Symbolic Mathematics

KIMIA NOORBAKHSH, MAHDI SHARIFI, MODAR SULAIMAN, SHUGE LEI

PROF. POOYAN JAMSHIDI, PROF. KALLOL ROY

MAY 24, 2021

What we have done…

- Chosen a compatible input and output layers to our data.
- Implemented a cross entropy loss function for evaluation.
- Trained an initial model on 10000 samples.
- Trained the same model on 1000000 samples.
- Trained the XOR task with longer input length and analyzed the results.

What we have done…

Chosen a compatible input and output layers to our data.

- Dictionary size: 91

```
words:
{'<s>': 0,
 '</s>': 1,
 '<pad>': 2,
 '(': 3, ')': 4,
 '<SPECIAL_5>': 5,
```

```
'rac': 60,
'sec': 61,
'sech': 62,
'sign': 63,
'sin': 64,
'sinh': 65,
'sqrt': 66,
```

```
'4': 85,
'5': 86,
'6': 87,
'7': 88,
'8': 89,
'9': 90}
```

- Input size: 91×768
- Output size: 768×91

```
gpt2 = GPT2Model.from_pretrained('gpt2')
in_layer = nn.Embedding(len(env.word2id), 768)
out_layer = nn.Linear(768, len(env.word2id))
```

What we have done…

Freezing the layers.

- For training the first 10000 samples:

```
for name, param in gpt2.named_parameters():
    # freeze all parameters except the layer norm and positional embeddings
    if 'ln' in name or 'wpe' in name:
        param.requires_grad = True
    else:
        param.requires_grad = False
```

- For the 1000000 samples, we also added ‘attn’ layers to the training layers.

What we have done…

Implemented a cross entropy loss function for evaluation.

- Question: $a + b = b + a$
- This could be handled using the **sympy** library and check whether two equations are the same or not.

```
optimizer = torch.optim.Adam(parameters)
loss_fn = nn.CrossEntropyLoss()
```

```
embeddings = in_layer(x.reshape(1, -1))
hidden_state = gpt2(inputs_embeds=embeddings).last_hidden_state[:, :]
logits = out_layer(hidden_state)[0]
loss = loss_fn(logits, y.reshape(y.shape[0]))
accuracies.append((logits.argmax(dim=-1) == y).float().mean().item())
```

What we have done…

Trained an initial model on 10000 samples.

- These are the results of the first and second epoch (no improvement!)

