

# Sheet 9

Jan Reckermann, Tim Weinreich, Julian Kusch

## 1

### a)

- Sentence Order Prediction (SOP): SOP directly teaches the model to grasp sentence relationships
- Token Boundary Detection (TBD): It can benefit from learning more explicitly where natural word boundaries occur (especially for german)

### b)

- Prefix Sequence Reconstruction (PSR): ability to encode earlier parts of a sequence conditioned on later context **Decoder only models can only predict sequentially?**
- Future Span Prediction (FSP): FSP forces the model to reason about future sequences more holistically instead of predicting token-by-token

## 2 Under the hood of LLMs: Llama 2.7B

Initilazation of the model

```
In [ ]: from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

#with open("access_token.txt", "r") as f:
#    access_token = f.read().strip()
access_token = "hf_jwkgzNtBaEmrTnboTOYXRaxNDXNwvWNn1L"
model = "meta-llama/Llama-2-7b-chat-hf"
tokenizer = AutoTokenizer.from_pretrained(model, token=access_token)
model = AutoModelForCausalLM.from_pretrained(model, token=access_token, torch_dtype=torch.float16)
print(model)
```

```
In [2]: for id in range(5100, 5110): #Printing the tokens 5100 up to 5110 for example
        print(f"{id=}, {tokenizer.decode([id])}")

print("\ntokenizer length:", len(tokenizer)) #the whole model contains 32000 tokens

sun_id = tokenizer.encode("sun", return_tensors="pt")[-1] #find the token number
print(f"\nsun_id={sun_id}")
```

```
print(tokenizer.decode(sun_id))

emb = model.get_input_embeddings()(sun_id.to(device))
print("embedding shape:", emb.shape) #shape of the token
```

```
id=5100, compet
id=5101, pair
id=5102, inglés
id=5103, Response
id=5104, Fig
id=5105, grad
id=5106, documentation
id=5107, cant
id=5108, appreci
id=5109, ãn
```

```
tokenizer length: 32000
```

```
sun_id=tensor([ 1, 6575])
<s> sun
embedding shape: torch.Size([2, 4096])
```

```
In [3]: sequence = "My favorite composer is"
model_inputs = tokenizer(sequence, return_tensors="pt").to(device) #transforms i
print(tokenizer.decode(model_inputs["input_ids"].tolist()[0])) #view decoded tok
with torch.no_grad():
    outputs = model(**model_inputs) #the model gets the input and generates the

logits = outputs['logits'][0, -1, :] #gets unnormalized probability for every to
print("\nlogits shape:", logits.shape)

probabilities = torch.nn.functional.softmax(logits, dim=-1) #normalizes
top_k = 7
top_prob, top_ind = torch.topk(probabilities, top_k)

print("\nOutputs:\n")

for i in range(top_k):
    print(f"{tokenizer.decode(top_ind[i].tolist())}: {top_prob[i]:.2f}") #prints
```

```
<s> My favorite composer is
```

```
logits shape: torch.Size([32000])
```

```
Outputs:
```

```
Moz: 0.25
Ch: 0.11
Be: 0.09
Ludwig: 0.08
Fr: 0.03
Wolfgang: 0.02
Ig: 0.02
```

```
In [4]: import regex as re

sequence = ""
model_inputs = tokenizer(sequence, return_tensors="pt").to(device) #input is emp

generated_answer = ""
for _ in range(30): #iterative generation of the next 30 tokens
```



```

with torch.no_grad():
    outputs = model(**model_inputs) #generate output bases on input
    logits = outputs['logits'][0, -1, :] #unnormalized probabilities for the
    probabilities = torch.nn.functional.softmax(logits, dim=-1) #normalizes
    next_token_id = torch.argmax(probabilities).unsqueeze(0) #take the token
    model_inputs["input_ids"] = torch.cat([model_inputs["input_ids"], next_t

#printing output/generated answer in readable form for every iteration
next_word = tokenizer.decode(next_token_id.tolist())
next_word = re.sub(r"^[^a-zA-Z0-9.?!]", "", next_word)
generated_answer += next_word
generated_answer += " "

print(generated_answer)

