Agenda:

1. NER → Bi-LSTM + CRF (code)
2. "Attention"-mechanism → LSTMs
   - Transformers
   - Multi-head Self attention
   - Terminology

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

(NER) → Health / Pharma

```
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:
  O O O O O O B-geo O O O O O B-geo O O O O O B-gpe O O O O O


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]


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.]]
```

Model Building - Bi-LSTM + CRF

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

```
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:
  O O O O O O B-geo O O O O O B-geo O O O O O B-gpe O O O O O


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]


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.]]
```

Model Building - Bi-LSTM + CRF

+ Code + Text Connect

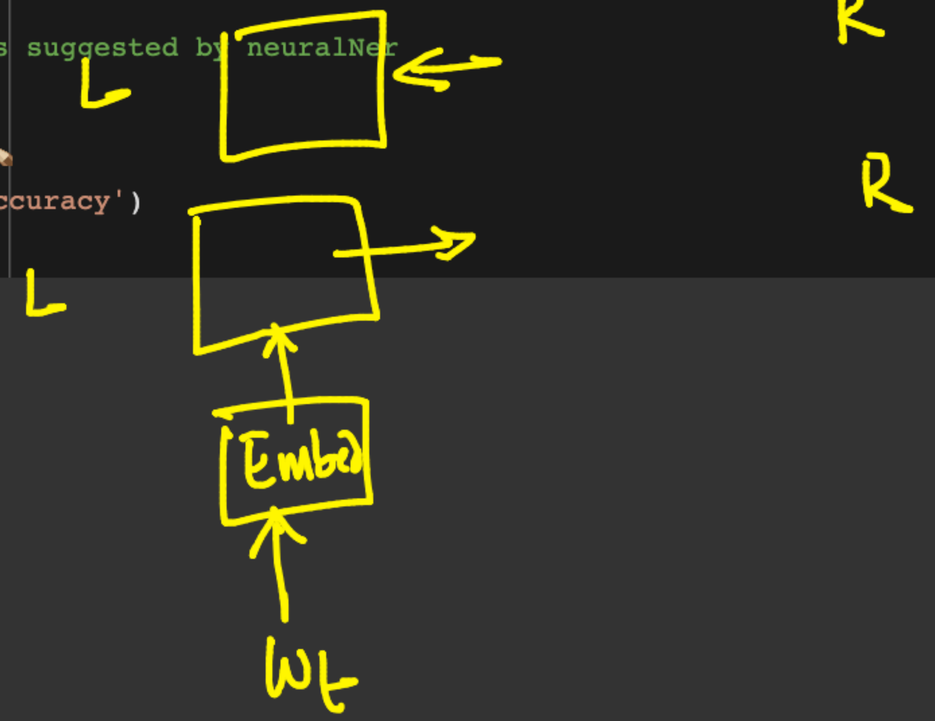
```
[ ] # from keras.models import Model, Input
    from keras.models import Model
    from tensorflow.keras.layers import Input
    from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
    #from keras_contrib.layers import CRF
    from tensorflow_addons.layers import 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  (None, 75, 100)           28400
 l)
```