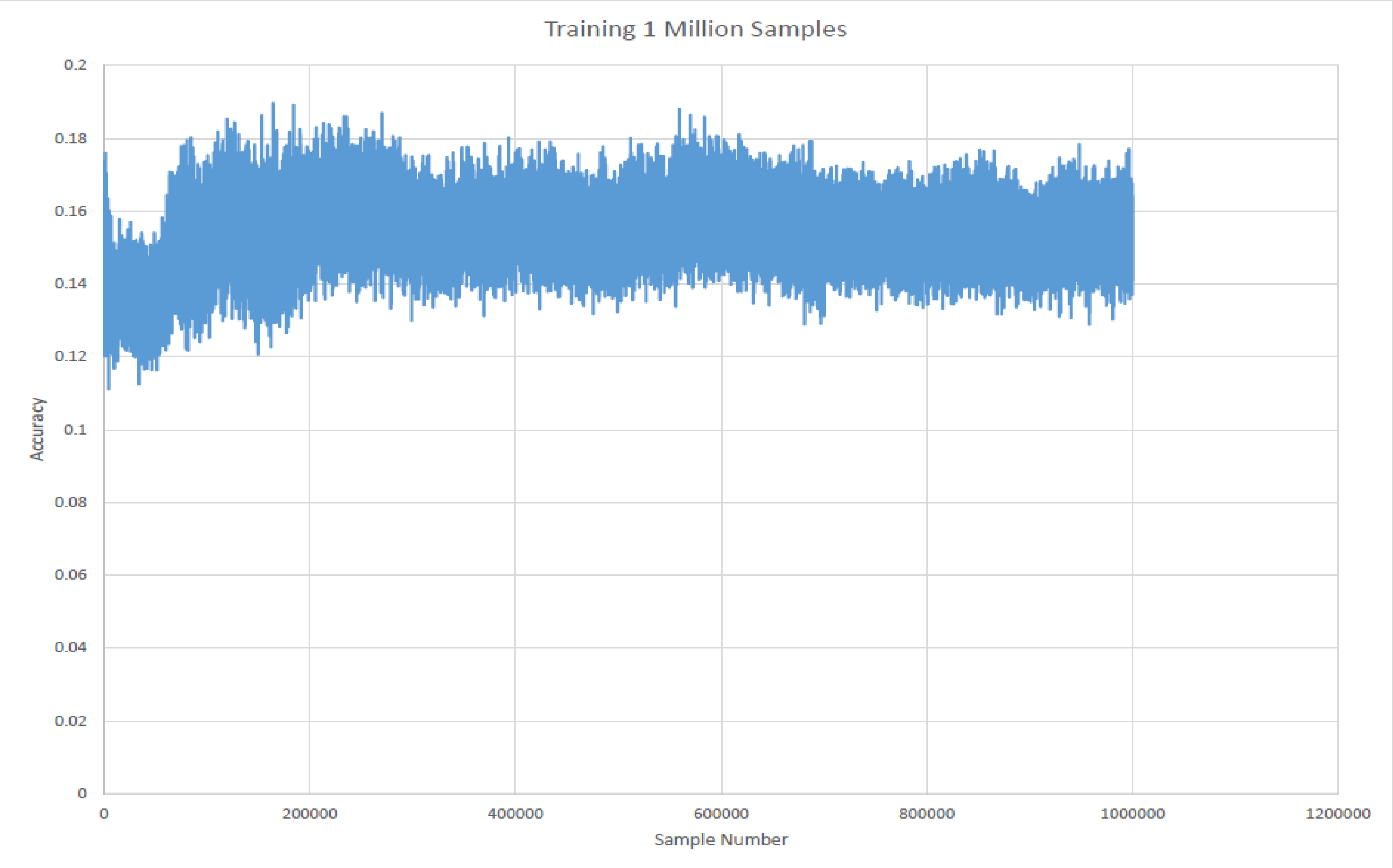
```
Samples: 500, Accuracy: 0.15763817325234414
Samples: 1000, Accuracy: 0.15360899686813353
Samples: 1500, Accuracy: 0.1508916337415576
Samples: 2000, Accuracy: 0.15609137535095216
Samples: 2500, Accuracy: 0.15047348447144032
Samples: 3000, Accuracy: 0.16530826970934867
Samples: 3500, Accuracy: 0.16306135967373847
Samples: 4000, Accuracy: 0.16064294055104256
Samples: 4500, Accuracy: 0.16799090638756753
Samples: 5000, Accuracy: 0.16350908167660236
Samples: 5500, Accuracy: 0.1545982526242733
Samples: 6000, Accuracy: 0.15577257606387l4
Samples: 6500, Accuracy: 0.15484916180372238
Samples: 7000, Accuracy: 0.1562862466275692
Samples: 7500, Accuracy: 0.15124963700771332
```

```
Samples: 10500, Accuracy: 0.13288296952843667
Samples: 11000, Accuracy: 0.14036015808582306
Samples: 11500, Accuracy: 0.1347443114593625
Samples: 12000, Accuracy: 0.14149298578500746
Samples: 12500, Accuracy: 0.14634867280721664
Samples: 13000, Accuracy: 0.15686407506465913
Samples: 13500, Accuracy: 0.15796048790216446
Samples: 14000, Accuracy: 0.1601380456984043
Samples: 14500, Accuracy: 0.16648880869150162
Samples: 15000, Accuracy: 0.15835930436849593
Samples: 15500, Accuracy: 0.15647333413362502
Samples: 16000, Accuracy: 0.16316208489239215
Samples: 16500, Accuracy: 0.15583015084266663
Samples: 17000, Accuracy: 0.15681038931012153
```

What we have done…

Trained the same model on 1000000 samples.

