Chapter 1

Statistics

1.1 Basics

- $\mathrm{E}(X) = \int x f(x)$ and $\mathrm{Var}(X) = \mathrm{E}((X-\mu)^2)$.
- Sample mean: $\overline{X} = \frac{1}{n}\sum X_i$.
- Sample variance: $S^2 = \frac{1}{n-1}\sum (X_i - \overline{X})^2$.
- Likelihood: Let X_i have joint pdf $f(\mathbf{x};\theta)$. Given observed values x_i of X_i, the *likelihood* of θ is $L(\theta) = f(\mathbf{x};\theta)$. The *maximum likelihood estimate* $\hat{\theta}(\mathbf{x})$ is the value of θ that maximizes $L(\theta)$.

1.2 Convergence of Random Variables

RVs X_n with cdf F_n.

- X_n *converges in distribution* to X (weakly) if

  $$\lim F_n(x) = F(x)$$

  for all x at which F is continuous.
- X_n *converges in probability* to X if for all $\epsilon > 0$,

  $$\lim P(|X_n - X| > \epsilon) = 0.$$

- X_n *converges almost surely* to X (strongly) if

  $$P(\lim X_n = X) = 1,$$

  i.e., events for which X_n does not converge to X have probability 0.

1.3 Parameter Estimation

An *estimator* is any statistic $\hat{\theta} = \hat{\theta}(X)$ used to estimate θ.

1.3.1 Properties of estimators

Mean squared error

- $\mathrm{MSE}(\hat{\theta}) = \mathrm{E}\left[(\hat{\theta} - \theta)^2\right]$.

Variance

- $\mathrm{Var}(\hat{\theta}) = \mathrm{E}\left[(\hat{\theta} - \mathrm{E}(\hat{\theta}))^2\right]$.

Bias

- $\mathrm{Bias}(\hat{\theta}) = \mathrm{E}(\hat{\theta}) - \theta$.
- Note that $\mathrm{MSE}(\hat{\theta}) = \mathrm{Var}(\hat{\theta}) + \mathrm{Bias}(\hat{\theta})^2$.
- The unbiased estimator with the smallest variance is known as the *minimum-variance unbiased estimator* (MVUE).

Consistency

- An estimator T_n of θ is *weakly consistent* if T_n converges in probability to θ.
- T_n is *strongly consistent* if T_n converges almost surely to θ.

Asymptotic normality

- T_n is *asymptotically normal* if

$$\sqrt{n}(T_n - \theta) \xrightarrow{D} N(0, V).$$

- Recall: CLT says that sample mean $\overline{X}$ is asymptotically normal.

Efficiency

- The *observed information* is

$$J(\theta) = -\frac{d^2 l}{d\theta^2}$$

  for scalar parameter θ or

$$J(\theta)_{ij} = -\frac{\partial^2 l}{\partial\theta_i \partial\theta_j}$$

  for $\theta = (\theta_1, \ldots, \theta_p)$.

- The larger $J(\hat{\theta})$ is, the more concentrated $l(\theta)$ is about $\hat{\theta}$.
- The *expected* or *Fisher information* is

$$I(\theta) = \mathrm{E}\left(-\frac{d^2 l}{d\theta^2}\right)$$

  or

$$I(\theta)_{ij} = \mathrm{E}\left(-\frac{\partial^2 l}{\partial\theta_i \partial\theta_j}\right).$$

- Equivalently, define *score* to be gradient of $l(\theta)$, i.e.,

$$s(\theta) = \frac{\partial l}{\partial \theta}.$$

Then expected value of score at θ is 0, i.e., $\mathrm{E}(s \mid \theta) = 0$. The Fisher information is defined to be the variance of the score $\mathrm{Var}(s(\theta)) = \mathrm{E}(s(\theta)s(\theta)^T)$.

- The *Cramer-Rao bound* says that for an unbiased estimator,

$$\mathrm{Var}(\hat{\theta}) \geq \frac{1}{I(\theta)}.$$

Generally, if $\mathrm{E}(\hat{\theta}) = \psi(\theta)$, then

$$\mathrm{Var}(\hat{\theta}) \geq \frac{(\psi'(\theta))^2}{I(\theta)} = \frac{(1 + b'(\theta))^2}{I(\theta)},$$

where b is the bias. In the unbiased multivariate case, the Cramer-Rao bound states that

$$\mathrm{cov}(T(X)) \geq I(\theta)^{-1},$$

where $M \geq 0$ means that M is positive semidefinite, i.e., $\mathbf{x}^T M \mathbf{x} \geq 0$ for all $\mathbf{x}$.

- The *efficiency* of an unbiased estimator is

$$e(T) = \frac{1/I(\theta)}{\mathrm{Var}(T)}.$$

By the Cramer-Rao bound, we know $e(T) \leq 1$. An estimator is *efficient* if $e(T) = 1$ for all θ, i.e., achieves Cramer-Rao bound for all θ. Thus, an efficient estimator is the MVUE (but not necessarily conversely).

Properties of MLEs

Let $\hat{\theta}$ be the MLE of θ. Then

- $\sqrt{n}(\hat{\theta} - \theta) \xrightarrow{D} N(0, I(\theta)^{-1})$, so $\hat{\theta}$ is asymptotically unbiased and asymptotically efficient.
- $\hat{\theta} \xrightarrow{(} P)\theta$, i.e., $\hat{\theta}$ is consistent.
- Invariance property: the MLE of $g(\theta)$ is $g(\hat{\theta})$.

