Report on Assignment 3 - Neural Networks
ELL784

Animesh Lohar - 2024EET2368
Humanshu D G - 2024EEN2016

May 7, 2025

1 Introduction

This report details the implementation and evaluation of Vision Transformer (ViT) models for the MNIST digit classification task. The assignment is structured into three parts, progressively introducing core ViT components, computational efficiency improvements, and multi-scale processing. The goal is to build various ViT architectures from scratch (leveraging deep learning framework building blocks but not pre-built transformer layers) and analyze their performance and characteristics on the MNIST dataset. The implementation is done using the PyTorch framework.

2 Part 1: Transformer for Images from Scratch

The first part focuses on building a basic Vision Transformer model for image classification. This involves designing a tokenization mechanism for images, implementing the multi-head self-attention module, and constructing the transformer encoder layer. Learnable positional encodings are also incorporated.

2.1 Image Tokenization Mechanism

Traditional Transformers process sequences of tokens. For images, we need to convert the 2D image data into a sequence of flattened patches, which serve as tokens. This process is handled by the 'PatchEmbedding' class in the provided code.

The image is divided into non-overlapping square patches of size $P \times P$. For an input image of size $H \times W$ with C channels, and a patch size P, the number of patches will be $\frac{H}{P} \times \frac{W}{P}$. Each patch is flattened into a 1D vector. To maintain spatial information and allow the transformer to process these patches, each flattened patch is linearly projected into a higher-dimensional embedding space of size D.

The 'PatchEmbedding' module uses a convolutional layer with a kernel size and stride equal to the patch size (P). This convolution effectively extracts non-overlapping patches and projects them into the embedding dimension D.

$$\text{Input Image: } \mathbf{X} \in \mathbb{R}^{B \times C \times H \times W}$$

$$\text{Convolution: } \mathbf{Y} = \text{Conv2d}(\mathbf{X}, \text{kernel} = P, \text{stride} = P) \in \mathbb{R}^{B \times D \times \frac{H}{P} \times \frac{W}{P}}$$

The output of the convolution is then flattened along the spatial dimensions and transposed to get a sequence of patches:

$$\mathbf{Z} = \text{Flatten}(\mathbf{Y}, \text{start\_dim} = 2) \in \mathbb{R}^{B \times D \times (\frac{H}{P} \times \frac{W}{P})}$$

$$\text{Token Embeddings: } \mathbf{T} = \text{Transpose}(\mathbf{Z}, 1, 2) \in \mathbb{R}^{B \times N \times D}$$

where $N = (\frac{H}{P} \times \frac{W}{P})$ is the number of patches.

Additionally, a learnable classification token (CLS token) is prepended to the sequence of patch embeddings. This token's embedding after passing through the transformer layers is used for the final classification task. Learnable positional embeddings are added to the patch and CLS tokens to incorporate their spatial position information.

$$\mathbf{T}_{\text{cls}} \in \mathbb{R}^{B \times 1 \times D}$$

$$\mathbf{T}_{\text{cat}} = \text{Concat}(\mathbf{T}_{\text{cls}}, \mathbf{T}) \in \mathbb{R}^{B \times (N+1) \times D}$$

$$\mathbf{P}_{\text{pos}} \in \mathbb{R}^{1\times(N+1)\times D}$$

$$\mathbf{E} = \mathbf{T}_{\text{cat}} + \mathbf{P}_{\text{pos}}$$

This matrix $\mathbf{E}$ serves as the input sequence to the transformer encoder layers.

2.2 Multi-Head Attention and Encoder Layer

The core of the transformer is the multi-head self-attention mechanism. The 'MultiHeadAttention' class implements this. It takes the input sequence $\mathbf{E}$ and computes Query (Q), Key (K), and Value (V) matrices by linearly projecting the input:

$$\mathbf{Q} = \mathbf{E}\mathbf{W}_Q$$

$$\mathbf{K} = \mathbf{E}\mathbf{W}_K$$

$$\mathbf{V} = \mathbf{E}\mathbf{W}_V$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{D\times D}$ are learnable weight matrices.