- The Training for 1 epoch took almost 24 hours with the Chameleon GPU. (batch size = 1)



A Big Mistake!

- The accuracies printed was wrong due to a mistake in shape of the output tensors.

```
accuracies.append((logits.argmax(dim=-1) == y).float().mean().item())
```

Wrong version

```
accuracies.append((logits.argmax(dim=-1) == y.reshape(-1)).float().mean().item())
```

Corrected!

Result for 2 epochs on 10000 Samples


```
Final accuracy: 0.41609758779406547
```

Batch size = 1

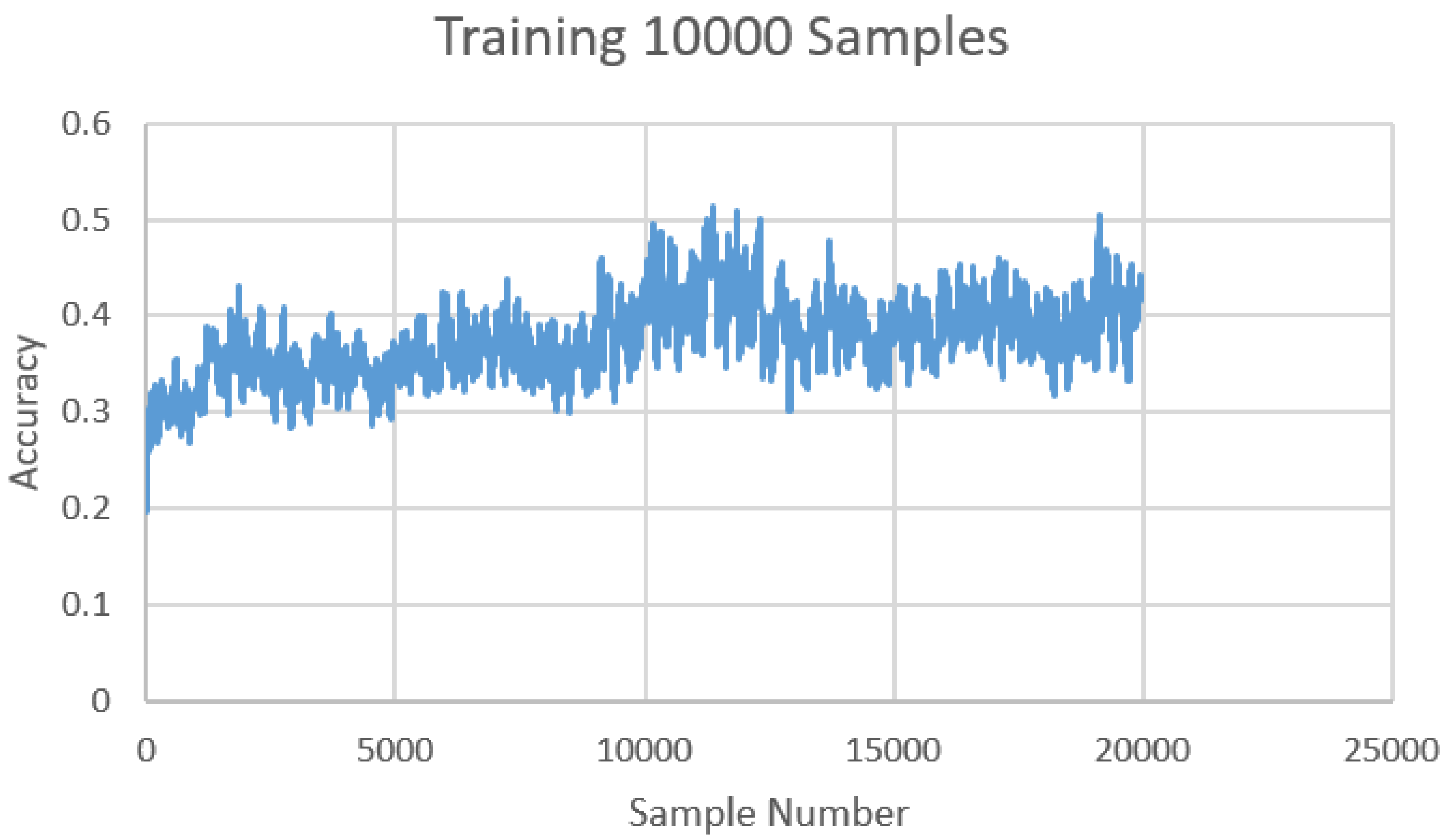
Epoch 1

```
Samples: 500, Accuracy: 0.28932218208909033
Samples: 1000, Accuracy: 0.3040719592571259
Samples: 1500, Accuracy: 0.32598347201943395
Samples: 2000, Accuracy: 0.3626392023265362
Samples: 2500, Accuracy: 0.3354347388446331
Samples: 3000, Accuracy: 0.3481183850765228
Samples: 3500, Accuracy: 0.35062968119978905
Samples: 4000, Accuracy: 0.3383585032820702
Samples: 4500, Accuracy: 0.33974564373493193
Samples: 5000, Accuracy: 0.36348727375268935
Samples: 5500, Accuracy: 0.38484443426132203
Samples: 6000, Accuracy: 0.4136078540980816
Samples: 6500, Accuracy: 0.38004438281059266
Samples: 7000, Accuracy: 0.3565161497890949
Samples: 7500, Accuracy: 0.41761154145002366
Samples: 8000, Accuracy: 0.34294311940670014
Samples: 8500, Accuracy: 0.3590305034816265
Samples: 9000, Accuracy: 0.361914224922657
Samples: 9500, Accuracy: 0.3862622971832752
Samples: 10000, Accuracy: 0.40854225382208825
```