1.4 Hypothesis Testing

X_i random sample from $f(x; \theta)$.

- Setup:
  - Null hypothesis $H_0\colon \theta \in \Theta_0$
  - Alternative hypothesis $H_1\colon \theta \in \Theta_1$
- Construct test statistic $t(\mathbf{X})$ such that large values of $t(\mathbf{X})$ cast doubt on H_0.

- Let $t_{\text{obs}} = t(\mathbf{x})$.
- The *p-value* or *significance level* is

$$p = P(t(\mathbf{X}) \geq t_{\text{obs}} \mid H_0).$$

- Small p-value = observed values unlikely under H_0.
- Can define *critical region* to be C such that we reject H_0 if and only if $\mathbf{x} \in C$.

1.4.1 Errors in hypothesis testing

Two types of error:

- Type I error: rejecting H_0 when H_0 is true
- Type II error: not rejecting H_0 when H_0 is false.

The *size* of a test is defined by

$$\begin{aligned} \alpha &= P(\text{type I error}) \\ &= P(\text{reject } H_0 \mid H_0 \text{ true}). \end{aligned}$$

The *power* of a test is $1 - \beta$, where

$$\begin{aligned} \beta &= P(\text{type II error}) \\ &= P(\text{don't reject } H_0 \mid H_1 \text{ true}). \end{aligned}$$

We want **small size** and **high power**.

For composite H_i, define the size

$$\alpha = \sup \theta \in \Theta_0 P(\text{reject } H_0 \mid \theta)$$

and the power function

$$w(\theta) = P(\text{reject } H_0 \mid \theta).$$

We want $w \approx 0$ for $\theta \in \Theta_0$ and $w \approx 1$ for $\theta \in \Theta_1$.

Student's t-test

Consider statistic

$$t = \frac{\overline{x} - \mu_0}{s/\sqrt{n}},$$

which has t-distribution with $n - 1$ degrees of freedom. Recall that a t-distribution is given by

$$T = \frac{Z}{\sqrt{V/\nu}},$$

where Z is standard normal and V is χ^2 with ν degrees of freedom.

Noncentral t-distribution

- Noncentral distributions describe how a test statistic is distributed when the null hypothesis is false.
- Noncentral t-distribution is given by
$$T = \frac{Z + \mu}{\sqrt{V/\nu}},$$
where μ is the noncentrality parameter.
- Under the alternative hypothesis, we get noncentral t-distribution with $n-1$ degrees of freedom and noncentrality parameter
$$\delta = \frac{\mu - \mu_0}{\sigma/\sqrt{n}}.$$
This allows us to set the power as
$$\text{Power} = P(\text{reject } H_0 \mid H_1).$$
Note that as n increases, power increases.

1.5 Bayesian Inference

- Unknown parameters are *random variables.*
- Probability model $f(\mathbf{x} \mid \theta)$ that is *conditional on* the value of θ.
- Prior density $\pi(\theta)$.
- After using observed data $\mathbf{x}$, get posterior density $\pi(\theta \mid \mathbf{x})$.
- We have
$$\begin{gathered} \pi(\theta \mid \mathbf{x}) \propto f(\mathbf{x} \mid \theta) \times \pi(\theta) \\ \text{posterior} \propto \text{likelihood} \times \text{prior} \end{gathered}$$

1.5.1 Prediction

- X_{n+1} future observation and $\mathbf{x} = (x_1, \ldots, x_n)$ observed data.
- Assume, conditional on θ, that X_{n+1} has density $f(x_{n+1} \mid \theta)$ independent of $X_i, \ldots, X_n$.
- The density of X_{n+1} given $\mathbf{x}$, called the *posterior predictive density*, is a conditional density denoted $f(x_{n+1} \mid \mathbf{x})$.
- Then
$$\begin{aligned} f(x_{n+1} \mid \mathbf{x}) &= \int f(x_{n+1}, \theta \mid \mathbf{x})\, d\theta \\ &= \int f(x_{n+1} \mid \theta, \mathbf{x}) \pi(\theta \mid \mathbf{x})\, d\theta \\ &= \int f(x_{n+1} \mid \theta) \pi(\theta \mid \mathbf{x})\, d\theta \end{aligned}$$

1.5.2 Credible interval

- Bayesian alternative to confidence interval.
- A $100(1-\alpha)\%$ credible set for θ is $jc \subset \Theta$ such that

$$P(\theta \in C \mid \mathbf{x}) = \int_C \pi(\theta \mid \mathbf{x})\, d\theta = 1 - \alpha.$$

1.5.3 Hypothesis testing and Bayes factors

Setup

- Prior probabilities $P(H_i)$ such that $P(H_0) + P(H_1) = 1$.
- A prior for θ_i under H_i, denoted $\pi(\theta_i \mid H_i)$.
- A model for data $\mathbf{x}$ under H_i, denoted $f(\mathbf{x} \mid \theta_i, H_i)$.

Bayes factor

- Prior odds of H_0 relative to H_1 is

$$\frac{P(H_0)}{P(H_1)}.$$

- Posterior odds of H_0 relative to H_1 is

$$\frac{P(H_0 \mid \mathbf{x})}{P(H_1 \mid \mathbf{x})}.$$

- Using Bayes' Theorem, get

$$\frac{P(H_0 \mid \mathbf{x})}{P(H_1 \mid \mathbf{x})} = \frac{P(\mathbf{x} \mid H_0)}{P(\mathbf{x} \mid H_1)} \times \frac{P(H_0)}{P(H_1)}$$
$$\text{posterior odds} = \text{Bayes factor} \times \text{prior odds}$$

  where the Bayes factor of H_0 relative to H_1 is

$$B_{01} = \frac{P(\mathbf{x} \mid H_0)}{P(\mathbf{x} \mid H_1)}.$$

- The quantity

$$P(\mathbf{x} \mid H_i) = \int_{\Theta_i} f(\mathbf{x} \mid \theta_i, H_i)\pi(\theta_i \mid H_i)\, d\theta$$

  is called the *marginal likelihood* for H_i.

