학번: 2024193011

이름: 신재완

2025.12.1

문제 (1)

CNN을 먼저 살펴보자.

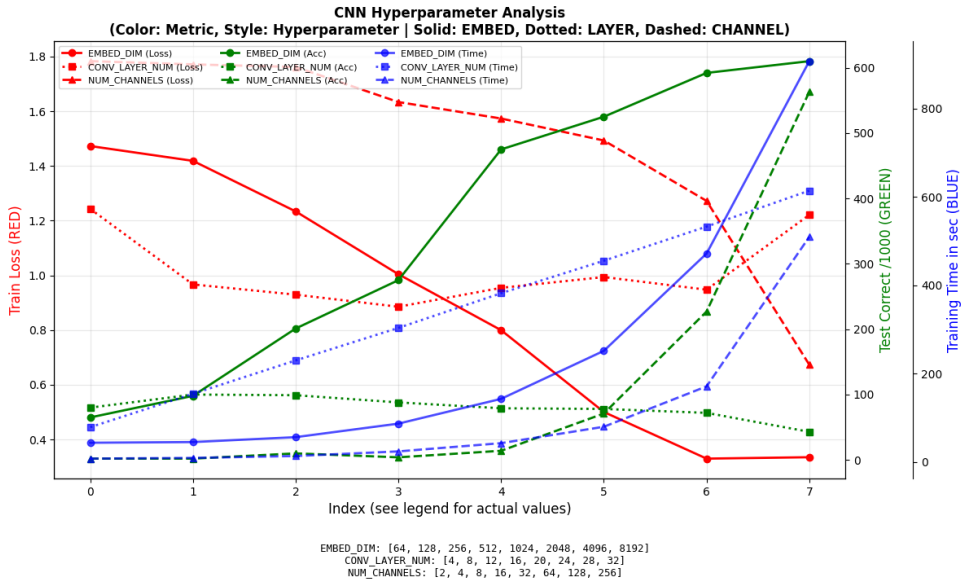

Figure 1: CNN Hyperparameter 변화에 따른 성능 비교

```
# Default values
DEFAULT_EMBED_DIM      = 16
DEFAULT_CONV_LAYER_NUM = 2
DEFAULT_NUM_CHANNELS   = 32
```

기본값은 위와 같이 고정해두고

```
embed_dims        = [64, 128, 256, 512, 1024, 2048, 4096, 8192]
conv_layers       = [4,   8,   12,  16,  20,   24,   28,   32  ]
num_channels_list = [2,   4,   8,   16,  32,   64,   128,  256 ]
```

위와 같이 각 값들을 변화시키면서 loss(빨간색), test accuracy(초록색), training time (파란색)으로 선을 그렸다. embed_dim 변화가 실선, conv_layer 변화는 점선, num_channel 변화는 파선이다.

점선을 보면 알 수 있듯 convolution layer를 늘려도 성능에 큰 변화가 없거나 오히려 성능이 떨어지는 것을 알 수 있다. 한편 layer 수가 늘어남에 따라 training 시간은 linear하게 늘어나는 것을 볼 수 있다.

두 번째로 성능에 큰 영향을 준 것은 파선인 channel의 수이다. 값이 클 때 loss가 빠르게 떨어지고 accuracy가 빠르게 오르는 것을 볼 수 있다. 2배씩 키우면서 본 training time은 거의 2배씩 느는 것을 보아 linear하며 같은 값과 비교했을 때 embedding dim보다 compute source를 많이 필요로 함을 알 수 있다.

가장 성능에 큰 영향을 준 것은 실선인 embedding dim이다. channel 수와 유사한 특징을 가지고 있지만 channel 수보다 더 좋은 성능을 보이고 있다.

