Project 3: Transformers

SW08 – Jannine Meier – FS24

Preprocessing: Winogrande dataset

"I moved the couch from the garage to the backyard to create space. The _ is small."

1. Replace the blank with each option and create two sequences

"… to create space. The couch is small."

"… to create space. The garage is small."

Stemming and Lemmatizing

2. Tokenize both sequences using BertTokenizer

-> lowercase, add special tokens, add padding to max_length (=45)

Remove punctuation and stopwords

" [CLS] i moved … to create space . the couch is small . [SEP] [PAD] [PAD]"

" [CLS] i moved … to create space . the garage is small . [SEP] [PAD] [PAD]"

3. Return for each sequence an input_id, attention_mask and label

-> text token to input_id conversion, attention_mask generation, label remapping to right option

Input_id_1: [101, 298, … 902, 4, 55, 102, 0, 0]
Attention_mask_1: [1, 1, … 1, 1, 1, 1, 0, 0]
Label: 0 (false option)

Input_id_2: [101, 298, … 147, 4, 55, 102, 0, 0]
Attention_mask_2: [1, 1, … 1, 1, 1, 1, 0, 0]
Label: 1 (true option)

Preprocessing: Anagram datasets

1. Split the sequence between the <sep>

b p k <sep> k p b

b p k

k p b

2. Tokenize both sequences at character level to IDs (no spaces)

[2, 3, 4]

exclude spaces for tokenization
ID 0 reserved for [PAD]
ID 1 reserved for <sep>

[4, 3, 2]

3. Return concatenated and padded input_id, attention_mask and label

-> concatenate sequences using <sep>, add padding to max_length (=25)

Input: [b, k, p, <sep>, k, p, b, [PAD], [PAD], [PAD]]
Input_id: [2, 3, 4, 1, 4, 3, 2, 0, 0, 0]
Attention_mask: [1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
Label: 1 (true)

Network Architecture: Transformer

6-Layer vanilla transformer econder

Positional Encoding

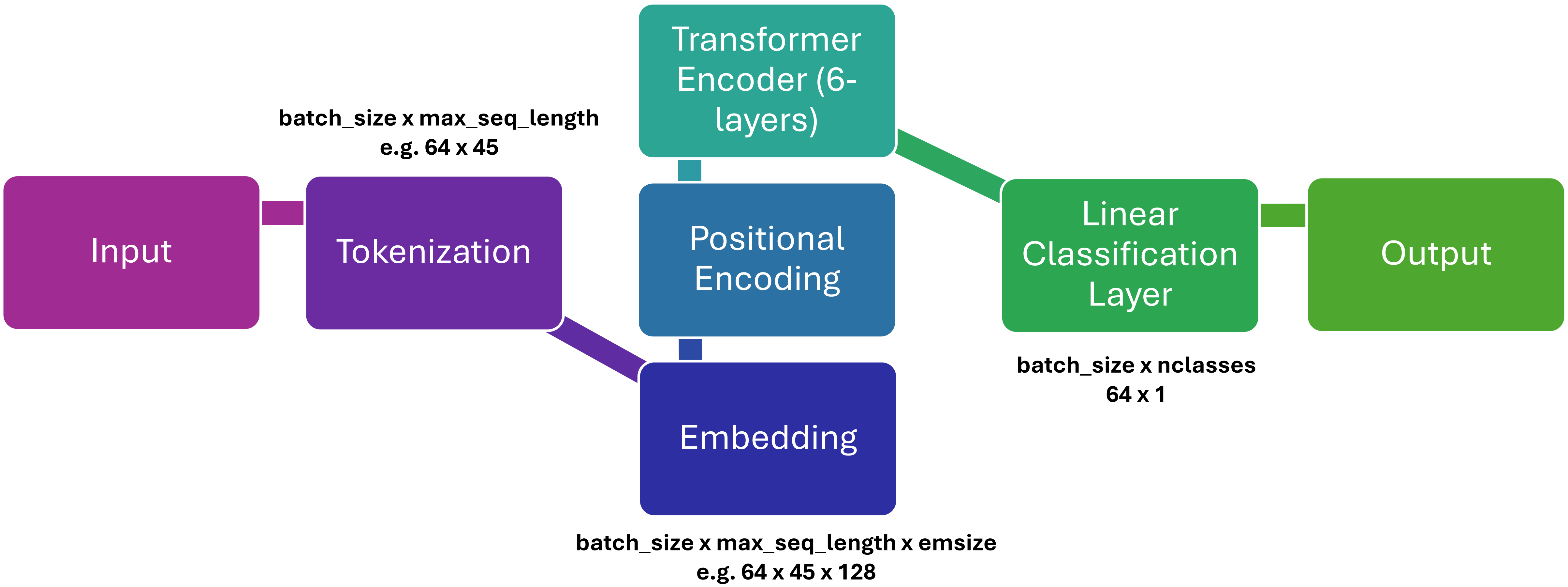
- add **positional encoding** to incorporate the concept of order
- include a **dropout layer** after adding the positional encodings to further enhance the model's generalization capabilities