Chapter 2

Linear Regression

- Observations
$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = x_i \cdot \beta + \epsilon_i,$$
i.e., $y = X\beta + \epsilon$.

- Loss function
$$L(\beta) = \mathrm{MSE}(\beta) = \frac{1}{n}\|y - X\beta\|^2.$$

- Want $\hat{\beta}$ that minimizes $L(\beta)$:
  - Find $\hat{\beta}$ such that
  $$\frac{\partial L}{\partial \beta} = 0.$$
  - Project y onto $\mathrm{Col}(X)$.

  Get $\hat{\beta} = (X^T X)^{-1} X^T y$. For simple linear regression, i.e. $y = \beta_0 + \beta_1 x + \epsilon$, we get
$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x},$$
$$\hat{\beta}_1 = \frac{\mathrm{cov}(x, y)}{\mathrm{Var}(x)} = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sum (x_i - \overline{x})^2}.$$

- Residual is $\hat{\epsilon} = y - X\hat{\beta}$. The residual/explained/total sum of squares is:
  - $\mathrm{RSS} = \|\hat{e}\|^2$, i.e., variation in the error between the observed data and modeled values.
  - $\mathrm{ESS} = \sum (\hat{y}_i - \overline{y}_i)$, i.e., how much variation there is in the modeled values.
  - $\mathrm{TSS} = \sum (y_i - \overline{y})$, i.e., how much variation there is in the observed data.
  - In linear regression, $\mathrm{TSS} = \mathrm{RSS} + \mathrm{ESS}$.

- Coefficient of determination is
$$R^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}}.$$
Note that the baseline model always predicts $\overline{y}$, so $R^2 = 0$. Furthermore, adding features weakly decreases RSS, hence weakly increases R^2. Instead, we could use adjusted R^2 :
$$\overline{R}^2 = 1 - \frac{\mathrm{RSS}/\mathrm{df}_{\mathrm{res}}}{\mathrm{TSS}/\mathrm{df}_{\mathrm{tot}}},$$
where df is degrees of freedom, so $\mathrm{df}_{\mathrm{res}} = n - p - 1$ and $\mathrm{df}_{\mathrm{tot}} = n - 1$.

- Assumptions:
  - **Weak exogeneity**: Predictor variables x can be treated as fixed values, and not random variables, i.e., x is error-free, i.e., $E(x\epsilon) = 0$. So no confounding variables.
  - **Linearity**: Mean of y is linear combination of parameters and x.
  - **Independence**: Observations are independent of each other.
  - **Homoscedasticity**: Constant variance of ϵ.
  - **Normality:** ϵ follow a normal distribution.
  - **No multicollinearity**: The independent variables are not highly correlated with each other.

Chapter 3

Natural Language Processing

- Solve sequence transduction, i.e., transforms an input sequence to an output sequence.
- Necessary to have some *memory*.

3.1 RNN

- At each time step, receive two inputs: word embedding of current word and hidden state.
- Let $x_t \in \mathbb{R}^n$ and $h_{t-1} \in \mathbb{R}^m$.
- Use weight matrix $W_x \in \mathbb{R}^{m \times n}$ and hidden-state-to-hidden-state matrix $W_h \in \mathbb{R}^{m \times m}$.
- Then

$$
\begin{aligned}
o_t &= W_{hh} h_{t-1} + W_{hx} x_t + b_h \\
h_t &= \tanh(o_t) = \tanh(W_h \cdot [h_{t-1}, x_t] + b_h) \\
y_t &= g(W_y h_t + b_y)
\end{aligned}
$$

  for some activation function g.
- We can show that

$$
\nabla_{W_{hh}}(h_t) = \sum_{t'=1}^{t-1} h_{t'} \left(W_{hh}^{t-t'-1} \tanh'(o_{t'+1}) \cdot \dots \cdot \tanh'(o_t) \right).
$$

  So the influence of $h_{t'}$ on h_t will be small if $t' \ll t$ as $\tanh(x)$ is small for $|x| > 2$, i.e., we have a *vanishing gradient problem*.

3.2 LSTM

- Introduce *cell state* to RNN.
- Information is added or removed to the cell state through *gates*.
- **Forget gate**:
  - Input: previous cell state C_{t-1} and x_t.

- Output: number in $[0, 1]$ for each entry in C_{t-1}, i.e., how much to forget.
- So we have

$$\begin{aligned} f_t &= \sigma(W_{hf}h_{t-1} + W_{xf}x_t + b_f) \\ &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \end{aligned}$$

where σ is the sigmoid function applied element-wise.

- **Input and update gate**:
  - Decide what and how much to add to the cell state.
  - What to add:
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C).$$
  - How much to add:
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i).$$
  - Update cell state:
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t,$$
where $\odot$ is element-wise multiplication.

