EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Homework 11

This homework is due on Wednesday, November 18, 2026, at 10:59PM.

1. Kernelized Linear Attention (Part II)

This is the second part of the "Kernelized Linear Attention" problem from HW10 (Problem 5). Please refer to Part I for notation and context.

In Part I of this problem, we considered ways to efficiently express the attention operation when sequences are long (e.g. a long document). Attention uses the following equation:

$$V'_{i\cdot} = \frac{\sum_{j=1}^{N} \text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right) V_{j\cdot}}{\sum_{j=1}^{N} \text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right)}. \tag{1}$$

We saw that when the similarity function is a kernel function (i.e. if we can write $\text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right) = \Phi(Q_{i\cdot})\Phi(K_{j\cdot})^T$ for some function Φ), then we can use the associative property of matrix multiplication to simplify the formula to

$$V'_{i\cdot} = \frac{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{N} \phi\left(K_{j\cdot}\right)^T V_{j\cdot}}{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{N} \phi\left(K_{j\cdot}\right)^T}. \tag{2}$$

If we use a polynomial kernel with degree 2, this gives a computational cost of $\mathcal{O}\left(ND^2M\right)$, which for very large N is favorable to the softmax attention computational cost of $\mathcal{O}\left(N^2 \max\left(D, M\right)\right)$. ($N$ is the sequence length, D is the feature dimension of the queries and keys, and M is the feature dimension of the values.) Now, we will see whether we can use kernel attention to directly approximate the softmax attention:

$$V'_{i\cdot} = \frac{\sum_{j=1}^{N} \exp(\frac{Q_{i\cdot}K_{j\cdot}^T}{\sqrt{D}})V_{j\cdot}}{\sum_{j=1}^{N} \exp(\frac{Q_{i\cdot}K_{j\cdot}^T}{\sqrt{D}})}. \tag{3}$$

(a) Approximating softmax attention with linearized kernel attention

i. As a first step, we can use Gaussian Kernel $\mathcal{K}_{\text{Gauss}}(q,k) = \exp(\frac{-||q-k||_2^2}{2\sigma^2})$ to rewrite the softmax similarity function, where $\text{sim}_{\text{softmax}}(q,k) = \exp(\frac{q^Tk}{\sqrt{D}})$. Let $\sigma^2 = \sqrt{D}$. **Rewrite the softmax similarity function using the Gaussian kernel.** (*Hint: You can write* $\exp(\frac{q^Tk}{\sqrt{D}})$ *as the product of the Gaussian kernel and two other terms.*)

Solution:

$$\text{sim}_{\text{softmax}}(q,k) = \exp(\frac{q^Tk}{\sqrt{D}}) = \exp(\frac{||q||_2^2}{2\sigma^2}) * \exp(\frac{-||q-k||_2^2}{2\sigma^2}) * \exp(\frac{||k||_2^2}{2\sigma^2})$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

$$= \exp(\frac{||q||_2^2}{2\sigma^2}) * \mathcal{K}_{\text{Gauss}}(q,k) * \exp(\frac{||k||_2^2}{2\sigma^2})$$

ii. However, writing softmax attention using a Gaussian kernel does not directly enjoy the benefits of the reduced complexity using the feature map. This is because the feature map of the Gaussian kernel usually comes from the Taylor expansion of $\exp(\cdot)$, which has infinitely many terms, so computing it is still expensive [1]. However, we can approximate the Gaussian kernel using a random feature map and then reduce the computation cost, using a trick from Random feature attention (2021)[2]. (Rahimi and Recht, 2007)[3] proposed random Fourier features to approximate a desired shift-invariant kernel. The method nonlinearly transforms a pair of vectors q and k using a **random feature map** $\phi_{\text{random}}()$; the inner product between $\phi(q)$ and $\phi(k)$ approximates the kernel evaluation on q and k. More precisely:

$$\phi_{\text{random}}(q) = \sqrt{\frac{1}{D_{\text{random}}}}\left[\sin\left(\mathbf{w}_1^T q\right), \ldots, \sin\left(\mathbf{w}_{D_{\text{random}}}^T q\right), \cos\left(\mathbf{w}_1^T q\right), \ldots, \cos\left(\mathbf{w}_{D_{\text{random}}}^T q\right)\right]^\top. \tag{4}$$

Here $\mathbf{w}_1, \ldots, \mathbf{w}_{D_{\text{random}}} \in \mathbb{R}^D$ are random vectors, sampled independently from $\mathcal{N}(\mathbf{0}, \sigma^{-2}\mathbf{I}_D)$. Then

$$\mathbb{E}_{\mathbf{w}_1,\ldots,\mathbf{w}_{D_{\text{random}}}}\left[\phi_{\text{random}}(q)^T \phi_{\text{random}}(k)\right] = \exp\left(-\|q-k\|^2/2\sigma^2\right). \tag{5}$$

Use ϕ_{random} to approximate the above softmax similarity function with Gaussian Kernel and derive the computation cost for computing all the V' here.

Solution:

$$\begin{aligned}\text{sim}_{\text{softmax}}(Q,K) &= \exp\left(\frac{||Q||_2^2}{2\sigma^2}\right) * \mathcal{K}_{\text{Gauss}}(Q,K) * \exp\left(\frac{||K||_2^2}{2\sigma^2}\right) \\ &\approx \exp\left(\frac{||Q||_2^2}{2\sigma^2}\right) * \phi_{\text{random}}(Q)^T \phi_{\text{random}}(K) * \exp\left(\frac{||K||_2^2}{2\sigma^2}\right)\end{aligned}$$

The computation cost of ϕ_{random} for a single vector is $O(D_{random}D)$ where D is for computing each coordinate $\sin\left(w_i^T q\right)$ or $\cos\left(w_i^T q\right)$, and there are $2D_{random}$ of coordinates.
Cost of computing the entire matrix $\phi_{random}(Q)$ is $O(D_{random}DN)$.
Cost of computing the entire matrix $\phi_{random}(K)$ is $O(D_{random}DN)$.
Cost of computing $\exp\left(\frac{||k||_2^2}{2\sigma^2}\right)$ is $O(D)$, due to the cost of taking the L2 norm. The cost of the whole $\exp\left(\frac{||K||_2^2}{2\sigma^2}\right)$ is $O(ND)$. It is a vector with N entries. As always, we only consider the **incremental cost** of computing – given that everything in previous paragraphs had been computed, how much more does it cost to compute everything in this paragraph?
Cost of $\exp\left(\frac{||Q||_2^2}{2\sigma^2}\right)$ is $O(ND)$. It is a vector with N entries.
Now, define the matrix $A := \exp\left(\frac{||Q||_2^2}{2\sigma^2}\right)\phi_{\text{random}}(Q)^T$ and the matrix $B := \phi_{\text{random}}(K)\exp\left(\frac{||K||_2^2}{2\sigma^2}\right)$.
They have shapes $N \times 2D_{random}$ and $2D_{random} \times N$ respectively. Cost of computing each is

[1] https://www.csie.ntu.edu.tw/~cjlin/talks/kuleuven_svm.pdf
[2] Peng, Hao, et al. "Random feature attention." arXiv preprint arXiv:2103.02143 (2021).
[3] Rahimi, Ali, and Benjamin Recht. "Random features for large-scale kernel machines." Advances in neural information processing systems 20 (2007).

 All Rights Reserved. This may not be publicly shared without explicit permission.

$O(D_{random}N)$.
Let B' be the $2D_{random} \times 1$ matrix obtained by summing the columns of B. Cost of computing is $O(D_{random}N)$.
Now it remains to do three matrix multiplications and one division to get V'. The ordering of the matrix multiplications is crucial.
Multiply BV. This costs $O(D_{random}NM)$.
Multiply $A(BV)$. This costs $O(ND_{random}M)$.
Multiply AB'. This costs $O(ND_{random})$.
Divide row i of $A(BV)$ by entry i of AB'. This costs $O(NM)$.
The total cost is found by summing all of them:

$$\mathcal{O}\left(N(M+D)D_{\text{random}}\right)$$

(b) (Optional) In autoregressive models such as GPT, attention is masked so that the i-th position can only be influenced by a position j if $j \leq i$. A position cannot be influenced by later positions. This type of attention masking is called *causal masking*.

