Documentation Notes

Instadeep nucleotide transformer classifier

In this project, I use the **InstaDeep Nucleotide Transformer**, a family of large transformer models pretrained on genomic sequences, to extract context-aware representations of DNA. These embeddings are then used to train a neural network classifier that distinguishes **promoter** regions (DNA sequences where gene transcription is initiated) from **non-promoter**sequences.

Rather than training a model directly on raw DNA tokens, this approach leverages pretrained biological knowledge encoded in the transformer embeddings.

Why we need Instadeep model?

The InstaDeep Nucleotide Transformer is pretrained on large-scale genomic data and learns biologically meaningful sequence patterns such as motifs, dependencies between distant bases, and regulatory structure.

In this project, I use the pretrained model `50M_multi_species_v2` to transform raw DNA sequences into dense vector embeddings. These embeddings serve as input features for a downstream classifier.

Using pretrained embeddings is beneficial because:

- They encode biologically relevant patterns.
- They capture long-range dependencies in DNA.
- They allow efficient training of a lightweight classifier on limited labeled data.

Training details

The following configuration is used for the nucleotide transformer:

```
model_name = '50M_multi_species_v2'
```

```
parameters, forward_fn, tokenizer, config = get_pretrained_model(
model_name=model_name,
...
embeddings_layers_to_save=(12,),
attention_maps_to_save=((1, 4), (7, 16)),
max_positions=512,
)
```

- **Layer 12 embeddings** are extracted, as later transformer layers typically encode higher-level, task-relevant representations.
- **Attention maps** are saved from selected layers and heads to allow interpretability analysis of sequence interactions.
- The maximum sequence length is set to 512 nucleotides.

The classifier is implemented as a **multilayer perceptron (MLP)** operating on the `[CLS]` token embedding produced by the transformer.

Architecture:

- Linear layer with hidden dimension **128**
- ReLU activation
- Linear layer with hidden dimension **64**
- ReLU activation
- Final linear layer with **1 output logit**

Training details:

- Batch size: **32**
- Number of epochs: **6**
- Optimizer: Optax (Adam)
- Loss function: **Binary Cross-Entropy (BCE)**

The BCE loss is defined as:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

Note that binary classification models outputs the label as a **Bernoulli random variable**. BCE corresponds to the negative log-likelihood of the Bernoulli distribution, making it the correct loss function for probabilistic binary classification.

Early stopping is used based on validation loss to prevent overfitting. The best-performing parameters and embeddings are saved for evaluation.

Model evaluation

Training and validation loss

The training and validation loss curves show stable convergence within six epochs, indicating effective learning without severe overfitting (see `/results/loss_curves.png`).

ROC curve

The **Receiver Operating Characteristic (ROC) curve** plots:

- **True Positive Rate (Recall)** on the y-axis
- **False Positive Rate** on the x-axis

The ROC curve illustrates how the classifier's sensitivity changes as the decision threshold varies.

A curve that bends toward the top-left corner indicates strong discriminative ability, whereas a diagonal line corresponds to random guessing.

The observed ROC curve rising well above the diagonal demonstrates that the classifier effectively separates promoter and non-promoter sequences across thresholds.

Precision–Recall curve

The **precision–recall (PR) curve** plots:

- **Precision** on the y-axis
- **Recall** on the x-axis

As recall increases, precision typically decreases because the classifier becomes more permissive.

The PR curve is especially informative for **imbalanced datasets**, as it focuses on performance for the positive class.

The downward trend observed in the PR curve reflects the expected tradeoff between precision and recall and indicates a reasonable balance between identifying promoters and avoiding false positives.

Confusion matrix

The confusion matrix is:

```
[[60  5]
 [17 68]]
```

This corresponds to:

- **60 true non-promoters correctly classified**
- **68 true promoters correctly classified**
- **5 false positives**
- **17 false negatives**

The relatively low number of false positives suggests high precision for promoter detection, while the remaining false negatives indicate some missed promoters, reflecting the recall–precision tradeoff.

Attention map analysis

Attention maps visualize how the transformer distributes attention between positions in the DNA sequence.

In this project, attention maps are plotted for **layer 1, head 4** for a representative promoter sequence.

- Each cell `(i, j)` represents how strongly position `i` attends to position `j`
- Diagonal dominance indicates local interactions
- Off-diagonal structure suggests long-range dependencies

Early layers showed relatively uniform attention, while later layers exhibited vertical attention patterns, indicating that multiple positions attend to a shared reference position. This behavior is consistent with global information aggregation in deep transformer layers. While attention maps provide insight into information flow, they were used here qualitatively rather than as definitive biological explanations.

Conclusion

This project demonstrates how pretrained genomic transformers can be combined with lightweight classifiers to solve biological sequence classification tasks. By leveraging learned DNA representations, the model achieves strong performance while maintaining interpretability through attention analysis.