TransformerClassifier

- use **nn.Embedding** for vectorization
- use Pytorchs **nn.TransformerEncoderLayer**
- use **BCEWithLogitsLoss** (already applies the sigmoid internally - no classifier activation function needed)
- use **Adam** optimizer
- initialize **weights** of the embedding and classifier layers with uniform distribution
- multiplying the embeddings by math.sqrt(emsize) helps in **balancing the magnitude of embeddings**

Network Architecture: Transformer

6-Layer vanilla transformer econder



Sweep Configurations

method: bayes

Parameters

- **learning_rate:** 1e-5, 1e-4, 1e-3, 1e-2 -> tried 'min': 1e-5, 'max': 1e-2
- batch_size: 64
- num_epochs: 100
- dropout: 0.1
- **dimension_size:** 128, 256, 512 -> controls nhid and emsize with a single parameter
- nhead: 2
- nlayers: 6
- nclasses: 1-> for binary classification with BCEWithLogitsLoss
- max_seq_length: 45 for winogrande, 25 for anagram
- vocab_size: 30’522 for winogrande, 30 for anagram

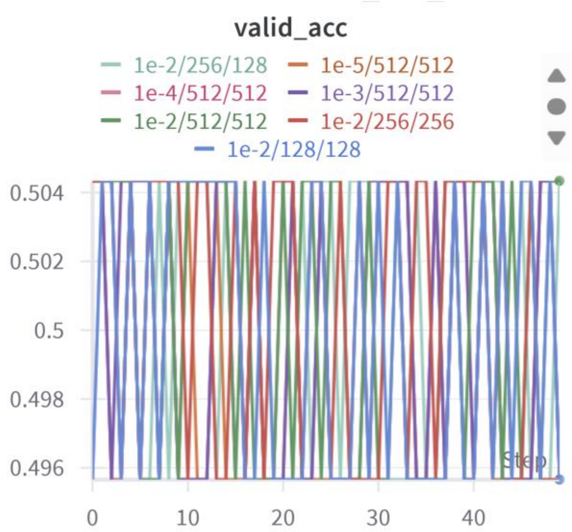
Results: Winogrande

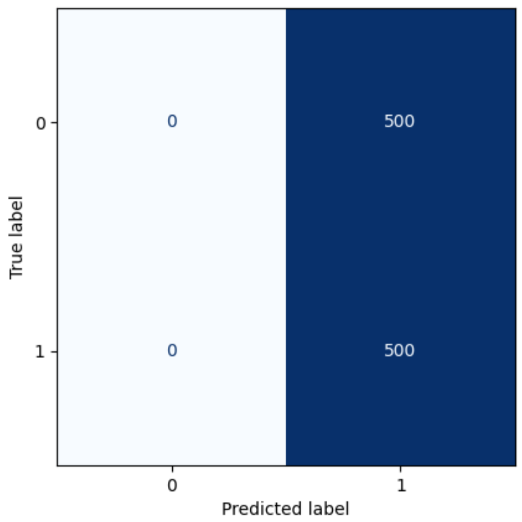
Best model

- Validation Accuracy: 0.5095%
- Test Accuracy: 50.00%

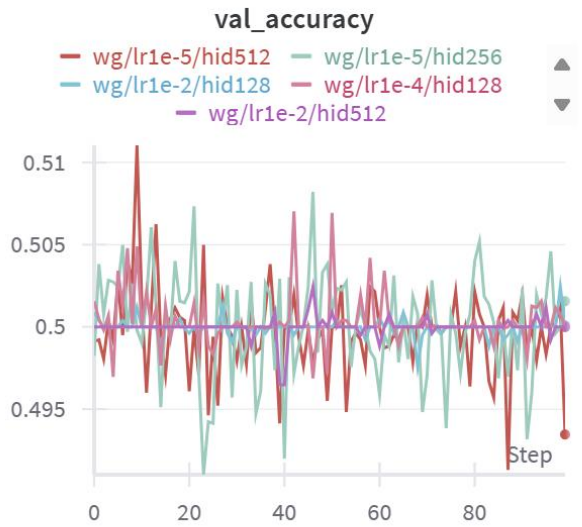
- Small increases/decreases in accuracy
- No longer predicting only one label