- **Output gate**:
  - Decide what parts of cell state to output:
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o).$$
  - Actual output:
$$h_t = o_t \odot \tanh(C_t)$$

- Drawbacks: computational complexity, overfitting, dropout harder to implement, sensitive to different random weight initializations, inability to handle temporal dependencies that are longer than a few steps,

3.3 Seq2seq

- Input $\to$ Encoder $\to$ Context Vector $\to$ Decoder $\to$ Output
- Encoder and decoder are both RNNs or LSTMs.
- Last hidden state of encoder is context vector, which initializes decoder RNN.
- At each time step, decoder receives previous token (input of RNN) and uses previous hidden state. The output of RNN is fed though FNN to get embedding of word.
- Stop when EOS token is outputted.
- For back-propagation, *teacher forcing* is used, i.e., plug in correct words to decoder and stop at correct length.
- Drawbacks:
  - Bottleneck problem: for long input sequences, information would tend to be lost.
  - For the decoder, different information may be relevant at different steps.

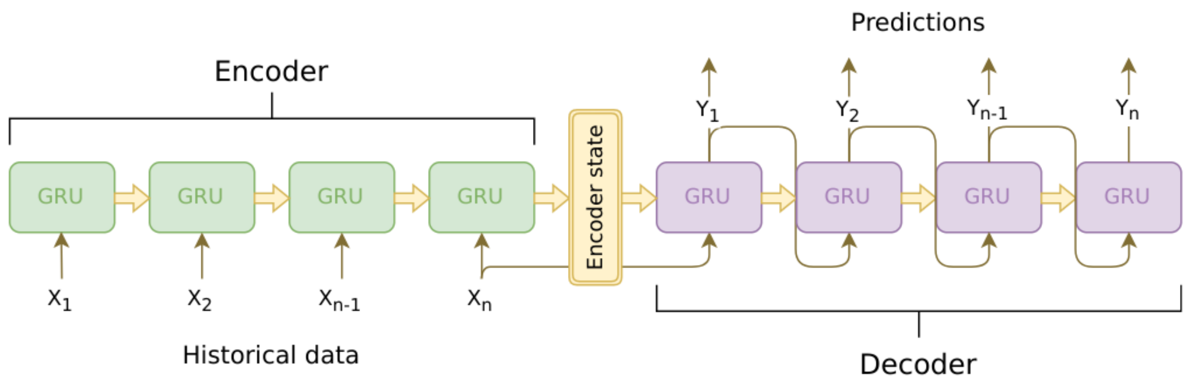

Image source: [5]

3.4 Seq2seq with Attention

- At each decoder step, it decides which source parts are more important
- Concretely, at each t,:
  (i) Use previous token in RNN to update hidden state h_t.
  (ii) Compute score(h_t, s_k) between decoder hidden state h_t and all encoder hidden states $s_1, \ldots, s_m$;
  (iii) Compute attention weights: softmax attention scores;
  (iv) Compute attention vector a_t: weighted sum of encoder states.
  (v) Use attention vector a_t and hidden state h_t in FNN to get output token.
- Popular score functions:
  - Dot-product: score$(h_t, s_k) = h_t^T s_k$.
  - Bilinear function: score$(h_t, s_k) = h_t^T W s_k$.
  - Multi-layer perceptron: score$(h_t, s_k) = v^T \tanh(W[h_t, s_k])$.
- Drawback: RNN is difficult to parallelize.

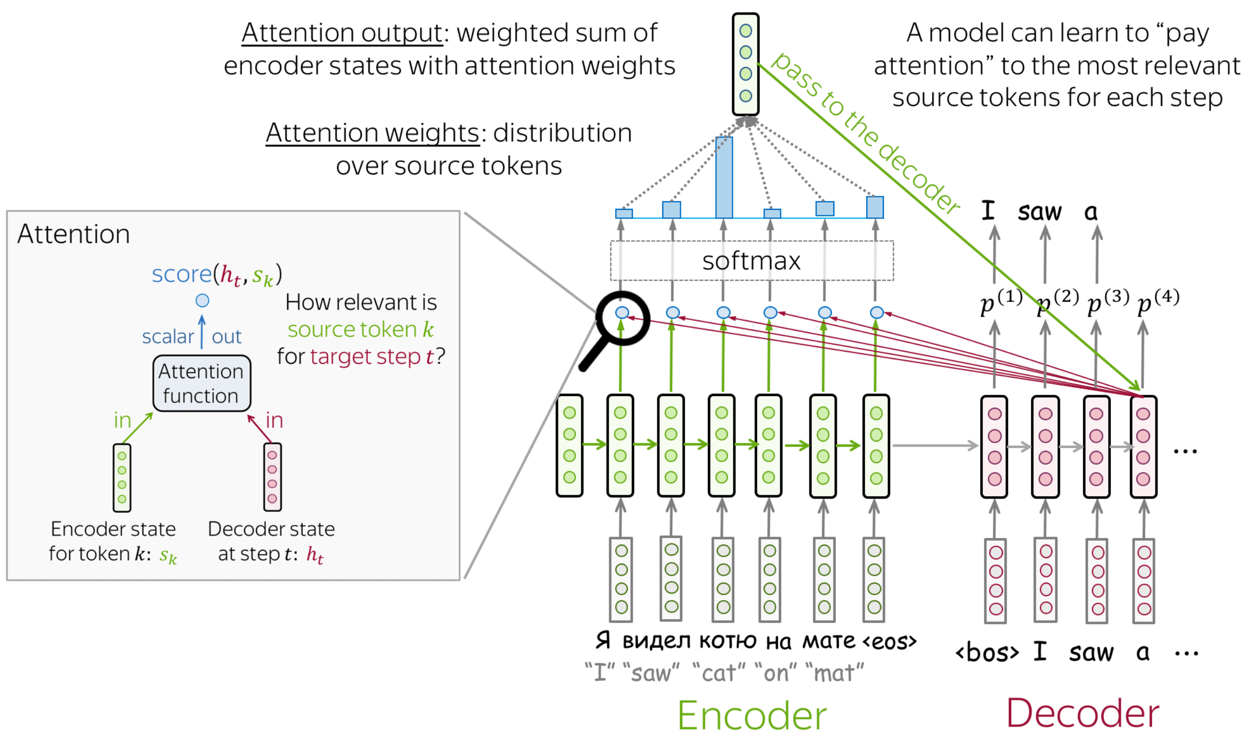

Image source: [15]