Derive how this causal masking changes equation 1 and 2. **Write equation 1 and 2 in terms of $S_{i\cdot}$ and $Z_{i\cdot}$**, which are defined as:

$$S_{i\cdot} = \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T V_{j\cdot}, \;\; Z_{i\cdot} = \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T, \tag{6}$$

to simplify the causal masking kernel attention and derive the computational complexity of this new causal masking formulation scheme.

Solution: The causal masking changes equation 1 as follows,

$$V'_{i\cdot} = \frac{\sum_{j=1}^{i} \text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right) V_{j\cdot}}{\sum_{j=1}^{i} \text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right)}. \tag{7}$$

We linearize the masked attention as described below,

$$V'_{i\cdot} = \frac{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T V_{j\cdot}}{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T}. \tag{8}$$

we can simplify equation 8 to

$$V'_{i\cdot} = \frac{\phi\left(Q_{i\cdot}\right) S_{i\cdot}}{\phi\left(Q_{i\cdot}\right) Z_{i\cdot}}. \tag{9}$$

Note that $S_{i\cdot} = S_{(i-1)\cdot} + \phi\left(K_{i\cdot}\right)^T V_{i\cdot}$ and $Z_{i\cdot} = Z_{(i-1)\cdot} + \phi\left(K_{i\cdot}\right)^T$. If ϕ maps to $\mathbb{R}^C$, each update costs $\mathcal{O}\left(CM\right)$, which does not depend on N, and computing $V'_{i\cdot}$ from $S_{i\cdot}$ and $Z_{i\cdot}$ also costs $\mathcal{O}\left(CM\right)$. So the total cost over all N positions is $\mathcal{O}\left(NCM\right)$, the same as without the mask and linear in the sequence length. For the degree-2 polynomial kernel ($C = \mathcal{O}\left(D^2\right)$) this is $\mathcal{O}\left(ND^2M\right)$.

2. Coding Question: Hand-Design Transformers

Please follow the instructions in this notebook. You will implement a simple transformer model (with a

 All Rights Reserved. This may not be publicly shared without explicit permission.

single attention head) from scratch and then create hand-designed attention heads of the transformer model capable of solving a basic problem.

Once you have finished the notebook, answer the following questions in your submission of the written assignment:

(a) Design a transformer that selects by contents. Compare the variables of your hand-designed Transformer with those of the learned Transformer. **Identify the similarities and differences between the two sets of variables and provide a brief explanation for each difference.**

Solution: Answers may vary, but weights, keys, and queries are likely more evenly distributed than those the student implemented. However, Vm, attention scores and values should be roughly the same.

Explanation: this occurs since the network is simple enough that there is only one correct attention pattern (only attend to matching positions) and only one correct thing to do with attended positions (use them to copy the original content into the output). In more complicated problems it's rare to be able to predict precisely what features the transformer learns.

(b) Design a transformer that selects by positions. Compare the variables of your hand-designed Transformer with those of the learned Transformer. **Identify the similarities and differences between the two sets of variables and provide a brief explanation for each difference.**

Solution: Answers may vary, but weights, keys, and queries are likely more evenly distributed than those the student implemented. However, attention scores and outputs should be roughly the same, since there is only one correct attention pattern (only attend to the first position) and only one correct thing to do with attended positions (use them to copy the first element into the output).

(c) (Optional) Design a transformer that selects by both content and position (the optional last section of the notebook, which outputs a positive number for each unique token and a negative number for each repeated token). **Comment on the similarities and differences between the weights and intermediate outputs of the learned and hand-coded models.**

Solution: Answers may vary, but all transformer outputs may look substantially different between the two models except the sign of the final output should be the same. This shows us that transformers can solve tasks in ways which are unintuitive to humans.

3. Scaling Laws of Batch Size

A common question in neural network training is, how should I select my hyperparameters? While a proper hyperparameter sweep will always provide the best answer, sweeps can become impractical especially at larger network sizes.

In this homework question, we will derive a simple scaling law for the optimal learning rate under varying batch sizes. Complete this notebook, and answer the following questions in your submission:

(a) **For linear regression using SGD, attach curves of learning rate vs MSE loss for all the considered batch sizes, and curve showing relationship between batch size and optimal learning rate. What function does this resemble?**

Solution: The numbers below come from the reference solution notebook. Students' numbers depend on their sweep and on how they pick the optimum, so grade the trend, not the exact values.

Each loss curve is U-shaped. With a learning rate (LR) that is too small, 100 steps are not enough to converge; with an LR that is too large, the SGD noise is large or the iteration diverges. Larger batches reach a lower loss (close to the noise floor $\sigma^2 = 25$), tolerate larger LRs before diverging, and have wider, flatter minima.

 All Rights Reserved. This may not be publicly shared without explicit permission.

For batch sizes $2, 4, 16, 64, 256, 1024, 2048$, the optimal LRs are about $0.012, 0.012, 0.018, 0.018, 0.023, 0.035, 0.028$. The optimal LR increases with the batch size, quickly for small batches, and then levels off. It resembles a logarithm, $\eta^* \approx a + b \ln B$ (a fit gives $b \approx 0.003$). Accept any increasing curve with diminishing returns (logarithmic, square-root-like, or saturating). At large batch sizes the minimum is flat, so the measured optimum is noisy there.

Why a logarithm (not required): with a fixed budget of $T = 100$ steps, the error left from the initialization shrinks like $e^{-4\eta T}$, while the SGD noise adds an error proportional to η/B. Minimizing $C_1 e^{-4\eta T} + C_2 \eta / B$ over η gives $\eta^* = \frac{1}{4T} \ln B + \text{const}$, a slope of $1/400 = 0.0025$, close to the fitted 0.003.

(b) **Attach the same curves but for an MLP instead of linear regression. How do the scaling laws differ?**

Solution: The trend is the same: the optimal LR increases with the batch size, much more slowly than linearly. In the reference run the optimal LR is about 0.05 for $B \leq 16$ and then about $0.08, 0.11, 0.14, 0.17$ for $B = 64, 256, 1024, 2048$, so it grows about $3\times$ while the batch size grows about $1000\times$. Compared with linear regression, the optimal LRs are larger, they barely change for small batches, and they keep growing up to $B = 2048$ instead of leveling off. The loss levels off near $\sigma^2 + 0.57 \approx 4.6$ instead of σ^2. Accept any answer that describes the student's plots correctly.

(c) **Finally, show the same curves when using the Adam optimizer instead of SGD. Does the scaling for learning rate change when using Adam?**

Solution: Yes, partly. With Adam the loss is much less sensitive to the LR: for $B \geq 64$, LRs from about 0.03 to about 2 give nearly the same loss, so the optimum is poorly determined (the notebook averages the LRs whose loss is within 0.05 of the best one). The optimal LR still increases with the batch size. In the reference run it is about $0.18, 0.13, 0.11, 0.28, 0.57, 0.65, 0.77$ for $B = 2, 4, 16, 64, 256, 1024, 2048$, which is several times larger than with SGD and grows faster (about $7\times$ from $B = 16$ to $B = 2048$, versus about $3\times$ for SGD), but the curve is noisy. With small batches ($B \leq 4$) the loss is noisy for large LRs. Accept answers that note that the optimal LR still grows with the batch size and that Adam tolerates a much wider range of LRs.

4. Read a Research Paper: FaceNet

In class, you have learnt how self-supervised learning can be used to learn useful representations from large datasets without labels (e.g., learning features from ImageNet). While these features may inherently pick out some notion of "similarity" between different images in the dataset, they are not incentivized to *cluster* different data points based on any interesting similarity measure.

The paper "FaceNet: A Unified Embedding for Face Recognition and Clustering" explores how we can view the task of face recognition through the lens of self-supervised (or to be more accurate, *slightly* supervised) learning.

