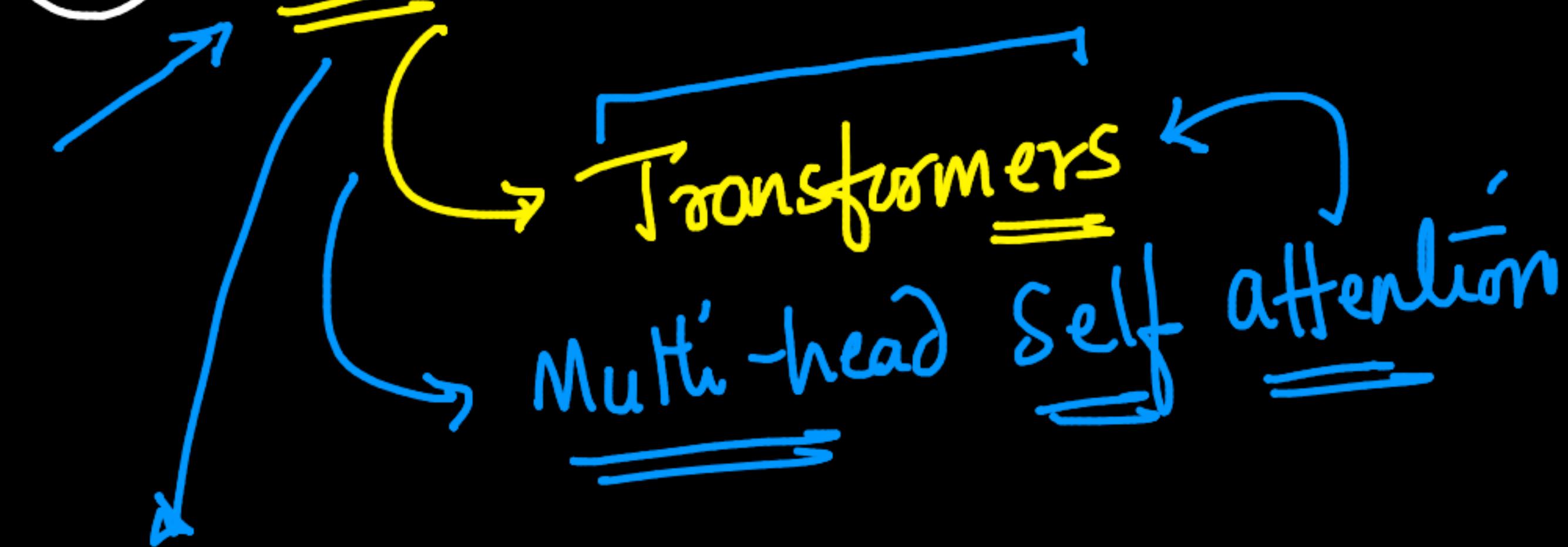


Agenda:

① NER → Bi-LSTM + CRF (code)

② "Attention"-mechanism → LSTMs



Terminology

+ Code + Text

```
[ ] print('\n  ')
[ ] print('After processing, labels:\n', y[0])
```

Connect ▾

(NER) → Health / Phase

Raw Sample:

Thousands of demonstrators have marched through London to protest the war in Iraq and demand the withdrawal of British troops from that

Raw Label:

0 0 0 0 0 0 B-geo 0 0 0 0 0 B-geo 0 0 0 0 0 B-geo 0 0 0 0 0

After processing, sample:

After processing, labels:

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
```

+ Code + Text

```
[ ] print('\n ')
print('After processing, labels:\n', y[0])
```

{x} Raw Sample:

Thousands of demonstrators have marched through London to protest the war in Iraq and demand the withdrawal of British troops from that

{x} Raw Label:

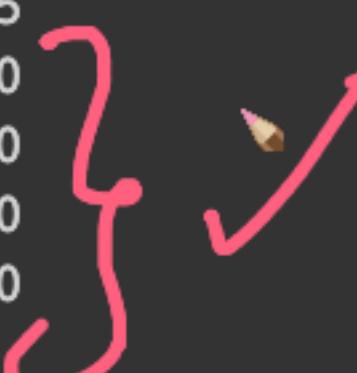
0 0 0 0 0 0 B-geo 0 0 0 0 0 B-geo 0 0 0 0 0 B-gpe 0 0 0 0 0

After processing, sample:

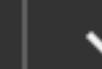
```
[10207 1277 6629 20616 15605 14764 14515 12779 1278 15048 17680 20918
23613 32038 13909 15048 6336 1277 28724 32859 9404 3914 7714 5905
 0      0      0      0      0      0      0      0      0      0      0      0
 0      0      0      0      0      0      0      0      0      0      0      0
 0      0      0      0      0      0      0      0      0      0      0      0
 0      0      0      0      0      0      0      0      0      0      0      0
 0      0      0      0]
```

After processing, labels:

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
```



+ Code + Text

Connect |  

```
[ ] # from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
#from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                   input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 75)]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectiona l1)	(None, 75, 100)	28400

+ Code + Text

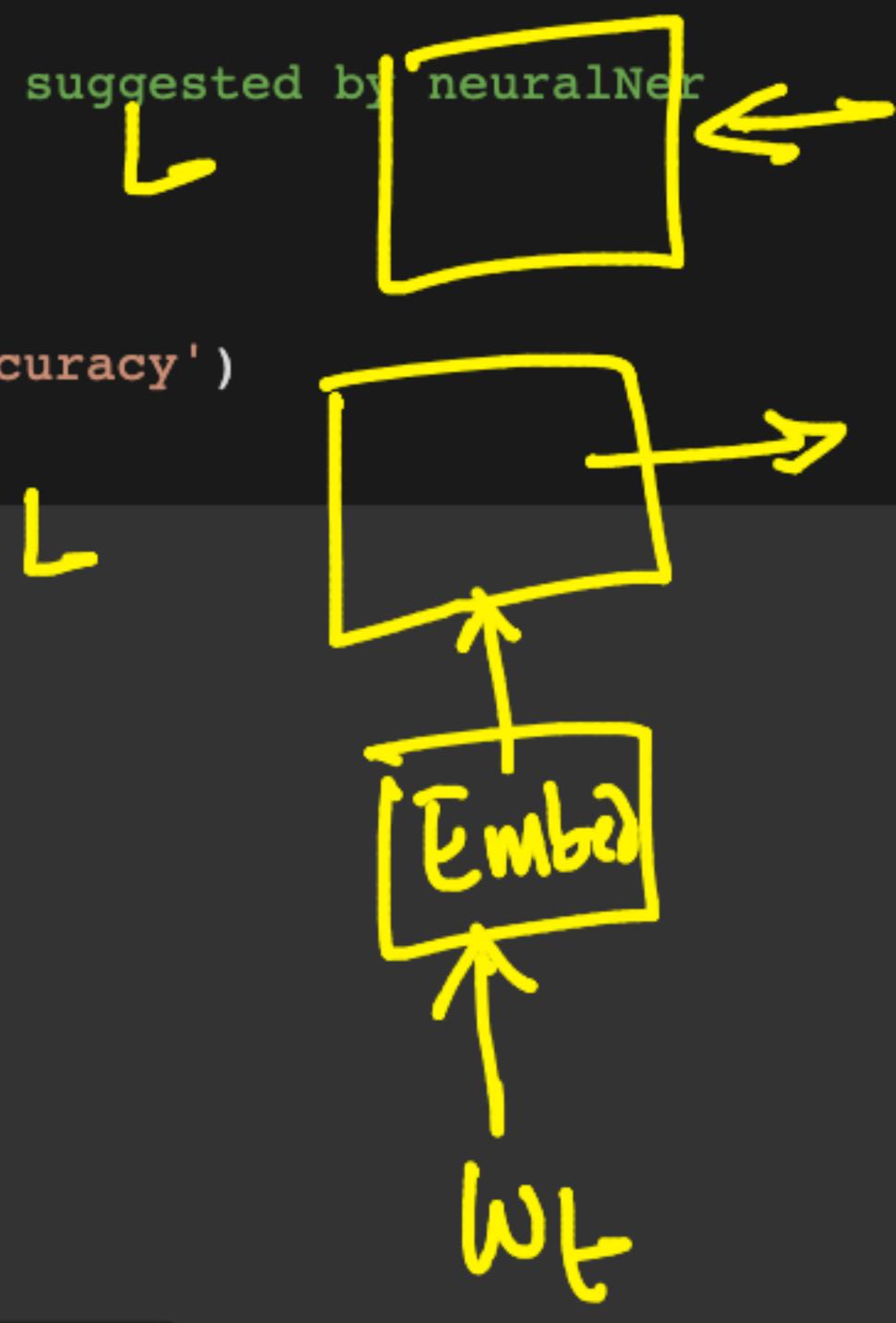
Connect |  Update

```
[ ] from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
# from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                   input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"

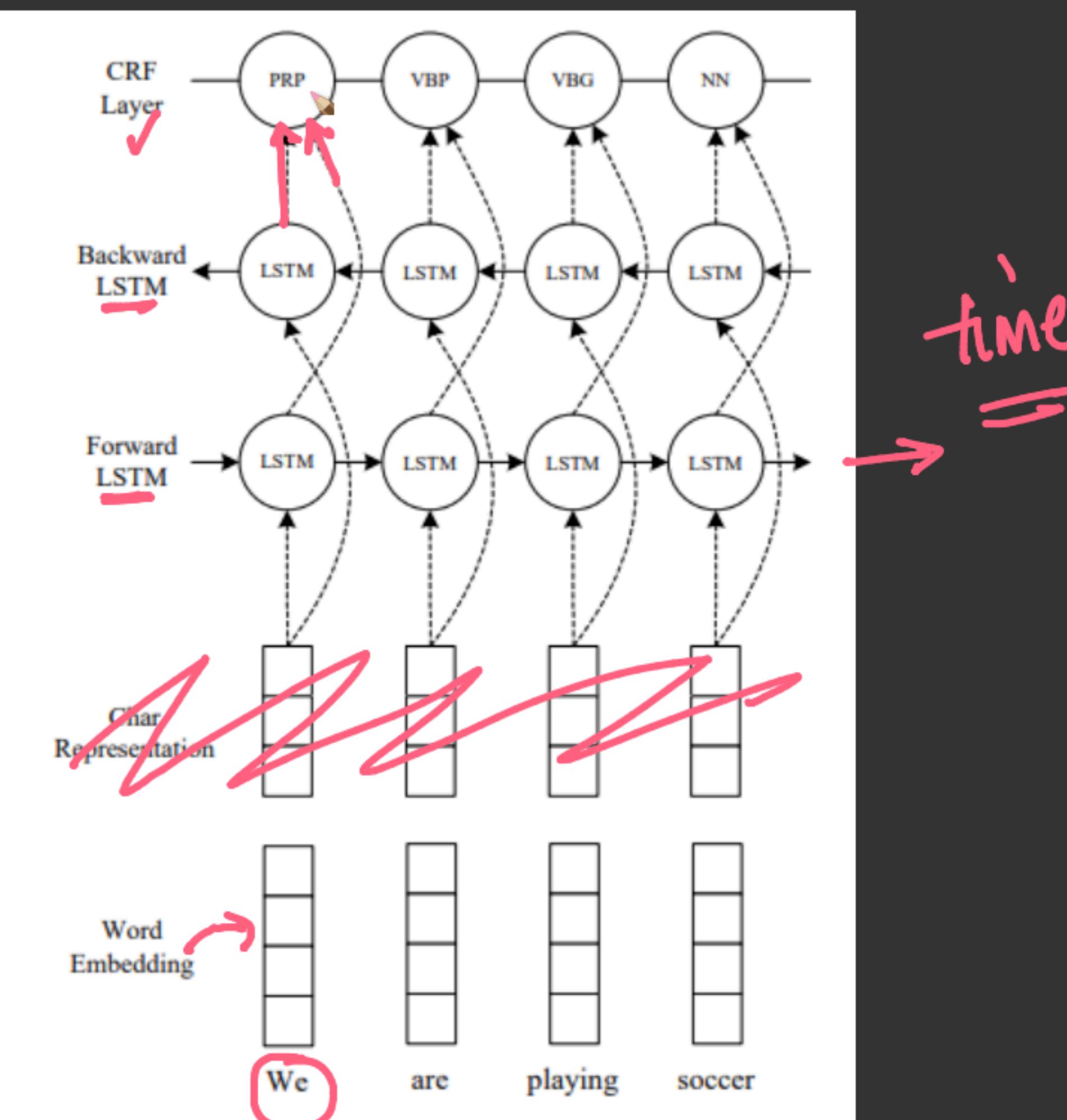
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 75]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectiona l1)	(None, 75, 100)	28400
time_distributed (TimeDistr	(None, 75, 50)	5050



+ Code + Text

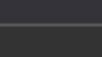
Connect | User Settings | More

- Therefore, we model label sequence jointly using a conditional random field (CRF) instead of decoding each label independently.



Hvbrid

+ Code + Text

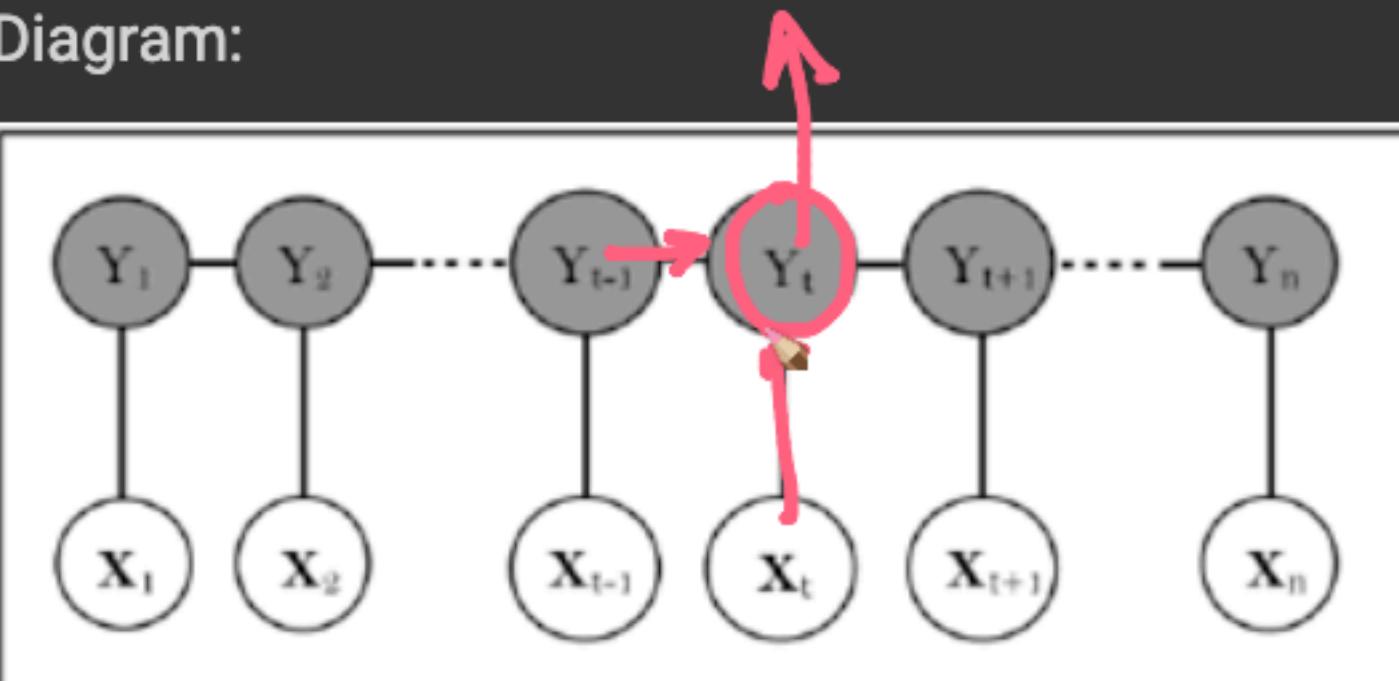
Connect |  

called as feature functions that will assist in generating unique features.