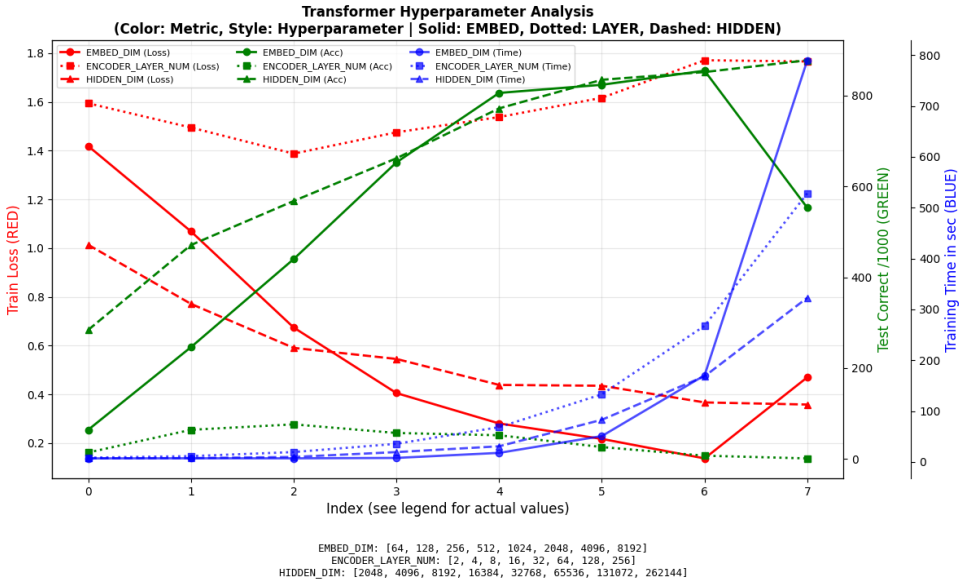

Figure 2: Transformer Hyperparameter 변화에 따른 성능 비교

Transformer는 위와 같이 나왔다.

```
# Default values
DEFAULT_EMBED_DIM          = 32
DEFAULT_ENCODER_LAYER_NUM  = 2
DEFAULT_HIDDEN_DIM         = 64
```

기본값은 위와 같이 고정해두고

```
embed_dims     = [64,   128,  256,  512,   1024,  2048,  4096,   8192  ]
encoder_layers = [2,    4,    8,    16,    32,    64,    128,    256   ]
hidden_dims    = [2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144]
```

위와 같이 hyperparameter를 변화시켰다.

Figure 2를 보면 CNN과 꽤나 유사한 형태를 띠는 것을 알 수 있다. conv layer 수와 마찬가지로 점선인 encoder layer 수같은 경우에는 아무리 늘려도 성능 향상이 나타나지 않거나 오히려 성능이 낮아지는 것을 알 수 있지만 늘릴 수록 연산 시간은 linear하게 늘어났다.

두 번째로 파선인 hidden dim은 성능에 꽤나 좋은 영향을 주는데, test accuracy가 최대 약 90% 가까이 나오는 것을 볼 수 있다.

실선인 embed dim은 CNN과 마찬가지로 성능에 긍정적인 영향을 주는 것을 알 수 있다. 한편 초기에 이 값을 크게 늘렸을 때는 loss가 매우 커지는 gradient explosion 현상이 발생하여 gradient norm이 1.0을 넘으면 잘라내는 gradient clipping과 layer norm을 추가했다.

문제 (2)

Figure 1과 Figure 2를 종합적으로 살펴보면 transformer가 일반적으로 test accuracy가 더 높으면서 training time은 더 짧은 것을 알 수 있다. 성능이 준수하게 나온 hyperparameter를 바탕으로 아래와 같이 SOTA model을 선정해봤다.

```
CNN_EMBED_DIM      = 4096
CNN_CONV_LAYER_NUM = 2
CNN_NUM_CHANNELS   = 64
```

```
TF_EMBED_DIM         = 4096
TF_ENCODER_LAYER_NUM = 2
TF_HIDDEN_DIM        = 64
```

