# Introduction

In this repository sparsity in the feed-forward architecture was tested. The method was implemented in an encoder based transformer model for NLP and a simple neural network for Mnist problem. Experiments were made to compare sparse architecture to the baseline feed-forward on speed and accuracy. The sparse feed forward component was a variation of that presented in the article: Jaszczur, S., et al. (2021). Sparse is Enough in Scaling Transformers.

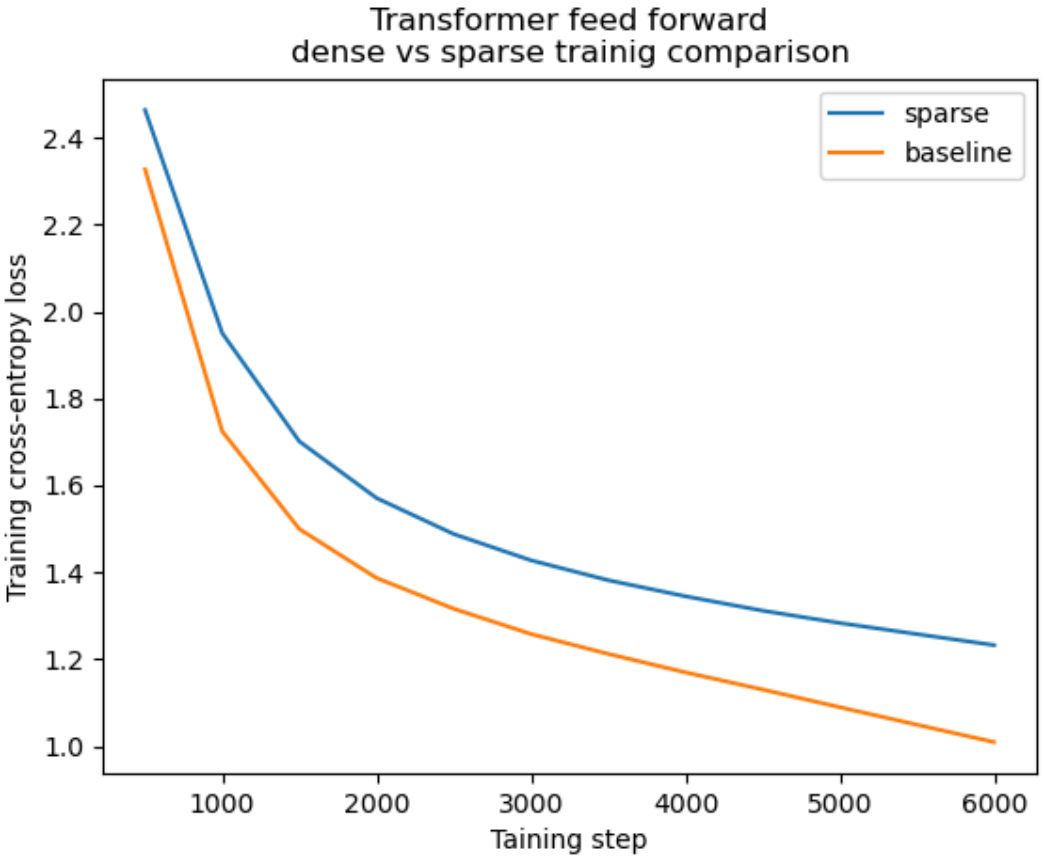
## Encoder based transformer

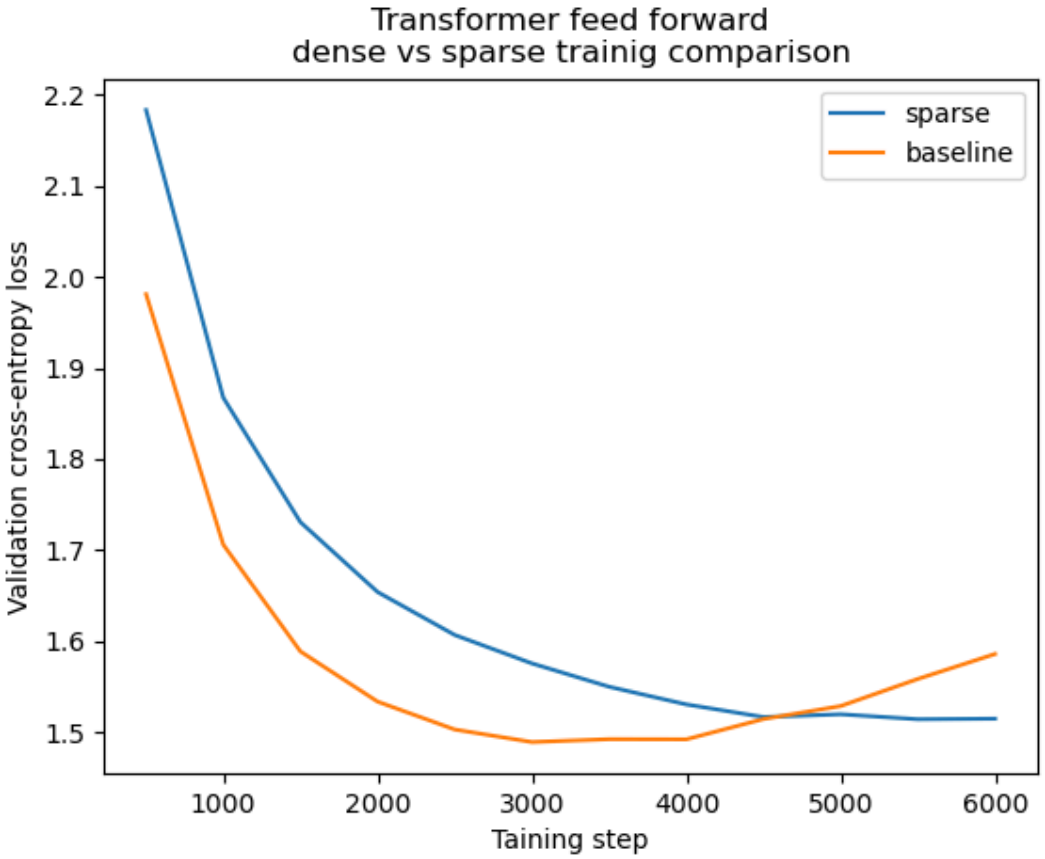
Transformer model parameters: d_model=384, n_heads=6, n_layers=6, context=256, tr_batch_size=64, lr=3e-4, dropout=0.2. During experiments with transformers in sparse models 1/64 of all parameters were used in feed forward layers per token inference (equals to 24 ReLu connections selection out of 1536 in FF[d_model x d_model*4 - ReLu - d_model*4 x d_model]).