You may find this blog post easier to follow than the paper. Its two figures, shown below, are particularly helpful.

Read the paper and answer the questions below.

(a) **What are the two neural network architectures considered by the authors?**

Solution: The first architecture is based on the Zeiler & Fergus model which consists of multiple interleaved layers of convolutions, non-linear activations, local response normalizations, and max pooling layers. The second architecture is based on the Inception model of Szegedy et al.

 All Rights Reserved. This may not be publicly shared without explicit permission.

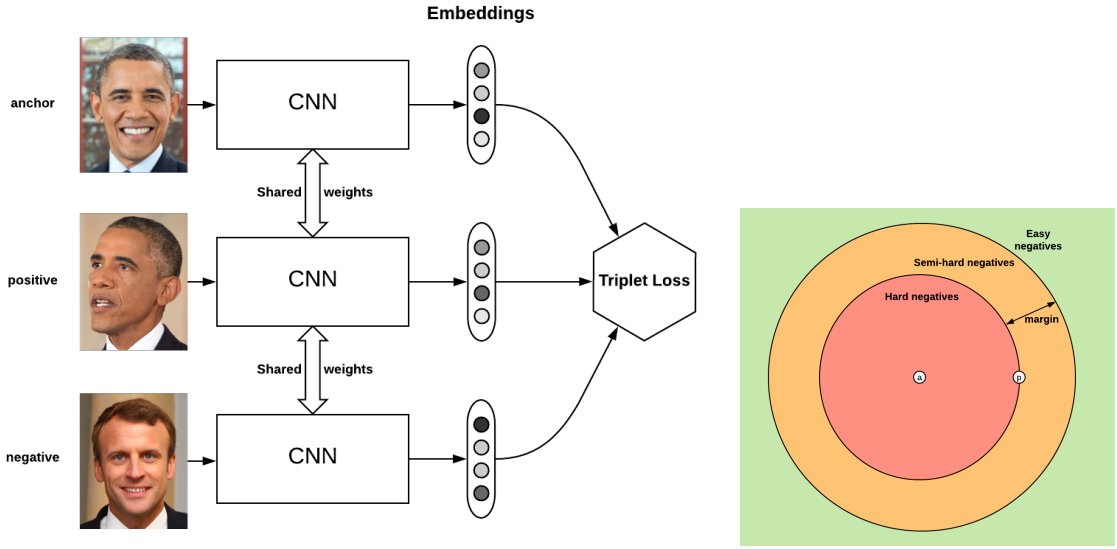


Figure 1: Figures from blog post.

(b) **Briefly describe the *triplet loss* and how it differs from a typical supervised learning objective.**

Solution: Open-ended question, there's no single "right" answer. The triplet loss takes an anchor x^a, a positive x^p (another image of the same person) and a negative x^n (an image of a different person), and penalizes the embedding f unless $\|f(x^a) - f(x^p)\|_2^2 + \alpha < \|f(x^a) - f(x^n)\|_2^2$ for a margin α. A typical supervised objective, such as softmax cross-entropy over a fixed set of identities, predicts a class label for each image. The triplet loss never predicts a class: it only uses whether two images show the same person, and it directly trains the distances between embeddings. So the learned embedding can be used to compare faces of people who were not in the training set.

(c) **What is the challenge with generating all possible triplets? Briefly describe how the authors address this challenge.**

Solution: Generating all possible triplets would simultaneously be computationally expensive (and likely infeasible for large datasets) and wasteful (since many triplets would trivially satisfy the condition set by equation 1). The authors use a variety of methods outlined in Section 3.3, including "hard" triplet mining for selecting the most informative triplets.

(d) **How many parameters and floating point operations (FLOPs) do the authors use for their neural network? How does this compare to a ResNet-50?** Read Table 1 of the original paper[4] on ResNet to find out the FLOPs of ResNet-50. Its parameter count can be found by searching online.

Solution: Table 1 suggests that the authors use a 140M (NN1) / 7.5M (NN2) parameter model for with 1.6B FLOPS. ResNet-50 uses approximately 26M parameters and 3.8B FLOPS.

(e) **What do the authors mean by *semi-hard* negatives?**

Solution: Semi-hard negatives are negatives that are farther from the anchor than the positive, but still within the margin α:

$$\|f(x^a) - f(x^p)\|_2^2 < \|f(x^a) - f(x^n)\|_2^2 < \|f(x^a) - f(x^p)\|_2^2 + \alpha.$$

These triplets still violate the margin, so they give a nonzero loss and a useful gradient. The authors use them because selecting the hardest negatives (those closer to the anchor than the positive) can lead

[4] He, Kaiming, et al. "Deep residual learning for image recognition." (2016).

 All Rights Reserved. This may not be publicly shared without explicit permission.

to bad local minima early in training, for example a collapsed model with $f(x) = 0$.

(f) **What are harmonic embeddings?**

Solution: Harmonic embeddings refer to a way of embedding data points across different neural network models such that they satisfy a special property: embeddings generated by different models v1 and v2 are compatible in the sense that they can be compared to each other. This compatibility greatly simplifies the path to using multiple models and comparing their results.

(g) **How does the performance vary with embedding dimensionality?**

Solution: The authors find that the performs improves slightly by increasing embedding dimensionality from 64 to 128, then drops on increasing it further. While it is natural to expect the performance to improve with higher dimensionality, the authors speculate that the performance drop may be due to larger embedding spaces requiring more training time to achieve similar (or better) performance. It is also likely that the difference in performance in Table 5 is statistically insignificant.

(h) **How does the performance vary with increasing amounts of training data?**

Solution: Expectedly, the validation performance increases with increasing amounts of training data, although Table 6 suggests plateauing of the performance. $10\times$ increase from 2.6M to 26M training images improves the performance by $\approx 9\%$, while a further $10\times$ increase only improves performance by 1%.

(i) Briefly share your favorite *emergent* property/result of the learned behavior with a triplet loss from the paper.

Solution: Open ended question, there's no "right" answer.

(j) **Which approach taken by the authors interested you the most? Why?** (≈ 100 words)

Solution: Open ended question, there's no "right" answer.

5. A Brief Introduction to Transformer Interpretability

Modern neural networks, particularly large language models like Transformers, achieve remarkable performance but are often treated as "black boxes". Mechanistic interpretability is a research field dedicated to reverse-engineering the specific circuits these models learn. Instead of just observing their input-output behavior, we aim to understand the internal mechanisms and computations that lead to their capabilities. This problem explores a mathematical framework proposed by Elhage et al. that treats Transformers as a collection of interacting "circuits" built from attention and MLP layers.

The Residual Stream: A Central Communication Channel

A core concept in this framework is the **residual stream**. At any given layer l, the residual stream, denoted $\mathbf{X}^l$, is a sequence of vectors representing the state of the computation for each token in the input. Crucially, the Transformer, like all modern post-ResNet architectures, is based on residual connections, meaning the output of each layer is *added* to the stream from the previous layer. For a layer l containing a component (e.g., an attention head) with function $f^{(l)}$, the update rule is:

$$\mathbf{X}^{l+1} = \mathbf{X}^l + f^{(l)}(\mathbf{X}^l)$$

The final output of an L-layer model is therefore the sum of the initial embedding and the outputs of all layers:

$$\mathbf{X}^{\text{final}} = \mathbf{X}^0 + \sum_{l=0}^{L-1} f^{(l)}(\mathbf{X}^l)$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

This additive structure is fundamental. It allows us to view the residual stream as a central communication bus or channel where different components read information from and additively write their results back to. This reframing is key to analyzing the function of individual components in relative isolation.

Throughout this problem, we refer to the following variables and their shapes.