In multi-head attention, these Q, K, V matrices are split into H "heads". For each head h, the Q, K, V vectors have a reduced dimension $d_k = d_v = D/H$. The attention scores are computed for each head independently:

$$\text{Attention}(Q_h, K_h, V_h) = \text{Softmax}\left(\frac{Q_h K_h^T}{\sqrt{d_k}}\right) V_h$$

The outputs from all heads are then concatenated and linearly projected back to the original embedding dimension D:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)\mathbf{W}_O$$

where $\text{head}_h = \text{Attention}(Q_h, K_h, V_h)$ and $\mathbf{W}_O \in \mathbb{R}^{D\times D}$.

The 'TransformerBlock' implements a standard transformer encoder layer. It consists of a multi-head self-attention sub-layer and a position-wise feed-forward network (FFN). Both sub-layers are followed by layer normalization and residual connections.

$$\mathbf{Y} = \text{LayerNorm}(\mathbf{E} + \text{MultiHead}(\mathbf{E}, \mathbf{E}, \mathbf{E}))$$

$$\mathbf{Z} = \text{LayerNorm}(\mathbf{Y} + \text{FFN}(\mathbf{Y}))$$

The FFN in the code is a simple two-layer MLP with a GELU activation and dropout.

The 'VisionTransformer' class stacks multiple 'TransformerBlock' layers. The output of the final block, specifically the embedding corresponding to the CLS token ($\mathbf{Z}_{:,0}$), is passed through a final layer normalization and a linear layer (head) to produce the class logits. A log-softmax is applied for the final output.

2.3 Learnable Positional Encoding

As implemented in the 'VisionTransformer' class, the positional encoding is a learnable parameter 'self.pos$_e$*mbed*'.*Itisatensorofshape*(1, N+1, D)*where*N*isthenumberofpatchesand*D*istheembeddingdimension.Thispara*

2.4 Performance on Test Set (Part 1 - Vanilla ViT)

After training the 'vanilla' Vision Transformer model for 3 epochs as per the combined code, the performance on the MNIST test set is reported.

The test accuracy for the vanilla Vision Transformer is:

```
vanilla Test: 98.66%
```

This indicates strong performance on the MNIST dataset using a basic ViT architecture.

3 Part 2: Reducing Computational Complexity

The vanilla self-attention mechanism has a computational complexity of $O(N^2 d)$, where N is the sequence length (number of tokens) and d is the embedding dimension. This quadratic dependency on N becomes a bottleneck for high-resolution images or larger patch sizes, which result in a large number of patches. Part 2 explores methods to reduce this complexity.

3.1 Design of Reduced Complexity Attention Layer

The assignment requires designing an attention layer with reduced complexity. The provided code implements two alternatives in the 'MultiHeadAttention' class: 'linformer' and 'uniform'. The 'linformer' attention mechanism is a method to approximate the standard attention with lower complexity.

3.1.1 Linformer Attention

Linformer reduces the complexity from $O(N^2d)$ to $O(Nkd)$, where $k \ll N$ is a predefined projection dimension. This is achieved by projecting the Key and Value matrices to a lower dimension before computing the attention scores.

Let the input to the attention layer be $\mathbf{X} \in \mathbb{R}^{B\times N\times D}$. The Q, K, V matrices are computed as before:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q \in \mathbb{R}^{B\times N\times D}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_K \in \mathbb{R}^{B\times N\times D}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_V \in \mathbb{R}^{B\times N\times D}$$

Instead of computing $\mathbf{Q}\mathbf{K}^T$, Linformer introduces learnable linear projection matrices $\mathbf{E} \in \mathbb{R}^{N\times k}$ and $\mathbf{F} \in \mathbb{R}^{N\times k}$ (or per-head/per-layer variations). The K and V matrices are projected along the sequence dimension:

$$\mathbf{K}_p = \mathbf{K}^T\mathbf{E} \in \mathbb{R}^{D\times k}$$

$$\mathbf{V}_p = \mathbf{V}^T\mathbf{F} \in \mathbb{R}^{D\times k}$$

The attention scores are then computed between $\mathbf{Q}$ and the projected $\mathbf{K}_p$:

$$\text{Scores} = \frac{\mathbf{Q}\mathbf{K}_p^T}{\sqrt{d_k}} \in \mathbb{R}^{B\times N\times k}$$

$$\text{Attention Weights} = \text{Softmax}(\text{Scores}) \in \mathbb{R}^{B\times N\times k}$$