- These features function can consider any logic(depends on programmer) but the output has to be either True:1 or False:0.

Optional: Math of Linear CRF

Diagram:



Fundamental Equation:

Consider the below formula:

CRF:

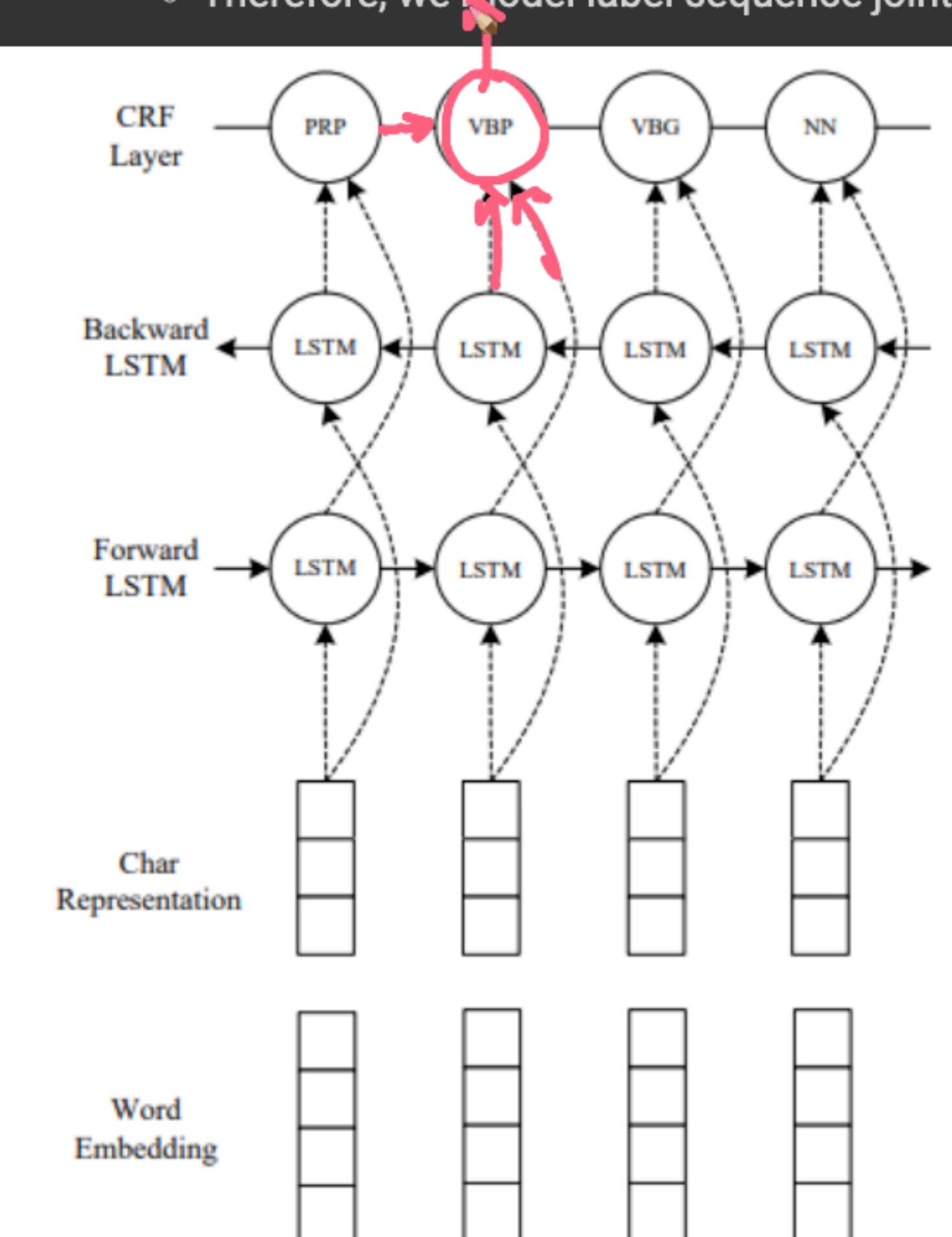
$$p_{\theta}(y|x) = \frac{\exp(\sum_j w_j F_j(x, y))}{\sum_{y'} \exp(\sum_j w_j F_j(x, y'))}$$

, where $F_j(x, y) = \sum_i^L f_j(y_{i-1}, y_i, x, i)$

+ Code + Text

sentence.

- For example, in POS tagging an adjective is more likely to be followed by a noun than a verb,
- In NER with standard BIO2 annotation I-ORG cannot follow I-PER.
- Therefore, we model label sequence jointly using a conditional random field (CRF) instead of decoding each label independently.



+ Code + Text

```
[ ] Downloading tensorflow_addons-0.18.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
[ ] | 1.1 MB 6.7 MB/s
```

```
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (from tensorflow-addons) (2.7.1)
```

```
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from tensorflow-addons) (21.3)
```

```
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->tensorflow-addons) (3.0)
```

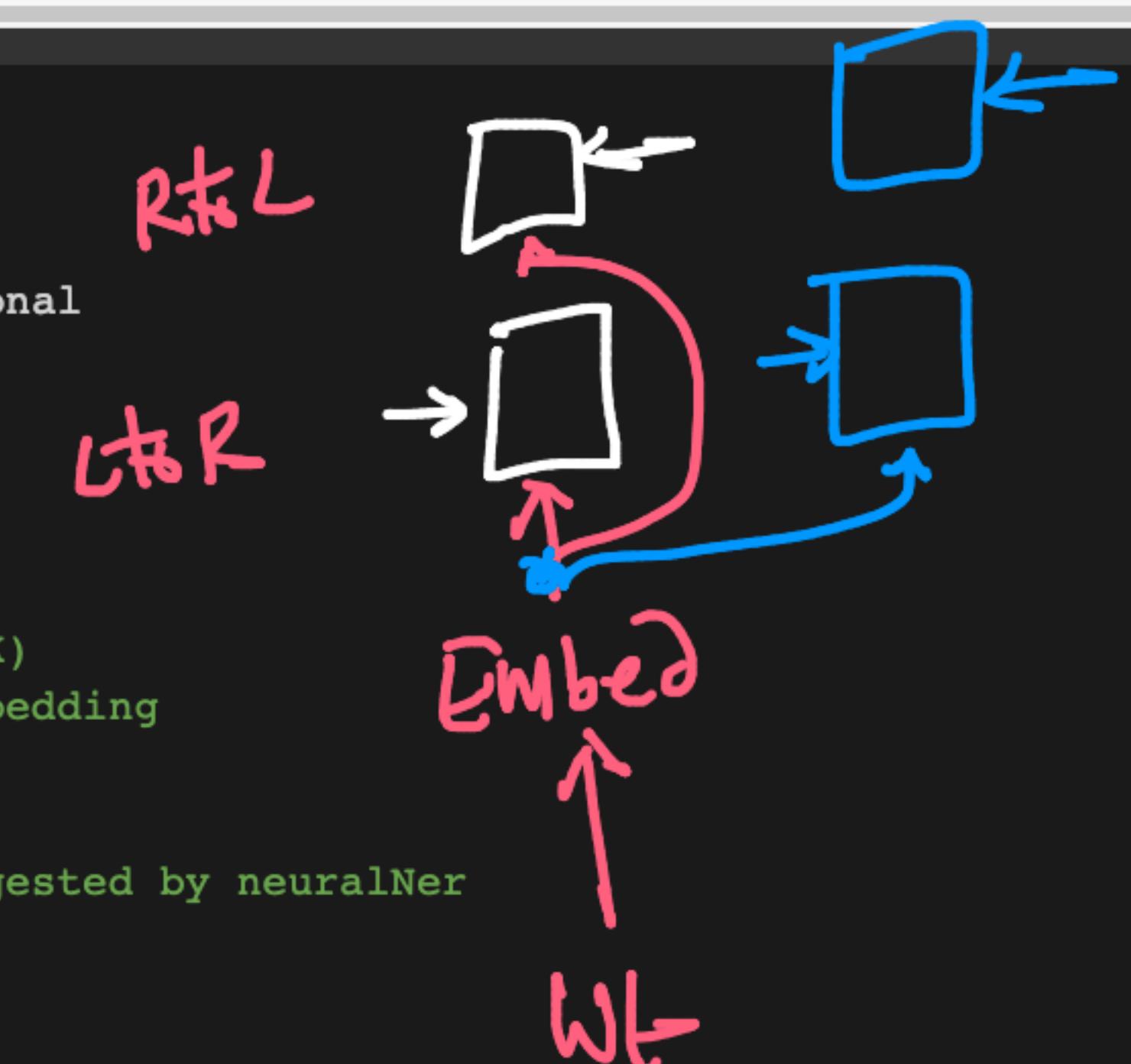
```
Installing collected packages: tensorflow-addons
```

```
Successfully installed tensorflow-addons-0.18.0
```

```
[ ] # from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
# from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                   input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"



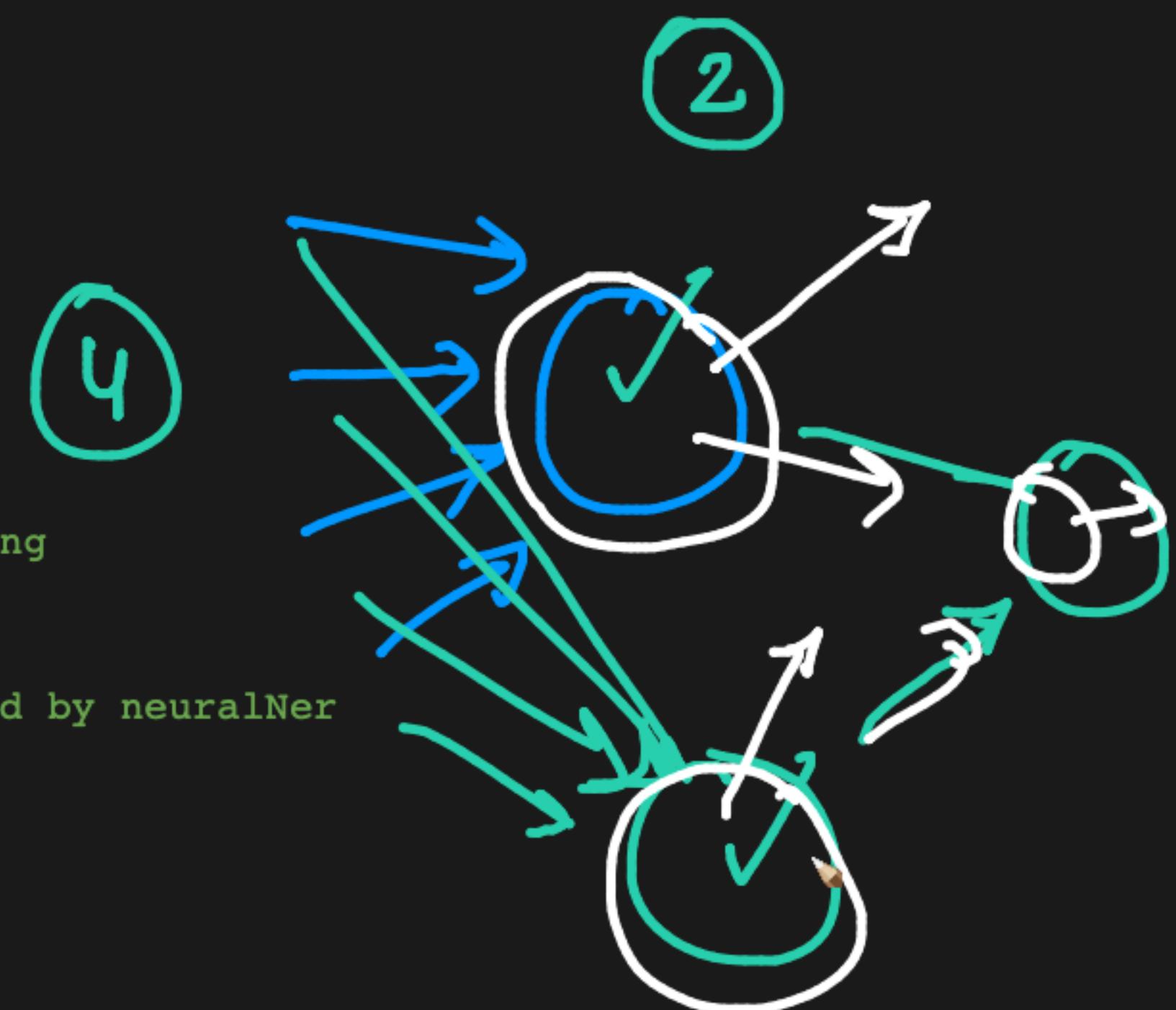
[] Downloading tensorflow_addons-0.18.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB) |██████████| 1.1 MB 6.7 MB/s

```
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (from tensorflow_addons) (2.7.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from tensorflow_addons) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->tensorflow_addons)
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.18.0
```

```
[ ] # from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
# from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                   input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"



colab.research.google.com/drive/1XGu6sNmU2PN7bD6PsG30kM_nN6GsgYa5#scrollTo=ftiqfF8BLroi

+ Code + Text Connect OK

SUCCESSFULLY INSTALLED TENSORFLOW-ADDITIONS-V1.0.0

```
[ ] # from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
# from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                   input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 75]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectiona	(None, 75, 100)	29400

colab.research.google.com/drive/1XGu6sNmU2PN7bD6PsG30kM_nN6GsgYa5#scrollTo=BfaepzMjLrog

+ Code + Text

Successfully installed tensorflow-addons==0.10.0

Q {x} C

from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
#from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
 input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
 recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 75]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectional)	(None, 75, 100)	29400

12/13

Connect |  

L08_NER_re.ipynb - Colaboratory | tfa.rnn.LayerNormLSTMCell | 1603.05118.pdf | Intro_to_Attention_Mechanism | Attention (machine learning) | LSTM layer | LSTM tensorflow | colab.research.google.com/drive/1XGu6sNmU2PN7bD6PsG30kM_nN6GsgYa5#scrollTo=BfaepzMjLrog | Update

+ Code + Text

SUCCESSFULLY INSTALLED TENSORFLOW-ADDITIONS-V1.0.0