Chapter 4

Transformers

- Encoding component is a stack (six in original paper) of encoders with same structure but different weights. Decoding component is a stack of decoders of the same number.
- Encoders: Self-Attention $\to$ FNN
  - First encoder receives embedding of words.
  - Other encoders receive output of encoder directly below.
- Decoders: Self-Attention $\to$ Encoder-Decoder Attention $\to$ FNN

4.1 Ingredients

4.1.1 Self-attention

Process

1. Create three vectors from each of the encoder's input vectors: Query, Key, and Value vectors.
2. Note: in original paper, input vectors are in $\mathbb{R}^{512}$, and QKV vectors are in $\mathbb{R}^{64}$.
3. Self-attention score between i-th and j-th word is $q_i \cdot k_j$ for all j.
4. Divide scores by square root of dimension (8; this gives more stable gradients) and softmax.
5. Multiply each value vector v_j by softmax scores.
6. Sum up the weighted value vectors. This is output of self-attention layer for i-th word.

In matrix form, suppose our input is $X \in \mathbb{R}^{n \times d_{\text{model}}}$, where n is the sequence length and d_{model} is the embedding dimension. (So rows of X are the inputs.) The QKV matrices are given by $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{k}}}$. Then the QKV vectors

$$
\begin{aligned}
Q &= XW_Q \\
K &= XW_K \\
V &= XW_V
\end{aligned}
$$

are in $\mathbb{R}^{n \times d_{\text{k}}}$ For each attention head, we have

$$
\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \times V \in \mathbb{R}^{n \times d_{\text{k}}},
$$

i.e., aggregate values weighted by attention.

4.1.2 Multi-headed attention

Do h parallel attention heads, with different weight matrices for each head.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \times W_O,$$

where

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$$

is output of each attention head and $W_O \in \mathbb{R}^{hd_{\text{k}} \times d_{\text{model}}}$. Note that $\text{MultiHead}(Q, K, V) \in \mathbb{R}^{n \times d_{\text{model}}}$.

4.1.3 Positional encoding

Transformers are permutation-invariant, so we add positional encoding to input embeddings.

4.1.4 FNN

Output of attention fed into FNN:

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

4.1.5 Residual connections & layer normalization

Each sub-layer (attention or feedforward) is followed by

$$\text{LayerNorm}(x + \text{Sublayer}(x)).$$

Given $x \in \mathbb{R}^d$, the layer norm is

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta,$$

where

$$\mu = \frac{\sum x_i}{d} \text{ (mean over features)}$$
$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{d} \text{ (variance over features)}$$
$$\gamma, \beta\text{: learned scaling and shifting parameters}$$
$$\epsilon\text{: small constant to prevent division by zero.}$$

4.2 Original Architecture

Encoder Layer:

- Input Embedding + Positional Encoding
- Multi-Head Self-Attention $\rightarrow$ Residual + Layer Norm
- Feed-Forward Network $\rightarrow$ Residual + Layer Norm

Decoder Layer:

- Masked Multi-Head Self-Attention (prevent peeking at future tokens)
- Multi-Head Encoder-Decoder Attention → Residual + Layer Norm
- Feed-Forward Network → Residual + Layer Norm

4.3 Modern Architecture

- Input Embedding
- Layer Norm → Self-Attention → Residual
  - Grouped-Query Attention
  - Rotary Embeddings
- Layer Norm → FNN → Residual

4.3.1 Grouped-query attention

- Use same W_K and W_V across heads, and each head has its own W_Q.
- Better: divide heads into groups. Heads in the same group share W_K and W_V

4.3.2 Rotary positional embeddings (RoPE)

- Limitations of absolute positional embedding:
  - Limited sequence length
  - Independence of positional embedding, e.g./ difference between position 1 and 2 is the same as the difference between position 1 and 500
- Alternative: *relative positional embeddings* [13]
  - Bias for positional offsets: use a bias to represent each possible positional offset
  - Relative attention becomes:

    $$\text{Attention}(Q, K, V)_i = \sum_{j=1}^{n} \text{softmax}(e_{ij}) \times (x_j W_V + a_{ij}^V),$$

    where

    $$e_{ij} = \frac{QK^T + x_i W_Q (a_{ij}^K)^T}{\sqrt{d_k}}.$$

  - Here, $a_{ij} \in \mathbb{R}^{1 \times d_a}$ is a vector of relative positional weights, i.e.,

    $$a_{ij} = w_{\text{clip}(j-i,k)}$$
    $$\text{clip}(x, k) = \max(-k, \min(k, x)).$$

    Calculate w for both keys and values.
  - Clipping allows scalability (i.e., arbitrarily long sequences)

- Limitations: slower; complicates key-value cache usage as each additional token changes the embedding for every other token.