The output is computed using the projected $\mathbf{V}_p$:

$$\text{Output} = \text{Attention Weights}\mathbf{V}_p^T \in \mathbb{R}^{B\times N\times D}$$

The complexity of computing $\mathbf{Q}\mathbf{K}_p^T$ is $O(Nkd)$. The complexity of computing the output is $O(Nkd)$. Since $k \ll N$, the total complexity is reduced to $O(Nkd)$.

In the provided code's 'MultiHeadAttention' class (when 'attention$_type$ =' $linformer$'), $theprojectionmatrices$'$self.E$ k), $whereNisthesequencelengthandkis$'$linformer_k$'.$Thekeyandvaluematricesareprojectedasshowninthef orwardpass$: $\mathbf{K}_{\text{proj}} = (\mathbf{K}^\intercal@\mathbf{E})^\intercal$

$$\mathbf{V}_{\text{proj}} = (\mathbf{V}^\intercal@\mathbf{F})^\intercal$$

and the attention is computed using $\mathbf{Q}$, $\mathbf{K}_{\text{proj}}$, and $\mathbf{V}_{\text{proj}}$.

3.1.2 Uniform Attention

The 'uniform' attention is a simple baseline where each token attends equally to all other tokens. The attention matrix is simply a uniform matrix where each element is $1/N$.

$$\text{Attention Matrix}_{ij} = \frac{1}{N}$$

This results in a complexity of $O(N^2d)$ for the matrix multiplication with V, but the computation of the attention matrix itself is trivial ($O(N^2)$ or even $O(1)$ per element). While not reducing the matrix multiplication complexity, it serves as a strong baseline to understand the contribution of learned attention weights.

3.2 Results and Comparison to Part 1

The combined code trains and evaluates the 'linformer' and 'uniform' models alongside the 'vanilla' model. The test accuracies are reported as follows:

```
vanilla Test: 98.66%
linformer Test: 98.63%
uniform Test: 92.57%
```

Comparing the results:

- The **Linformer** model (with 'linformer$_k = 16$')$achieves a test accuracy of 98.63\%, which is very close to the 98.66\% of the *$
$*Vanilla**model.This suggests that the Linformer approximation is effective in maintaining performance while reducing$
$*Uniform**attention model achieves a significantly lower test accuracy of 92.57\%.This highlights the importance of learne$

  The Linformer successfully reduces complexity while preserving performance, making it a viable alternative for larger inputs. Uniform attention, while simple, serves as a clear demonstration of the value of learned attention.

3.3 Attention Map Visualization and Inferences

The code attempts to visualize the attention map from the first attention head of the first transformer block for a sample image. The visualization part in the provided code has a slight issue where 'cls$_a ttn$'$is used without being defined, and the reshaping assumes a specific number of patches (16 for a 28 \times 28$ image with 7×7 patches). Assuming the intention is to visualize the attention weights from the CLS token to the patch tokens, we can analyze what the different attention mechanisms might show.

For an input image (e.g., an MNIST digit), patch size 7 on a 28x28 image gives $4 \times 4 = 16$ patches. Including the CLS token, the sequence length is 17. The attention map for the first head would be a 17×17 matrix (or 1×17 if focusing on CLS token attention).