```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
# from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                   input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNet
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 75]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectiona	(None, 75, 100)	29400

✓ Functional AP ✓

```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
# from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                   input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()

Model: "model"

Layer (type)           Output Shape        Param #
=====                ======             =====
input_1 (InputLayer)   [(None, 75)]        0
embedding (Embedding)  (None, 75, 20)      703600
bidirectional (Bidirectiona (None, 75, 100)  29400

```

Diagram illustrating a Bidirectional LSTM architecture:

- Input:** Input sequence of length MAX_LEN.
- Embedding:** Input is embedded into a 20-dimensional space (labeled "wt").
- LSTM Layers:** Two parallel LSTM layers process the sequence in opposite directions. Each layer has 50 units (labeled "Lstm").
- Concatenation:** The outputs from both LSTM layers are concatenated at each time step.
- Dense Layer:** A dense layer with 50 units (ReLU activation) is applied across all time steps.
- CRF Layer:** A CRF layer (n_tags+1) is used to produce the final output.

colab.research.google.com/drive/1XGu6sNmU2PN7bD6PsG30kM_nN6GsgYa5#scrollTo=BfaepzMjLrog

+ Code + Text

Successfully installed tensorflow-addons==0.10.0

Run Cell

```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
#from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                  input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 75]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectional)	(None, 75, 100)	29400

15 / 16

colab.research.google.com/drive/1XGu6sNmU2PN7bD6PsG30kM_nN6GsgYa5#scrollTo=BfaepzMjLrog

+ Code + Text

Successfully installed tensorflow-addons==0.10.0

Run Cell

```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
#from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                  input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 75]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectional)	(None, 75, 100)	29400

17 / 17

`+ Code + Text`

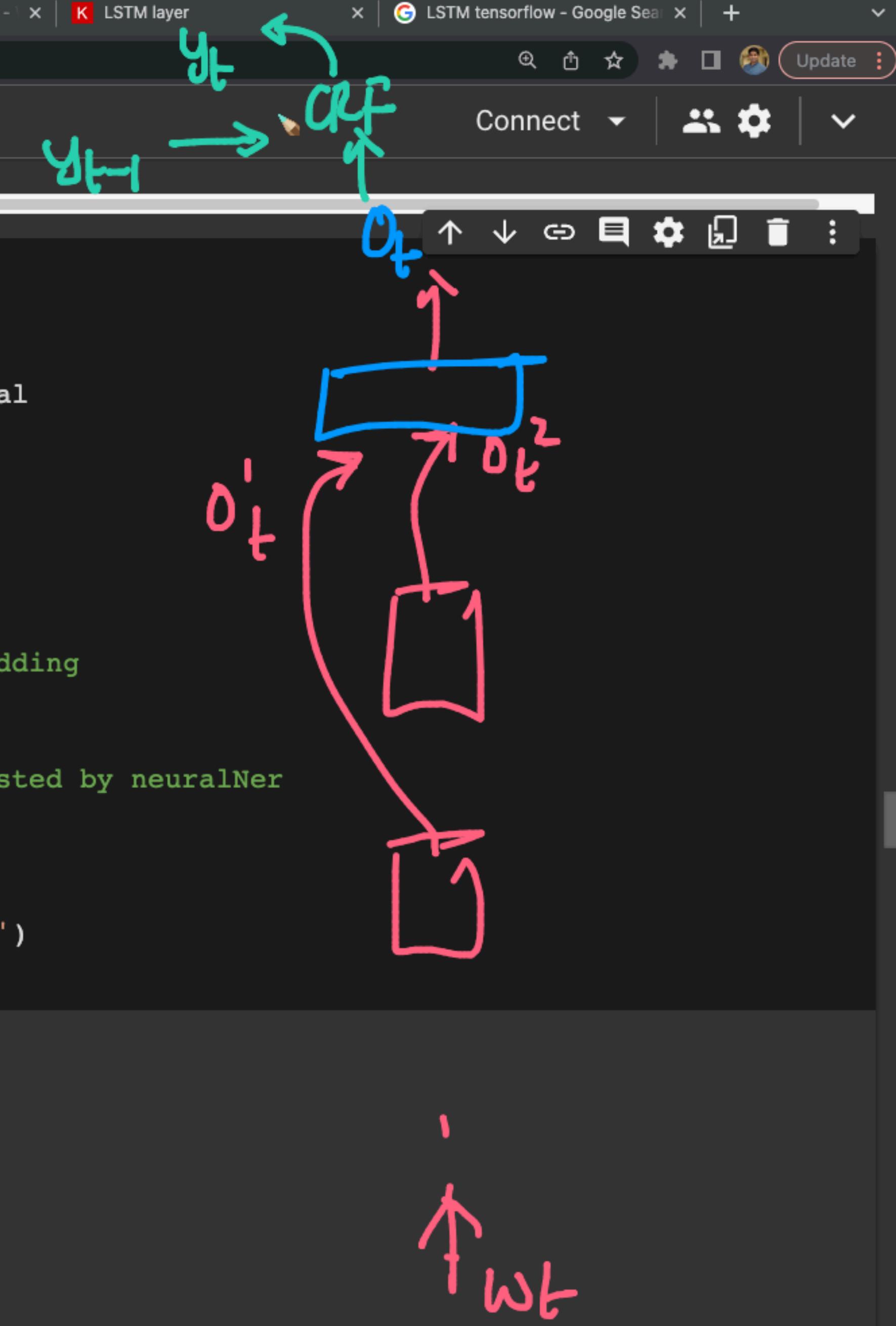
SUCCESSFULLY INSTALLED TENSORFLOW-ADDITIONS-V1.10.0

```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
# from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                  input_length=MAX_LEN, mask_zero=True)(input) # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model) # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model) # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1) # CRF layer, n_tags+1(PAD)
out = crf(model) # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics = 'accuracy')
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 75]	0
embedding (Embedding)	(None, 75, 20)	703600
bidirectional (Bidirectiona	(None, 75, 100)	29400



+ Code + Text

1. drugName

2. condition

3. review Let's take a look at one of review, and understand how a SME would have labelled it manually.

```
[ ] print('Review Number:', raw_df['uniqueID'][0])
print('Drug Name:', raw_df['drugName'][0])
print('Review:', raw_df['review'][0])
```

Sentence → NER

```
Review Number: 206461.0
Drug Name: Valsartan
Review: "It has no side effect, I take it in combination of Bystolic 5 Mg and Fish Oil"
```

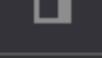
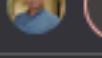
And we see, for review number 206461, there were **no side-effects or problem** being caused by the drug Valsartan. Let's take a look at another example.

```
[ ] print('Review Number:', raw_df['uniqueID'][15])
print('Drug Name:', raw_df['drugName'][15])
print('Review:', raw_df['review'][15])
```

```
Review Number: 81890.0
Drug Name: Liraglutide
Review: "I have been taking Saxenda since July 2016. I had severe nausea for about a month once I got up to the 2.6 dosage. It has since
```

Liraglutide is a molecule used to treat type 2 diabetes and chronic weight management. Now, we can see there are some side-effects attached to it

+ Code + Text

Connect |  

Disease		
	S800	s800
Clinical	i2b2-2010	i2b2
	Radiology	radiology

SPECIES
PROBLEM, TEST, TREATMENT
ANATOMY, OBSERVATION, ANATOMY_MODIFIER,
OBSERVATION_MODIFIER, UNCERTAINTY

The 16 entity types in the BioNLP13CG model include, some of which are:

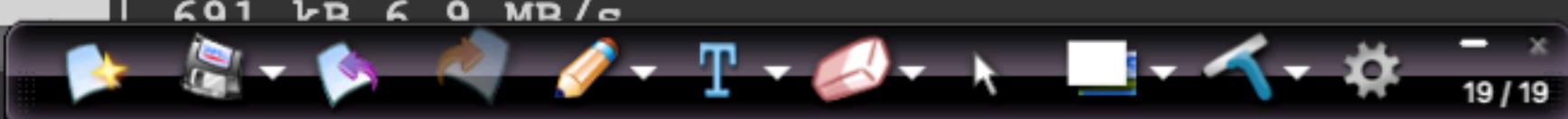
- ✓ • ORGAN
- ✓ • ORGANISM
- ✓ • PROBLEM
- ✓ • TREATMENT
- DISEASE
- ✓ • CHEMICAL

Let's import stanza

```
[ ] !pip install stanza
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting stanza
```