Problem: Tiny Shakespeare text generation

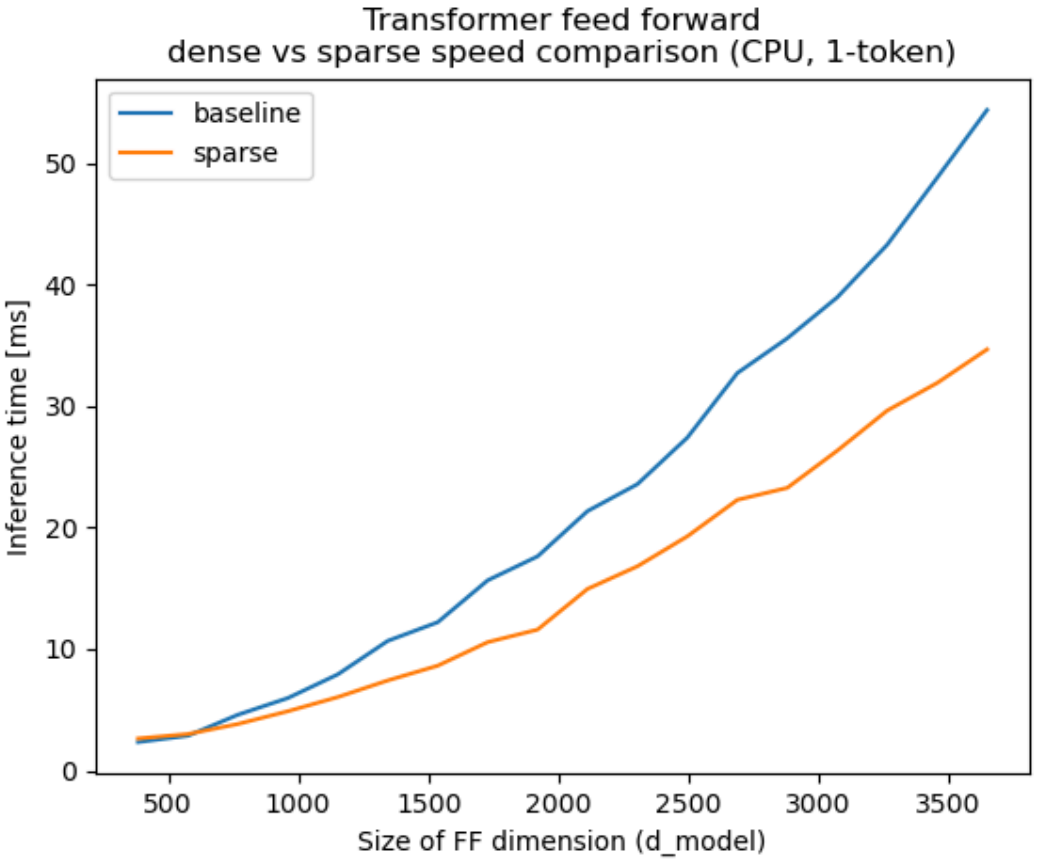
### Training

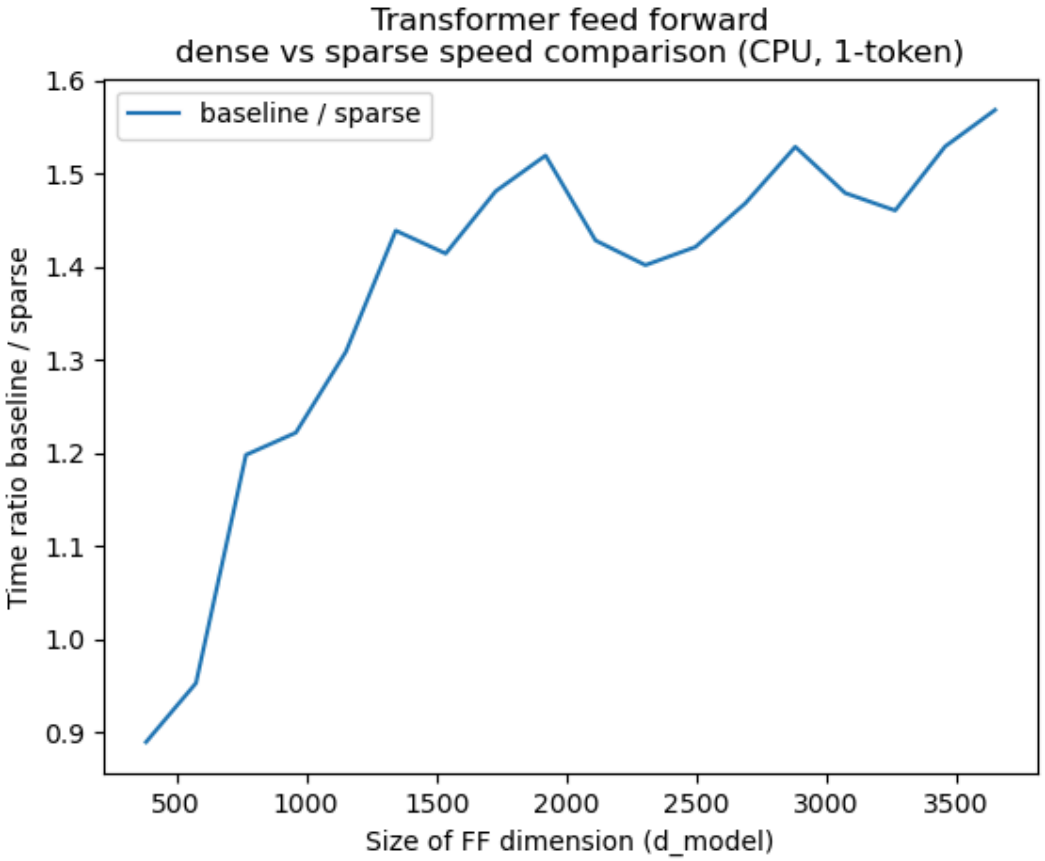
Baseline model training time (6K steps): 21.35 min
Sparse model training time (6K steps): 25.28 min