Variable	Shape	Description
$\mathbf{T}$	$\mathbb{R}^{n_{\text{vocab}} \times n_{\text{context}}}$	Matrix of one-hot encoded input tokens. Each column is a token.
$\mathbf{X}^l$	$\mathbb{R}^{d_{\text{model}} \times n_{\text{context}}}$	"Residual stream" or "embedding" vectors of model at layer l. Each column is a vector for a token.
$\mathbf{W}_E$	$\mathbb{R}^{d_{\text{model}} \times n_{\text{vocab}}}$	Token embedding matrix. Maps one-hot vectors to the residual stream.
$\mathbf{W}_U$	$\mathbb{R}^{n_{\text{vocab}} \times d_{\text{model}}}$	Token unembedding matrix. Maps final residual stream vectors to logits.
$\mathbf{W}_Q^h, \mathbf{W}_K^h, \mathbf{W}_V^h$	$\mathbb{R}^{d_{\text{head}} \times d_{\text{model}}}$	Query, Key, and Value weight matrices for attention head h.
$\mathbf{W}_O^h$	$\mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$	Output weight matrix for attention head h. This weight matrix left multiplies the output from attention head h and projects it into the dimension of the residual stream.
$\mathbf{A}^h$	$\mathbb{R}^{n_{\text{context}} \times n_{\text{context}}}$	The attention pattern matrix (post-softmax) for attention head h.
$\mathbf{W}_{OV}^h$	$\mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$	The Output-Value matrix for attention head h, defined as $\mathbf{W}_{OV}^h = \mathbf{W}_O^h \mathbf{W}_V^h$.
$\mathbf{W}_{QK}^h$	$\mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$	The Query-Key matrix for attention head h, defined as $\mathbf{W}_{QK}^h = \mathbf{W}_Q^{hT} \mathbf{W}_K^h$.

Note that the superscript h is omitted when context is clear about which attention head we are referring to.

(a) **The Simplest Transformer (A Zero-Layer Model)**

i. Given a matrix of one-hot input tokens $\mathbf{T} \in \mathbb{R}^{n_{\text{vocab}} \times n_{\text{context}}}$, write the mathematical expression for the final logits, $\mathbf{L} \in \mathbb{R}^{n_{\text{vocab}} \times n_{\text{context}}}$. Your expression should be in terms of $\mathbf{T}$, the token embedding matrix $\mathbf{W}_E$, and the unembedding matrix $\mathbf{W}_U$.

Solution: The computation proceeds in two steps:

(1) **Embedding:** The one-hot tokens $\mathbf{T}$ are mapped into the residual stream. Since there are no layers, this initial stream is also the final stream.

$$\mathbf{X}^{\text{final}} = \mathbf{X}^0 = \mathbf{W}_E \mathbf{T}$$

(2) **Unembedding:** The final residual stream vectors are mapped to logits.

$$\mathbf{L} = \mathbf{W}_U \mathbf{X}^{\text{final}}$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

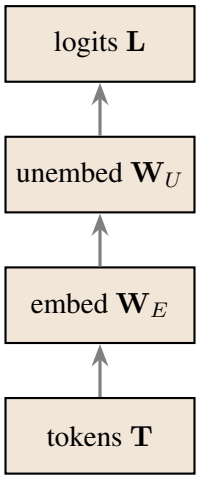


Figure 2: The Simplest Transformer: A Zero-Layer Model

Substituting the first equation into the second gives the full expression:

$$\mathbf{L} = \mathbf{W}_U \mathbf{W}_E \mathbf{T}$$

ii. In simple terms, what algorithm does this zero-layer model implement? What information does the prediction for the token at position t depend on? This reveals the baseline functionality of the Transformer architecture before any contextual processing is introduced.

Solution: This model implements a simple, position-wise word lookup and prediction. To see this clearly, let's analyze the computation for a single token at position t. The t-th column of $\mathbf{T}$, denoted $\mathbf{T}_t$, is a one-hot vector in $\mathbb{R}^{n_{\text{vocab}}}$. The product $\mathbf{W}_E \mathbf{T}_t$ selects the column of $\mathbf{W}_E$ corresponding to the input token at that position, giving its embedding $\mathbf{X}_t$. The product $\mathbf{W}_U \mathbf{X}_t$ then computes the logits $\mathbf{L}_t$ for the output token at position t. It is important to note that the calculation of logits $\mathbf{L}_t$ **only depends on the input token** $\mathbf{T}_t$. It has no access to any other tokens in the context. Therefore, this zero-layer model is equivalent to a simple bigram model (predicting the next word based only on the current word) applied independently at every position in the sequence. It has no contextual understanding. All contextual processing in a Transformer comes from the attention and MLP layers. This zero-layer computation is sometimes called the "direct path" in the mechanistic interpretability literature.

(b) **Multi-Head Attention: Concatenation vs. Addition**

In the original Vaswani et al. paper on transformers, a multi-head attention layer is described differently than our "independent circuits" view. There, the outputs of all heads are concatenated and then multiplied by a single large output matrix. This question asks you to prove this is equivalent to an "additive and independent" view.

Consider an attention layer with H heads. Let the value computation output for head h be $\mathbf{r}^h \in \mathbb{R}^{d_{\text{head}} \times n_{\text{context}}}$. In the "concatenation" view, these are stacked vertically to form a matrix $\mathbf{R}^H \in \mathbb{R}^{d_{\text{head}} \cdot H \times n_{\text{context}}}$ and multiplied by a single large output matrix $\mathbf{W}_O^H \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}} \cdot H}$, and the final output is $\mathbf{H} = \mathbf{W}_O^H \mathbf{R}^H$. Typically, the dimension $d_{\text{head}} \cdot H$ is equal to d_{model} where $H = d_{\text{model}}/d_{\text{head}}$. In the "additive and independent" view, each head h has its own output matrix, $\mathbf{W}_O^h \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$, and the total output is $\mathbf{H} = \sum_{h=1}^{H} \mathbf{W}_O^h \mathbf{r}^h$.

(i) Show that these formulations are equivalent. Specifically, demonstrate how $\mathbf{W}_O^H$ can be constructed from the individual $\mathbf{W}_O^h$ matrices to make the two expressions for $\mathbf{H}$ identical.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Solution: We want to show that $\mathbf{W}_O^H \mathbf{R}^H = \sum_{h=1}^H \mathbf{W}_O^h \mathbf{r}^h$.
To make the formulations equivalent, we construct the large output matrix $\mathbf{W}_O^H$ by horizontally concatenating the individual head output matrices:

$$\mathbf{W}_O^H = \begin{bmatrix} \mathbf{W}_O^1 & \mathbf{W}_O^2 & \cdots & \mathbf{W}_O^H \end{bmatrix}$$

where each block $\mathbf{W}_O^h$ has dimensions $d_{\text{model}} \times d_{\text{head}}$.
Now, we compute the product using the rules of block matrix multiplication:

$$\begin{aligned} \mathbf{H} = \mathbf{W}_O^H \mathbf{R}^H &= \begin{bmatrix} \mathbf{W}_O^1 & \mathbf{W}_O^2 & \cdots & \mathbf{W}_O^H \end{bmatrix} \begin{bmatrix} \mathbf{r}^1 \\ \mathbf{r}^2 \\ \vdots \\ \mathbf{r}^H \end{bmatrix} \\ &= \mathbf{W}_O^1 \mathbf{r}^1 + \mathbf{W}_O^2 \mathbf{r}^2 + \cdots + \mathbf{W}_O^H \mathbf{r}^H \\ &= \sum_{h=1}^H \mathbf{W}_O^h \mathbf{r}^h \end{aligned}$$

This final expression is identical to the "additive and independent" formulation. This proves that the two perspectives are mathematically equivalent, showing us that we can analyze the contributions of each head independently.

(ii) What is an advantage and disadvantage of each view?

Solution: The concatenation view is more computationally efficient because it only requires a single matrix multiplication. However, it is less interpretable because it obscures the independent contribution of each head. The additive view makes it explicit that each head independently writes to the residual stream, which is more interpretable and aligns with the "residual stream as communication channel" perspective. However, it requires H separate matrix multiplications, which is less efficient.

For the following parts, we consider a Transformer with a single attention layer with H heads and no normalizations as shown in Figure 3.

(c) **The QK Circuit: Determining Attention Patterns**

An attention head can conceptually be split into two independent operations: one that decides *where to look* (the QK circuit) and one that decides *what information to move* (the OV circuit). Let's first analyze the "where to look" mechanism.