```
  Downloading stanza-1.4.2-py3-none-any.whl (691 kB)
```



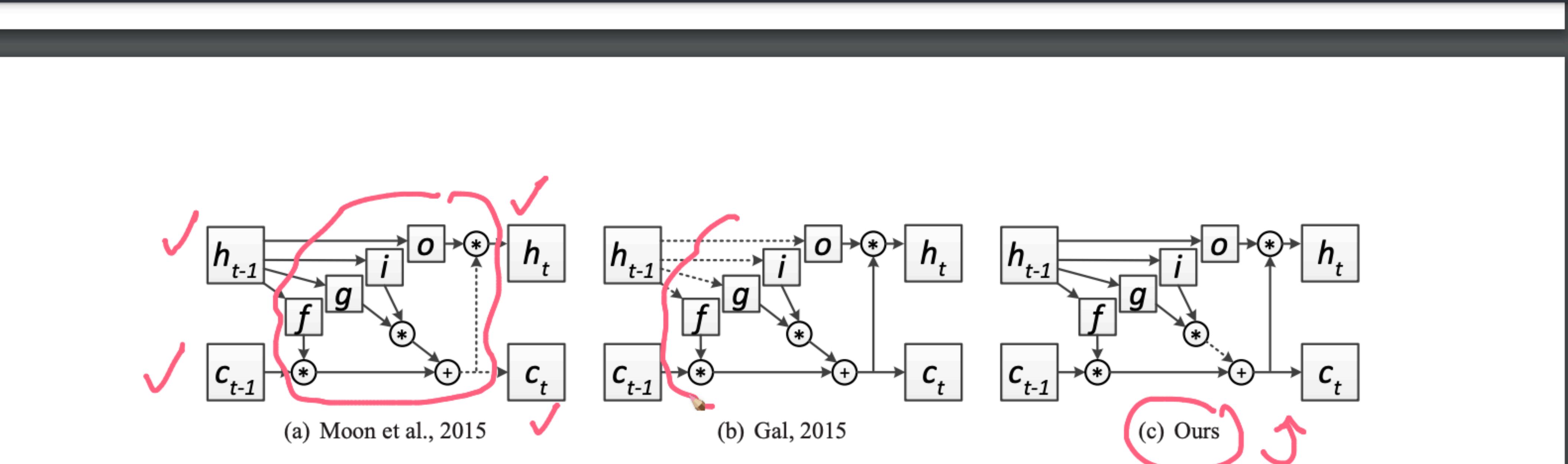


Figure 1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propose to apply dropout to the hidden state updates vector \mathbf{g}_t :

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * d(\mathbf{g}_t) \quad (13)$$

We found that previous hidden states correctly as proposed in the paper by Moon et al. (2015) have achieved an

the network is able to learn long-term dependencies, it is not capable of exploiting them during test phase. Note that our assumption of all gates being equal to 1 helps the network to preserve hidden state, since in a real network gate values lie within $(0, 1)$ interval. In practice trained networks tend to saturate gate values (Karpathy et al., 2015)

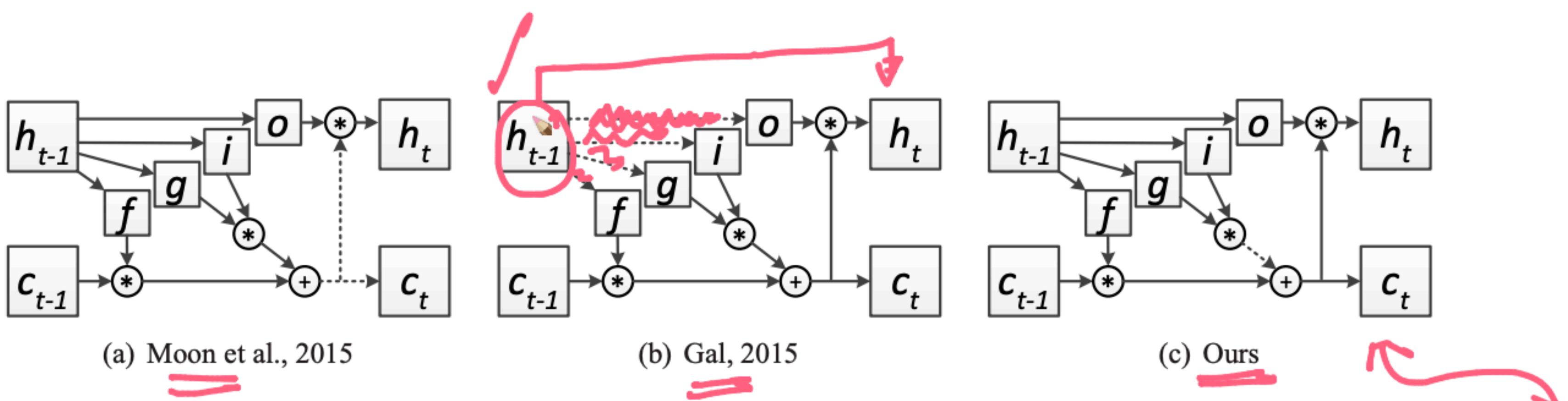


Figure 1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propose to apply dropout to the hidden state updates vector \mathbf{g}_t :

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * d(\mathbf{g}_t) \quad (13)$$

We found that an image of a face with a neutral expression can behave as binary switches. Previous hidden states correctly as proposed in Figure 1. The final Montreal (2015) have achieved an

the network is able to learn long-term dependencies, it is not capable of exploiting them during test phase. Note that our assumption of all gates being equal to 1 helps the network to preserve hidden state, since in a real network gate values lie within $(0, 1)$ interval. In practice trained networks tend to saturate gate values (Karpathy et al., 2015)

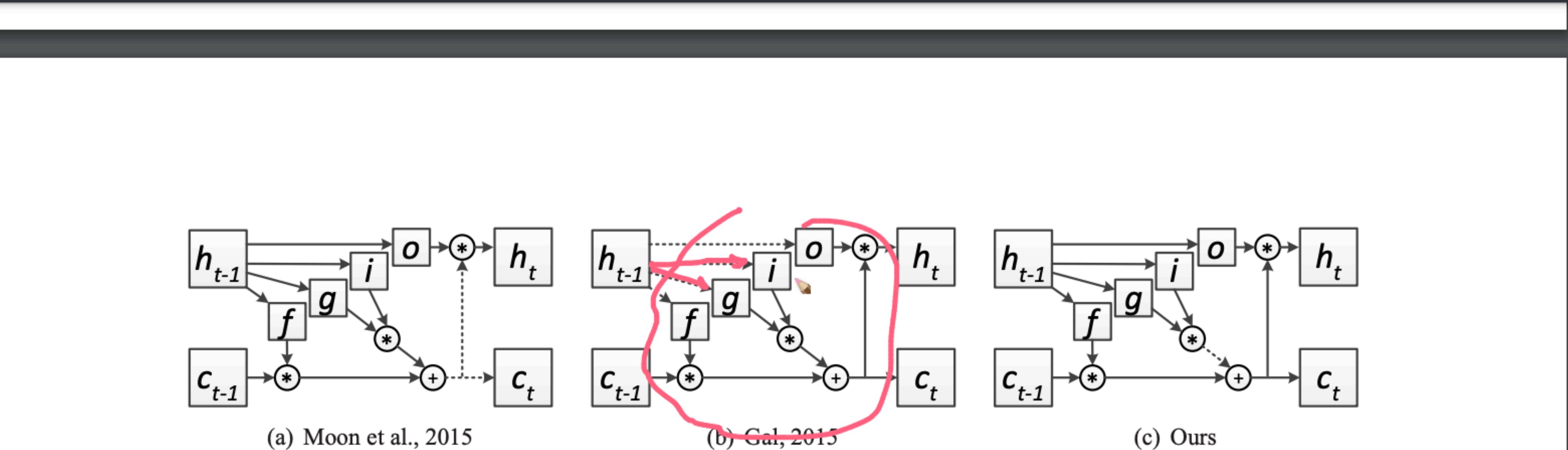


Figure 1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propose to apply dropout to the hidden state updates vector \mathbf{g}_t :

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * d(\mathbf{g}_t) \quad (13)$$

the network is able to learn long-term dependencies, it is not capable of exploiting them during test phase. Note that our assumption of all gates being equal to 1 helps the network to preserve hidden state, since in a real network gate values lie within $(0, 1)$ interval. In practice trained networks tend to saturate gate values (Karpathy et al., 2015)

We found that applying dropout to previous hidden states directly as proposed in the racing car (Moon et al., 2015) have achieved an

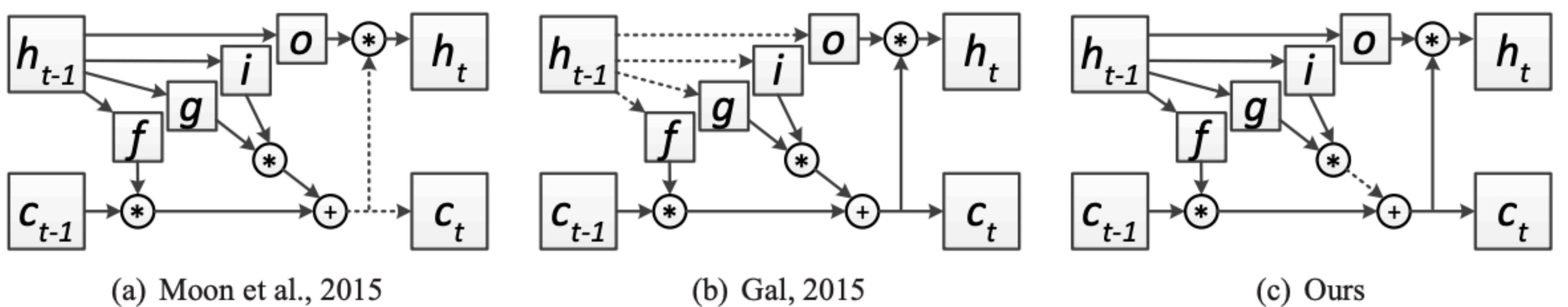


Figure 1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propose to apply dropout to the hidden state updates vector \mathbf{g}_t :

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * d(\mathbf{g}_t) \quad (13)$$

We found that applying dropout to previous hidden states directly as proposed in the racing car (Moon et al., 2015) have achieved an

the network is able to learn long-term dependencies, it is not capable of exploiting them during test phase. Note that our assumption of all gates being equal to 1 helps the network to preserve hidden state, since in a real network gate values lie within $(0, 1)$ interval. In practice trained networks tend to saturate gate values (Karpathy et al., 2015)

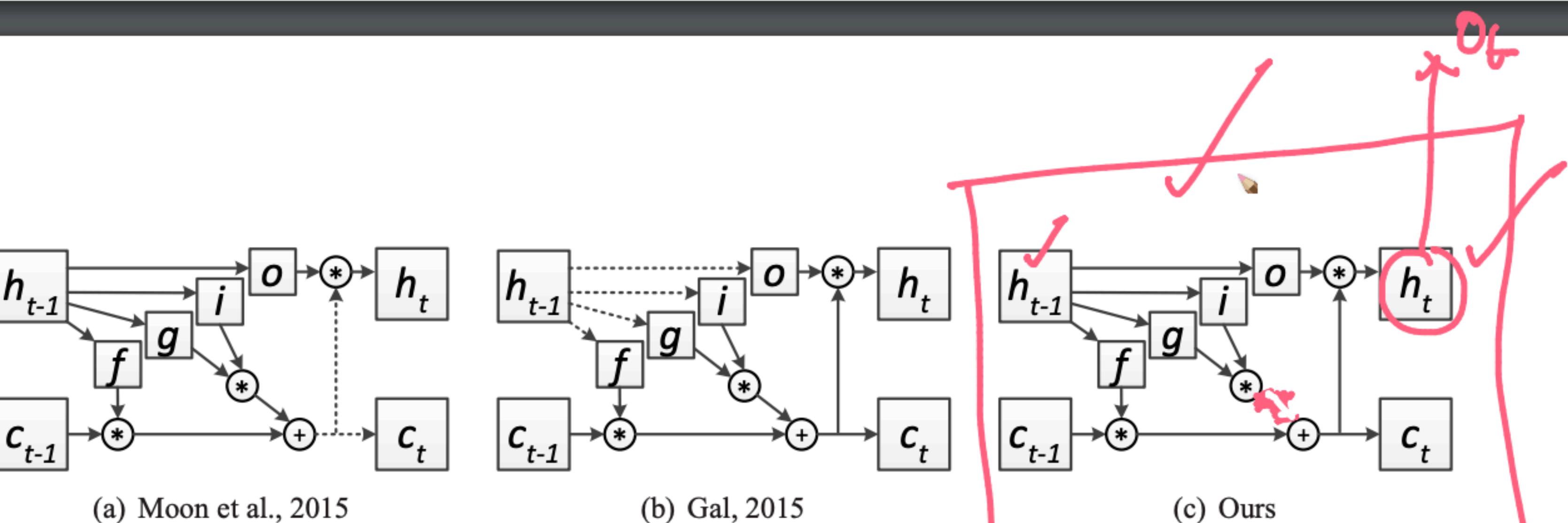


Figure 1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propose to apply dropout to the hidden state updates vector \mathbf{g}_t :

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * d(\mathbf{g}_t) \quad (13)$$

We found that an image of a single neuron can behave as binary switches. Previous hidden states directly as proposed in the original Moon et al. (2015) have achieved an

the network is able to learn long-term dependencies, it is not capable of exploiting them during test phase. Note that our assumption of all gates being equal to 1 helps the network to preserve hidden state, since in a real network gate values lie within $(0, 1)$ interval. In practice trained networks tend to saturate gate values (Karpathy et al., 2015)

+ Code + Text

Name: drugName, dtype: int64

So we see the top 5 drugName with the highest number of side-effects are :

- Trazodone
- Lamotrigine
- Tamsulosin
- Amitriptyline
- Dulcolax

Scope for improvement

1. Use BIO-Bert for Relationship extraction We can see some the problems are solved by that Drug as well, so if we can establish a relationship between the drugName and the side-effect that will give us a much refined insight about the drug

Further more we can also involve

2. Knowledge Graph for Question and Answering purpose

Conclusion summary

Links:

<https://www.persistent.com/blogs/building-named-entity-recognition-models-for-healthcare/>

<https://arxiv.org/pdf/1511.08308v4.pdf>

A series of handwritten annotations in pink and orange ink. A large orange circle contains the word 'stanza'. An orange arrow points from the word 'Transform' to the circle. To the right of the circle is the text 'CRF & LSTMs'. Below the circle is the text 'Bi-directional LSTM + CRF' with an orange checkmark. Below that is the text 'Transform' with an orange checkmark. At the bottom left is the text 'i2b2 dataset' with an orange checkmark. There is also a small orange arrow pointing towards the top left.

+ Code + Text

Name: drugName, dtype: int64

So we see the top 5 drugName with the highest number of side-effects are :

- Trazodone
- Lamotrigine
- Tamsulosin
- Amitriptyline
- Dulcolax

Scope for improvement

1. Use BIO-Bert for Relationship extraction We can see some the problems are solved by that Drug as well, so if we can establish a relationship between the drugName and the side-effect that will give us a much refined insight about the drug
2. Knowledge Graph for Question and Answering purpose

Conclusion summary

Links:

<https://www.persistent.com/blogs/building-named-entity-recognition-models-for-healthcare/>

<https://arxiv.org/pdf/1511.08308v4.pdf>

1 human expert ✓

sent → Stanza → BiLSTM + QAF

26 / 26

+ Code + Text

Connect | User Settings

Name: drugName, dtype: int64

So we see the top 5 drugName with the highest number of side-effects are :

- Trazodone
- Lamotrigine
- Tamsulosin
- Amitriptyline
- Dulcolax

Scope for improvement

1. Use **BIO-Bert** for Relationship extraction We can see some the problems are solved by that Drug as well, so if we can establish a relationship between the drugName and the side-effect that will give us a much refined insight about the drug

Further more we can also involve

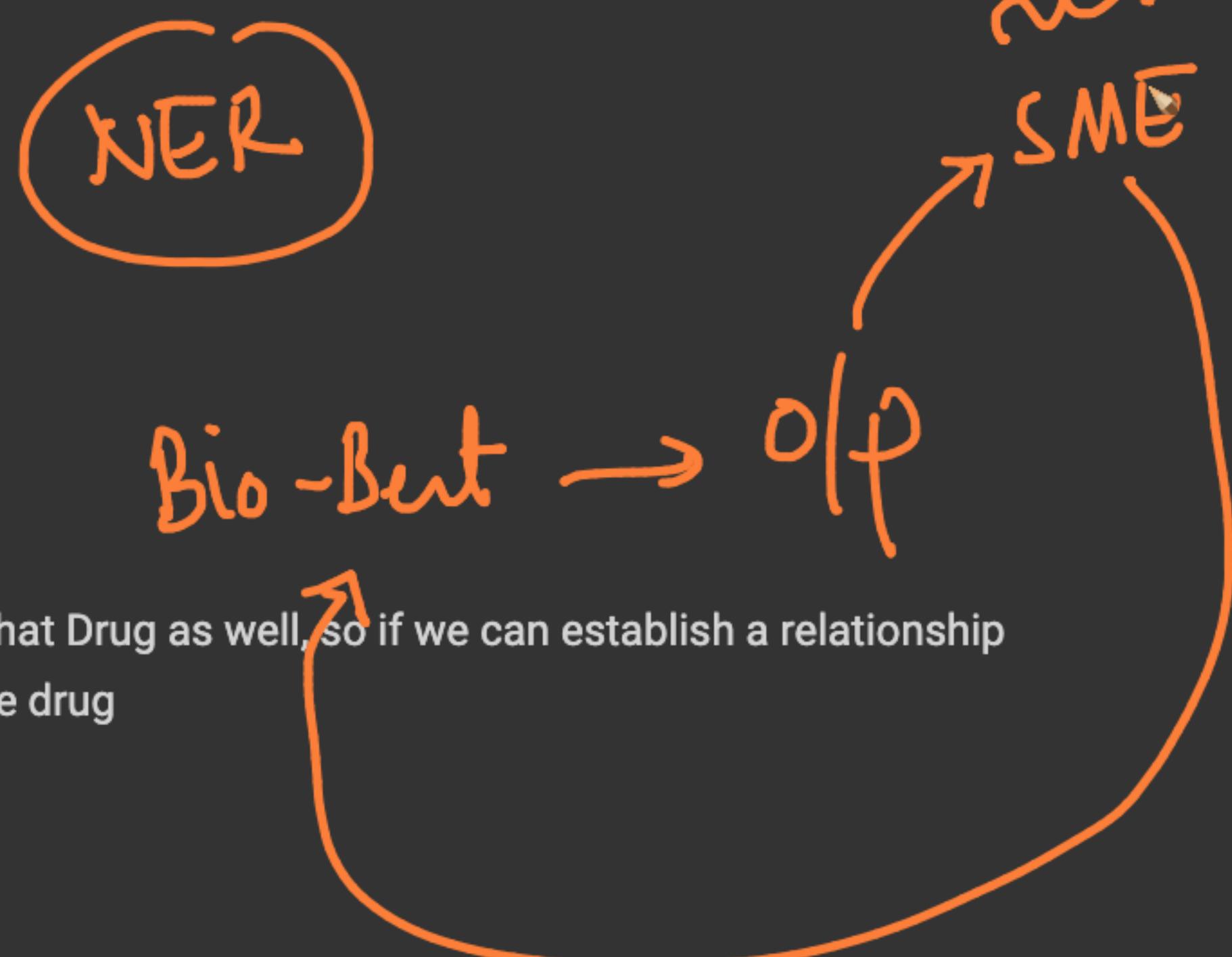
2. Knowledge Graph for Question and Answering purpose

Conclusion summary

Links:

<https://www.persistent.com/blogs/building-named-entity-recognition-models-for-healthcare/>

<https://arxiv.org/pdf/1511.08308v4.pdf>



what is 'aspect' in ML?

- the way in which a problem, idea, etc., may be considered
- like your goal is to do classification based on some aspect. see below examples.

```
INPUT = {'text': 'food is delicious but price is high',  
         'aspect': 'food',  
         }  
  
OUTPUT = {'aspect_Sentiment' : 'Positive'  
         }
```

Problem:

encoder

decoder

positive/neutral

Aspect Based Sentiment Analysis

Input Text

ambience was nice but service was not so great

+ Code + Text

Connect |  

```
[ ] #load glove word embedding vectors
glove_vectors = gensim.downloader.load('glove-twitter-25')
```

1. Get KEY*

SourceSentence = "food is delicious but price is high"

Tokenized = [food, is, delicious, but, price, is, high]

Key = *representation*(*Tokenized*)

Here, *representation* is nothing but a word embedding model or output from recurrent model.