```

Unterscheidung

Unterscheidung zwischen

Unterscheidung zwischen

Unterscheidung zwischen K

Unterscheidung zwischen K raft

Unterscheidung zwischen K raft

Unterscheidung zwischen K raft und

Unterscheidung zwischen K raft und

Unterscheidung zwischen K raft und K

Unterscheidung zwischen K raft und K raft

Unterscheidung zwischen K raft und K raft werk

Unterscheidung zwischen K raft und K raft werk

Unterscheidung zwischen K raft und K raft werk

Unterscheidung zwischen K raft und K raft werk

Unterscheidung zwischen K raft und K raft werk

Unterscheidung zwischen K raft und K raft werk K

Unterscheidung zwischen K raft und K raft werk K raft

Unterscheidung zwischen K raft und K raft werk K raft

Unterscheidung zwischen K raft und K raft werk K raft und

Unterscheidung zwischen K raft und K raft werk K raft und

Unterscheidung zwischen K raft und K raft werk K raft und K

Unterscheidung zwischen K raft und K raft werk K raft und K raft

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk s

ind

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk s

ind two

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk s

ind two different

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk s

ind two different German

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk s

ind two different German words

Unterscheidung zwischen K raft und K raft werk K raft und K raft werk s

ind two different German words that

### 3 Flow-based modeling



Zettel 9

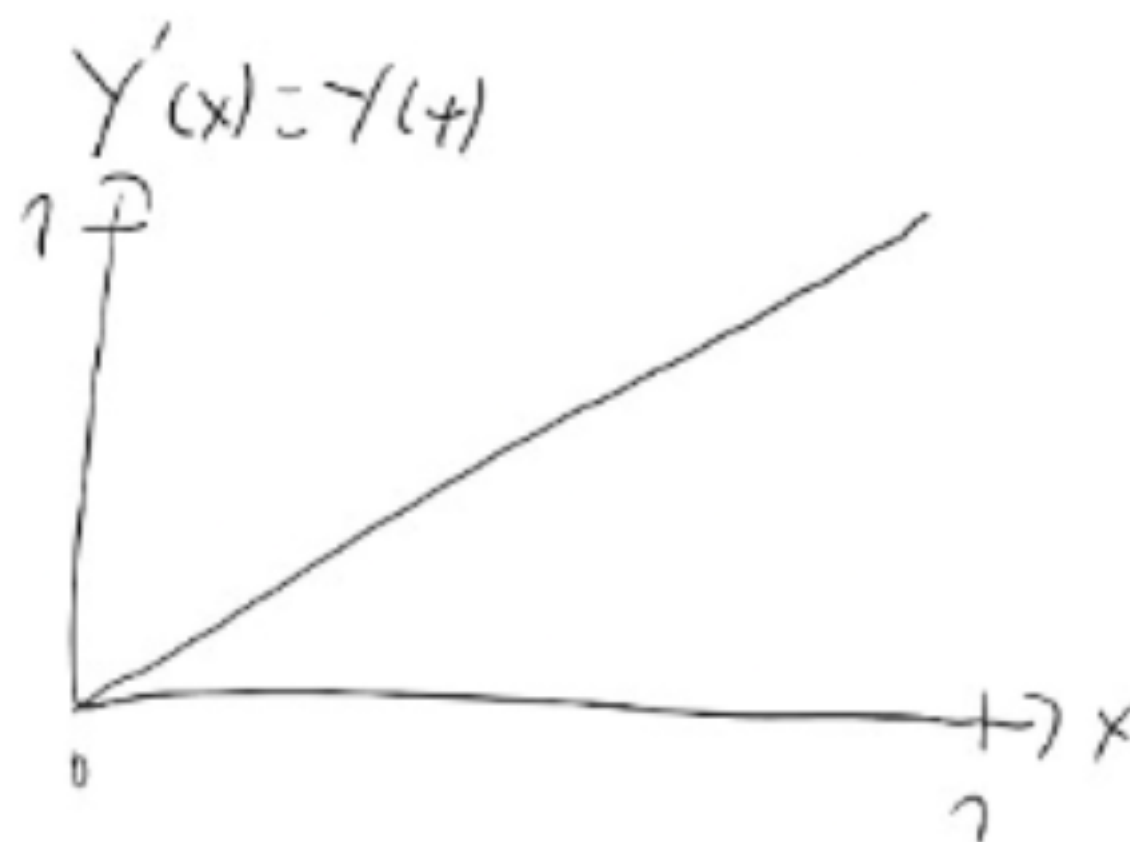
Donnerstag, 19. Dezember 2024

11:25

Aufg. 3  
a)  $P(x)$ 

$$Y(x) = \int_0^x 1 dx' = x$$

$$Y'(x) = Y^{-1}(x) = x$$



add:

$$p dP(Y) = \frac{dY}{dx} = 1$$

$$p dP(Y') = \frac{dY'}{dx} = 1$$

Y's pdf is arbitrary, it is not in general uniform

With this form you can transform the distributions given a relation between the variables. Here you want to do the opposite: Find a relation given the transform of prob. distr.

$$b) \quad P_Y(Y') = P_X(x) \left| \frac{dx}{dY} \right| \quad \begin{aligned} X &= g^{-1}(Y) \\ Y &= g(X) \end{aligned}$$

Example:

$$P_X(x) = \frac{1}{2}x \quad P_Y(y) = -\frac{1}{2}y + 1$$

$$g(x): \quad P_X(x) \stackrel{!}{=} P_Y(y) \Rightarrow \frac{1}{2}x = -\frac{1}{2}y + 1 \Rightarrow x = 2 - y$$

$$g(x) = 2 - x \quad g^{-1}(y) = 2 - y$$

$$P_Y(Y') = P_X(g^{-1}(x)) \left| \frac{d(g^{-1}(x))}{dY} \right| = \frac{1}{2}(2 - y) \left| \frac{d(2 - y)}{dY} \right|$$