The pre-softmax attention score from a "query" token at position i to a "key" token at position j is computed as:

$$\mathbf{S}_{ij} = \mathbf{Q}_i^T \mathbf{K}_j$$

where $\mathbf{Q}_i = \mathbf{W}_Q \mathbf{X}_i$ and $\mathbf{K}_j = \mathbf{W}_K \mathbf{X}_j$ are the query and key vectors. (We drop the $1/\sqrt{d_{\text{head}}}$ scaling, which does not change the analysis.)

(i) Starting from the definitions above, derive the expression for $\mathbf{S}_{ij}$ as a bilinear form involving the residual stream vectors $\mathbf{X}_i$ and $\mathbf{X}_j$, and a single "virtual" weight matrix $\mathbf{W}_{QK}$. Explicitly define $\mathbf{W}_{QK}$ in terms of $\mathbf{W}_Q$ and $\mathbf{W}_K$.

Solution: The pre-softmax attention score $\mathbf{S}_{ij} = \mathbf{Q}_i^T \mathbf{K}_j$. We substitute the definitions of query and key vectors, $\mathbf{Q}_i = \mathbf{W}_Q \mathbf{X}_i$ and $\mathbf{K}_j = \mathbf{W}_K \mathbf{X}_j$:

$$\mathbf{S}_{ij} = (\mathbf{W}_Q \mathbf{X}_i)^T (\mathbf{W}_K \mathbf{X}_j)$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

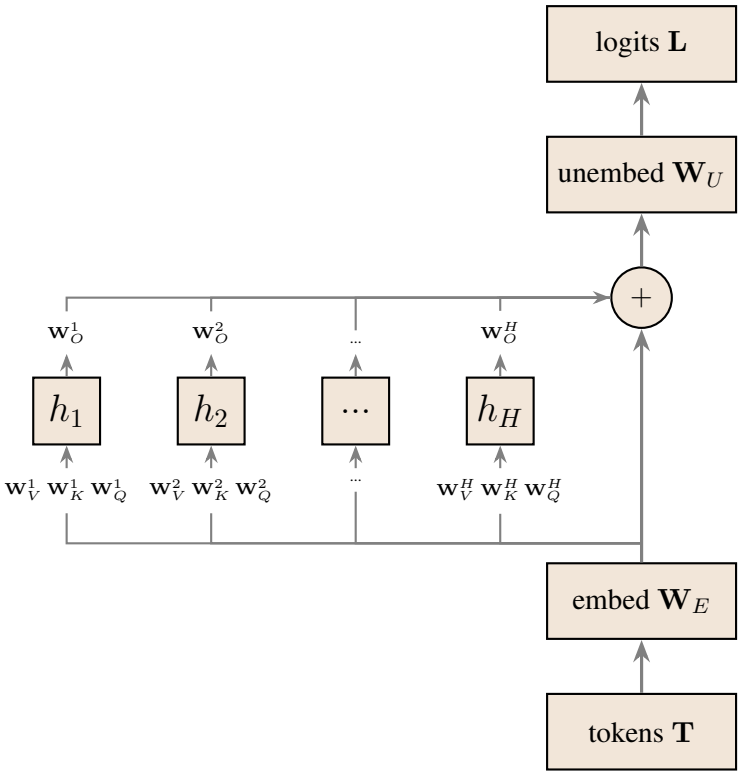


Figure 3: A Transformer With A Single Attention Layer with No Normalizations

Using the property $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$, we get:

$$\mathbf{S}_{ij} = \mathbf{X}_i^T\mathbf{W}_Q^T\mathbf{W}_K\mathbf{X}_j$$

We can define the virtual "Query-Key" matrix $\mathbf{W}_{QK}$ as $\mathbf{W}_Q^T\mathbf{W}_K$, which leads us to the final expression

$$\mathbf{S}_{ij} = \mathbf{X}_i^T\mathbf{W}_{QK}\mathbf{X}_j$$

(ii) The matrix $\mathbf{W}_{QK}$ can be interpreted as defining the "question" the attention head asks to determine which tokens to attend to. Consider a toy scenario where $d_{\text{model}} = 3$.

(1) If $\mathbf{W}_{QK} = \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, what kind of relationship between $\mathbf{X}_i$ and $\mathbf{X}_j$ would lead to a high attention score $\mathbf{S}_{ij} = \mathbf{X}_i^T\mathbf{W}_{QK}\mathbf{X}_j$? Describe in words what this head "looks for."

Solution: If $\mathbf{W}_{QK} = \mathbf{I}$, then $\mathbf{S}_{ij} = \mathbf{X}_i^T\mathbf{I}\mathbf{X}_j = \mathbf{X}_i^T\mathbf{X}_j$. This is the dot product of the two residual stream vectors. A high score means the vectors are pointing in a similar direction. This head "looks for" tokens whose residual stream vectors are most similar to its own.

(2) If $\mathbf{W}_{QK} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, what kind of relationship would lead to a high attention score? What happens to information in the second and third dimensions?

 All Rights Reserved. This may not be publicly shared without explicit permission.

Solution: Let $\mathbf{X}_i = \begin{bmatrix} \mathbf{X}_{i1} \\ \mathbf{X}_{i2} \\ \mathbf{X}_{i3} \end{bmatrix}$ and $\mathbf{X}_j = \begin{bmatrix} \mathbf{X}_{j1} \\ \mathbf{X}_{j2} \\ \mathbf{X}_{j3} \end{bmatrix}$. The score is:

$$\begin{aligned} \mathbf{S}_{ij} &= \mathbf{X}_i^T \mathbf{W}_{QK} \mathbf{X}_j \\ &= \begin{bmatrix} \mathbf{X}_{i1} & \mathbf{X}_{i2} & \mathbf{X}_{i3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{j1} \\ \mathbf{X}_{j2} \\ \mathbf{X}_{j3} \end{bmatrix} \\ &= \mathbf{X}_{i1} \mathbf{X}_{j1} \end{aligned}$$

This head only cares about the first dimension of the residual stream vectors. Information in the second and third dimensions is completely ignored—effectively "deleted" from the attention computation. This demonstrates how the QK circuit can selectively attend to specific features while ignoring others.

(3) If $\mathbf{W}_{QK} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$, what pattern does this head look for? How does this differ from the identity case?

Solution: The score is:

$$\begin{aligned} \mathbf{S}_{ij} &= \mathbf{X}_i^T \mathbf{W}_{QK} \mathbf{X}_j \\ &= \mathbf{X}_{i1} \mathbf{X}_{j1} + \mathbf{X}_{i2} \mathbf{X}_{j2} - \mathbf{X}_{i3} \mathbf{X}_{j3} \end{aligned}$$

This head rewards similarity in the first two dimensions but *dissimilarity* in the third dimension. For example, it would give a high score if $\mathbf{X}_i$ and $\mathbf{X}_j$ are both positive in dimensions 1 and 2, but have opposite signs in dimension 3. This demonstrates that the QK circuit can implement complex logical queries that combine multiple conditions, including both positive and negative correlations.

(4) If $\mathbf{W}_{QK} = -\mathbf{I} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$, what pattern does this head look for? What is the effect of this on the attention pattern after softmax?

Solution: The score is:

$$\mathbf{S}_{ij} = \mathbf{X}_i^T (-\mathbf{I}) \mathbf{X}_j = -\mathbf{X}_i^T \mathbf{X}_j$$

This is the *negative* dot product. High scores occur when vectors point in *opposite* directions. Importantly, this "suppresses" or "deletes" attention to similar tokens. After softmax, positions with similar residual stream vectors will receive low attention weights, while positions with dissimilar vectors will receive higher weights. This demonstrates that the QK circuit can be programmed to *avoid* certain patterns, not just seek them out. Such "negative queries" could be useful for preventing the model from attending to redundant information or for implementing more complex logical operations like "attend to tokens that are different from the current one."