```
[ ] #Input
sentences = ['food', 'is', 'delicious', 'but', 'price', 'is', 'high']

#Dimensionality reduction for easy visualization.
pca = PCA(n_components=5)

# Key sequence
KEY = [glove_vectors[i].tolist() for i in sentences]
KEY = pca.fit_transform(KEY)

# PLOT
plt.figure(figsize=(10.5, 7.5))
key_df = pd.DataFrame(np.transpose(np.matrix(KEY)), columns = sentences)
```

+ Code + Text Connect |  Update

```
[ ] #load glove word embedding vectors
glove_vectors = gensim.downloader.load('glove-twitter-25')
```

1. Get KEY

SourceSentence = "food is delicious but price is high"
Tokenized = [food, is, delicious, but, price, is, high]
Key = representation(*Tokenized*)

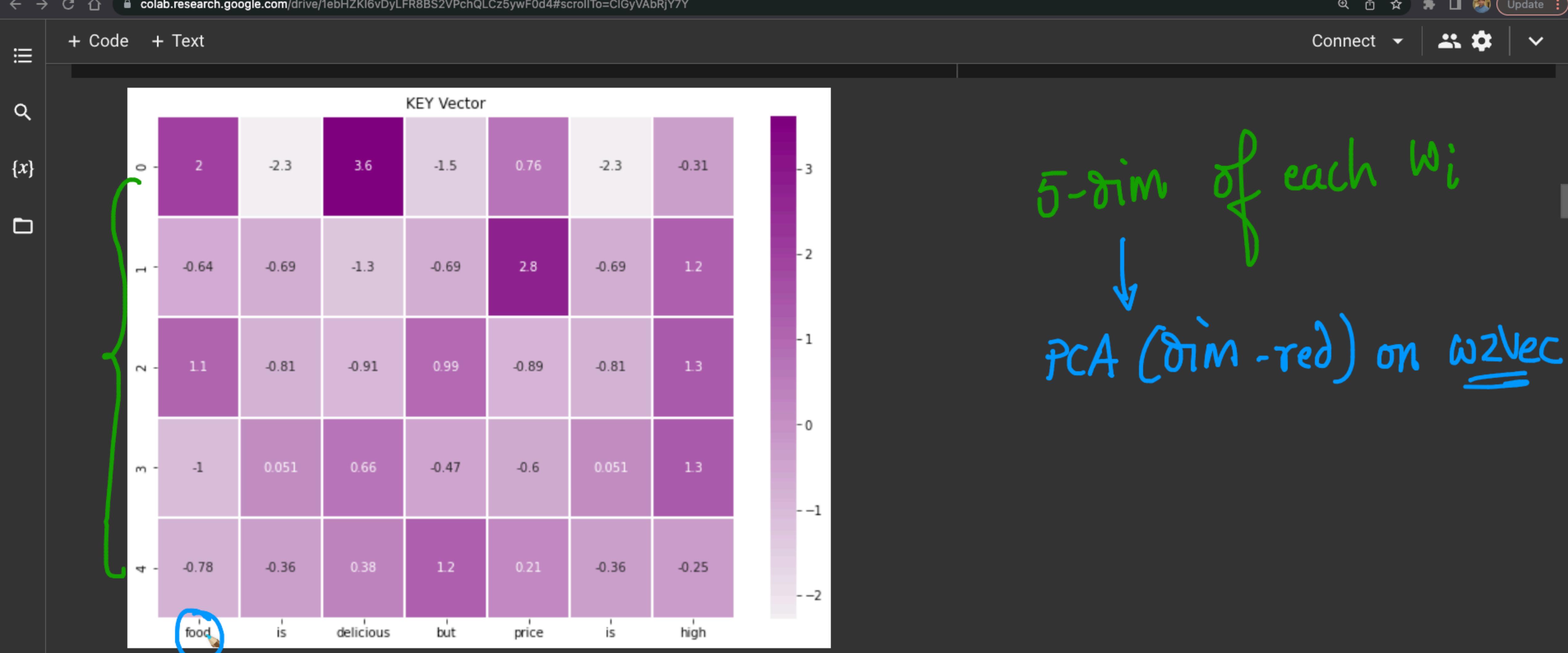
Here, *representation* is nothing but a word embedding model or output from recurrent model.

```
[ ] #Input
sentences = ['food', 'is', 'delicious', 'but', 'price', 'is', 'high']

#Dimensionality reduction for easy visualization.
pca = PCA(n_components=5)

# Key sequence
KEY = [glove_vectors[i].tolist() for i in sentences]
KEY = pca.fit_transform(KEY)

# PLOT
plt.figure(figsize=(10.5, 7.5))
key_df = pd.DataFrame(np.transpose(np.matrix(KEY)), columns = sentences)
```



2. Get Query

```
target = [tasty, cost]
```

+ Code + Text

Connect ▾

I tokenize = [food, is, delicious, but, price, is, high]

Key = representation(Tokenized)

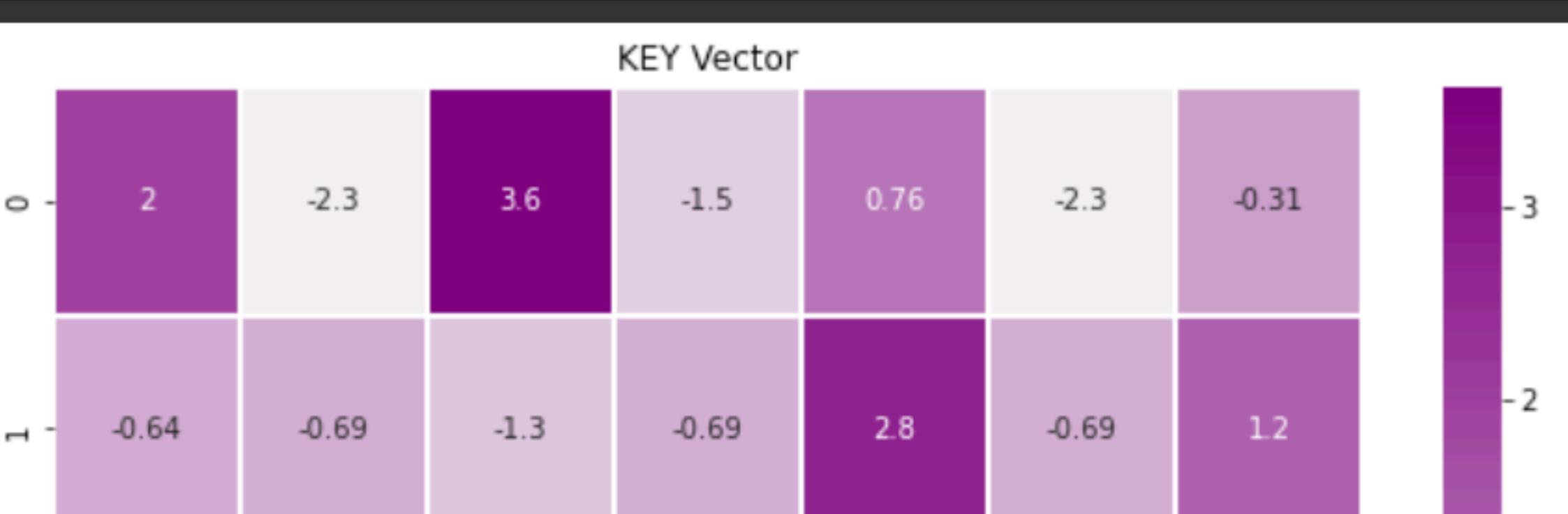
Here, *representation* is nothing but a word embedding model or output from recurrent model.

```
#Input
sentences = ['food', 'is', 'delicious', 'but', 'price', 'is', 'high']

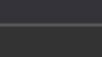
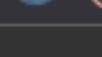
✓ #Dimensionality reduction for easy visualization.
pca = PCA(n_components=5)

# Key sequence ✓
KEY = [glove_vectors[i].tolist() for i in sentences]
KEY = pca.fit_transform(KEY)

# PLOT
plt.figure(figsize=(10.5, 7.5))
key_df = pd.DataFrame(np.transpose(np.matrix(KEY)), columns = sentences)
ax = sns.heatmap(key_df, annot=True, cmap=sns.light_palette("purple", as_cmap=True), linewidths=1, ).set_title('KEY Vector')
```



+ Code + Text

Connect |  

2. Get Query

target : ---

ABSA

target = [tasty, cost]

QUERY = representation(target)

Here, *representation* is nothing but a word embedding model.

#Query text
target = ['cost']

#Query embedding and dimensionality reduction
QUERY = [glove_vectors[i].tolist() for i in target]
QUERY = pca.transform(QUERY)

#Plot
plt.figure(figsize=(10.5, 7.5))
query_df = pd.DataFrame(np.transpose(QUERY), columns = target)
ax = sns.heatmap(query_df, annot=True, cmap=sns.light_palette("grey", as_cmap=True), linewidths=1,).set_title('QUERY Vector',)

QUERY Vector

0.76

-2.0

-1.5

33 / 33

colab.research.google.com/drive/1ebHZKI6vDyLFR8BS2VPchQLCz5ywF0d4#scrollTo=ClGyVAbRjY7Y

+ Code + Text Connect |  

2. Get Query

{x} target = [tasty, cost]
QUERY = representation(target)

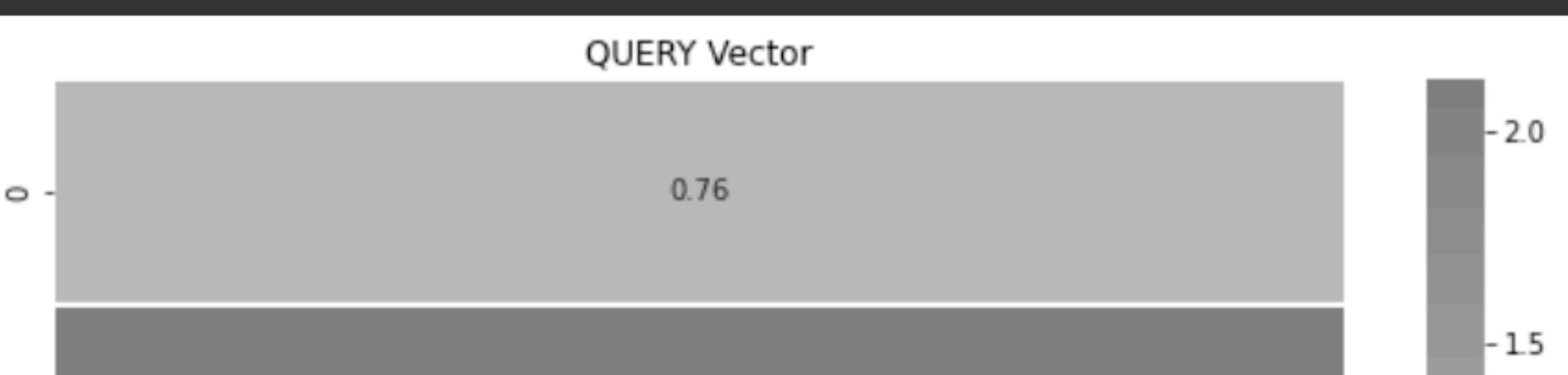
Here, *representation* is nothing but a word embedding model.

```
[ ] #Query text
target = [ 'cost']

#Query embedding and dimensionality reduction
QUERY = [glove_vectors[i].tolist() for i in target]
QUERY = pca.transform(QUERY)

#Plot
plt.figure(figsize=(10.5, 7.5))
query_df = pd.DataFrame(np.transpose(QUERY), columns = target)
ax = sns.heatmap(query_df, annot=True, cmap=sns.light_palette("grey", as_cmap=True), linewidths=1, ).set_title('QUERY Vector', )
```

QUERY Vector

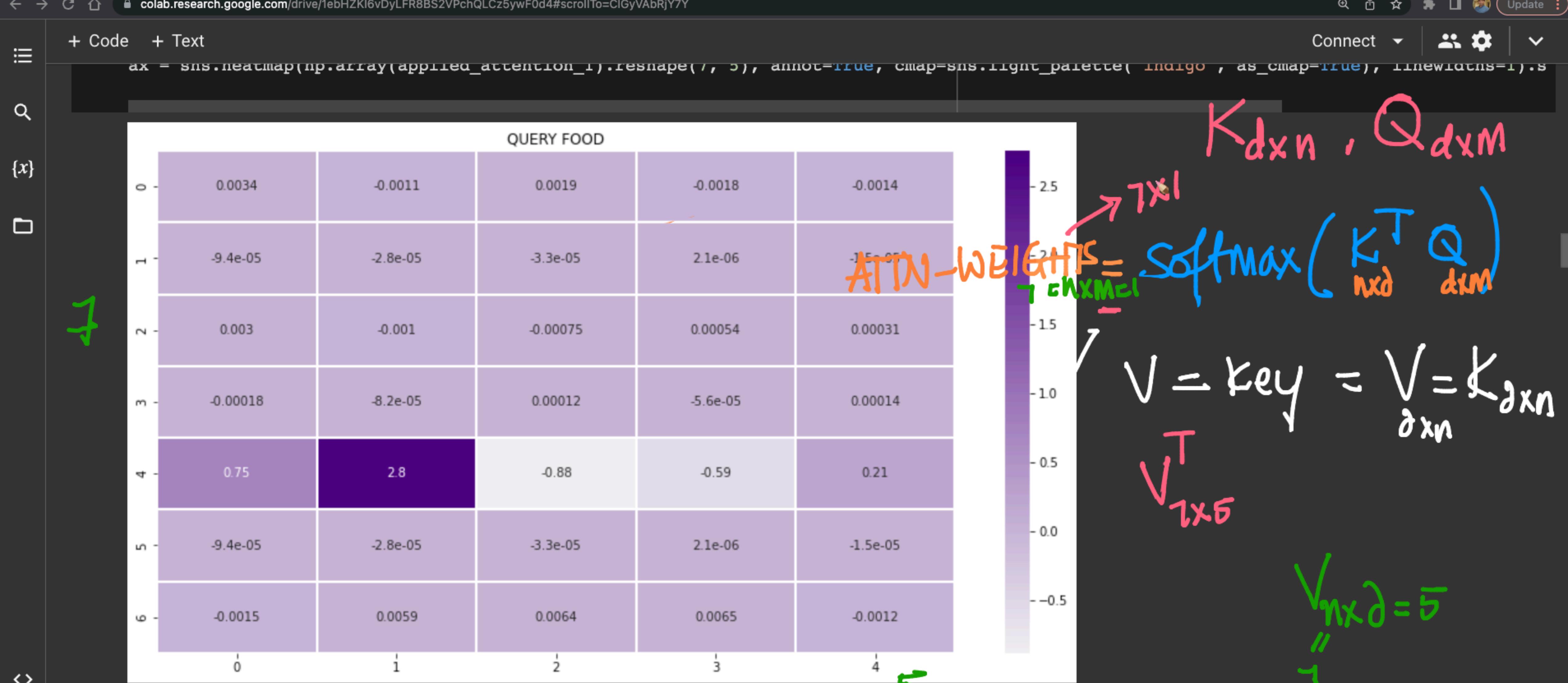


0.76

-2.0
-1.5

34 / 34





