Subject: Re: Overview of AI on Z- Session 1
Date: Monday, June 3, 2024 at 12:36:27 PM Mountain Daylight Time
From: Swamy Sivasubramaniyan
To: Unnikrishnan B (HCL Mainframe CoE)
Attachments: image001.png, image002.png, image003.png, image004.png, image006.png, NIPS-2017-attention-is-all-you-need-Paper.pdf, Transformer — The Encoder Stack Explained _ by Sandaruwan Herath _ Data Science and Machine Learning _ Apr, 2024 _ Medium.pdf, The decoder stack in the Transformer model _ by Sandaruwan Herath _ Data Science and Machine Learning _ May, 2024 _ Medium.pdf

Perfect! Good job, you get 100% for your first assignment. 😊

1. *The tokens you put in your notes in Page 13 [101, 16849, 2100, 7459, 2000, 4553, 2035, 2154, 102]. These are just examples or did you calculate it. I have read that these are generated by some complex calculations. Just curious.*
   Those were generated for a BERT model. Here is the code below.
   Don't worry about the complex calculations. We, as application developers, do not have to deal with them directly. There is lots of 'magic', aka abstractions.

Step 1 : Import the pyhton models: BertTokerizer, BertModel, pyTorch,

```
from transformers import BertTokenizer, BertModel
import torch
```

if any of these packages are "not found", use

```
pip install -q -U transformers
pip install -q -U scikit-learn
```

If you are doing this in a Jupyter notebook add ! in front of the command

```
!pip install -q -U scikit-learn
```

Step 2 : Get an instance of the Model.

```
# Let's load a vanilla BERT-base model.
model = BertModel.from_pretrained('bert-base-uncased')
```

Step 3 : Get an instance of the Tokenizer.

```
# load the bert-base uncased tokenizer.
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased') # Uncased -->
PreTrainer converts all to lowercase
```

Step 4: Use the Tokenizer from Step#3 to generate the tokens representing a sentence.

```
tokens = tokenizer.encode('Swamy loves to learn all day')  # tokenize a
simple sequence
```

Step 5: View the token generated

```
print(tokens)
```

Step 6: Decode the tokens and generate the text (reverse)

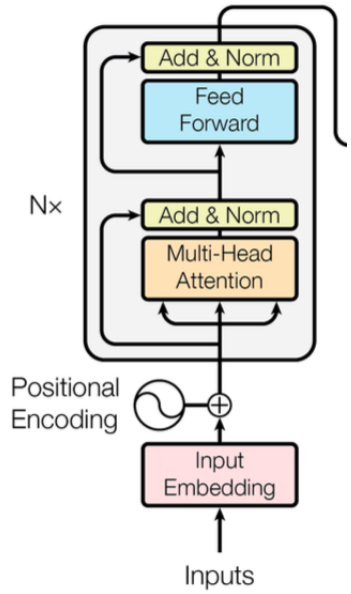
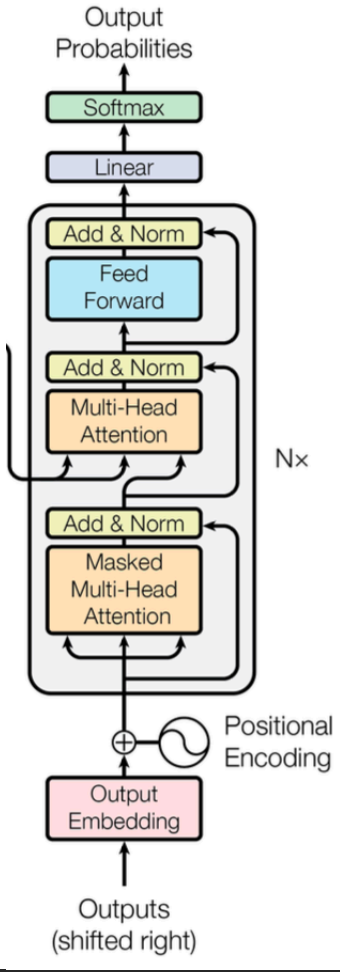
```
txt = tokenizer.decode(tokens)
```

```
print(txt)
```

2. *It says in Page 14 that some use encoders and some decoders and some both. How does that work as most will have to understand and also generate text output?*

   First: Update your mental model by dropping the *word meaning* of encoder and decoder (as in – encoder making it cryptic, and decoder converting cryptic bit stream to text).