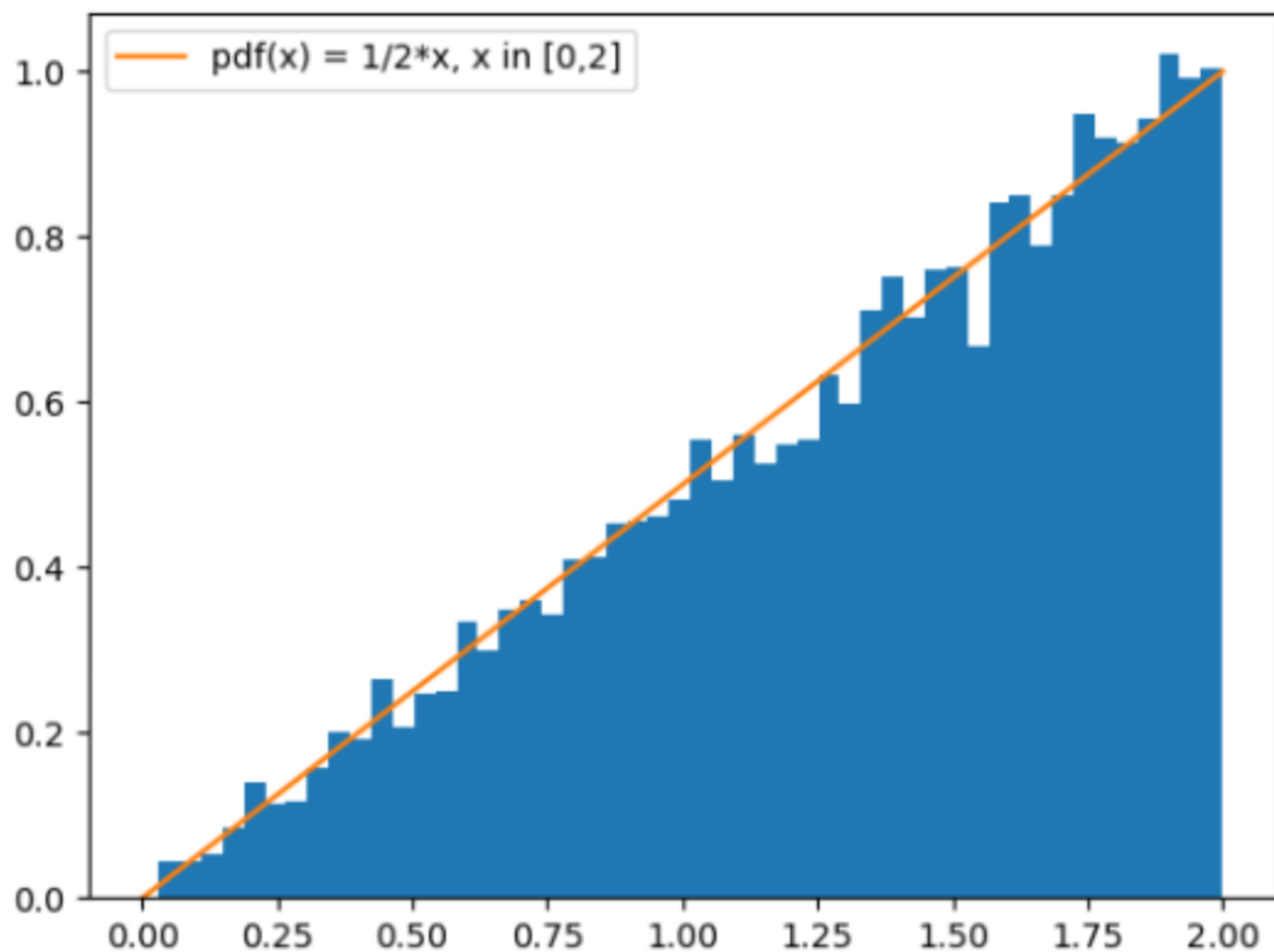
$$= 1 - \frac{1}{2} \cdot y \quad | \cdot (-1) = 1 - \frac{1}{2} \cdot y$$

(b)

```
In [7]: import numpy as np
import matplotlib.pyplot as plt

# Load the 1d samples:
samples = np.load("data/samples_1d.npy")

x_lin = np.linspace(0, 2, 1000)
plt.hist(samples, bins=50, density=True)
plt.plot(x_lin, 1/2 * x_lin, label="pdf(x) = 1/2*x, x in [0,2]")
plt.legend()
plt.show()
```



```
In [11]: def g(x):
    return 2-x
samples_trans = g(samples)
x_lin = np.linspace(0, 2, 1000)
plt.hist(samples_trans, bins=50, density=True)
plt.plot(x_lin, -1/2 * x_lin + 1, label="pdf(y) = -1/2*y + 1, y in [0,2]")
plt.legend()
plt.show()
```

You should get  $2 - 2\sqrt{1 - (x/2)^2}$   
 Interesting this works as well