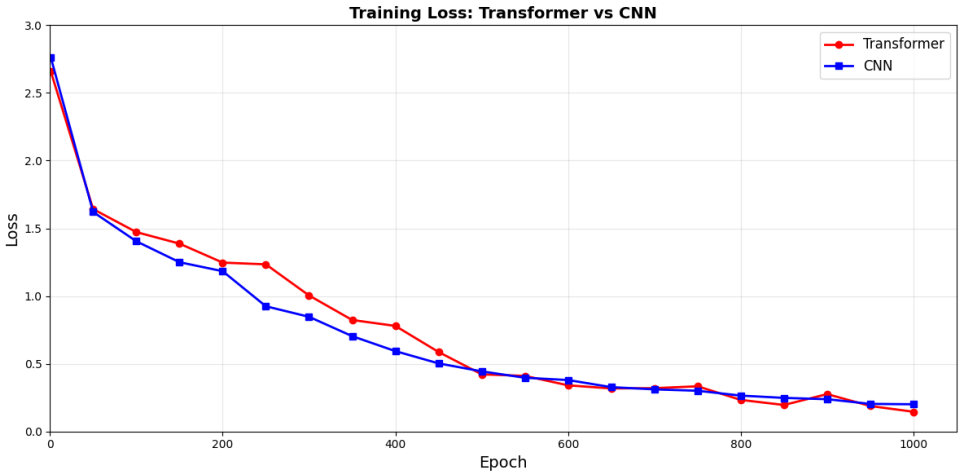

Figure 3: Loss 비교

Training loss 감소는 두 모델이 유사한 형태를 보여주고 있다.

이후 이 결과를 바탕으로 00 + 00 부터 99 + 99까지 가능한 모든 경우에 대해 prediction을 해봤다.

Table 1: SOTA 모델 성능 비교

Metric	CNN	Transformer
Training time (1k epochs)	16m 50.3s	3m 27.8s
Total inference time	287.8670 s	70.7286 s
Time per iteration	28.7867 ms	7.0729 ms
Precision	78.49% (7849/10000)	82.21% (8221/10000)
Deviation mean	-0.4912	-0.5869
Deviation std	13.5331	13.5519

결과를 살펴보면 Transformer가 연산 시간이 훨씬 적게 걸리면서 정확도는 더 높음을 알 수 있다.

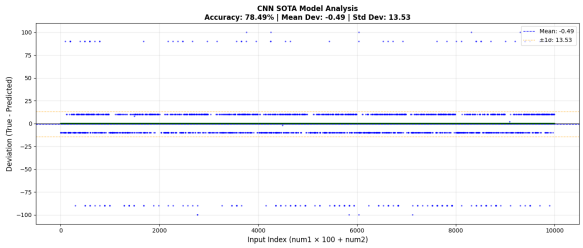

Figure 4: CNN SOTA Scatter Plot

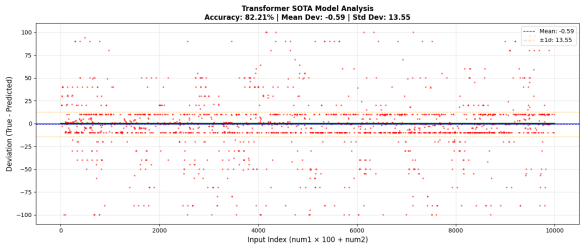

Figure 5: Transformer SOTA Scatter Plot

전수 조사 scatter를 살펴보면 그 특징이 보다 뚜렷하게 나타난다. 주목할만한 점은 CNN은 ±10에 주로 error가 몰려 있고 일부 ±90에 error가 분포되어 있지만 transformer는 ±10에 error가 주로 있어도 전체적으로 error가 퍼져 있는 것을 알 수 있다.