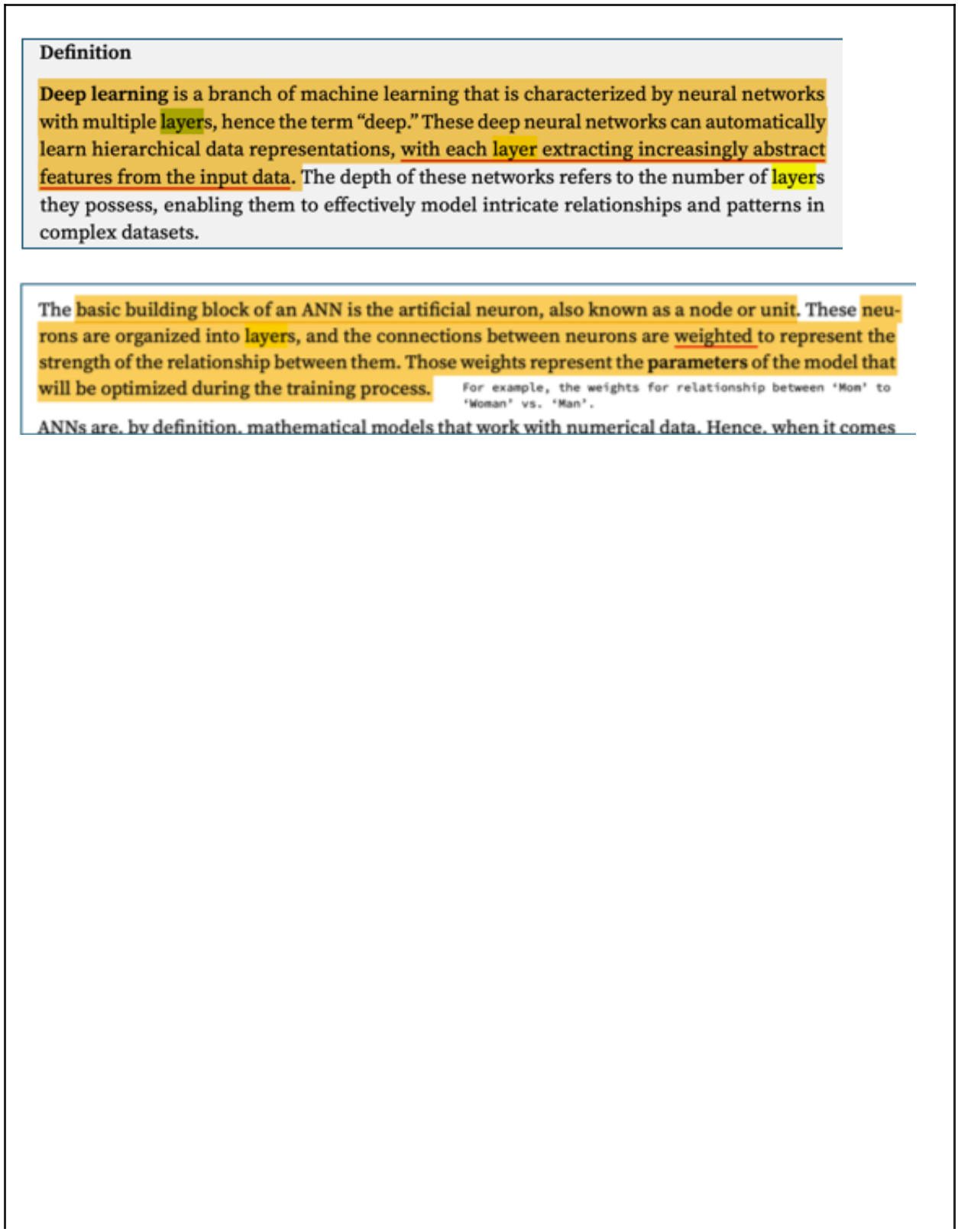
   In the context of the Transformer model, both encoder and decoder are very similar. One difference is that the decoder uses right-shifted embeddings as the starting point, compared to the encoder.
   Source: The original paper that introduced the Transformer model - Attention Is All You Need (2017). PDF attached – 'NIPS-2017-attention-is-all-you-need-Paper.pdf'.
   Also, attaching a couple of articles on the topic (I have not read them yet!)