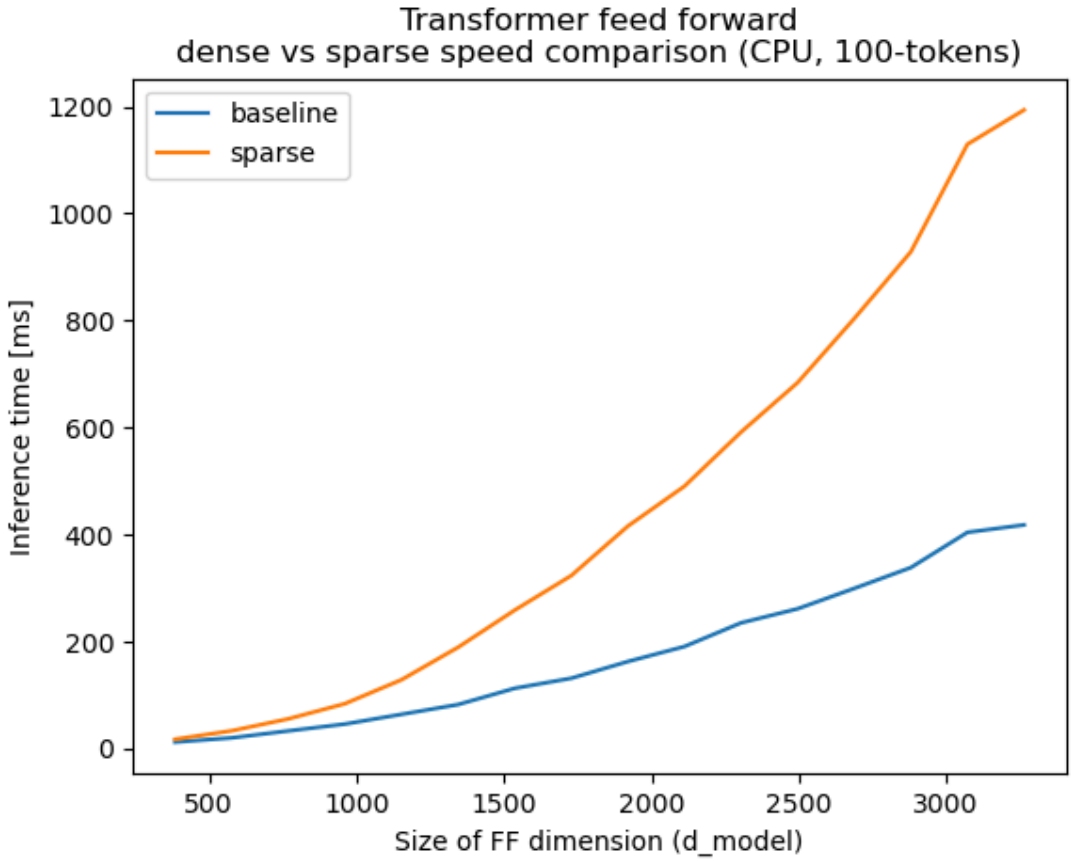

Transformer feed forward
dense vs sparse trainig comparison
sparse
baseline
Training cross-entropy loss
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
1000
2000
3000
4000
5000
6000
Taining step