```
[ ] def calculate_attention_vector(applied_attention):
    return np.sum(applied_attention, axis=0)
```

+ Code + Text

Connect |  

6. Context Vectors

$A = \text{softmax}(K^T Q)$

```
def apply_attention_scores(attention_weights, annotations):
    # Multiple the annotations by their attention weights
    return [annotations[:, i] * attention_weights[i] for i in range(len(attention_weights))]
```

$\overbrace{V^T}^{A}$

applied_attention_1 = apply_attention_scores(ATN_WEIGHTS, np.transpose(VALUE))

plt.figure(figsize=(14.5, 7.5))
ax = sns.heatmap(np.array(applied_attention_1).reshape(7, 5), annot=True, cmap=sns.light_palette("indigo", as_cmap=True), linewidths=1).s

QUERY FOOD

	0	1	2	3	4
0	0.0034	-0.0011	0.0019	-0.0018	-0.0014
1	-9.4e-05	-2.8e-05	-3.3e-05	2.1e-06	-1.5e-05
2	0.003	-0.001	-0.00075	0.00054	0.00031
3	-0.00018	-8.2e-05	0.00012	-5.6e-05	0.00014

0.5 1.0 1.5 2.0 2.5

37 / 39

+ Code + Text Connect |  

6. Context Vectors

$\text{softmax}(K^T \cdot Q)$

```
def apply_attention_scores(attention_weights, annotations):
    # Multiple the annotations by their attention weights
    return [annotations[:, i] * attention_weights[i] for i in range(len(attention_weights))]

applied_attention_1 = apply_attention_scores(ATTN_WEIGHTS, np.transpose(VALUE))

plt.figure(figsize=(14.5, 7.5))
ax = sns.heatmap(np.array(applied_attention_1).reshape(7, 5), annot=True, cmap=sns.light_palette("indigo", as_cmap=True), linewidths=1).s
```

QUERY FOOD

	0	1	2	3	4
0	0.0034	-0.0011	0.0019	-0.0018	-0.0014
1	-9.4e-05	-2.8e-05	-3.3e-05	2.1e-06	-1.5e-05
2	0.003	-0.001	-0.00075	0.00054	0.00031
3	-0.00018	-8.2e-05	0.00012	-5.6e-05	0.00014

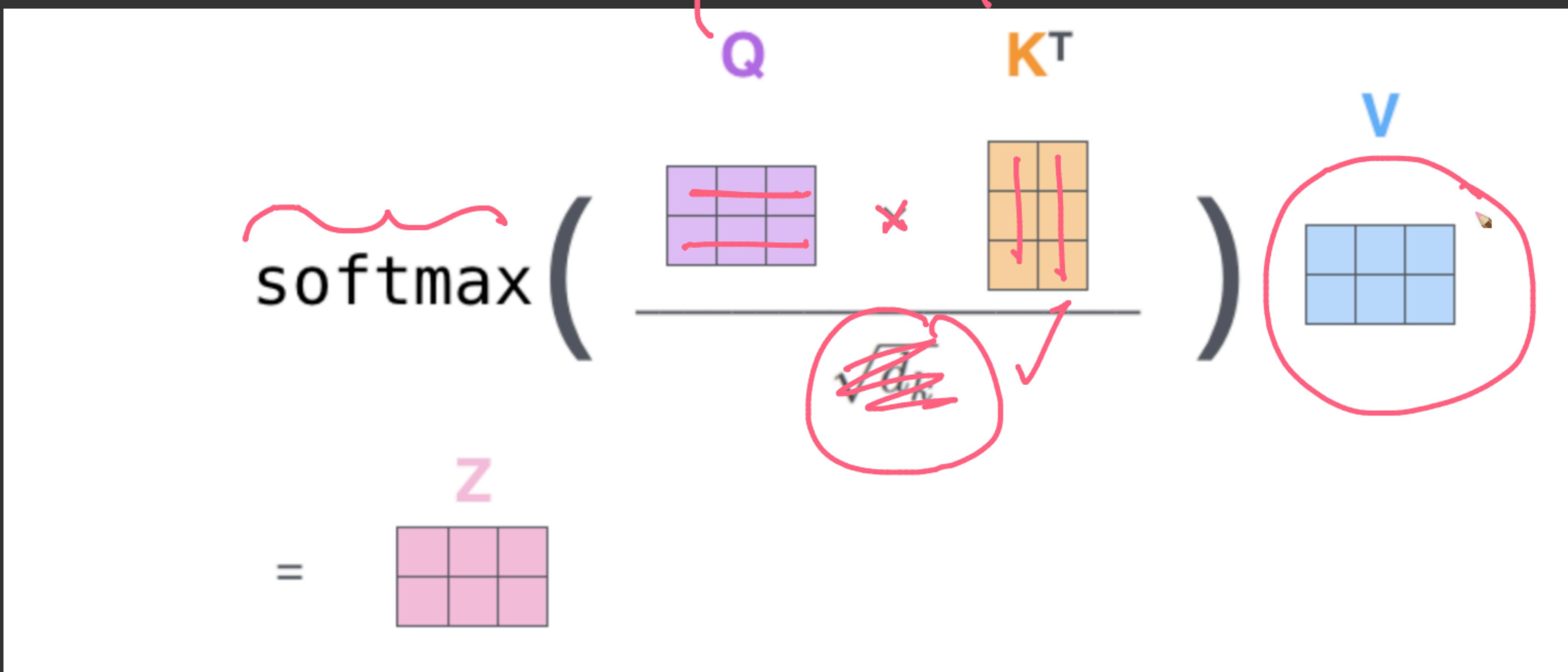
+ Code + Text

$\mathbf{K} = \mathbf{key}$,

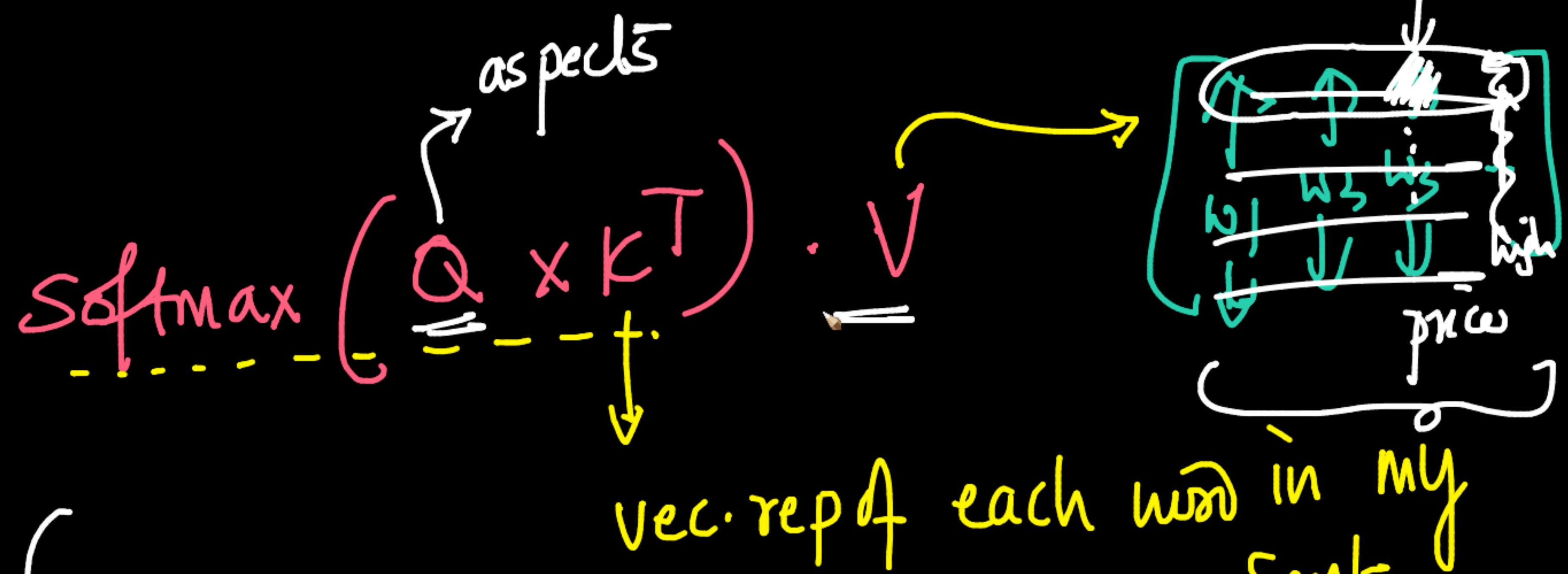
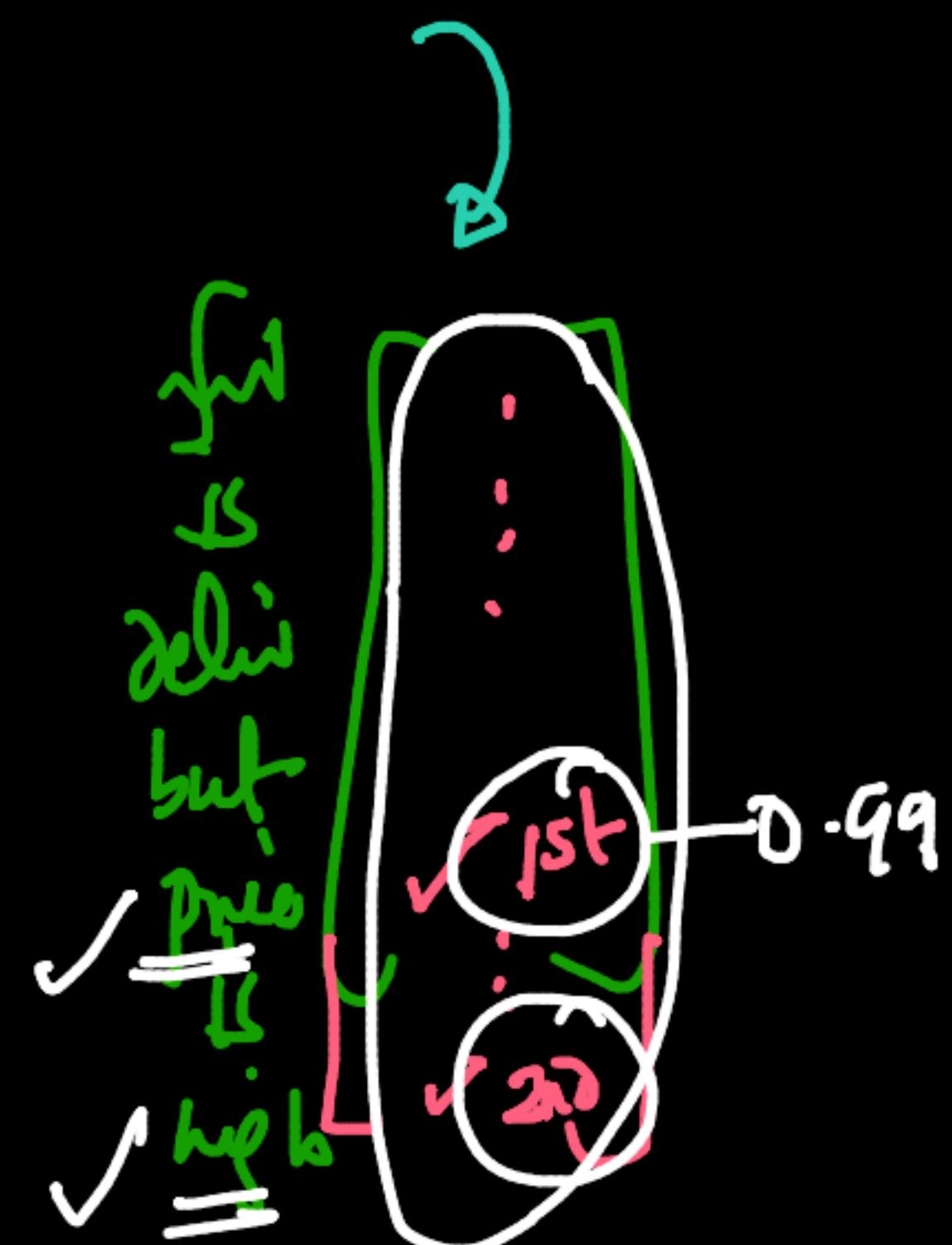
$\mathbf{V} = \mathbf{Value}$,

where $\mathbf{K} == \mathbf{Q} == \mathbf{V}$

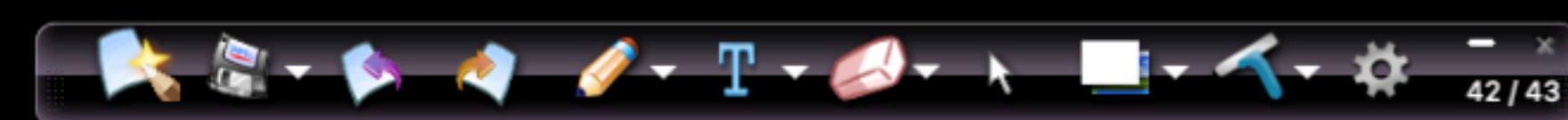
$\mathbf{K} = \mathbf{V}$

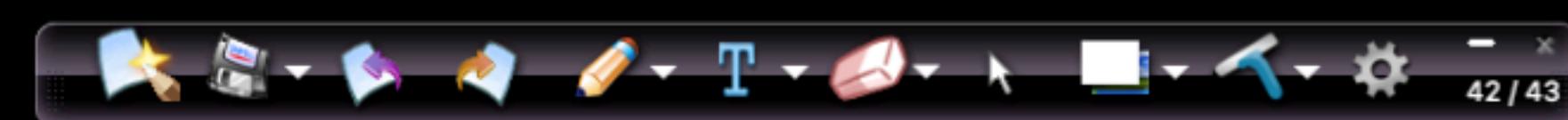


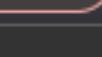
ABSA



for every aspect, which words in the sent are most similar





+ Code + Text Connect |   | 

3. Compute Energy Function.

{x} Here, Energy Function is *DotProduct*.

K $S \times T$

```
[ ] def compute_energy_function(vect, mat):
    # calculate energy/attention fuction
    return np.matmul( mat, np.transpose(vect), )
```

Q $S \times 1$

```
ENERGY_SCORE = compute_energy_function(QUERY, KEY)
ENERGY_SCORE
```

K $S \times T$

```
array([[ 0.51131411],
       [-3.23661494],
       [-0.24969932],
       [-2.17613397],
       [ 6.85134696],
       [-3.23661494],
       [ 1.53640209]])
```

4. Compute Attention weights

