = (0.3, 0.9).$$

The concatenated vector is

$$u = (0.8, 0.6, 0.3, 0.9).$$

Let

$$W^O = \begin{bmatrix} 0.5 & 0 \\ 0.5 & 0 \\ 0 & 0.5 \\ 0 & 0.5 \end{bmatrix}.$$

Then

$$\begin{aligned} \mathrm{MHA} &= uW^O \\ &= (0.5(0.8+0.6), 0.5(0.3+0.9)) \\ &= (0.7, 0.6). \end{aligned}$$

$$\boxed{\mathrm{MHA} = (0.7, 0.6).}$$

Different heads can represent different relationships before their information is combined.

2. Sinusoidal Positional Encoding

Self-attention alone does not identify token order. Classical positional encoding uses

$$\boxed{PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)},$$

$$\boxed{PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)}.$$

Worked example: position 2 with $d_{model} = 4$.

For $i = 0$,

$$PE(2, 0) = \sin(2) = 0.9093,$$

$$PE(2, 1) = \cos(2) = -0.4161.$$

For $i = 1$,

$$10000^{2/4} = 100,$$

$$PE(2,2) = \sin(2/100) = \sin(0.02) \approx 0.0200,$$

$$PE(2,3) = \cos(0.02) \approx 0.9998.$$

Therefore,

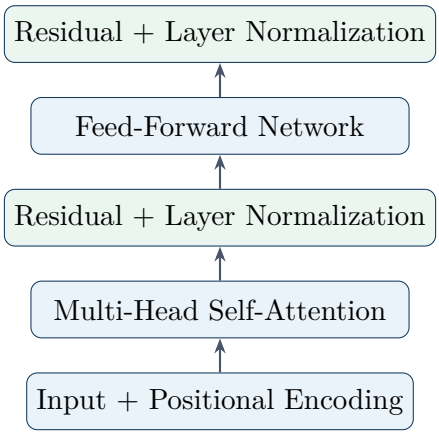
$$\boxed{PE(2) = (0.9093, -0.4161, 0.0200, 0.9998).}$$

If the token embedding is $e = (0.5, 0.1, -0.2, 0.3)$, the position-aware input is

$$e + PE(2) = (1.4093, -0.3161, -0.1800, 1.2998).$$

VIII. Transformer Encoder and Decoder

An encoder layer contains multi-head self-attention, a residual connection, layer normalization, and a position-wise feed-forward network. A decoder adds masked self-attention and encoder-decoder cross-attention.



1. Residual Connection and Layer Normalization

A residual connection forms

$$r = x + \text{Sublayer}(x).$$

Layer normalization is

$$\boxed{\text{LN}(r_i) = \gamma \frac{r_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta}.$$

Worked example: residual plus normalization.

Let

$$x = (1, 2), \qquad z_{\text{att}} = (0.4, 0.6).$$

The residual vector is

$$r = x + z_{\text{att}} = (1.4, 2.6).$$

Its mean is

$$\mu = \frac{1.4 + 2.6}{2} = 2.$$

Its variance is

$$\sigma^2 = \frac{(1.4 - 2)^2 + (2.6 - 2)^2}{2} = 0.36.$$

With $\gamma = 1$, $\beta = 0$, and negligible ϵ,

$$\text{LN}(r) = \left(\frac{-0.6}{0.6}, \frac{0.6}{0.6}\right) = (-1, 1).$$

$$\boxed{\text{LN}(x + z_{\text{att}}) = (-1, 1).}$$

2. Position-Wise Feed-Forward Network

The Transformer feed-forward network is

$$\boxed{\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2}.$$

It is applied independently to every sequence position.

Worked example: one token through an FFN.

Let

$$x = \begin{bmatrix} 1 & -1 \end{bmatrix}, \quad W_1 = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix}, \quad b_1 = (0, 0).$$

Then

$$xW_1 = (2, 1).$$

After ReLU,

$$u = \text{ReLU}(2, 1) = (2, 1).$$

Let

$$W_2 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \qquad b_2 = 0.$$

Therefore,

$$\mathrm{FFN}(x) = 2(0.5) + 1(1) = 2.$$

$$\boxed{\mathrm{FFN}(x) = 2.}$$

3. Decoder Probability

The Transformer decoder models

$$\boxed{P(y \mid x) = \prod_t P(y_t \mid y_{<t}, x)}.$$

Worked example: softmax output and cross-entropy.
Suppose the vocabulary logits at one decoding step are

$$z = (2.0, 1.0, 0.0).$$

Softmax gives

$$p_i = \frac{e^{z_i}}{e^2 + e^1 + e^0}.$$

Since

$$e^2 + e^1 + e^0 = 7.389 + 2.718 + 1 = 11.107,$$

we obtain

$$p = (0.665, 0.245, 0.090).$$

If the first token is correct, the cross-entropy loss is

$$L = -\log(0.665) = 0.408.$$

$$\boxed{p = (0.665, 0.245, 0.090), \qquad L = 0.408.}$$

IX. RNNs versus Transformers

Property	RNN, LSTM, or GRU	Transformer
Processing	Sequential recurrence	Parallel across positions during training
Memory	Hidden or cell state	Attention-based representation
Long dependencies	May be difficult	Direct attention path
Streaming	Natural	Requires causal or chunked design
Long-sequence cost	Sequential in T	Standard attention is quadratic in T
Small data	Often effective	Often benefits from pretraining