Encoder	Decoder
	

3. *Are layers in LLM (page 16) the vertical connected neurons in the LLM? Just to have a visual feel.*

This is going to be a bit hard to explain in an email... let me try anyway 😊 [*don't rake your brain if my explanation makes no sense!*]

Take a look at some of the content from earlier pages, I will should an actual example..



Definition

Deep learning is a branch of machine learning that is characterized by neural networks with multiple layers, hence the term "deep." These deep neural networks can automatically learn hierarchical data representations, with each layer extracting increasingly abstract features from the input data. The depth of these networks refers to the number of layers they possess, enabling them to effectively model intricate relationships and patterns in complex datasets.

The basic building block of an ANN is the artificial neuron, also known as a node or unit. These neurons are organized into layers, and the connections between neurons are weighted to represent the strength of the relationship between them. Those weights represent the **parameters** of the model that will be optimized during the training process.

For example, the weights for relationship between 'Mom' to 'Woman' vs. 'Man'.

ANNs are, by definition, mathematical models that work with numerical data. Hence, when it comes

Once we have the vectorized input, we can pass it into the multi-layered neural network. There are three main types of layers:

- **Input layer**: The first layer of the neural network receives the input data. Each neuron in this layer corresponds to a feature or attribute of the input data.
- **Hidden layers**: Between the input and output layers, there can be one or more hidden layers. These layers process the input data through a series of mathematical transformations and extract relevant patterns and representations from the data.
- **Output layer**: The final layer of the neural network produces the desired output, which could be predictions, classifications, or other relevant results depending on the task the neural network is designed for.

Hidden layers: One layer might focus on the grammer subject, verb and noun relationships between the tokens.

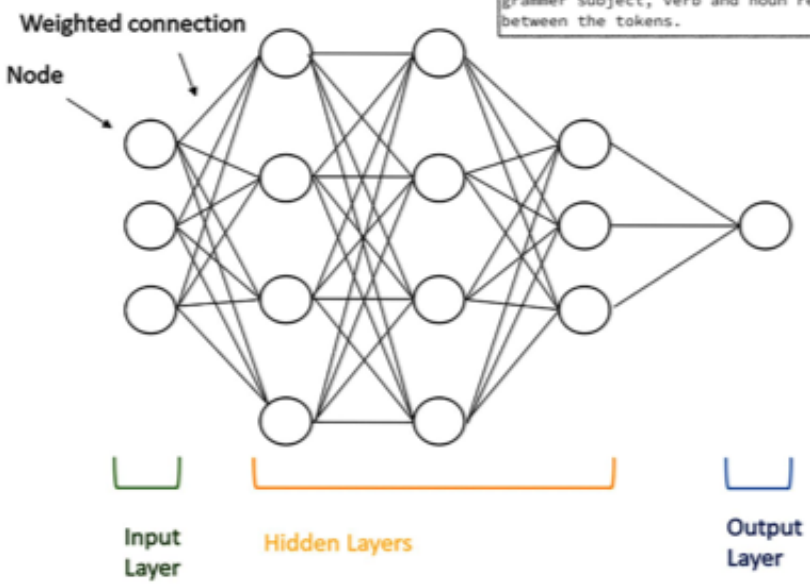


Figure 1.6: High-level architecture of a generic ANN

The code example is based on the BERT model we used to see the tokens.

Once we generated the model, we can call a few functions to interrogate the model deeper.

```
# Let's load a vanilla BERT-base model.
model = BertModel.from_pretrained('bert-base-uncased')
```

1. Get the number of 'named parameters' in the 'bert-base-uncased' model.

```
named_params = list(model.named_parameters())
print('The BERT model has {:} different named
parameters.\n'.format(len(named_params)))
```

The output:
`The BERT model has 199 different named parameters.`

2. Sample of the Embedding layers and their size.

```
print('==== Embedding Layer ====\n')
for p in named_params[0:5]:
```

```
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

Output:

```
==== Embedding Layer ====

embeddings.word_embeddings.weight                       (30522, 768)
embeddings.position_embeddings.weight                     (512, 768)
embeddings.token_type_embeddings.weight                     (2, 768)
embeddings.LayerNorm.weight                                   (768,)
embeddings.LayerNorm.bias                                     (768,)
```

3. Get the parameters from the Output layer.