- **Vanilla Attention Map:** We would expect to see varying attention weights from the CLS token to different image patches. Patches corresponding to the digit itself (the foreground) would likely have higher attention weights than background patches. The vanilla attention can capture intricate relationships between any pair of tokens.
- **Linformer Attention Map:** The visualization here would show the attention scores between the Query of the CLS token (or any token) and the projected Keys of all tokens. While the direct $N \times N$ attention map isn't computed, the resulting attention weights after softmax ($N \times k$) still influence the weighted sum of the projected Values. The visualization of the $N \times k$ matrix might be less directly interpretable in terms of spatial relationships than the $N \times N$ map. However, if we visualize the attention from the CLS token to the *original* patches after the Linformer calculation (which requires post-processing the $N \times k$ attention matrix), we might still see higher attention on relevant patches, but potentially less sharp or more global patterns due to the projection. The provided code's visualization focuses on the attention from the CLS token to the patches (index 1 onwards). For Linformer, this attention is $1 \times k$. To visualize this spatially as a 4×4 grid of patches, the $1 \times k$ vector would need to be somehow mapped or broadcast back to the 16 patch locations. The code's current approach of reshaping assumes a direct mapping after padding, which might not accurately reflect the spatial attention.
- **Uniform Attention Map:** The attention map from the CLS token to any patch would be uniform, i.e., $1/(N+1)$ for all patches. Visualizing this as a 4×4 grid would show a completely flat, uniform heat map. This confirms that the uniform attention mechanism distributes attention equally across all tokens, regardless of their content or position.

Inferences from Attention Maps (Conceptual based on mechanism):

- Vanilla attention learns to focus on the most relevant parts of the image (the digit pixels) when classifying.
- Linformer attention aims to approximate this focused attention with lower computational cost by using projections. While the exact spatial pattern might differ slightly due to the approximation, it should ideally still exhibit some level of focus on salient features if the approximation is good.

- Uniform attention does not learn to focus; it treats all patches equally. This explains its lower performance compared to learned attention mechanisms, as it cannot prioritize information from the important parts of the image.

The visualization code snippet needs correction (‘cls$_attn‘isnotdefined, likelyintendedtobe‘attn_map[0]‘), buttheprincipl

4 Part 3: Introducing Multi-Scale in Attention Heads (Bonus)

Part 3 explores incorporating multi-scale processing into the Vision Transformer. This allows the model to capture features at different granularities simultaneously.

4.1 Design and Code of Multi-Scale Encoder

The multi-scale approach is implemented by using patch embeddings of different sizes and processing these different-scale token sequences within the transformer blocks.

4.1.1 Multi-Scale Patch Embedding

The ‘MultiScalePatchEmbedding‘ module takes a list of patch sizes (e.g., $[7, 14]$). For each patch size P_i, it creates a separate convolutional layer to extract and embed patches of that size.

$$\text{For each } P_i \in \text{patch\_sizes}:$$

$$\mathbf{Y}_i = \text{Conv2d}(\mathbf{X}, \text{kernel} = P_i, \text{stride} = P_i) \in \mathbb{R}^{B \times D \times \frac{H}{P_i} \times \frac{W}{P_i}}$$

$$\mathbf{T}_i = \text{Flatten}(\mathbf{Y}_i, 2).Transpose(1, 2) \in \mathbb{R}^{B \times N_i \times D}$$

where $N_i = (\frac{H}{P_i} \times \frac{W}{P_i})$ is the number of patches for scale i. The ‘MultiScalePatchEmbedding‘ returns a list of embedded sequences: $[\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{num\_scales}]$.

4.1.2 Multi-Scale Multi-Head Attention

The ‘MultiScaleMultiHeadAttention‘ module is designed to process these multiple sequences. It assumes the total number of attention heads is distributed among the different scales. For ‘num$_scales‘scalesand‘num_heads‘totalhe
$num_heads//num_scales‘heads$.

Instead of a single QKV projection, it has separate QKV projections for each scale's sequence. Each scale's sequence $\mathbf{T}_i$ is processed independently by its allocated attention heads:

$$\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i = \text{Linear}_i(\mathbf{T}_i)$$

These are then split into ‘heads$_per_scale‘heads, andstandardself-attentionisperformedwithineachscale'sheads.$Attentio
$\text{MultiHead}_{\text{scale } i}(\mathbf{T}_i, \mathbf{T}_i, \mathbf{T}_i) \in \mathbb{R}^{B \times N_i \times D}$ The module returns a list of the output sequences from each scale's attention: $[\text{Attention}_1, \text{Attention}_2, \ldots, \text{Attention}_{num\_scales}]$.