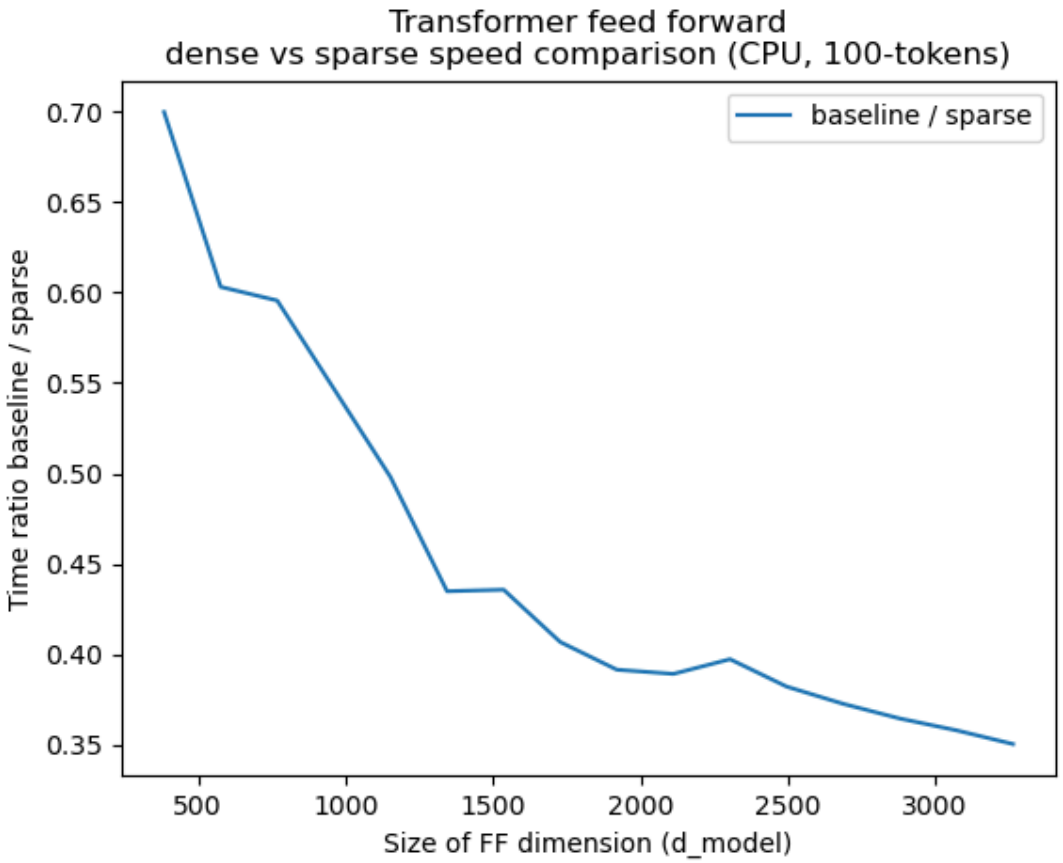

Transformer feed forward
dense vs sparse trainig comparison
sparse
baseline
Validation cross-entropy loss
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1000
2000
3000
4000
5000
6000
Taining step

Inference


Transformer feed forward
dense vs sparse speed comparison (CPU, 1-token)
baseline
sparse
Inference time [ms]
0
10
20
30
40
50
500
1000
1500
2000
2500
3000
3500
Size of FF dimension (d_model)


Transformer feed forward
dense vs sparse speed comparison (CPU, 1-token)
baseline / sparse
Time ratio baseline / sparse
Size of FF dimension (d_model)
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
500
1000
1500
2000
2500
3000
3500


Transformer feed forward
dense vs sparse speed comparison (CPU, 100-tokens)
baseline
sparse
Inference time [ms]
1200
1000
800
600
400
200
0
500
1000
1500
2000
2500
3000
Size of FF dimension (d_model)




Summary

Sparse transformer model is 10% bigger than the baseline (11M vs 10M params). Training of sparse models is 15.5% slower and it takes significantly more steps to train it properly than baseline.

During inference when loading only 1/64 FF parameters per token, the sparse model is 40% faster when processing a batch of 1 token, and much slower when it needs to process many tokens or batches concurrently. Solutions to that could be to process models input tokens with all parameters loaded, implement caching of previously computed tokens and infer next tokens loading 1/64 FF parameters. It would speed-up generation of new tokens by 40% and does not slow down computation of input tokens.

# Experiments:

## Simple NN

Architecture of the model:

- linear(784, d_model)
- feed-forward or sparse FF (d_model, d_model*4)
- linear(d_model, 10)