```
print('\n==== Output Layer ====\n')
for p in named_params[-2:]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

Output:

```
==== Output Layer ====

pooler.dense.weight                                     (768, 768)
pooler.dense.bias                                         (768,)
```

Regards,

Swamy Sivasubramaniyan
1-650-308-4493 Desk
1-720-239-3720 Mobile
Swamy.Sivasubramaniyan@USVenturesLtd.com

From: "Unnikrishnan B (HCL Mainframe CoE)" <unnikrishnanb@hcltech.com>
Date: Monday, June 3, 2024 at 10:34 AM
To: Swamy Sivasubramaniyan <Swamy.Sivasubramaniyan@usventuresltd.com>
Subject: Re: Overview of AI on Z- Session 1

Swamy,
I too finished the first chapter. Many things went over my head… But got an idea. Should I know everthing mentioned or a general idea should do and we can build as we progress?

Few Qs –

1. The tokens you put in your notes in Page 13 [101, 16849, 2100, 7459, 2000, 4553, 2035, 2154, 102]. These are just examples or did you calculate it. I have read that these are generated by some complex calculations. Just curious.
2. It says in Page 14 that some use encoders and some decoders and some both. How does that work as most will have to understand and also generate text output?
3. Are layers in LLM (page 16) the vertical connected neurons in the LLM? Just to have a visual

feel.
Another thing, I hardly understood the transformer architecture. It went over my head.. (page 11,12)

Tomorrow will start the next chapter.

Unni

From: Swamy Sivasubramaniyan <Swamy.Sivasubramaniyan@usventuresltd.com>
Date: Monday, 3 June 2024 at 8:52 AM
To: Unnikrishnan B (HCL Mainframe CoE) <unnikrishnanb@hcltech.com>
Subject: Re: Overview of AI on Z- Session 1

I attached the wrong PDF; the previous research paper explains SuperGLUE – an LLM evaluation technique.

Regards,

Swamy Sivasubramaniyan
1-650-308-4493 Desk
1-720-239-3720 Mobile
Swamy.Sivasubramaniyan@USVenturesLtd.com

From: Swamy Sivasubramaniyan <Swamy.Sivasubramaniyan@usventuresltd.com>
Date: Sunday, June 2, 2024 at 9:11 PM
To: "Unnikrishnan B (HCL Mainframe CoE)" <unnikrishnanb@hcltech.com>
Subject: Re: Overview of AI on Z- Session 1

Good morning Unni,
I started looking for good sources to understand LLM and how to add additional domain-specific training to an open-source model.
I landed on this book: "Building LLM Powered Applications" I have cloned the git repository with all the sample code (Jupyter notebooks) - https://github.com/sswamyn/Building-LLM-Powered-Applications

I'm thrilled to share that I've just completed Chapter 1 of the book. 😊 I've attached the PDF version with my highlights and notes for your Reference.
I have a PDF version of the book since I bought it. Please be careful when sharing the PDF since it is directly linked to my account.

Regards,

Swamy Sivasubramaniyan

1-650-308-4493 Desk
1-720-239-3720 Mobile
Swamy.Sivasubramaniyan@USVenturesLtd.com

From: Swamy Sivasubramaniyan <Swamy.Sivasubramaniyan@usventuresltd.com>
Date: Saturday, June 1, 2024 at 7:49 PM
To: "Unnikrishnan B (HCL Mainframe CoE)" <unnikrishnanb@hcltech.com>
Subject: Re: Overview of AI on Z- Session 1

I found a short video on “Fine-tuning LLM,” which is exactly what you are looking for.
It is a very high-level one-pager since it should give you some talking points. 😊

Regards,

Swamy Sivasubramaniyan
1-650-308-4493 Desk
1-720-239-3720 Mobile
Swamy.Sivasubramaniyan@USVenturesLtd.com

From: "Unnikrishnan B (HCL Mainframe CoE)" <unnikrishnanb@hcltech.com>
Date: Friday, May 31, 2024 at 9:30 PM
To: Swamy Sivasubramaniyan <Swamy.Sivasubramaniyan@usventuresltd.com>
Subject: FW: Overview of AI on Z- Session 1