- For **RoPE** [14], the intuition is to rotate each embedding by $m\theta$, where m is the position of the word in the sequence.

- Benefits:
  - Scalability: adding new words does not change the embedding of previous words
  - The dot product of the embeddings of two words does not depend on absolute position.

- Mathematically, we first work in $\mathbb{C}^{d_k/2}$. Let M_j be the rotation matrix by $m\theta_j$. Then the output of RoPE for the m-th word is just

$$Q_m\Theta_m = Q_m \times \begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_{d_k/2} \end{pmatrix},$$

  where Q_m is the m-th row of Q (i.e., query vector for m-th word).

- Do the same for key vector.

4.4 Other Techniques

4.4.1 Sparse attention

Instead of global autoregressive self-attention, use local autoregressive self-attention in some layers.

4.4.2 Mixture of experts (MOE)

- Instead of a single monolithic feedforward layer in the transformer block, use a set of expert networks, and route the input to only a few of them.

- A router (a smaller FNN) calculates which experts to turn on.

- Could do model merging by doing a weighted average of experts using weights from the router.

4.5 Complexity

4.5.1 Number of parameters and FLOPs

Suppose input length is n and n_{layer} is the number of layers. Recall $X \in \mathbb{R}^{n\times d_{\text{model}}}$; $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}}\times d_{\text{k}}}$; and $Q, K, V \in \mathbb{R}^{n\times d_{\text{k}}}$. As in [9], we have the following steps in the transformer architecture:

1. Embed: token and positional embeddings.
   - Notation:
     - d_{model} is model dimension, i.e., dimension of the residual stream.
     - n_{vocab} is the number of tokens in the vocabulary.

- Parameters: $n_{\text{vocab}} \times d_{\text{model}}$ for token embedding, and $n \times d_{\text{model}}$ for positional embedding $\Rightarrow (n_{\text{vocab}} + n)d_{\text{model}}$ parameters.
- FLOPs: $O(d_{\text{model}})$ per token

2. Attention (QKV): project into QKV vectors.
   - Notation:
     - $d_{\text{k}} = d_{\text{v}}$ is the dimension of the projections.
     - n_{heads} is the number of attention heads.
     - $d_{\text{attn}} = d_k n_{\text{heads}}$ is the output of the multi-headed attention (usually $= d_{\text{model}}$).
   - Parameters: $3d_{\text{model}}d_{\text{k}}$ per head per layer $\Rightarrow 3n_{\text{layer}}d_{\text{model}}d_{\text{attn}}$ parameters.
   - FLOPs: we are doing three instances of $[n, d_{\text{model}}] \times [d_{\text{model}}, d_{\text{k}}]$, so get $3 \times 2 \times nd_{\text{model}}d_{\text{k}}$ per head per layer $\Rightarrow 6d_{\text{model}}d_{\text{attn}}$ per token per layer $\Rightarrow 6n_{\text{layer}}d_{\text{model}}d_{\text{attn}}$ per token.

3. Attention (Mask): dot-product query and key, i.e., QK^T.
   - FLOPs: doing $[n, d_{\text{k}}] \times [d_{\text{k}}, n]$, so get $2n^2 d_{\text{k}}$ per head per layer $\Rightarrow 2nn_{\text{layer}}d_{\text{attn}}$ per token.

4. Attention (Project): project output of attention heads to d_{model} by multiplying with $W_O \in \mathbb{R}^{d_{\text{attn}} \times d_{\text{model}}}$
   - Parameters: $d_{\text{attn}}d_{\text{model}}$ per layer $\Rightarrow n_{\text{layer}}d_{\text{attn}}d_{\text{model}}$ parameters.
   - FLOPs: doing $[n, d_{\text{attn}}] \times [d_{\text{attn}}, d_{\text{model}}]$, so get $2nd_{\text{attn}}d_{\text{model}}$ per layer $\Rightarrow 2n_{\text{layer}}d_{\text{attn}}d_{\text{model}}$ per token.

5. Feedforward: two linear layers in dense neural network.
   - Notation:
     - d_{ff} is output size of first linear layer (usually $d_{\text{ff}} = 4d_{\text{model}}$).
   - Parameters: first weight matrix gives $d_{\text{model}}d_{\text{ff}}$ and second gives $d_{\text{ff}}d_{\text{model}}$ per layer $\Rightarrow 2n_{\text{layer}}d_{\text{model}}d_{\text{ff}}$ parameters.
   - FLOPs: doing $[n, d_{\text{model}}] \times [d_{\text{model}}, d_{\text{ff}}] \times [d_{\text{ff}}, d_{\text{model}}]$ per layer, so get $4nd_{\text{model}}d_{\text{ff}}$ per layer $\Rightarrow 4n_{\text{layer}}d_{\text{model}}d_{\text{ff}}$ per token.

6. De-embed: go back to vocabulary.
   - FLOPs: doing $[n, d_{\text{model}}] \times [d_{\text{model}}, d_{\text{vocab}}]$, so get $2nd_{\text{model}}d_{\text{vocab}} \Rightarrow 2d_{\text{model}}d_{\text{vocab}}$ per token.