```
[ ] def softmax(x):
    # Compute attention weights
    x = np.array(x, dtype=np.float64)
    e_x = np.exp(x)
```

43 / 43

6. Context Vectors

$A_{7 \times 1}$ $V_{5 \times 1}$

```

def apply_attention_scores(attention_weights, annotations):
    # Multiple the annotations by their attention weights
    return [annotations[:i] * attention_weights[i] for i in range(len(attention_weights))]

applied_attention_1 = apply_attention_scores(ATTN_WEIGHTS, np.transpose(VALUE))

```

$K_{7 \times 5}$: all words

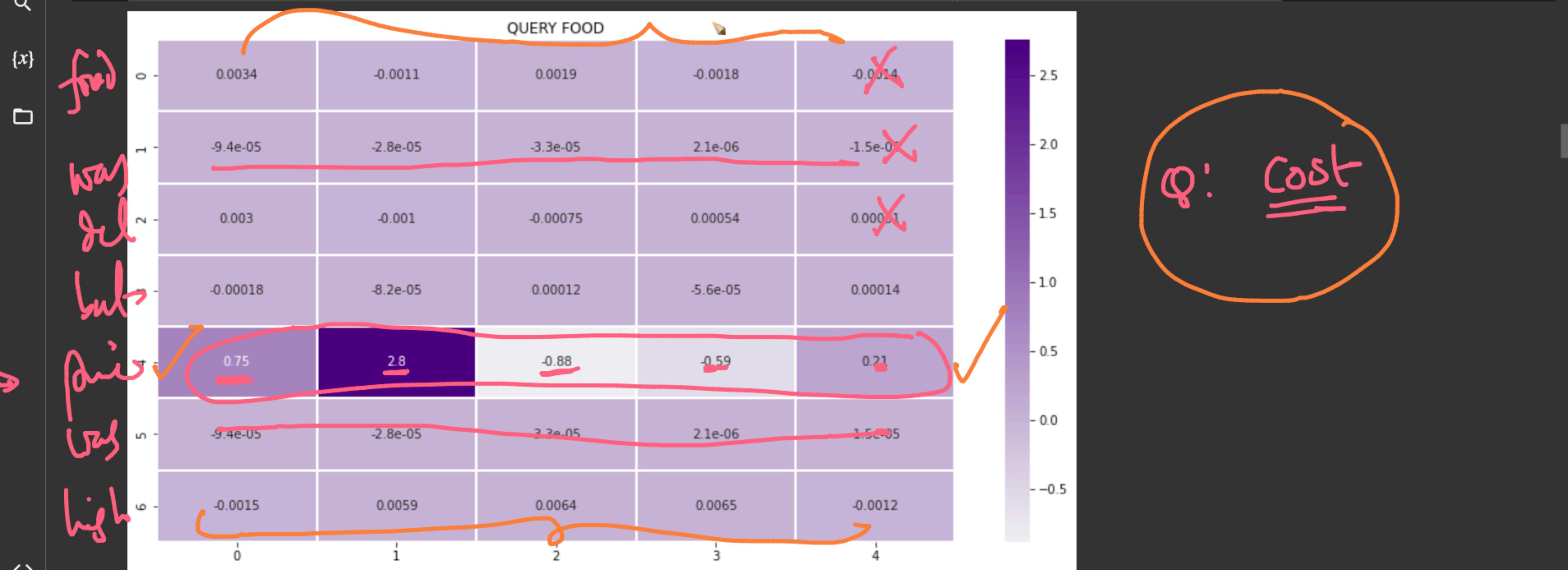
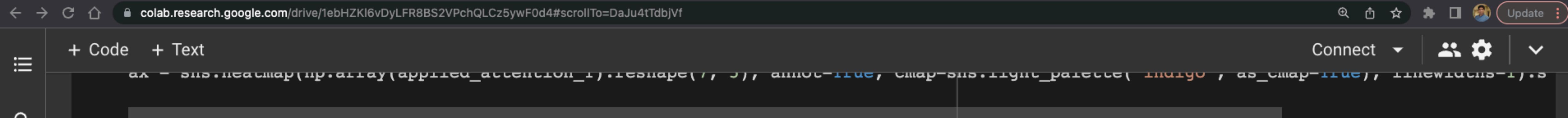
$Q_{1 \times 5} \rightarrow \text{cost}$

QUERY FOOD

	0	1	2	3	4
0	0.0034	-0.0011	0.0019	-0.0018	-0.0014
1	-9.4e-05	-2.8e-05	-3.3e-05	2.1e-06	-1.5e-05
2	0.003	-0.001	-0.00075	0.00054	0.00031
3	-0.00018	-8.2e-05	0.00012	-5.6e-05	0.00014

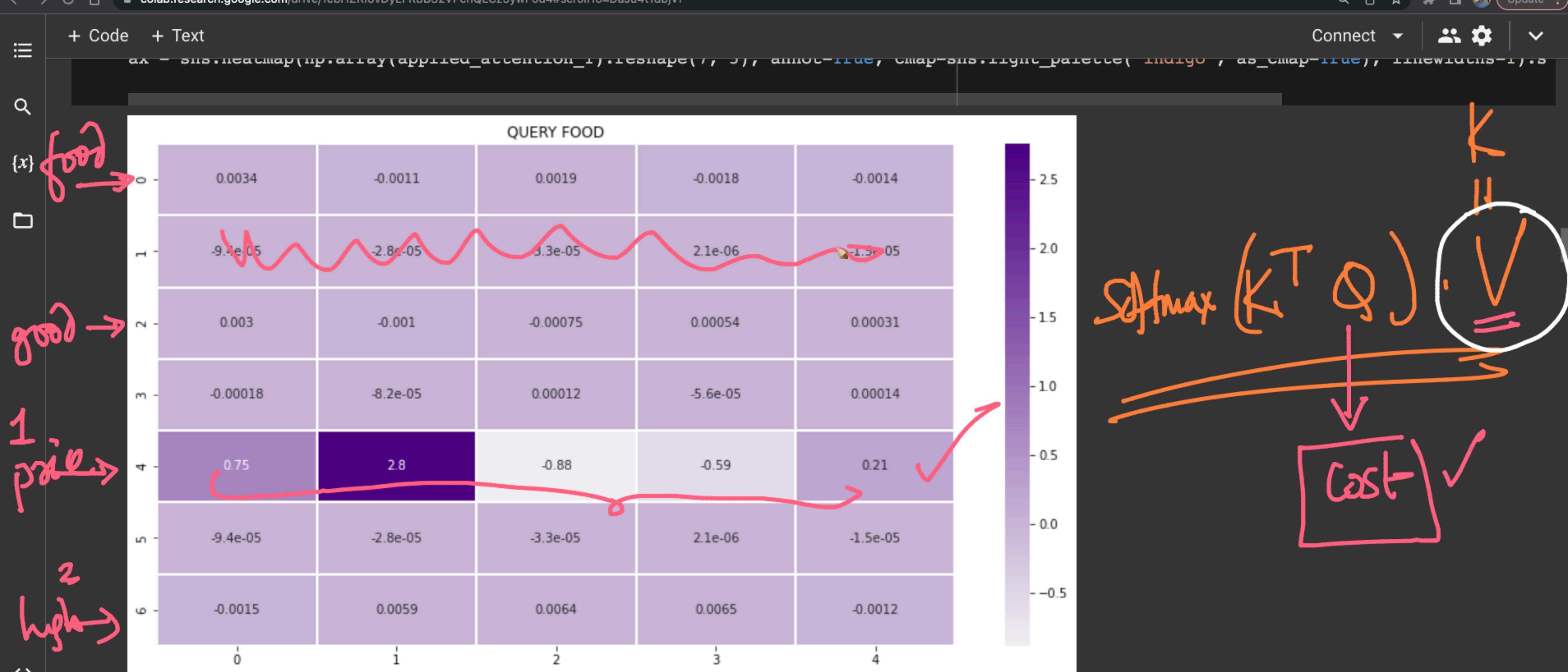
Softmax $(K_{7 \times 5} \cdot Q_{5 \times 1}^T) = A_{7 \times 1}$

$V_{7 \times 5} = \text{key}$

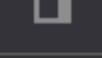
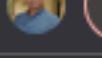


```
[ ] def calculate_attention_vector(applied_attention):  
    return np.sum(applied_attention, axis=0)
```





```
[ ] def calculate_attention_vector(applied_attention):
    return np.sum(applied_attention, axis=0)
```

+ Code + Text Connect |  

- These Alignment functions based on a notion of comparing query representations with key representations.
- compute either the cosine similarity or the dot product between the key and query representations.
- To account for varying lengths of representation, scaled dot product can be employed that normalizes the dot product by the representation vector length.
- Note that these functions assume that key and query have the same representation vector space.

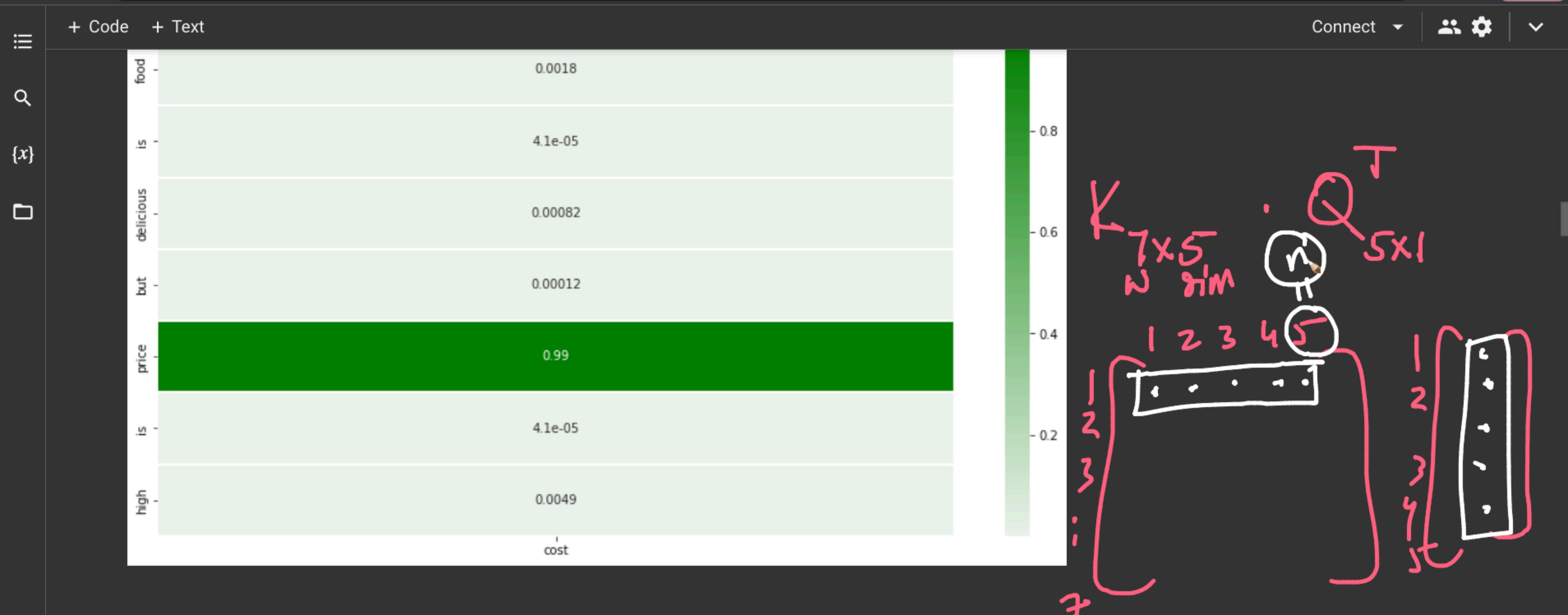
<i>Attention_Function</i>	<i>Equation</i> $f(Q, K)$
Similarity/(e. g. CosineSim)	$Sim(Q, K)$
<u>Dot Product/Multiplicative</u>	$Q^T K$
Scaled_Dot Product	$\frac{Q^T K}{\sqrt{n}}$

$\frac{Q^T K}{\sqrt{n}}$

2. General alignment function

1. General alignment
2. Biased general alignment
3. Activated general alignment

- General alignment extends dot product to keys and queries with different representations by introducing a learnable transformation matrix W that maps queries to the vector space of keys.
- Biased general alignment allows to learn the global importance of some keys irrespective of the query by introducing a bias term.



5. VALUE

Here, VALUE = KEY