Worked Example: Computational Scaling

For an approximate comparison, recurrent hidden-state computation contains a term proportional to

$$Td^2,$$

while the attention-score computation contains a term proportional to

$$T^2d.$$

Let $T = 100$ and $d = 64$.
RNN-style recurrent work is approximately

$$100(64^2) = 409{,}600$$

multiplication units, performed sequentially across 100 steps.
Attention-score work is approximately

$$100^2(64) = 640{,}000$$

multiplication units, but the token-pair computations can be parallelized.
For $T = 1000$,

$$T^2d = 1000^2(64) = 64{,}000{,}000,$$

showing why standard attention becomes expensive for very long sequences.

Length T	Recurrent term Td^2	Attention-score term T^2d
100	409,600	640,000
1000	4,096,000	64,000,000

Model-selection principle. Transformers provide short attention paths and parallel training, but recurrent models remain useful for compact causal or streaming systems. The correct choice depends on data size, sequence length, latency, causality, hardware, and pretraining availability.

X. Applications and Model Selection

Domain	Applications	Suitable models
Natural language	Sentiment, translation, summarization, QA	BiRNN, LSTM, Transformer
Speech	Recognition, synthesis, translation	LSTM, GRU, Transformer
Time series	Forecasting, monitoring, anomaly detection	RNN, LSTM, Transformer
Recommendation	Next-item and click prediction	GRU, sequential Transformer
Computer vision	Classification and detection	Vision Transformer
Healthcare	Patient-event prediction	LSTM, Transformer
Multimodal AI	Text-image and audio-text modeling	Transformer architectures

Worked Example: Choosing a Model

Suppose three applications have the following requirements.

Application	Important conditions	Recommended starting model
Live sensor prediction	One sample at a time, strict causality, limited hardware	GRU
Machine translation	Long context, large parallel hardware, pretrained models	Transformer
Small medical sequence set	Limited labeled data, moderate sequence length	Regularized LSTM or GRU

The recommendation follows the constraints, not a claim that one architecture is universally best.

XI. Practice Problems

1. For $h_t = \tanh(0.4x_t + 0.5h_{t-1})$, calculate h_1 when $x_1 = 2$ and $h_0 = 0$.
2. A recurrent derivative factor is 0.7. Calculate its magnitude after eight steps.
3. In an LSTM, $f_t = 0.8$, $c_{t-1} = 0.5$, $i_t = 0.4$, and $\tilde{c}_t = 0.6$. Calculate c_t.
4. In a GRU, $z_t = 0.25$, $h_{t-1} = 0.8$, and $\tilde{h}_t = 0.2$. Calculate h_t.
5. A three-token target has conditional probabilities 0.9, 0.8, and 0.7. Calculate its sequence probability.
6. Convert attention scores $(0, 1)$ into softmax weights.
7. For $q = (1, 0)$ and $k = (2, 3)$, calculate the dot-product attention score.
8. For sequence length $T = 50$, how many entries are in the full $T \times T$ attention-score matrix?
9. Compute $PE(0, 0)$ and $PE(0, 1)$ for sinusoidal positional encoding.
10. A model gives probability 0.8 to the correct token. Calculate its cross-entropy loss.

Answer check.

1. $h_1 = \tanh(0.8) \approx 0.6640$.

2. $(0.7)^8 \approx 0.05765$.
3. $c_t = 0.8(0.5) + 0.4(0.6) = 0.64$.
4. $h_t = (1 - 0.25)(0.8) + 0.25(0.2) = 0.65$.
5. $0.9(0.8)(0.7) = 0.504$.
6. $\text{softmax}(0, 1) = (1/(1 + e), e/(1 + e)) \approx (0.269, 0.731)$.
7. $q^\top k = 1(2) + 0(3) = 2$.
8. $50^2 = 2500$ entries.
9. $PE(0, 0) = \sin(0) = 0$ and $PE(0, 1) = \cos(0) = 1$.
10. $L = -\log(0.8) \approx 0.223$.

XII. Key Takeaways

1. RNNs share recurrent parameters and summarize earlier information in hidden states.
2. BPTT creates products of derivatives, which can vanish or explode across long sequences.
3. LSTM and GRU gates regulate forgetting, writing, and exposing recurrent information.
4. Basic Seq2Seq compresses a source sequence into a context vector and factorizes target probability autoregressively.
5. Attention constructs a different weighted context for each query.
6. Transformer attention uses query, key, and value matrices with scaled dot products.
7. Causal masks prevent a decoder from using future target tokens.
8. Multi-head attention learns several interaction patterns, while positional encoding supplies order information.
9. Residual connections, layer normalization, and feed-forward networks complete the Transformer layer.
10. Architecture selection should consider sequence length, causality, latency, data volume, hardware, and pretraining.

Prepared By
Md. Atikuzzaman
Lecturer, Department of Computer Science and Engineering
Green University of Bangladesh
Email: atik@cse.green.edu.bd