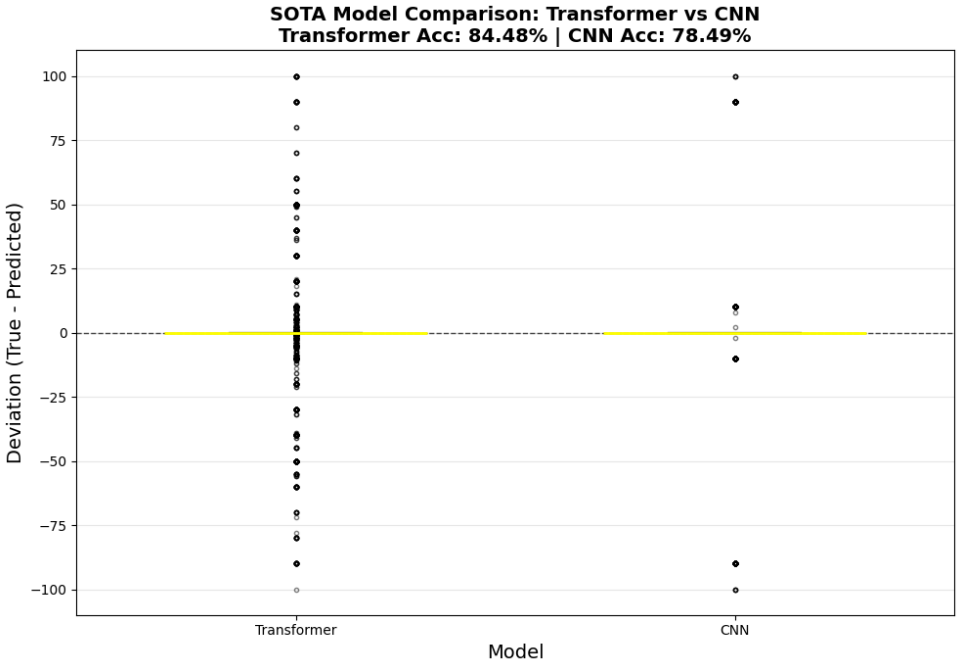

Figure 6: Box Plot 비교

상자 수염 그림을 봐도 그 차이를 알 수 있다.

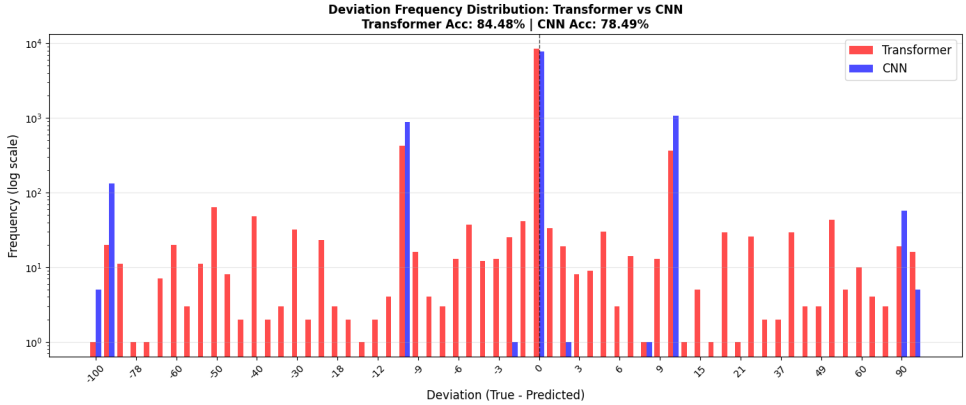

Figure 7: 최빈값 그래프

최빈값 그래프를 보면 앞서 언급한 특징이 보다 두드러진다.

Table 2: Top 5 Most Frequent Deviations

CNN			Transformer		
Deviation	**Count**	**Ratio**	**Deviation**	**Count**	**Ratio**
+0	7849	78.49%	+0	8448	84.48%
+10	1070	10.70%	-10	423	4.23%
-10	878	8.78%	+10	363	3.63%
-90	133	1.33%	-50	63	0.63%
+90	57	0.57%	-40	48	0.48%

최빈값 순위는 위와 같다.

이러한 특징이 나타나는 이유를 분석해보면 이와 같다. CNN에서는 2x2 grid로 볼 때 각 자릿수의 연산은 아주 잘 한다. 하지만 다음 자릿수에 올림을 하는 것을 잘 못한다. 그 이유는 각 숫자의 덧셈과 달리 자릿수 올림인 carry는 보다 복잡한 관계를 학습해야하기 때문이다. 이를 위해서는 보다 deep한 layer가 필요할 수 있을 것이다. 현재 model은 layer가 2개뿐이라 각 자릿수 연산은 쉽게 학습한 것이다. 한편 아예 deep하게 가면 모든 연산에서 핵심적인 각 자릿수 학습이 어려워져서 문제 (1)에서 살펴보았듯 성능이 잘 안 나온다.