Note that we are doing FLOPs *per token*. So the total number of parameters is

$$2n_{\text{layer}}d_{\text{model}}(2d_{\text{attn}} + d_{\text{ff}}) + d_{\text{model}}(n_{\text{vocab}} + n) \approx 2n_{\text{layer}}d_{\text{model}}(2d_{\text{attn}} + d_{\text{ff}}) =: N$$

and the total FLOPs per token for the forward pass becomes

$$2N + 2d_{\text{model}}d_{\text{vocab}} + 2nn_{\text{layer}}d_{\text{attn}} \approx 2N + 2nn_{\text{layer}}d_{\text{attn}} =: C_{\text{forward}}.$$

The FLOPs for the backwards pass is about twice the FLOPs for the forward pass, so the total number of FLOPs per token is

$$C \approx 6N.$$

Then the total number of FLOPs is

$$6ND,$$

where D is the number of tokens in the corpus.

If we let τ be the throughput, i.e., inputs processed per unit time (in FLOPs per second) and T be the total training time, then we get

$$\tau T \approx 6ND$$

4.5.2 Time complexity of self-attention

- Calculating Q, K, V are all $O(nd_{\text{model}}d_{\text{k}})$.
- Calculating QK^T is $O(n^2 d_{\text{k}})$.
- Doing softmax is $O(n^2)$.
- Multiplying result with V is $O(n^2 d_{\text{k}})$.

Overall, the time complexity of the self-attention mechanism is $O(nd_{\text{model}}d_{\text{k}} + n^2 d_{\text{k}})$. If $n \gg d_{\text{model}}, d_{\text{k}}$, this reduces to just $O(n^2)$.

Remark. In [10], the authors show that the time complexity of self-attention is necessarily quadratic in the input length, unless the Strong Exponential Time Hypothesis (SETH) is false.

Chapter 5

Fine-Tuning Techniques

Full fine-tuning, of course, has the best performance but is computationally expensive. Need parameter-efficient fine-tuning (PEFT).

- Adapter-based methods
  - Adapters: small, trainable models inserted into pre-trained transformers.
  - Freeze original model weights.
- Instruction tuning

5.1 Low-Rank Adaptation (LoRA)

Introduced in [8].

- Intuition: only train low rank perturbations of the (selected) weight matrices.
- Let $W_0 \in \mathbb{R}^{d \times k}$ be a pre-trained weight matrix.
- Update: $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times d}, r \ll \min(d, k)$.
- Advantages:
  - Roughly converges to full fine-tuning as r increases.
  - No additional inference latency: when needing to switch to another downstream task, can recover W_0 by subtracting BA, and then we can add a new $B'A'$.
- Notes [6]:
  - Optimal placement highly dependent on the dataset and model architecture.
  - For transformers, applying LoRA exclusively to attention layers provides the most stability and mitigates the risk of divergence.
  - For MoE, applying LoRA to each expert individually boosts performance, but significantly increases memory usage. Applying to router gives limited success.
  - Could optimize memory by using same B across different A.

5.2 QLoRA

Quantized LoRA, introduced in [4].

5.2.1 4-bit NormalFloat Quantization

- Better quantization data type for normally distributed data.
- In general, for k-bit NormalFloat, equally divide the normal distribution into 2^k quantiles.
- Note: to ensure 0 has an exact representation, actually divide the negative part into 2^{k-1} quantiles and the positive part into $2^{k-1}+1$ quantiles, then remove one of the two overlapping zeros.
- Then
$$X^{\text{NF4}} = \text{round}\left(\frac{1}{\text{absmax}(X^{\text{FP32}})} X^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} X^{\text{FP32}}),$$
where c^{FP32} is the *quantization constant*.

5.2.2 Block-wise Quantization

- One problem is quantization is that outliers severely impacts the scaling, and the full range of the lower precision data type may not be effectively used.
- Solution: Chunk the input tensor into blocks that are independently quantized, each with its own quantization constant.

5.2.3 Double Quantization

- Helps reduce the memory footprint of quantization constants.
- Block-wise k-bit quantization for the quantization constants c^{FP32}.
- Gives $c_2^{k\text{-bit}}$, with second level of quantization constants c_1^{FP32}.

5.2.4 Implementation

- Recall: LoRA has
$$Y = XW + XL_1L_2$$
for low rank matrices L_i.
- For QLoRA, do
$$Y^{\text{BF16}} = X^{\text{BF16}} \text{doubleDequant}(c_1^{\text{FP32}}, c_2^{k\text{-bit}}, W^{\text{NF4}}) + X^{\text{BF16}} L_1^{\text{BF16}} L_2^{\text{BF16}},$$
where
$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{k\text{-bit}}, W^{\text{NF4}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{k\text{-bit}}), W^{\text{NF4}}) = W^{\text{BF16}}$$
- Original paper [4] uses FP8 for c_2, block size of 64 for W, and block size of 256 for c_2.
- Computations done in BF16 and storage of W in NF4.

5.2.5 $(\mathbf{IA})^3$

- Introduced in [12], Infused Adapter by Inhibiting and Amplifying Inner Activations is intended to improve over LoRA.
- Notation: if $l \in \mathbb{R}^d$ and $A \in \mathbb{R}^{m \times d}$, then $(l \odot A)_{ij} = l_j A_{ij}$, i.e., element-wise multiplication of l and rows of A.
- Learned vectors $l_{\mathrm{v}}, l_{\mathrm{k}}, l_{\mathrm{ff}}$.
- Attention layer:
$$\mathrm{softmax}\left(\frac{Q(l_{\mathrm{k}} \odot K^T)}{\sqrt{d_k}}\right) \times (l_{\mathrm{v}} \odot V)$$
- Feedforward layer:
$$(l_{\mathrm{ff}} \odot \gamma(W_1 x))W_2,$$
where γ is the FFN nonlinearity.
- Advantages:
  - Enables mixed tasks.
  - For single-task models, no additional computational cost compared to original, because one can simply replace W with $l \odot W$.