(d) **The OV Circuit: Reading and Writing Information**

Now let's analyze the information-moving part of the attention head. Once the attention pattern $\mathbf{A}^h$ is

 All Rights Reserved. This may not be publicly shared without explicit permission.

determined (by the QK circuit), the head reads information from attended tokens and writes it to the destination. This is governed by the OV circuit. Note that the output of head h is:

$$h(\mathbf{X}^0) = \mathbf{W}_O^h \mathbf{W}_V^h \mathbf{X}^0 \mathbf{A}^{hT} = \mathbf{W}_{OV}^h \mathbf{X}^0 \mathbf{A}^{hT}$$

where $\mathbf{A}^h$ is the attention pattern (post-softmax) determined by the QK circuit.

(i) **[Understanding the Attention-Weighted Average]**
Before proceeding, let's clarify what $\mathbf{X}^0\mathbf{A}^T$ computes. Recall that:

- $\mathbf{X}^0 \in \mathbb{R}^{d_{\text{model}} \times n_{\text{context}}}$ where the j-th column $\mathbf{X}_j^0$ is the residual stream vector for the token at position j.
- $\mathbf{A} \in \mathbb{R}^{n_{\text{context}} \times n_{\text{context}}}$ is the attention pattern where $\mathbf{A}_{ij}$ is the attention weight from destination position i to source position j.
- Therefore, $\mathbf{A}^T \in \mathbb{R}^{n_{\text{context}} \times n_{\text{context}}}$ where $(\mathbf{A}^T)_{ji} = \mathbf{A}_{ij}$.

Show that the i-th column of $\mathbf{X}^0\mathbf{A}^T$ is an attention-weighted average of the source token vectors. Specifically, show:

$$(\mathbf{X}^0\mathbf{A}^T)_i = \sum_{j=1}^{n_{\text{context}}} \mathbf{A}_{ij}\mathbf{X}_j^0$$

Solution: The i-th column of the matrix product $\mathbf{X}^0\mathbf{A}^T$ is obtained by multiplying $\mathbf{X}^0$ with the i-th column of $\mathbf{A}^T$:

$$(\mathbf{X}^0\mathbf{A}^T)_i = \mathbf{X}^0 \cdot (\mathbf{A}^T)_i$$

The i-th column of $\mathbf{A}^T$ equals the i-th row of $\mathbf{A}$ (written as a column vector):

$$(\mathbf{A}^T)_i = \begin{bmatrix} \mathbf{A}_{i1} \\ \mathbf{A}_{i2} \\ \vdots \\ \mathbf{A}_{i,n_{\text{context}}} \end{bmatrix}$$

This vector contains all the attention weights from destination position i to each source position j.
Now, we can write $\mathbf{X}^0$ as a block matrix of its columns:

$$\mathbf{X}^0 = \begin{bmatrix} \mathbf{X}_1^0 & \mathbf{X}_2^0 & \cdots & \mathbf{X}_{n_{\text{context}}}^0 \end{bmatrix}$$

The matrix-vector product is:

$$\begin{aligned}(\mathbf{X}^0\mathbf{A}^T)_i &= \begin{bmatrix} \mathbf{X}_1^0 & \mathbf{X}_2^0 & \cdots & \mathbf{X}_{n_{\text{context}}}^0 \end{bmatrix} \begin{bmatrix} \mathbf{A}_{i1} \\ \mathbf{A}_{i2} \\ \vdots \\ \mathbf{A}_{i,n_{\text{context}}} \end{bmatrix} \\ &= \mathbf{A}_{i1}\mathbf{X}_1^0 + \mathbf{A}_{i2}\mathbf{X}_2^0 + \cdots + \mathbf{A}_{i,n_{\text{context}}}\mathbf{X}_{n_{\text{context}}}^0 \\ &= \sum_{j=1}^{n_{\text{context}}} \mathbf{A}_{ij}\mathbf{X}_j^0\end{aligned}$$

This is precisely an attention-weighted average! The i-th column of $\mathbf{X}^0\mathbf{A}^T$ combines the source

 All Rights Reserved. This may not be publicly shared without explicit permission.

token vectors $\mathbf{X}_j^0$ with weights $\mathbf{A}_{ij}$. Since attention weights satisfy $\sum_j \mathbf{A}_{ij} = 1$ (from softmax), this is a convex combination of the source vectors.

Interpretation: For each destination position i, the attention pattern $\mathbf{A}$ determines how much to "gather" from each source position j. The result is a new vector that blends information from multiple source positions according to the attention weights.

(ii) Write the full expression for the final residual stream, $\mathbf{X}^{\text{final}}$, in terms of the initial stream $\mathbf{X}^0$ and the heads' weight matrices.

Solution: The final residual stream $\mathbf{X}^{\text{final}}$ is the sum of the initial stream $\mathbf{X}^0$ and the outputs of all attention heads:

$$\begin{aligned}\mathbf{X}^{\text{final}} &= \mathbf{X}^0 + \sum_{i=1}^{H} h_i(\mathbf{X}^0) \\ &= \mathbf{X}^0 + \sum_{i=1}^{H} \mathbf{W}_O^i \mathbf{W}_V^i \mathbf{X}^0 \mathbf{A}^{iT} \\ &= \mathbf{X}^0 + \sum_{i=1}^{H} \mathbf{W}_{OV}^i \mathbf{X}^0 \mathbf{A}^{iT}\end{aligned}$$

(iii) The update vector added to the residual stream at destination position t by head i is the t-th column of the head's output, denoted $h_i(\mathbf{X}^0)_t$. Prove that this update vector must lie within the column space of $\mathbf{W}_{OV}^i$. That is, show that $h_i(\mathbf{X}^0)_t \in \text{Col}(\mathbf{W}_{OV}^i)$. This demonstrates that the head can only write to a low-dimensional subspace of the d_{model}-dimensional residual stream.

Solution: Let the head's output be $h_i(\mathbf{X}^0) = \mathbf{W}_{OV}^i \mathbf{X}^0 \mathbf{A}^{iT}$. Define an intermediate matrix $\mathbf{Z}^i = \mathbf{X}^0 \mathbf{A}^{iT} \in \mathbb{R}^{d_{\text{model}} \times n_{\text{context}}}$, which represents the attention-weighted average of the residual stream vectors. The head's output can then be written as:

$$h_i(\mathbf{X}^0) = \mathbf{W}_{OV}^i \mathbf{Z}^i$$

Let $h_i(\mathbf{X}^0)_t$ be the t-th column of $h_i(\mathbf{X}^0)$ and $\mathbf{Z}_t^i$ be the t-th column of $\mathbf{Z}^i$. By the definition of matrix-matrix multiplication, the t-th column of the product is:

$$h_i(\mathbf{X}^0)_t = \mathbf{W}_{OV}^i \mathbf{Z}_t^i$$

A matrix-vector product $\mathbf{W}_{OV}^i \mathbf{Z}_t^i$ is, by definition, a linear combination of the columns of $\mathbf{W}_{OV}^i$, where the coefficients are the elements of $\mathbf{Z}_t^i$:

$$h_i(\mathbf{X}^0)_t = \mathbf{W}_{OV}^i \mathbf{Z}_t^i = \sum_{j=1}^{d_{\text{model}}} (\mathbf{W}_{OV}^i)_j (\mathbf{Z}_t^i)_j$$

where $(\mathbf{W}_{OV}^i)_j$ is the j-th column of $\mathbf{W}_{OV}^i$ and $(\mathbf{Z}_t^i)_j$ is the j-th element of $\mathbf{Z}_t^i$.

Since $h_i(\mathbf{X}^0)_t$ is a linear combination of the columns of $\mathbf{W}_{OV}^i$, it must lie within the column space of $\mathbf{W}_{OV}^i$, i.e., $h_i(\mathbf{X}^0)_t \in \text{Col}(\mathbf{W}_{OV}^i)$. This holds for any position t and any head i, so every update vector is constrained to this "write subspace."

(e) **Formalizing Read and Write Subspaces via SVD**

 All Rights Reserved. This may not be publicly shared without explicit permission.