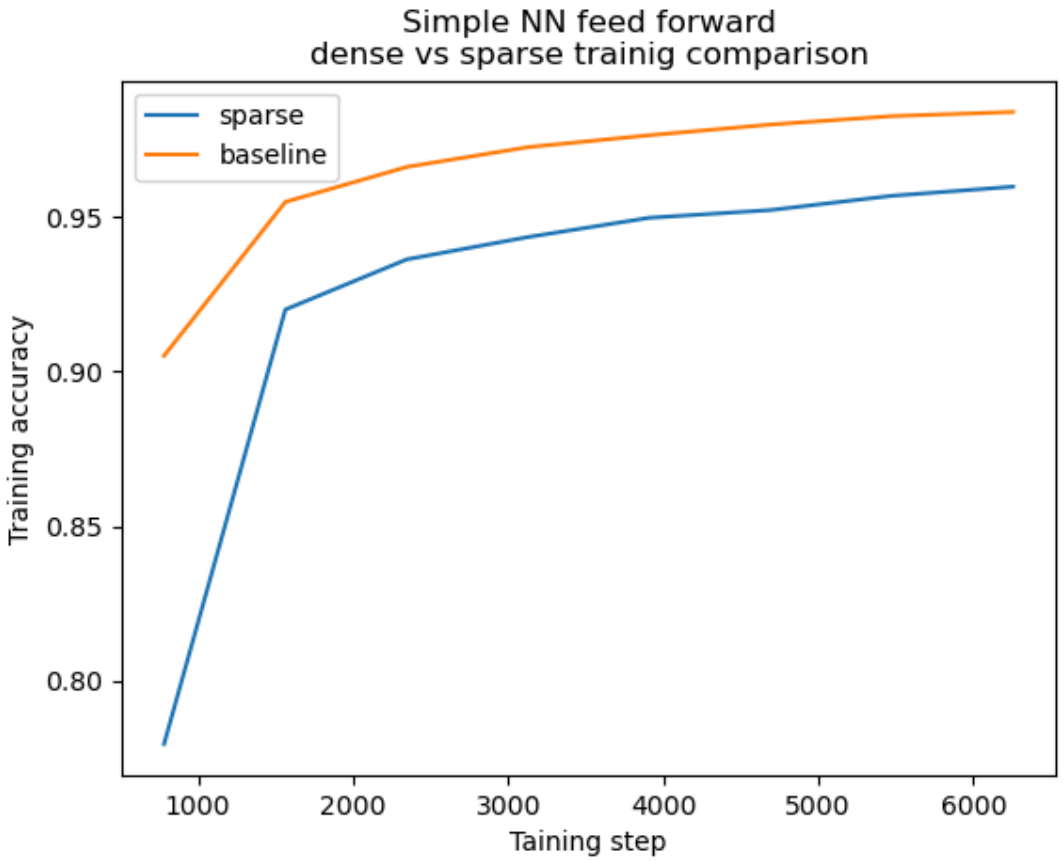
Problem:

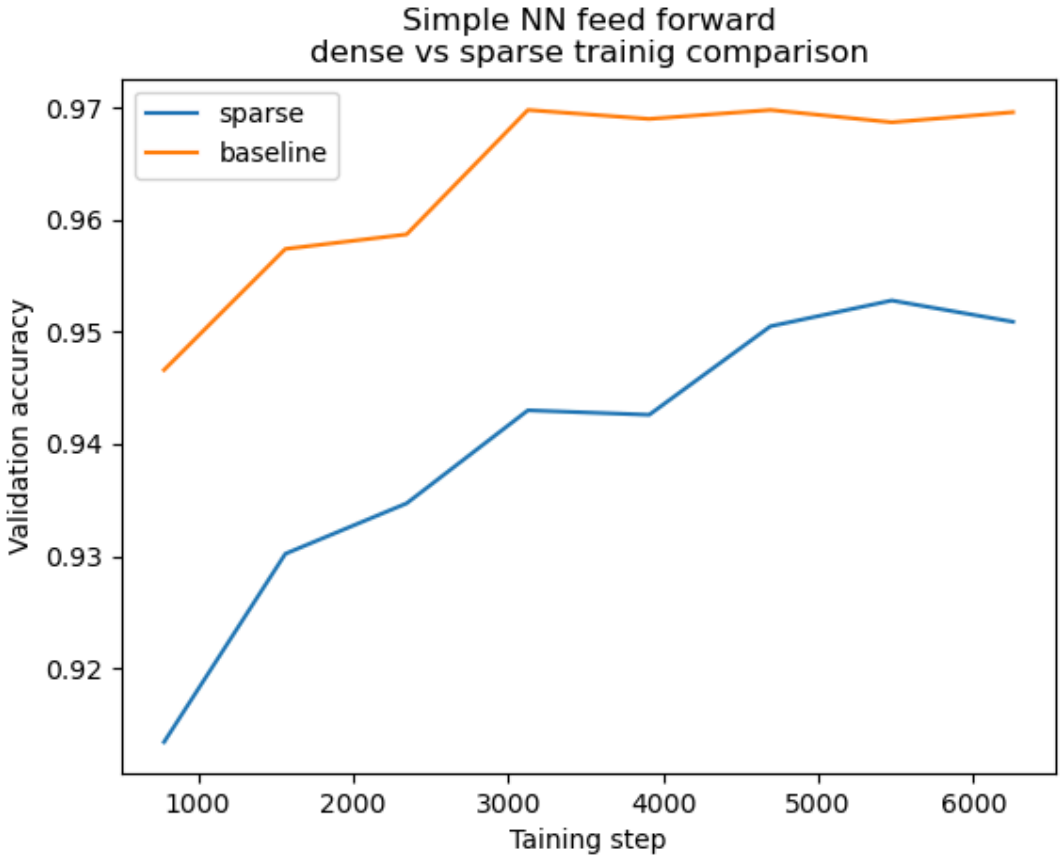
- Mnist images classification

During training (accuracy validation) d_model of 100 was used

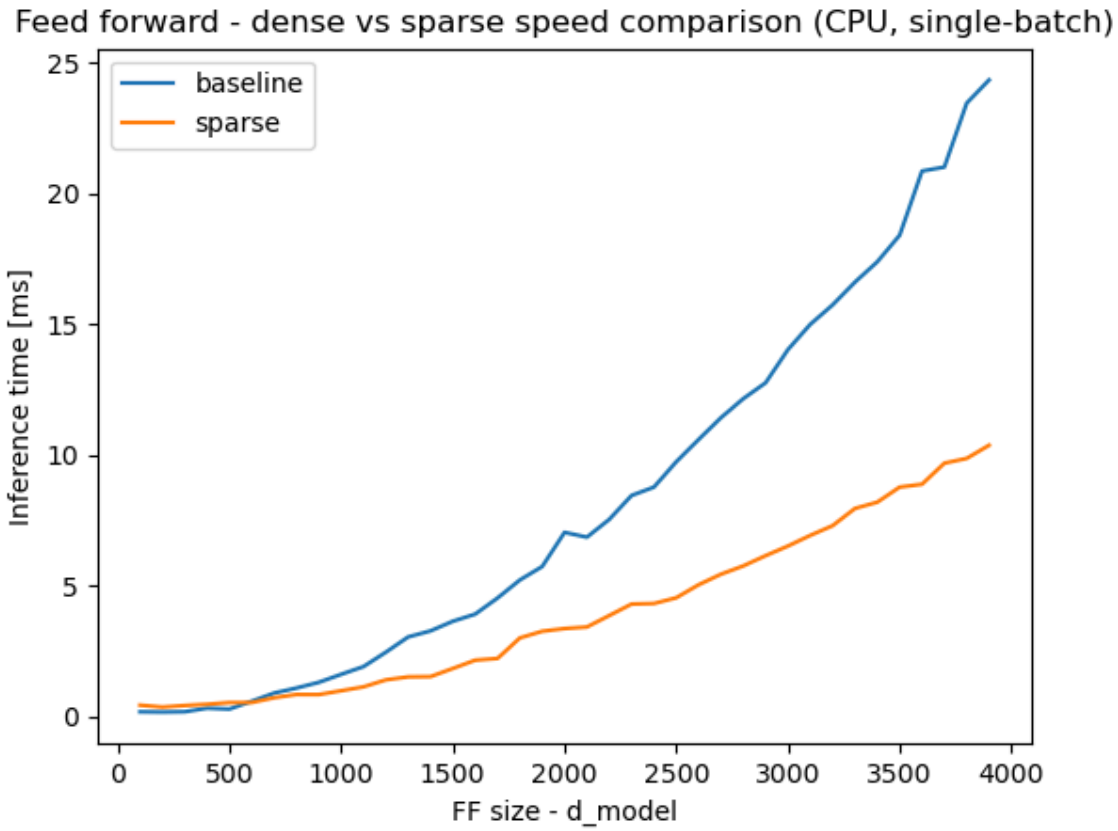
Sparse model uses and trains 1/50 FF layer parameters per sample/image.