```
[ ] VALUE = np.matrix(KEY)
```

+ Code + Text Connect |  

1. Similarity based Alignment functions

1. Similarity e.g. Cosine Similarity
2. Dot Product
3. Scaled Dot Product

- These Alignment functions based on a notion of comparing query representations with key representations.
- compute either the cosine similarity or the dot product between the key and query representations.
- To account for varying lengths of representation, scaled dot product can be employed that normalizes the dot product by the representation vector length.
- Note that these functions assume that key and query have the same representation vector space.

Attention_Function Equation $f(Q, K)$

Similarity/(e. g. CosineSim) $Sim(Q, K)$

DotProduct/Multiplicative $Q^T K$

Scaled_DotProduct $\frac{Q^T K}{\sqrt{n}}$



A handwritten note showing the square root symbol over the variable 'n'.

2. General alignment function

1. General alignment
2. Biased general alignment

+ Code + Text Connect |  

1. Similarity based Alignment functions

1. Similarity e.g. Cosine Similarity
2. Dot Product
3. Scaled Dot Product

- These Alignment functions based on a notion of comparing query representations with key representations.
- compute either the cosine similarity or the dot product between the key and query representations.
- To account for varying lengths of representation, scaled dot product can be employed that normalizes the dot product by the representation vector length.
- Note that these functions assume that key and query have the same representation vector space.

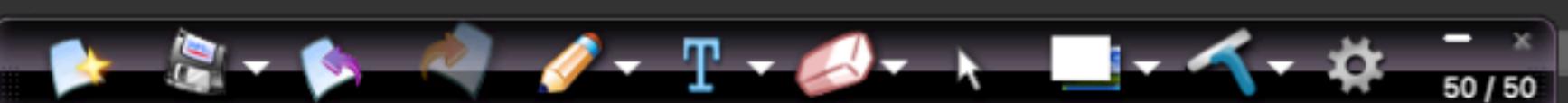
Attention_Function	Equation $f(Q, K)$
Similarity/(e. g. CosineSim)	$\text{Sim}(Q, K)$
DotProduct/Multiplicative	$Q^T K$
Scaled_DotProduct	$\frac{Q^T K}{\sqrt{n}}$

<> 2. General alignment function

1. General alignment

2. Biased general alignment

Connect |  



50 / 50

+ Code + Text

1. General alignment
2. Biased general alignment
3. Activated general alignment

- General alignment extends dot product to keys and queries with different representations by introducing a learnable transformation matrix W that maps queries to the vector space of keys.
- Biased general alignment allows to learn the global importance of some keys irrespective of the query by introducing a bias term.
- Activated general alignment adds a nonlinear activation layer such as hyperbolic tangent, rectifier linear unit, or scaled exponential linear unit.

Attention_Function Equation $f(Q, K)$

<i>General_alignment</i>	$Q^T W K$
<i>Biased_general_alignment</i>	$K^T (W \cdot Q + b)$
<i>Activated_general_alignment</i>	$\text{activation}(Q^T W K + b)$

$$\text{softmax}\left(\frac{Q^T K}{\sqrt{n}}\right) \cdot V$$

3. Additive alignment function

1. Concat
2. Additive Alignment
3. Deep alignment

- combine keys and query to form a joint representation.

+ Code + Text

Connect |  

2. Dot Product

3. Scaled Dot Product

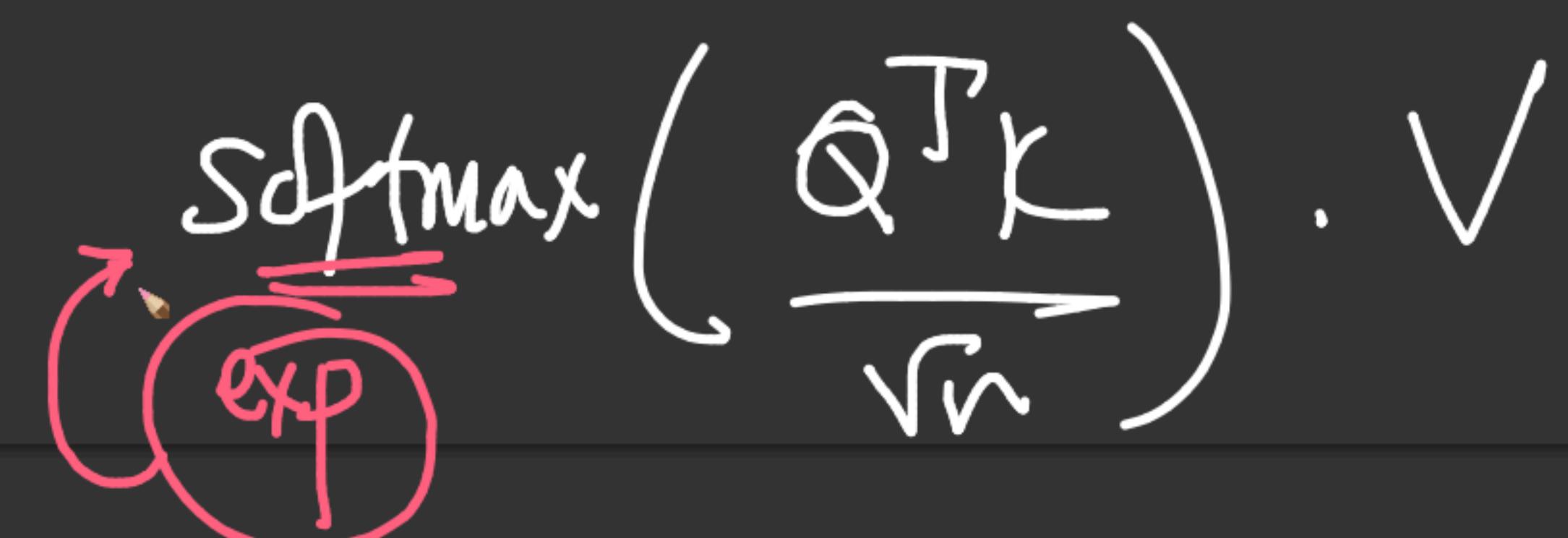
- These Alignment functions based on a notion of comparing query representations with key representations.
- compute either the cosine similarity or the dot product between the key and query representations.
- To account for varying lengths of representation, scaled dot product can be employed that normalizes the dot product by the representation vector length.
- Note that these functions assume that key and query have the same representation vector space.

Attention_Function *Equation* $f(Q, K)$

Similarity/(e.g. CosineSim) $Sim(Q, K)$

DotProduct/Multiplicative $Q^T K$

Scaled_DotProduct $\frac{Q^T K}{\sqrt{n}}$



$$\text{softmax} \left(\frac{Q^T K}{\sqrt{n}} \right) \cdot V$$

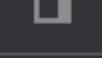
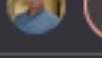
2. General alignment function

1. General alignment
2. Biased general alignment
3. Activated general alignment

- General alignment extends dot product to keys and queries with different representations by introducing a learnable transformation matrix

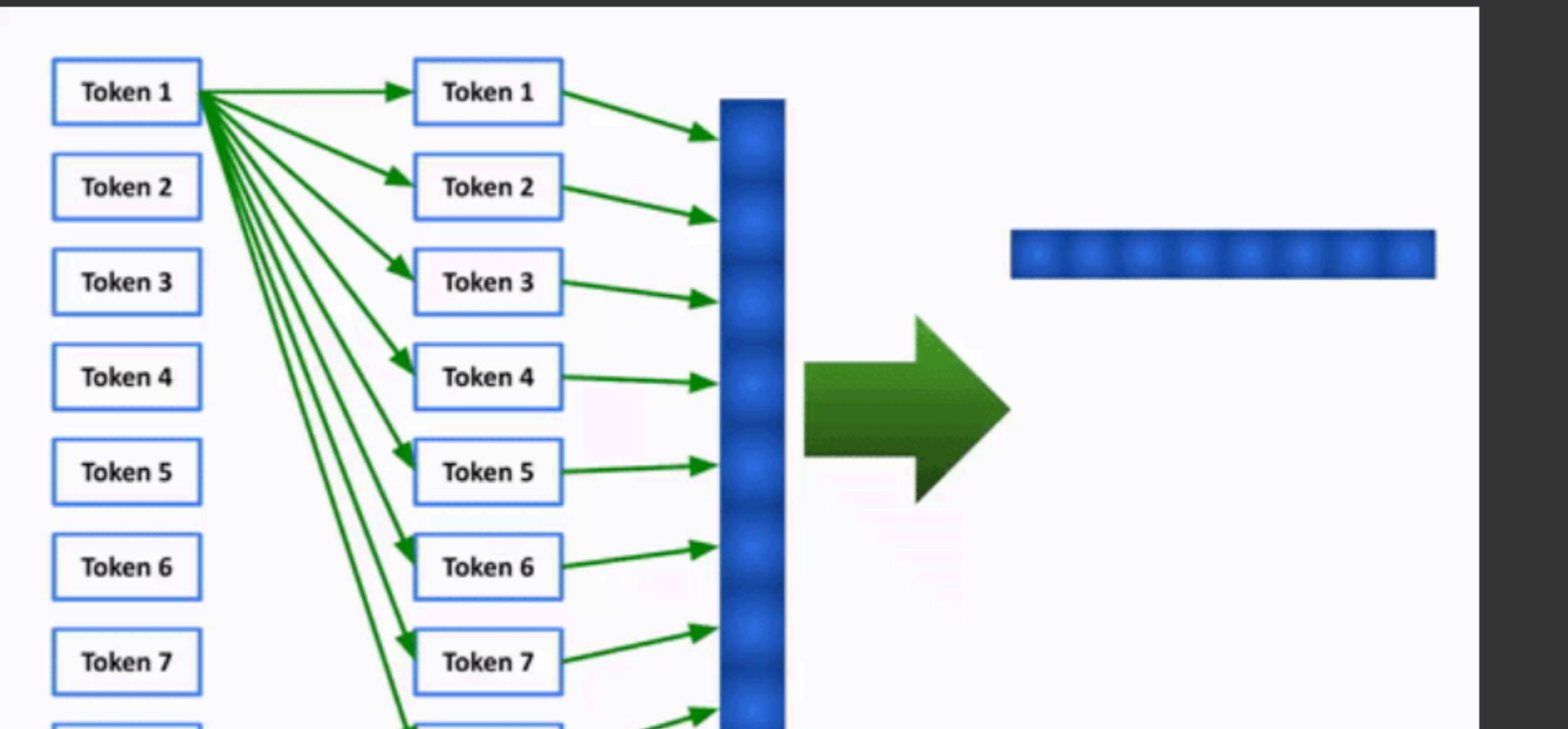
W that maps queries to the vector space of keys

colab.research.google.com/drive/1ebHZKI6vDyLFR8BS2VPchQLCz5ywF0d4#scrollTo=Zv17C6PKuK1K

+ Code + Text Connect |  

2. Self Attention AM

- In Self-Attention both KEY and QUERY sequence is Same.
- Basically, QUERY = KEY = VALUE
- **How to Compute Self AM?**
 - Compute KEY, QUERY and VALUE Vectors
 - Compute Energy function
 - Compute Attention Score

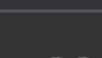


The diagram illustrates the Self-Attention mechanism. On the left, there is a vertical stack of seven blue rectangular boxes, each labeled "Token 1" through "Token 7". From each token box, a green arrow points to a second identical stack of seven boxes, also labeled "Token 1" through "Token 7". This second stack is positioned directly above the first. A large, solid blue vertical bar, representing a weight matrix or projection layer, is positioned between the two stacks. Green arrows from each token in the first stack point towards this blue bar. A large green arrow points from the blue bar to a final horizontal blue bar on the right, which represents the output context vector. The entire process is enclosed in a light gray rounded rectangle.

53 / 53

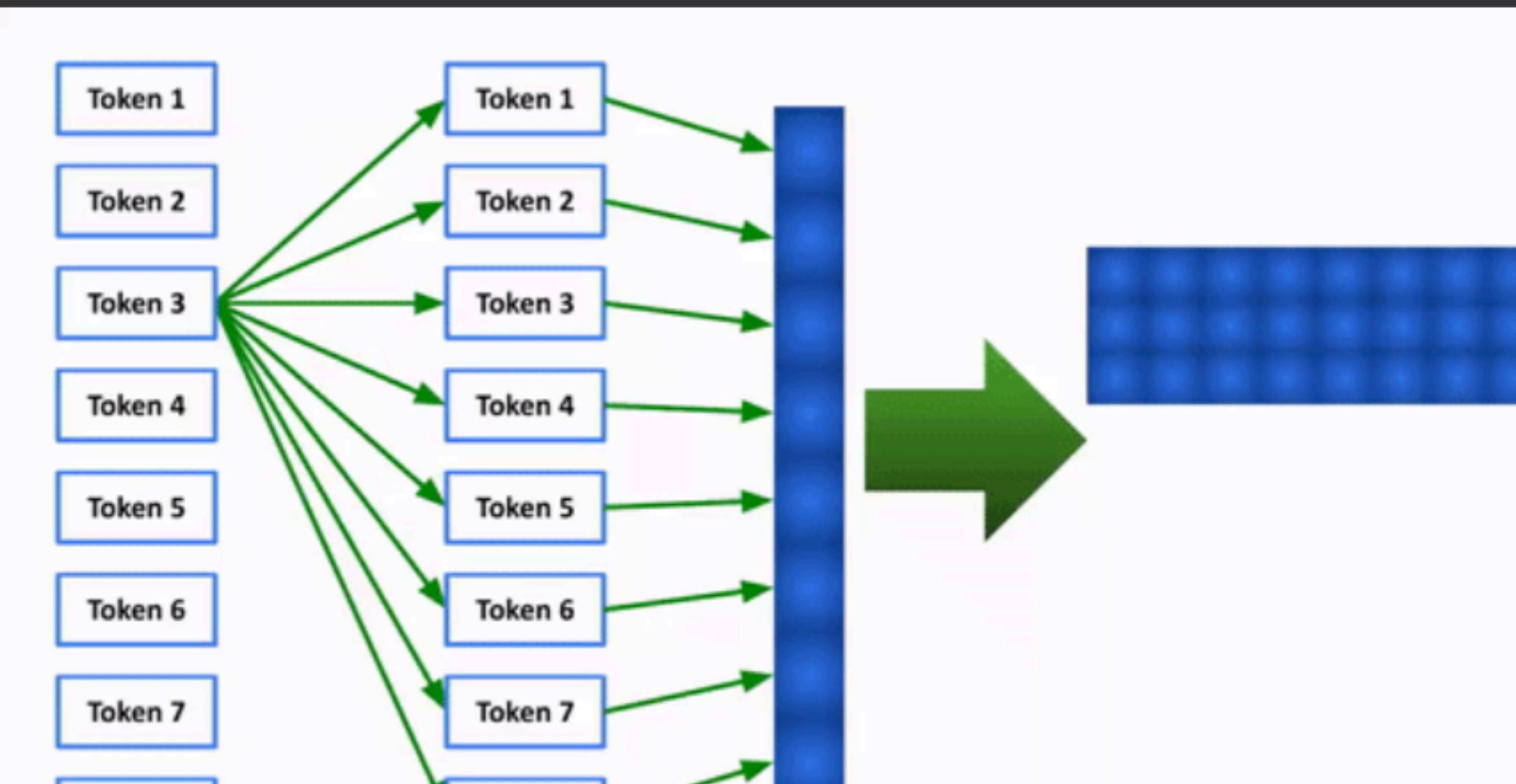
colab.research.google.com/drive/1ebHZKI6vDyLFR8BS2VPchQLCz5ywF0d4#scrollTo=Zv17C6PKuK1K

+ Code + Text

Connect |  

2. Self Attention AM

- In Self-Attention both KEY and QUERY sequence is Same.
- Basically, QUERY = KEY = VALUE
- **How to Compute Self AM?**
 - Compute KEY, QUERY and VALUE Vectors
 - Compute Energy function
 - Compute Attention Score



The diagram illustrates the Self-Attention mechanism. On the left, a vertical stack of boxes represents tokens: Token 1, Token 2, Token 3, Token 4, Token 5, Token 6, Token 7, and Token 8. From each token, three green arrows point to the right. The top arrow points to a second box labeled 'Token 1', the middle arrow points to a second box labeled 'Token 2', and the bottom arrow points to a second box labeled 'Token 3'. This indicates that each token is being used as both a query and a key. All three arrows from each token point to a single large blue vertical bar, which then points to a large blue rectangular matrix. This matrix represents the attention scores or context vectors for each token, where rows and columns correspond to the tokens in the input sequence.

54 / 54

$$\text{softmax} \left(\frac{\mathbf{k}^T \mathbf{Q}}{\sqrt{n}} \right) \cdot \mathbf{V}$$

Intro_to_Attention_Mechanisms_V1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text Connect Editing

Open in Colab

```
import pandas as pd
import numpy as np
import string
import re
import random
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from gensim.models import word2vec, FastText
import gensim.downloader
from sklearn.decomposition import PCA
```

Problem Statement

- You are a data scientist in travel agency company.
- Your company is trying to figure out the sentiment of people about the Hotel and Restaurant listed on their website on the basis of entity

Intro_to_Attention_Mechanisms_V1.ipynb 56 / 56