Epoch 2

```
Samples: 10500, Accuracy: 0.37648009195923804
Samples: 11000, Accuracy: 0.4336681571602821
Samples: 11500, Accuracy: 0.4163236790895462
Samples: 12000, Accuracy: 0.41636047303676604
Samples: 12500, Accuracy: 0.3708689112961292
Samples: 13000, Accuracy: 0.3882619747519493
Samples: 13500, Accuracy: 0.41208639204502107
Samples: 14000, Accuracy: 0.3775967198610306
Samples: 14500, Accuracy: 0.3613714224100113
Samples: 15000, Accuracy: 0.4055298209190369
Samples: 15500, Accuracy: 0.41404973894357683
Samples: 16000, Accuracy: 0.42975750789046285
Samples: 16500, Accuracy: 0.4081644794344902
Samples: 17000, Accuracy: 0.38103840544819834
Samples: 17500, Accuracy: 0.4448875626921654
Samples: 18000, Accuracy: 0.37044109985232354
Samples: 18500, Accuracy: 0.3873744860291481
Samples: 19000, Accuracy: 0.39758136838674546
Samples: 19500, Accuracy: 0.42351166293025017
Samples: 20000, Accuracy: 0.41609758779406547
```

Result for 2 epochs on 10000 Samples



What we have done …

Trained the XOR task with longer input length and analyzed the results.

- Increased the input sequence size of the bit-xor task in the universal computations code from 5 to 50 to see how fast the accuracies grow… (note that the bit-xor task is much simpler than our task!)