반면 transformer는 self attention에서 모든 숫자들이 서로 연관되어 있기 때문에 CNN보다는 carry를 보다 잘 학습한 것을 알 수 있다. 하지만 동시에 CNN보다는 각 자리의 공간적 이해가 부족하기 때문에 전체적으로 더 퍼져 있다.

결론적으로 보자면 각 자릿수 연산이 매우 중요하다면 CNN이 더 좋은 구조이겠지만 실제로 딱 맞아 떨어지는 정확도와 훈련 속도, 추론 속도까지 전반적인 것들을 고려한다면 transformer가 보다 적합한 구조라 할 수 있다.

문제 (1) 재시도

지훈과 토의를 하면서 몇가지를 더 개선했다. 이전 내용에서 성능이 안 나왔던 이유로는 learning rate가 너무 작아서였던 것으로 알아냈다. 지훈은 5e-3으로 했는데 나는 기존과 동일하게 1e-4로 했기 때문이다. lr이 너무 작아서 학습이 제대로 안 되고 있었던 것이다. 추가로 convolution을 3중 for문이 아니라 전부 matrix 연산으로 바꿔서 속도가 더 빨라졌다. transformer에서도 마찬가지로 lr이 문제였는데 기존에 positional encoding을 넣었다가 성능이 안 나와서 다시 뺐는데 lr을 높였을 때는 positional encoding이 마지막 한 자릿수 정확도에서 중요했다.

먼저 CNN을 다시 살펴보겠다.

```
DEFAULT_EMBED_DIM      = 32
DEFAULT_CONV_LAYER_NUM = 2
DEFAULT_NUM_CHANNELS   = 32
```

위와 같이 고정하고

```
embed_dims        = [4, 8, 16, 32, 64, 128, 256, 512]
conv_layers       = [1, 2, 3,  4,  5,  6,   7,   8  ]
num_channels_list = [4, 8, 16, 32, 64, 128, 256, 512]
```

위와 같은 값을 변화시켜봤다.

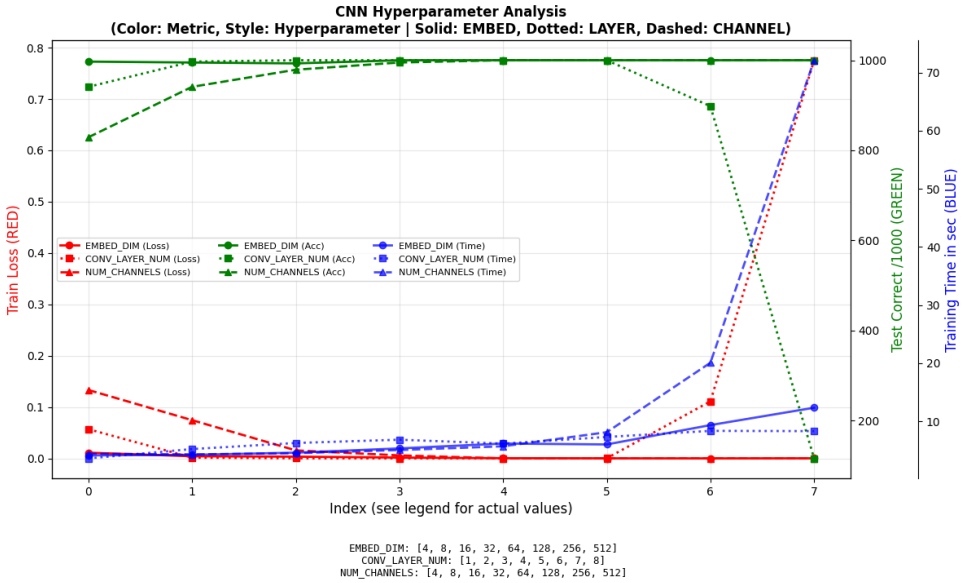

Figure 8: CNN Hyperparameter 변화에 따른 성능 비교 (재시도)

결과는 위와 같이 나왔다.

```
DEFAULT_EMBED_DIM         = 64
DEFAULT_ENCODER_LAYER_NUM = 2
DEFAULT_HIDDEN_DIM        = 128
```

```
embed_dims     = [4, 8, 16, 32, 64, 128, 256, 512]
encoder_layers = [1, 2, 3,  4,  5,  6,   7,   8  ]
hidden_dims    = [4, 8, 16, 32, 64, 128, 256, 512]
```

Transformer는 위와 같이 변화시켜봤다.