4.1.3 Multi-Scale Transformer Block

The ‘MultiScaleTransformerBlock‘ processes the list of embedded sequences. It contains separate layer normalization and feed-forward networks for each scale. Given input sequences $[\mathbf{E}_1, \ldots, \mathbf{E}_{num\_scales}]$, the block computes:

$$\text{AttnOuts} = \text{MultiScaleMultiHeadAttention}([\mathbf{E}_1, \ldots, \mathbf{E}_{num\_scales}])$$

$$\text{For each scale } i:$$

$$\mathbf{Y}_i = \text{LayerNorm}_i(\mathbf{E}_i + \text{AttnOuts}_i)$$

$$\mathbf{Z}_i = \text{LayerNorm2}_i(\mathbf{Y}_i + \text{FFN}_i(\mathbf{Y}_i))$$

The block returns a list of the processed sequences: $[\mathbf{Z}_1, \ldots, \mathbf{Z}_{num\_scales}]$.

4.1.4 Multi-Scale Vision Transformer

The 'MultiScaleVisionTransformer' uses the 'MultiScalePatchEmbedding' to get initial sequences for each scale. It then passes this list of sequences through a stack of 'MultiScaleTransformerBlock' layers.

After the transformer blocks, the output sequences from all scales are concatenated along the sequence dimension.

$$\mathbf{E}_{\text{final}} = \text{Concat}(\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_{num\_scales}) \in \mathbb{R}^{B \times (\sum N_i) \times D}$$

A CLS token is prepended to this concatenated sequence, and positional embeddings are added.

$$\mathbf{T}_{\text{cat}} = \text{Concat}(\mathbf{T}_{\text{cls}}, \mathbf{E}_{\text{final}}) \in \mathbb{R}^{B \times (1+\sum N_i) \times D}$$

$$\mathbf{E}_{\text{final\_pos}} = \mathbf{T}_{\text{cat}} + \mathbf{P}_{\text{pos}}$$

This sequence is then passed through a final layer normalization, and the CLS token embedding is used for the classification head.

The design processes different scales independently within the attention and FFN layers of each block. Information is aggregated only at the end by concatenating the features from all scales before the final classification layer.

4.2 Performance on Test Set (Part 3 - Multi-Scale ViT)

The combined code trains and evaluates the 'multiscale' model. The test accuracy is reported as follows:

```
multiscale Test: 98.71%
```

Comparing this to the vanilla ViT (98.66%), the multi-scale model achieves a slightly higher test accuracy on MNIST. While the improvement is marginal on this relatively simple dataset, the multi-scale approach demonstrates the potential to capture information at different resolutions, which is often beneficial for more complex vision tasks.

5 Conclusion

This assignment successfully demonstrated the implementation of various Vision Transformer architectures from scratch (using basic PyTorch modules).

- **Part 1:** A basic Vision Transformer was built and achieved high accuracy on MNIST (98.66%), validating the core ViT approach.
- **Part 2:** Computational efficiency was addressed by implementing Linformer and Uniform attention. Linformer maintained comparable performance (98.63%) while offering reduced complexity, showing its effectiveness as an approximation. Uniform attention (92.57%) served as a baseline, highlighting the importance of learned attention patterns.
- **Part 3:** A multi-scale architecture was designed and implemented by processing patches of different sizes concurrently. This model achieved a slightly better test accuracy (98.71%) than the vanilla ViT on MNIST, indicating the potential benefits of multi-scale feature processing in vision transformers.

The code provided a clear framework for defining and comparing these models. The training and evaluation loop allowed for direct comparison of their performance metrics. The conceptual analysis of attention maps provided insight into how different mechanisms handle token interactions. While the visualization code had a minor issue, the principles of interpreting attention maps for different models remain valid.

The assignment provided valuable hands-on experience in building key components of Vision Transformers and exploring variations for efficiency and feature representation.