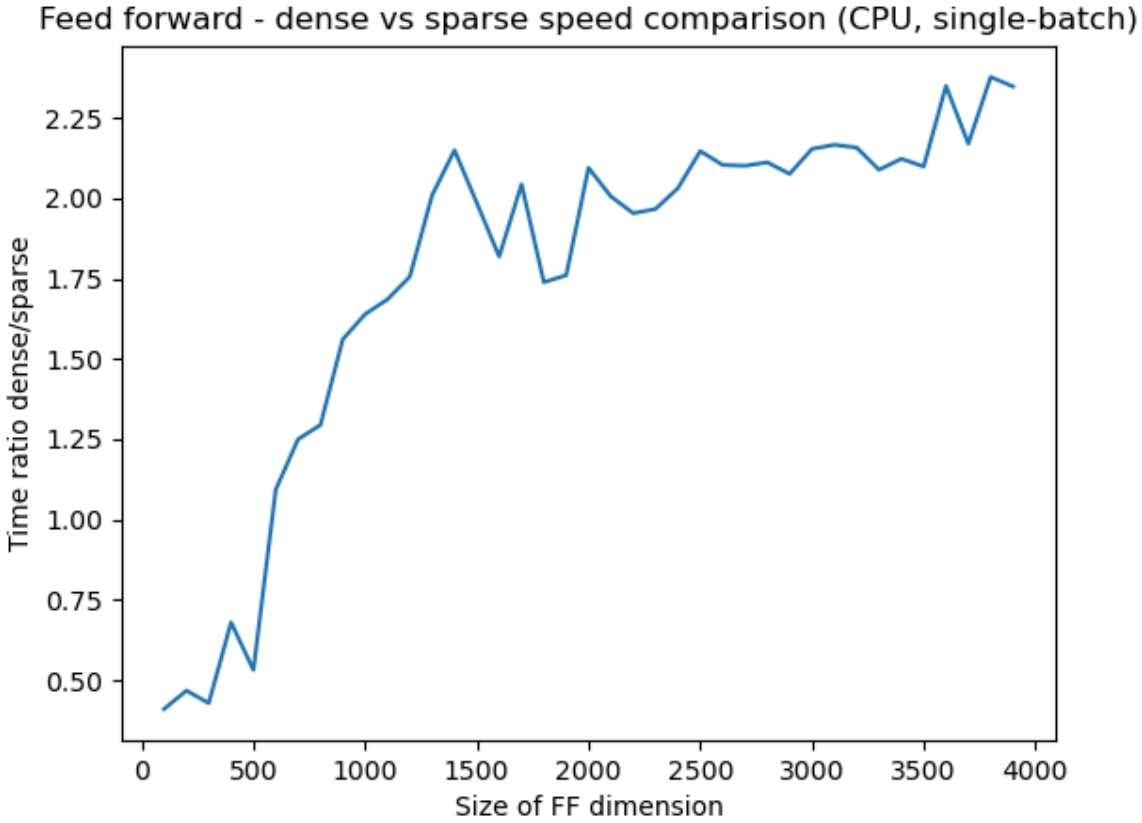
## Training




Simple NN feed forward
dense vs sparse trainig comparison
sparse
baseline
Validation accuracy
0.97
0.96
0.95
0.94
0.93
0.92
1000
2000
3000
4000
5000
6000
Taining step

Inference


Feed forward - dense vs sparse speed comparison (CPU, single-batch)
baseline
sparse
Inference time [ms]
FF size - d_model
0
5
10
15
20
25
0
500
1000
1500
2000
2500
3000
3500
4000


Feed forward - dense vs sparse speed comparison (CPU, single-batch)
Time ratio dense/sparse
Size of FF dimension
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
0
500
1000
1500
2000
2500
3000
3500
4000

Summary

Using 1/50 FF params speedup of 100% was achieved.

# Conclusion

Sparsity in feed forward layers is a promising method. It could find usage in specific situations, where longer training and slightly less accuracy can be traded for a faster inference. Same sparse FF can be 2x faster than baseline FF during inference.

For a beneficial usage of this method in a transformer architecture, additional implementation of caching processed tokens and distinctive processes for input and generated token processing needs to be made. Sparse FF in the transformer can speed up its generated token computing by 40%.

Quality and speed of o model is highly dependable on sparsity percentage. 2% sparsity provides presented speedups but when tested 4% a speedup was not observed. Sparsity of 10% has a similar quality to the baseline, but is significantly slower.