Accuracy is growing fast!

$$n = 5$$

```
Samples: 500, Accuracy: 0.6320000129938126
Samples: 1000, Accuracy: 0.600000016093254
Samples: 1500, Accuracy: 0.6560000130534172
Samples: 2000, Accuracy: 0.7320000123977661
Samples: 2500, Accuracy: 0.6400000116229058
Samples: 3000, Accuracy: 0.6960000142455101
Samples: 3500, Accuracy: 0.7640000116825104
Samples: 4000, Accuracy: 0.7520000123977661
Samples: 4500, Accuracy: 0.7560000121593475
Samples: 5000, Accuracy: 0.8280000087618827
Samples: 5500, Accuracy: 0.9000000059604645
Samples: 6000, Accuracy: 0.9440000027418136
Samples: 6500, Accuracy: 0.9520000028610229
Final accuracy: 0.9920000004768371
```

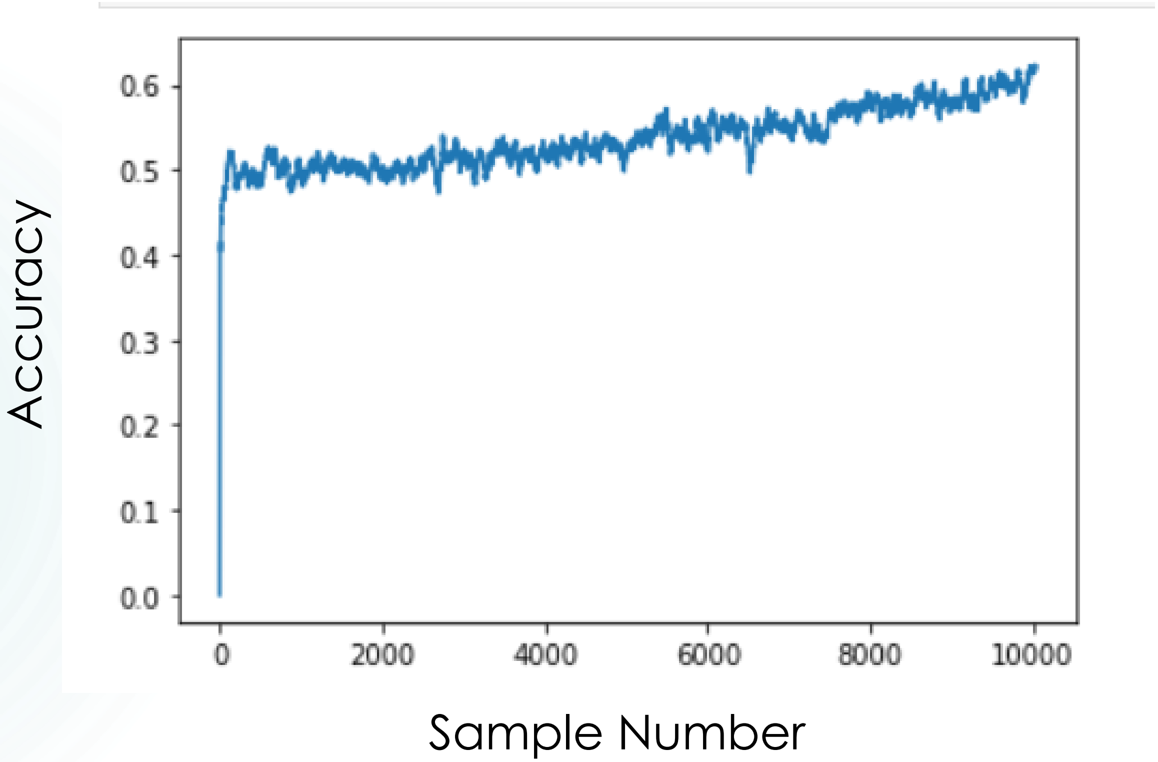
$$n = 50$$

```
Samples: 500, Accuracy: 0.4899999940395356
Samples: 1000, Accuracy: 0.4931999635219575
Samples: 1500, Accuracy: 0.5003999990224838
Samples: 2000, Accuracy: 0.4995999985933304
Samples: 2500, Accuracy: 0.5035999983549118
Samples: 3000, Accuracy: 0.5195999985933304
Samples: 3500, Accuracy: 0.5264000052213669
Samples: 4000, Accuracy: 0.5180000001192093
Samples: 4500, Accuracy: 0.5400000011920929
Samples: 5000, Accuracy: 0.5176000010967254
Samples: 5500, Accuracy: 0.5608000022172928
Samples: 6000, Accuracy: 0.5320000004768372
Samples: 6500, Accuracy: 0.5196000039577484
Samples: 7000, Accuracy: 0.5468000048398971
Samples: 7500, Accuracy: 0.5636000013351441
Samples: 8000, Accuracy: 0.5720000016689301
Samples: 8500, Accuracy: 0.5668000030517578
Samples: 9000, Accuracy: 0.5703999996185303
Samples: 9500, Accuracy: 0.6036000001335144 0
Samples: 10000, Accuracy: 0.6160000000834465
```