Important: Separating the QK and OV Circuits
In this analysis, we focus exclusively on the **OV circuit** $\mathbf{W}_{OV}$, which characterizes *what* information is moved and *how* it is transformed.
Recall from part (d) that the full output of an attention head is:

$$h(\mathbf{X}^0) = \mathbf{W}_{OV}\mathbf{X}^0\mathbf{A}^T$$

This can be decomposed into two independent operations:

- **QK Circuit** (analyzed in part c): Computes the attention pattern $\mathbf{A}$, which determines *which* source tokens to attend to at each destination position.
- **OV Circuit** (analyzed here): The linear transformation $\mathbf{W}_{OV}$ applied to source tokens, determining *what* information is extracted and *where* it is written.

In our SVD analysis below, when we write $\mathbf{W}_{OV}\mathbf{X}_{\text{src}}$, we are analyzing what happens to information from a *single* source token. The attention pattern then determines which source tokens this transformation is applied to and with what weights. These two circuits operate *independently*: you can understand what the OV circuit does to each source token without knowing which tokens will be attended to.

In part (d), we showed that each attention head can only write to the column space of $\mathbf{W}_{OV}$, a low-dimensional subspace. But this raises deeper questions: *which* directions in the source token does the head read from? And *which* directions in the destination does it write to? The Singular Value Decomposition (SVD) provides a complete answer, revealing that attention heads are specialized communication channels between specific subspaces.

(i) Prove that $\text{rank}(\mathbf{W}_{OV}) \leq d_{\text{head}}$. Why does this imply that $\mathbf{W}_{OV}$ is a low-rank matrix (in typical transformer architectures)?

Solution: Recall from linear algebra that the rank of a matrix product is bounded by the minimum rank of its factors:

$$\text{rank}(\mathbf{AB}) \leq \min(\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B}))$$

Here, $\mathbf{W}_{OV} = \mathbf{W}_O\mathbf{W}_V$ where $\mathbf{W}_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$ and $\mathbf{W}_V \in \mathbb{R}^{d_{\text{head}} \times d_{\text{model}}}$.
The rank of any matrix is bounded by its smallest dimension. Since $\mathbf{W}_O$ has d_{head} columns, $\text{rank}(\mathbf{W}_O) \leq d_{\text{head}}$. Similarly, since $\mathbf{W}_V$ has d_{head} rows, $\text{rank}(\mathbf{W}_V) \leq d_{\text{head}}$.
Therefore:

$$\text{rank}(\mathbf{W}_{OV}) \leq \min(d_{\text{head}}, d_{\text{head}}) = d_{\text{head}}$$

In typical architectures, $d_{\text{head}} \ll d_{\text{model}}$ (e.g., $d_{\text{head}} = 64$ while $d_{\text{model}} = 768$), making $\mathbf{W}_{OV}$ a very low-rank approximation of a full $d_{\text{model}} \times d_{\text{model}}$ matrix. This low-rank structure is what allows us to interpret the head as a specialized channel.

(ii) Because $\mathbf{W}_{OV}$ is low-rank, its SVD has a special structure. Let $\mathbf{W}_{OV} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be the (compact) SVD, where $\mathbf{U} \in \mathbb{R}^{d_{\text{model}} \times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$, and $\mathbf{V} \in \mathbb{R}^{d_{\text{model}} \times r}$, with $r = \text{rank}(\mathbf{W}_{OV}) \leq d_{\text{head}}$. Consider a source token with residual stream vector $\mathbf{X}_{\text{src}} \in \mathbb{R}^{d_{\text{model}}}$. Show that the output of the OV circuit can be decomposed as:

$$\mathbf{W}_{OV}\mathbf{X}_{\text{src}} = \sum_{k=1}^{r} \sigma_k(\mathbf{V}_k^T\mathbf{X}_{\text{src}})\mathbf{U}_k \tag{10}$$

where $\mathbf{U}_k$ and $\mathbf{V}_k$ are the k-th columns of $\mathbf{U}$ and $\mathbf{V}$, respectively, and σ_k is the k-th singular value.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Solution: Starting with the SVD:

$$\begin{aligned}\mathbf{W}_{OV}\mathbf{X}_{\text{src}} &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)\mathbf{X}_{\text{src}} \\ &= \mathbf{U}(\mathbf{\Sigma}(\mathbf{V}^T\mathbf{X}_{\text{src}}))\end{aligned}$$

We analyze this computation in three steps:
Step 1 - Reading: $\mathbf{V}^T\mathbf{X}_{\text{src}}$ computes the projections of the source vector onto each right singular vector. Since $\mathbf{V} = [\mathbf{V}_1\ \mathbf{V}_2\ \cdots\ \mathbf{V}_r]$ with orthonormal columns, we have:

$$\mathbf{V}^T\mathbf{X}_{\text{src}} = \begin{bmatrix}\mathbf{V}_1^T\mathbf{X}_{\text{src}} \\ \mathbf{V}_2^T\mathbf{X}_{\text{src}} \\ \vdots \\ \mathbf{V}_r^T\mathbf{X}_{\text{src}}\end{bmatrix} \in \mathbb{R}^r$$

Each element $\mathbf{V}_k^T\mathbf{X}_{\text{src}}$ is a scalar measuring how much of $\mathbf{X}_{\text{src}}$ lies in the direction $\mathbf{V}_k$.
Step 2 - Weighting: $\mathbf{\Sigma}$ scales each projection by its importance:

$$\mathbf{\Sigma}(\mathbf{V}^T\mathbf{X}_{\text{src}}) = \begin{bmatrix}\sigma_1(\mathbf{V}_1^T\mathbf{X}_{\text{src}}) \\ \sigma_2(\mathbf{V}_2^T\mathbf{X}_{\text{src}}) \\ \vdots \\ \sigma_r(\mathbf{V}_r^T\mathbf{X}_{\text{src}})\end{bmatrix} \in \mathbb{R}^r$$

Step 3 - Writing: $\mathbf{U}$ maps these weighted projections to the output:

$$\mathbf{U}\begin{bmatrix}\sigma_1(\mathbf{V}_1^T\mathbf{X}_{\text{src}}) \\ \vdots \\ \sigma_r(\mathbf{V}_r^T\mathbf{X}_{\text{src}})\end{bmatrix} = \sum_{k=1}^{r}\sigma_k(\mathbf{V}_k^T\mathbf{X}_{\text{src}})\mathbf{U}_k$$

This final expression shows that the output is a linear combination of the left singular vectors $\{\mathbf{U}_k\}$, with coefficients determined by how much the input aligns with each right singular vector $\{\mathbf{V}_k\}$.

(iii) Based on the decomposition above, define precisely:

(1) The **read subspace**: Which directions in $\mathbf{X}_{\text{src}}$ can the head extract information from?

(2) The **write subspace**: Which directions in the destination residual stream can the head write to?

(3) What happens to information in $\mathbf{X}_{\text{src}}$ that is orthogonal to the read subspace?

Hint: Think about which matrix "touches" the input first (this determines what can be read) and which matrix determines the form of the output (this determines what can be written).

Solution: **Intuition:** When we compute $\mathbf{W}_{OV}\mathbf{X}_{\text{src}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{X}_{\text{src}}$ from right to left:

- $\mathbf{V}^T$ operates on the *input*, extracting information by projecting onto its rows (the $\mathbf{V}_k$ vectors). If $\mathbf{V}_k^T\mathbf{X}_{\text{src}} = 0$, that direction cannot be read. So $\mathbf{V}$ defines what can be *read*.
- $\mathbf{U}$ produces the *output* as a linear combination of its columns (the $\mathbf{U}_k$ vectors). The output must lie in span($\mathbf{U}$). So $\mathbf{U}$ defines what can be *written*.

From $\mathbf{W}_{OV}\mathbf{X}_{\text{src}} = \sum_{k=1}^{r}\sigma_k(\mathbf{V}_k^T\mathbf{X}_{\text{src}})\mathbf{U}_k$:
(1) **Read Subspace:** span$(\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_r) \subseteq \mathbb{R}^{d_{\text{model}}}$

 All Rights Reserved. This may not be publicly shared without explicit permission.