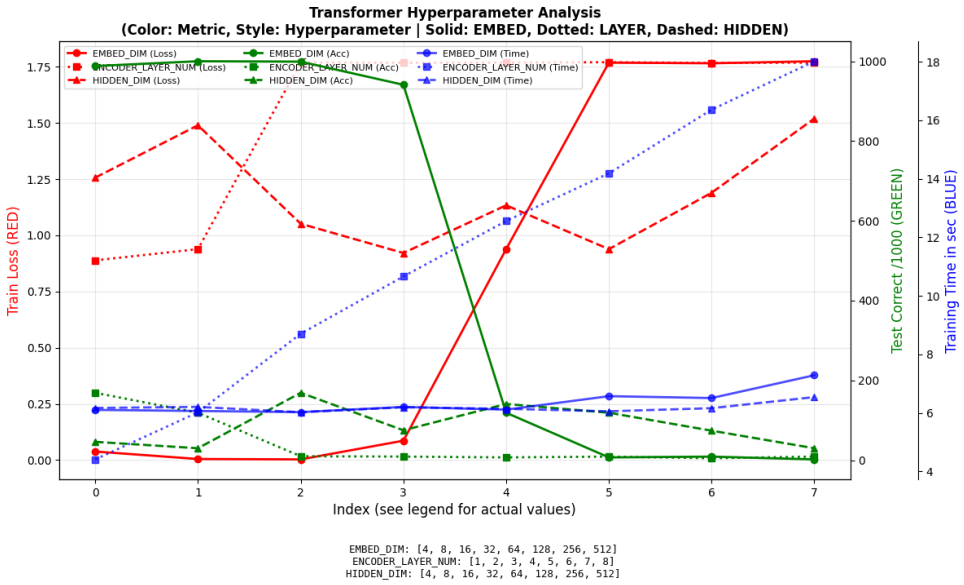

Figure 9: Transformer Hyperparameter 변화에 따른 성능 비교 (재시도)

결과는 위와 같다.

문제 (2) 재시도

```
EMBED_DIM      = 128
CONV_LAYER_NUM = 2
NUM_CHANNELS   = 64
LEARNING_RATE  = 8e-3
```

여러번 테스트를 해보면서 CNN은 최종적으로 위와 같이 설정했고 650 epoch에서 loss가 0.0000이 나왔다.

```
EMBED_DIM          = 8
ENCODER_LAYER_NUM  = 2
HIDDEN_DIM         = 32
LEARNING_RATE      = 6e-3
```

Transformer는 위와 같이 했고 5000 epoch에서도 loss가 0.0001로 나왔다.

Table 3: SOTA 모델 성능 비교 (재시도)

Metric	**CNN**	**Transformer**
Total computation time	9.3049 s	14.6648 s
Time per iteration	0.9305 ms	1.4665 ms
Precision	100.00% (10000/10000)	100.00% (10000/10000)
Deviation mean	0.0000	0.0000
Deviation std	0.0000	0.0000
Checkpoint size	482 KB	23 KB

전수 조사 결과 위 Table과 같이 결과가 나왔다. lr을 높이니 결론이 뒤바뀌었다. CNN은 여러 parameter에서도 잘 학습이 되는데 transformer는 좀 더 학습이 까다로웠다. 사실 CNN이든 transformer든 이렇게 간단한 task를 못할 리가 없는데 지금 와서 보니 참 웃기다. lr이 중요함을 깨닫게 되는 것 같다.

한편 model weight checkpoint size는 거의 20배 가까이 transformer가 가볍게 나왔다. hyper-parameter를 적게 해서 그런 것 같다. 그래서 한 번 CNN도 더 줄여봤는데 2, 2, 16으로 해도 거의 비슷한 결과가 나왔고 size는 17KB가 나왔다.

결론적으로 속도, 정확도, 모델 크기, 학습의 용이성 모든 측면에서 CNN이 좋다고 결론내릴 수 있겠다.