Accuracy is growing slower!

What we have done …

Trained the XOR task with longer input length and analyzed the results.



Questions

- The warning after loading the pre-traines GPT2, why attentions are not initialized?

Might be so important, we should figure it out!

```
gpt2 = GPT2Model.from_pretrained('gpt2')
in_layer = nn.Embedding(len(env.word2id), 768)
out_layer = nn.Linear(768, len(env.word2id)) # or flattening or softmax or non-linear !
# [1, 45, 76, 2, 4] = ['sin', 'sub',...]
# sin     sub     cos      add
# 0.9     0.05    0.05       0  -> sin 0.9 =  1/sum(1+45+...)
# 0       0.8     0.2        0 -> sub
```

```
HBox(children=(FloatProgress(value=0.0, description='Downloading', max=665.0, style=ProgressStyle(description_…

HBox(children=(FloatProgress(value=0.0, description='Downloading', max=548118077.0, style=ProgressStyle(descri…

Some weights of GPT2Model were not initialized from the model checkpoint at gpt2 and are newly initialized: ['h.0.attn.masked_bias', 'h.1.attn.masked_bias', 'h.2.attn.masked_bias', 'h.3.attn.masked_bias', 'h.4.attn.masked_bias', 'h.5.attn.masked_bias', 'h.6.attn.masked_bias', 'h.7.attn.masked_bias', 'h.8.attn.masked_bias', 'h.9.attn.masked_bias', 'h.10.attn.masked_bias', 'h.11.attn.masked_bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Questions

- Why in the xor task, they use the last n outputs of the last hidden state? Why not the first n outputs?
- Last_hidden_state[:, :n]

```
accuracies = [0]
while sum(accuracies[-50:]) / len(accuracies[-50:]) < .99:
    x, y = generate_example(n)
    x = torch.from_numpy(x).to(device=device, dtype=torch.long)
    y = torch.from_numpy(y).to(device=device, dtype=torch.long)

    embeddings = in_layer(x.reshape(1, -1))
    hidden_state = gpt2(inputs_embeds=embeddings).last_hidden_state[:,n:]
    logits = out_layer(hidden_state)[0]

    loss = loss_fn(logits, y)
    accuracies.append((logits.argmax(dim=-1) == y).float().mean().item())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if len(accuracies) % 500 == 0:
        accuracy = sum(accuracies[-50:]) / len(accuracies[-50:])
        print(f'Samples: {len(accuracies)}, Accuracy: {accuracy}')

print(f'Final accuracy: {sum(accuracies[-50:]) / len(accuracies[-50:])}')
```

Questions

- What would be the appropriate batch size for our task and how to deal with input and output lengths?

Future Plan

- Train new code on more data! (did not have time to do that for today ☹)
- Increase batch size from 1 to 2, 4, 8, 16, …
- Check equality of outputs using sympy for more accuracy
- Try to first study and then add beam search at the end for more accuracy!
- What else? **Plan after the meeting:**
- **Use signed randomization for initializing the weights.**
- **Increase the batch size (25 or 32).**
- **Use larger epochs (~1000).**
- **Increase the learning rate.**
- **Norm Regularization.**

Thank you!

All of the code is available at https://github.com/softsys4ai/differentiable-proving