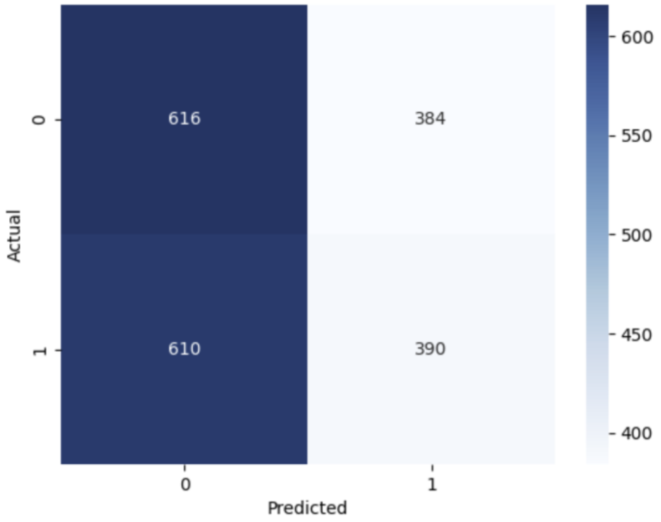
Project 2 RNNs





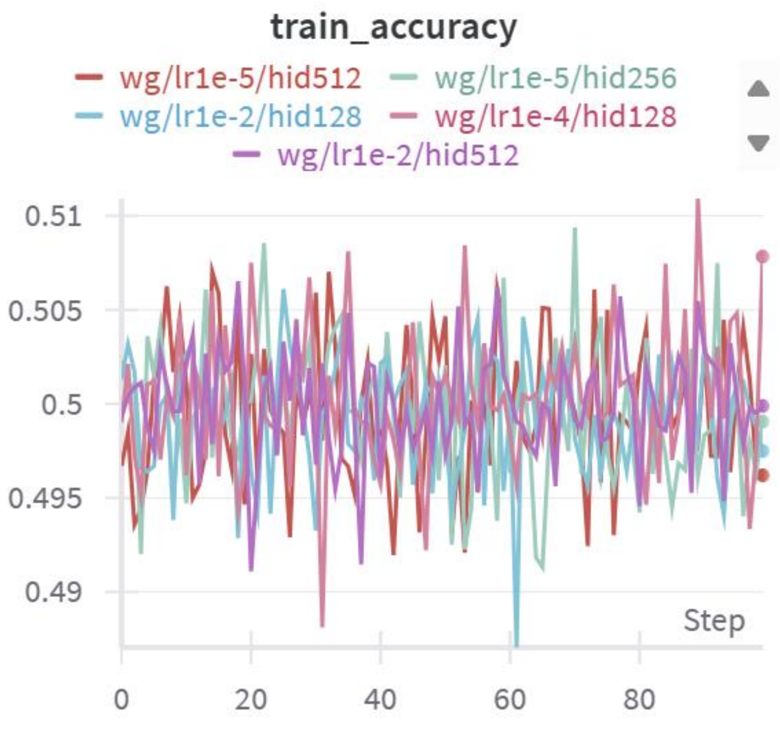
Project 3 Transformers

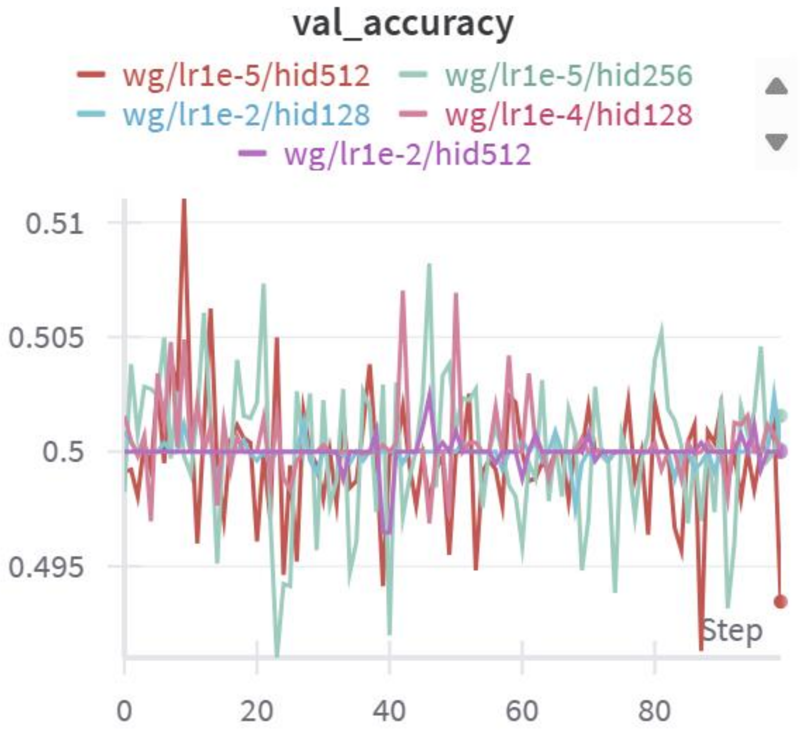


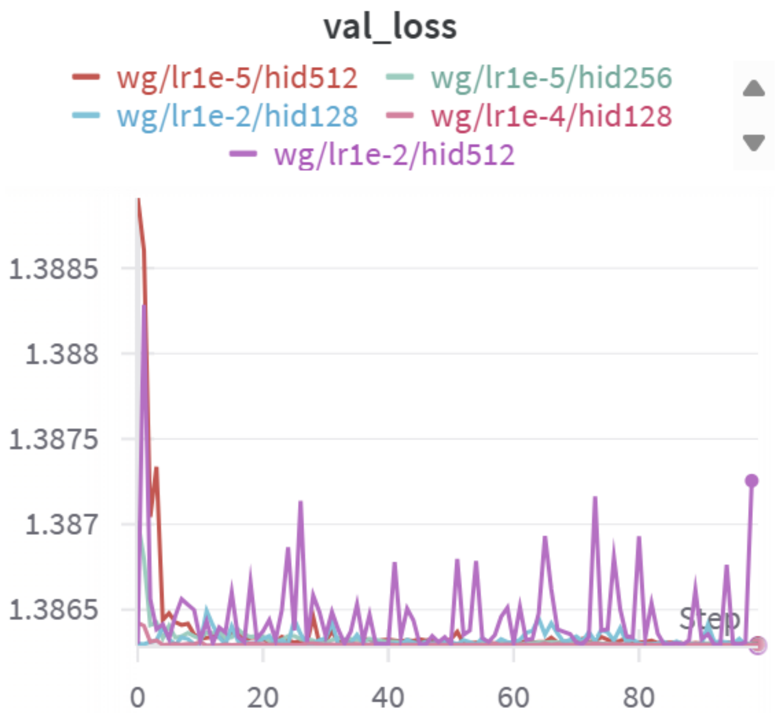