The head extracts information by projecting $\mathbf{X}_{\text{src}}$ onto each right singular vector $\mathbf{V}_k$ via the inner products $\mathbf{V}_k^T\mathbf{X}_{\text{src}}$. Only components of $\mathbf{X}_{\text{src}}$ that lie in this r-dimensional subspace can influence these projections. The read subspace has dimension $r \leq d_{\text{head}} \ll d_{\text{model}}$.
(2) **Write Subspace:** $\text{span}(\mathbf{U}_1, \mathbf{U}_2, \ldots, \mathbf{U}_r) \subseteq \mathbb{R}^{d_{\text{model}}}$
The output is explicitly a linear combination $\sum_{k=1}^{r} c_k \mathbf{U}_k$ where $c_k = \sigma_k(\mathbf{V}_k^T\mathbf{X}_{\text{src}})$ are scalars. By definition, any such linear combination must lie in $\text{span}(\mathbf{U}_1, \ldots, \mathbf{U}_r)$. This confirms our earlier result from part (d): $\text{span}(\mathbf{U}_1, \ldots, \mathbf{U}_r) = \text{Col}(\mathbf{W}_{OV})$.
(3) If $\mathbf{X}_{\text{src}}$ has a component $\mathbf{X}_\perp$ orthogonal to all $\mathbf{V}_k$, then $\mathbf{V}_k^T\mathbf{X}_\perp = 0$ for all k. This component contributes nothing to the sum, so it is completely ignored—the head is "blind" to information outside its read subspace.

(iv) **[Connecting to the Value Projection]** The value vector computed by the head is $\mathbf{v} = \mathbf{W}_V\mathbf{X}_{\text{src}}$, which lives in $\mathbb{R}^{d_{\text{head}}}$. Explain in 2-3 sentences how the read subspace $\text{span}(\mathbf{V}_1, \ldots, \mathbf{V}_r)$ relates to what $\mathbf{W}_V$ can "see" in the source token. Why is it natural that the read subspace dimension is at most d_{head}?

Solution: The value projection $\mathbf{W}_V$ maps the d_{model}-dimensional residual stream to a d_{head}-dimensional value vector, so it can only extract at most d_{head} independent pieces of information from $\mathbf{X}_{\text{src}}$. The read subspace is the row space of $\mathbf{W}_{OV} = \mathbf{W}_O\mathbf{W}_V$, which is contained in the row space of $\mathbf{W}_V$: a direction of $\mathbf{X}_{\text{src}}$ that $\mathbf{W}_V$ maps to zero is also mapped to zero by $\mathbf{W}_{OV}$. When $\mathbf{W}_O$ has full column rank d_{head} (the typical case), the two row spaces are equal, so the read subspace is exactly the set of directions the value vector can see. The row space of $\mathbf{W}_V$ has dimension at most d_{head}, which is why the read subspace has dimension at most d_{head}.

(v) **[Synthesis]** The paper describes attention heads as moving information "from the residual stream of one token to another." In 2-3 sentences, explain how the SVD $\mathbf{W}_{OV} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ provides a complete characterization of this information movement. What role does each component ($\mathbf{V}$, $\mathbf{\Sigma}$, $\mathbf{U}$) play, and how does this relate to the attention pattern $\mathbf{A}$?

Solution: The SVD reveals that information movement happens in three stages: (1) The right singular vectors $\mathbf{V}$ define the *read subspace*—which specific directions in each source token the head extracts information from. (2) The singular values $\mathbf{\Sigma}$ weight how important each extracted direction is. (3) The left singular vectors $\mathbf{U}$ define the *write subspace*—where this information is written in the destination token's residual stream. This OV circuit characterization applies to *every* source token identically and is completely independent of the attention pattern $\mathbf{A}$ (determined by the QK circuit), which only controls *which* source tokens to move information from and *how much weight* to give each. Together, the QK circuit (where to look) and OV circuit (what to move) fully specify the attention head's operation.

6. LoRA

A common strategy for adapting a large pre-trained model to a new task is to update only a subset of its parameters, keeping the rest frozen. Low-Rank Adaptation (LoRA) offers a more flexible approach to this idea. In this problem, we focus on a single weight matrix W with m rows and ℓ columns, where W_0 is the pre-trained value. During LoRA-based training, W is replaced by $W_0 + AB$, where W_0 remains frozen and only A and B are learnable. Here, A is an $m \times k$ matrix and B is a $k \times \ell$ matrix, typically with $k \ll \min(m, \ell)$.

(a) Suppose you are using LoRA to adapt a pretrained deep neural net to a new task and observe that the model is "underfitting" the training data. **What would you try to adjust in the LoRA to get better performance?**

Solution: To address underfitting in LoRA, consider the following strategies:

 All Rights Reserved. This may not be publicly shared without explicit permission.

i. Increase the rank k of the low-rank matrices A and B to allow for more expressive power.
Increasing the rank adds parameters to the LoRA and this extended expressive power generally allows better fitting.

ii. Change the initialization you're using to leverage the SVD of either the original pretrained weight matrix or the matrix-valued gradient for a large amount of training data.
By aligning the initializations to the natural gradient directions in either the training data or the original matrix, we are likely to be able to have more significant impacts faster.

iii. Change the learning rate or other hyperparameters to ensure better convergence.
We know that learning rate is generally the single most important hyperparameter to tune. And underfitting can be a manifestation of simply a too-small learning rate. Furthermore, we know from recent work that there are benefits from having a higher learning rate for the "second" matrix in a LoRA — in this case the A matrix. (See the paper: LoRA+: Efficient Low Rank Adaptation of Large Models, arXiv:2402.12354)

(b) Suppose both A and B are initialized to all zeros. **Why will this cause problems for LoRA-based finetuning?**
Remember, this is going to be trained using SGD-style updates over a training set with a loss function.

Solution: If both A and B are initialized to zero, the LoRA adaptation starts as a zero matrix and the gradients for both A and B will also be zero at initialization. This means the parameters will not update during training, preventing the model from learning any adaptation.
The gradients are zero for B because no change in B will have any effect on the output because it will be multiplied by a zero matrix. The gradients are zero for A because the input activations to A are always zero since they are multiplied by a zero matrix.

(c) Consider the following pseudocode for LoRA initialization:

```
A = torch.nn.Parameter(torch.empty(m, k))
B = torch.nn.Parameter(torch.empty(k, l))
torch.nn.init.xavier_uniform_(A)
torch.nn.init.xavier_uniform_(B)
```

Why might LoRA fine-tuning not work well with this initialization?

Solution: Since we are fine-tuning a pretrained model, the weights of A and B should be such that the product AB is 0. Why? Because we want to leverage the features that the pretrained model has.
Xavier initialization does not provide this property, as it initializes A and B independently to be nonzero, leading to a nonzero product. The scale of this shift is not infinitesimal, and so we could move the activations to the subsequent layer far outside of the distribution it was trained for.
The resulting model behavior is likely very different from the pretrained model at initialization.

(d) **How much memory is required to store the LoRA adaptation weights (A and B)?** Assume we are using floats (4 bytes per real number) and give your answer in bytes. **How does this compare to storing a single full-rank adaptation matrix?**

Solution: Storing the LoRA weights requires memory for A ($m \times k$) and B ($k \times \ell$), for a total of $mk + k\ell$ parameters. In bytes, this is $4k(\ell + m)$ bytes. In contrast, storing a full-rank adaptation would require $m\ell$ parameters and thus $4m\ell$ bytes. When $k \ll \min(m, \ell)$, LoRA uses much less memory.

7. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

 All Rights Reserved. This may not be publicly shared without explicit permission.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework?**

Contributors: Gireeja Ranade, Sheng Shen, Shaojie Bai, Angelos Katharopoulos, Hao Peng, Olivia Watkins, Jake Austin, Dhruv Shah, Anant Sahai, Linyuan Gong, Sultan Daniels, Kevin Frans, Patrick Mendoza, Naman Jain.

 All Rights Reserved. This may not be publicly shared without explicit permission.