Chapter 6

Fourier Transform

- Let $x(t)$ be a complex-valued function. Then its Fourier transform is

  $$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}\, dt.$$

  The inverse Fourier transform is

  $$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}\, d\omega.$$

- For a sequence of discrete time signals, use discrete-time Fourier transform:

  $$X(\omega) = \sum_{-\infty}^{\infty} x[n]e^{-i\omega n}.$$

  The output is continuous in ω and periodic. The inverse DTFT is:

  $$x[n] = \frac{1}{2\pi}\int_{2\pi} X(\omega)e^{i\omega n}\, d\omega.$$

- Discrete Fourier transform converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the DTFT. It is given by

  $$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n]e^{-2\pi ikn/N} \\ &= \sum_{n=0}^{N-1} x[n]W_N^{kn}, \end{aligned}$$

  where $W_N = e^{-2\pi i/N}$. The inverse transform is given by

  $$\frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn}.$$

- The fast Fourier transform is based on the idea that the N-th roots of unity W_N^k have nice properties when N is a power of 2. In vector form,

  $$\mathbf{X}[k] = \mathbf{W_N}\mathbf{x}[n],$$

  where $W_N = V(W_N^0, \ldots, W_N^{N-1})$ is a Vandermonde matrix. The computational complexity is $O(N \log N)$ vs. $O(N^2)$ of definition of DFT.

- Limitation of Fourier transform is that it lacks temporal resolution. A way to overcome this is the short-time Fourier transform.
  - Continuous case:
  $$\text{STFT}\{x(t)\}(\tau, \omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-i\omega t}\, dt,$$
  where w is the window function, usually Hann or Gaussian window.
  - Discrete case:
  $$\text{STFT}\{x[n]\}(k, \omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-k]e^{-i\omega n}.$$

  Uncertainty principle: there is a trade-off between temporal and frequency resolution.

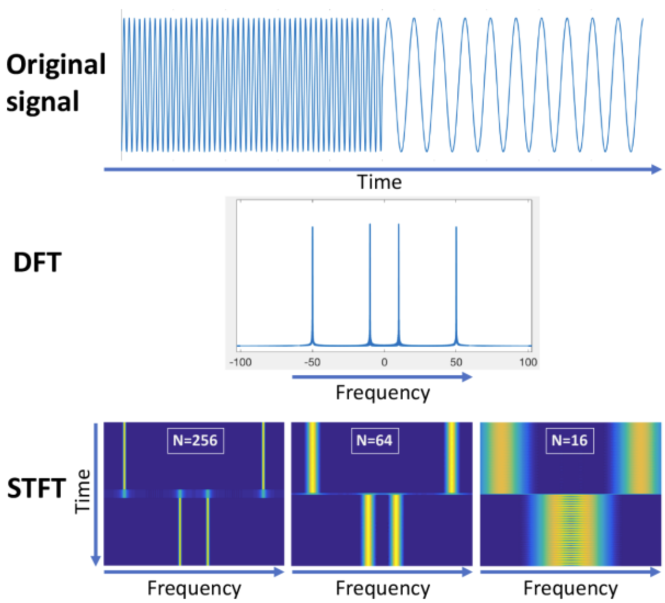

Image source: [7]

- Spectrograms:
  - Divide a time-domain signal into segments of equal length.
  - Apply FFT to each segment, transforming the data from the time domain to the frequency domain.
  - Each segment corresponds to vertical line in spectogram.
  - For window width w, spectogram$(t) = |\,\text{STFT}(t)|^2$.
- Nyquist–Shannon sampling theorem
  - Sampling rate must be at least twice the bandwidth of the signal to avoid aliasing.
  - Let $x(t)$ have Fourier transform $X(f)$. Suppose $X_{1/T}(f)$ is the DTFT of sample sequence $x[n]$.
  - For DTFT, copies of X_f are shifted by multiples of the sampling rate f_s and added.

- If Nyquist–Shannon is not satisfied, copies will overlap.
- Any frequency component above $f_s/2$ is indistinguishable from a lower-frequency component (i.e., alias).

Bibliography

[1] Jay Alammar. Online Blog. https://jalammar.github.io/.

[2] Jay Alammar and Maarten Grootendorst. How Transformer LLMs Work. Online Course. https://www.deeplearning.ai/short-courses/how-transformer-llms-work.

[3] Aziz Belaweid. Complete Summary of Absolute, Relative and Rotary Position Embeddings! Online Blog. https://azizbelaweid.substack.com/p/complete-summary-of-absolute-relative.

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[5] Joseph Eddy. Forecasting with Neural Networks - An Introduction to Sequence-to-Sequence Modeling Of Time Series. https://jeddy92.github.io/ts_seq2seq_intro/.

[6] Vlad Fomenko, Han Yu, Jongho Lee, Stanley Hsieh, and Weizhu Chen. A note on lora, 2024.

[7] Diego Hernando. Lecture 16: Limitations of the Fourier Transform: STFT. Mathematical Methods in Medical Physics Lecture Notes, April 2016. https://qiml.radiology.wisc.edu/wp-content/uploads/sites/760/2022/12/notes_016_STFT.pdf.

[8] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021.

[9] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[10] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention, 2022.

[11] Neil Laws. Part A Statistics. Lecture Notes, April 2016.

[12] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022.

[13] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018.

[14] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021.

[15] Lena Voita. Sequence to Sequence (seq2seq) and Attention. NLP Course. https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html.