Results: Winogrande

No learning progress in train and validation!







Results: Anagram_small

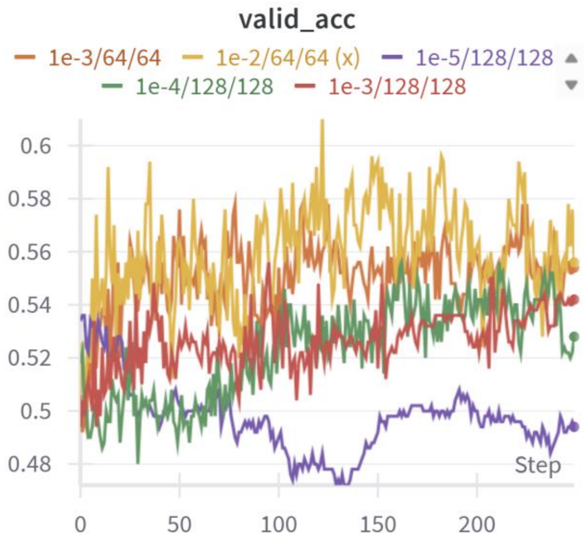
Best model

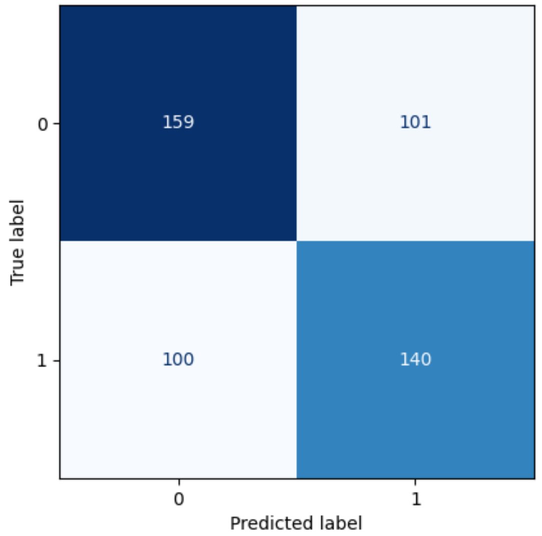
- Validation accuracy: 0.5643
- Test accuracy: 0.5360

Confusion-matrix

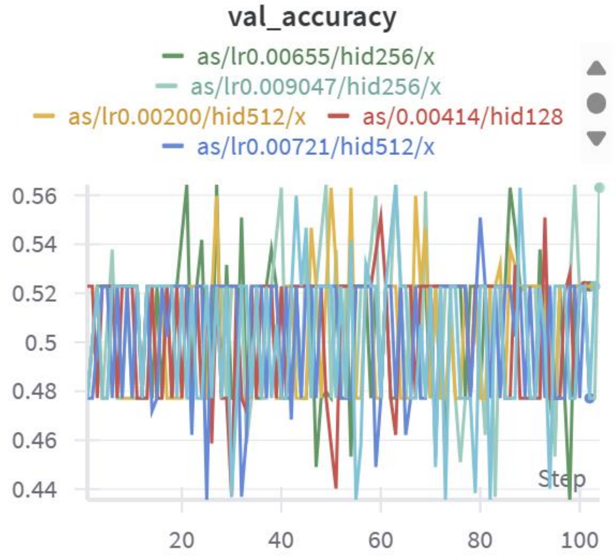
- Never predicts an anagram as non-anagram
- Predicts almost everything as anagram
- Performs worse than project 2

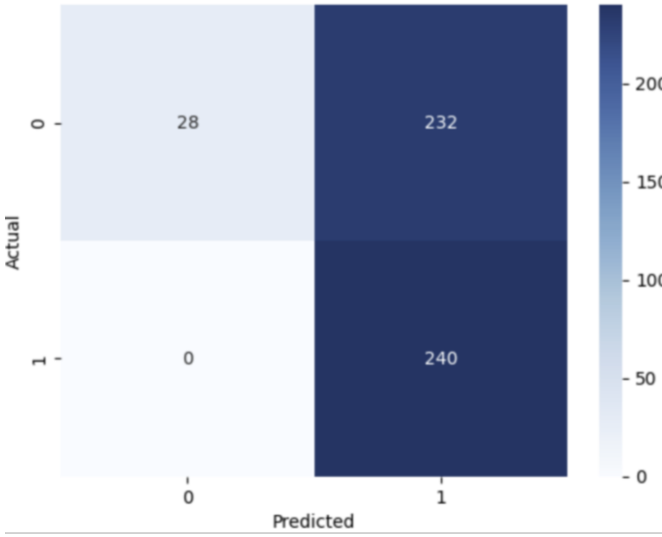
Project 2 RNNs





Project 3 Transformers

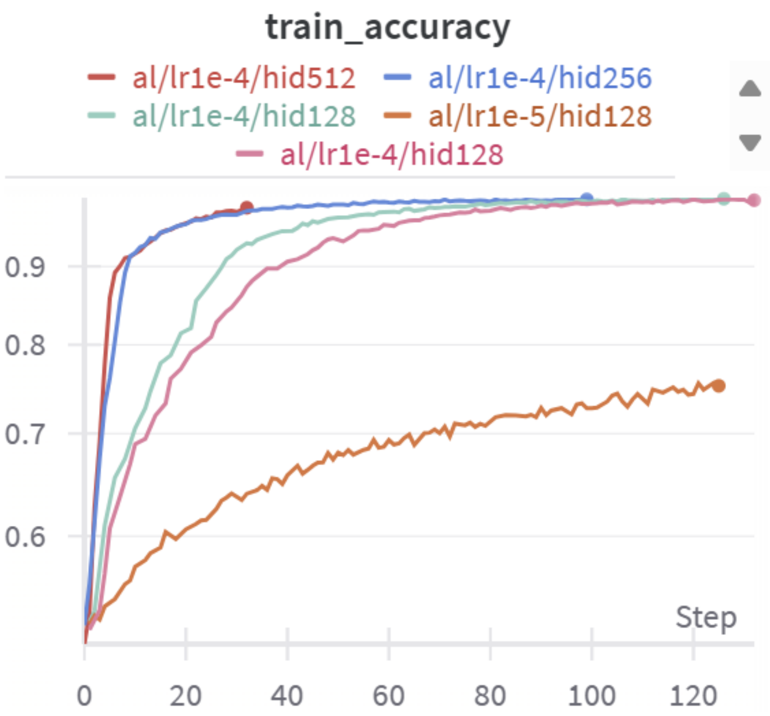


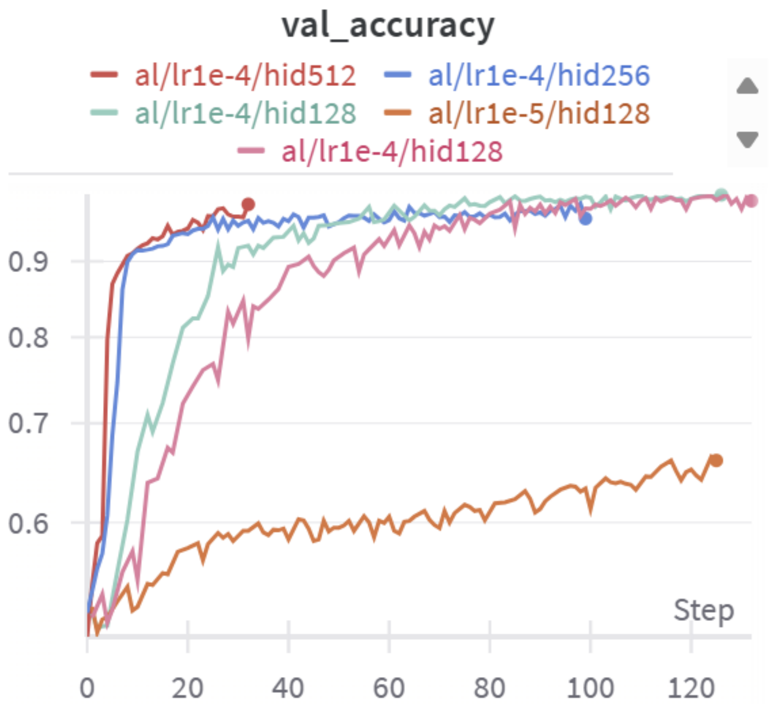


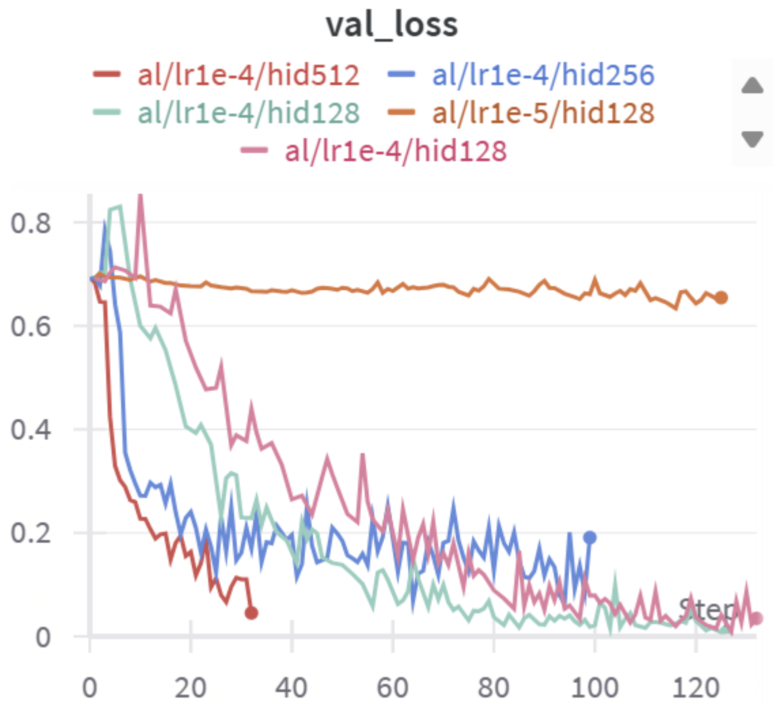
Results: Anagram_large

Best model

- Validation accuracy: 0.9999
- Test accuracy: 0.9720







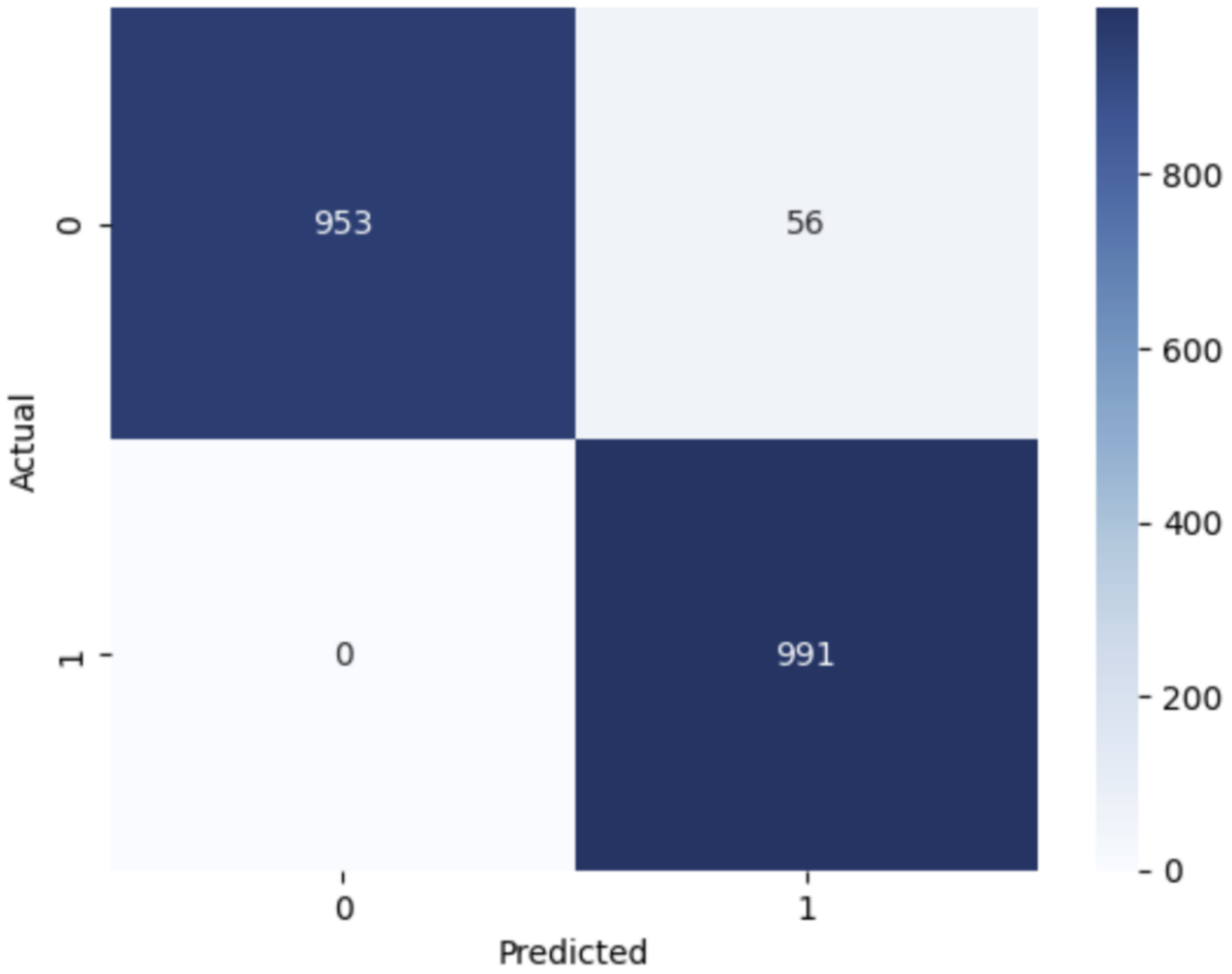
Results: Anagram_large

Best model

- Almost perfect performance
- Again: Never predicts an anagram as non-anagram
- Rarley predicts a non-anagram as anagram

Configuration

- Small learning rates (1e-5 and 1e-4) showed best results



Conclusions

Interpretation

Winogrande dataset

- task is still too complex for our architechture

Anagram small dataset

- not enough data to learn from
- RNN performance better than Transformer performance

Anagram large dataset

- enough data to learn from
- almost perfect performance when using small learning rates

Lessons Learned

Long run time

Winogrande takes about 50 minutes per run

➔optimize code to get faster runs

Exploding gradients

➔occur when the gradients during backpropagation become too large, leading to numerical instability and wildly oscillating training loss because the weight updates are disproportionately large.

➔use gradient clipping and a smaller number of attention heads

Train data size has huge impact on performance

➔see difference in anagram performance