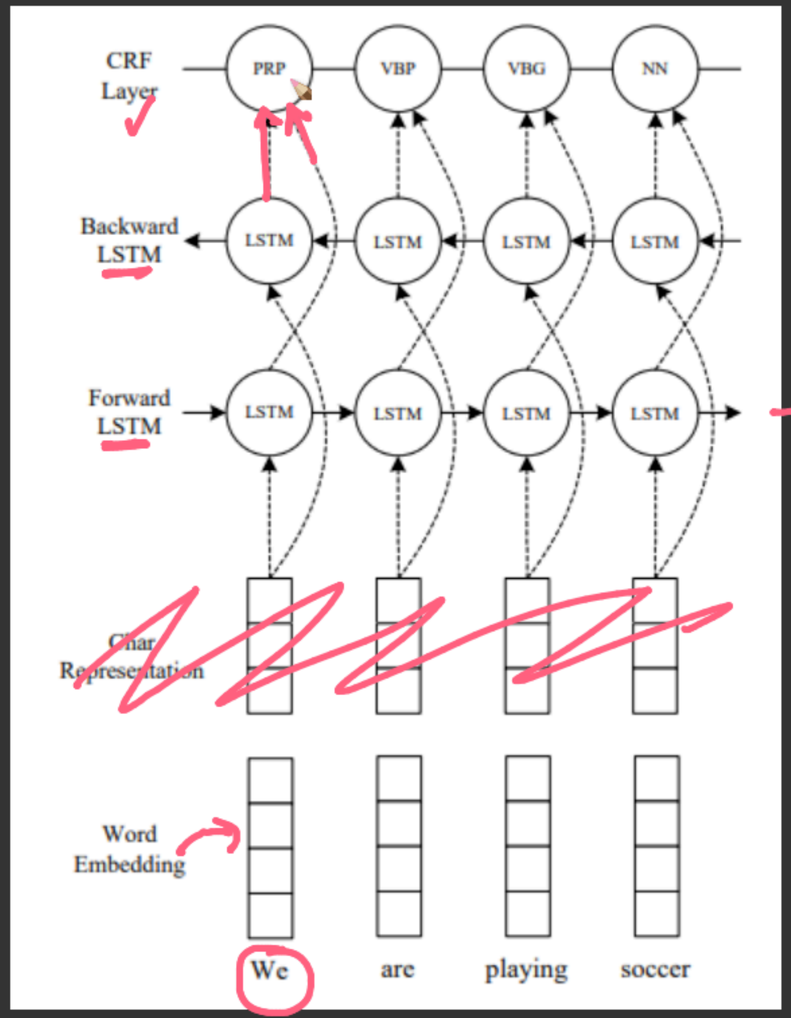
+ Code + Text Connect

```
[ ]
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)          28400
 l)

 time_distributed (TimeDistr  (None, 75, 50)           5050
```



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



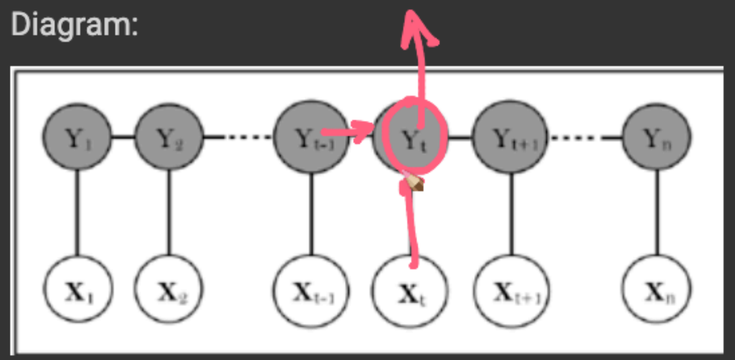
Hybrid

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:

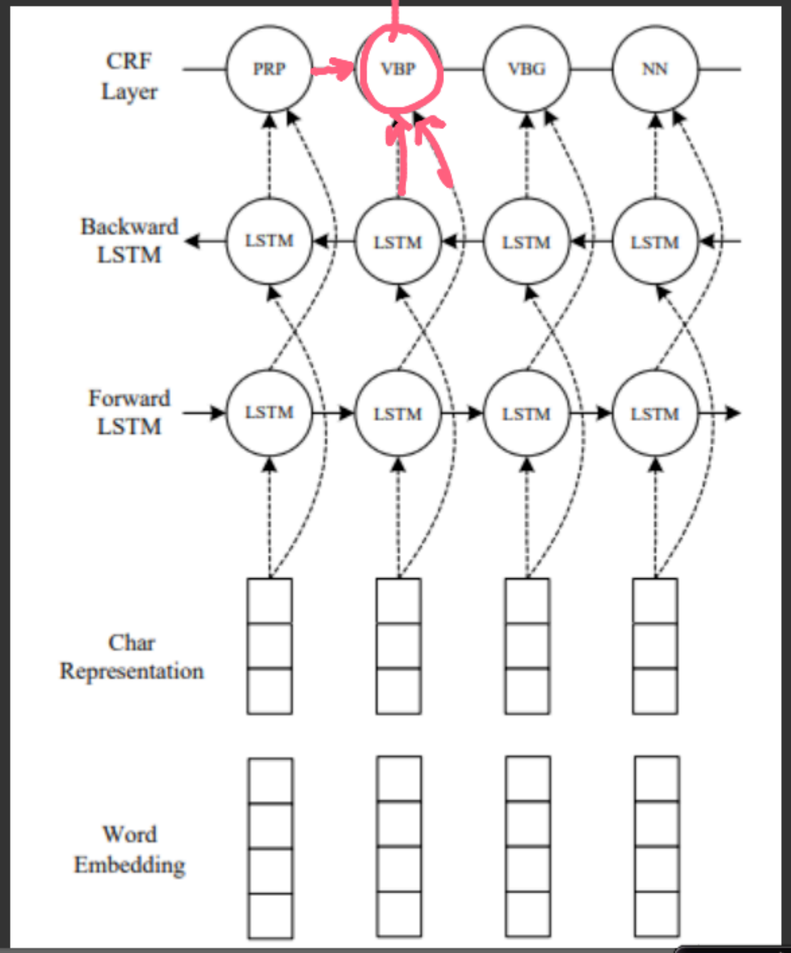
CRF:

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

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

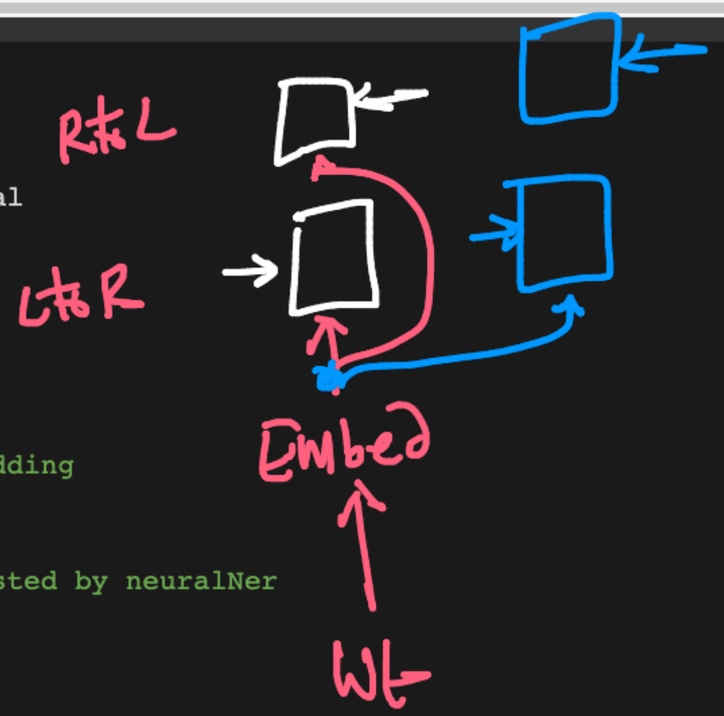
sentence.

- 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.



```
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"
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help

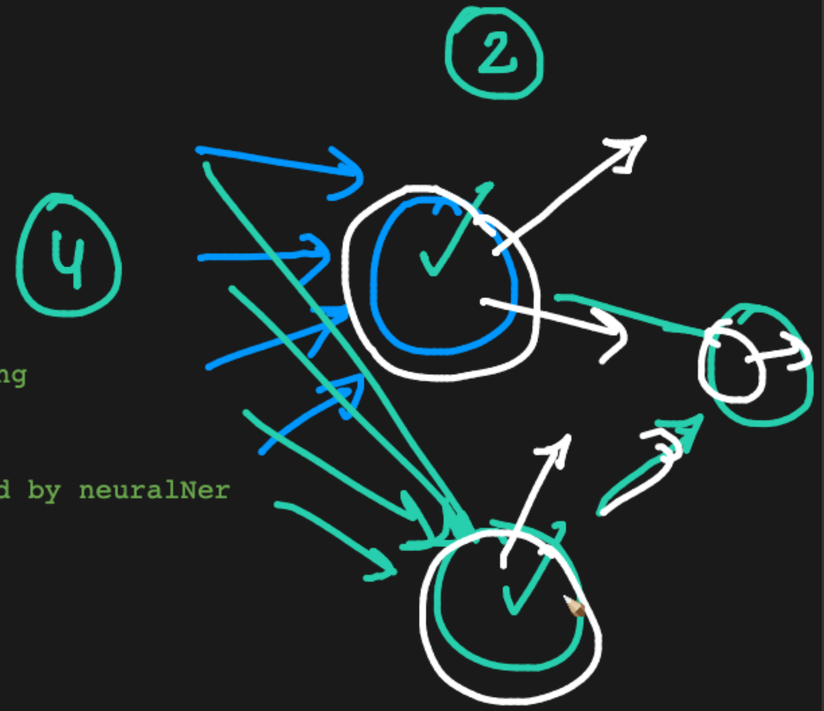
L08_NER_re.ipynb - Colaborat | tfa.rnn.LayerNormLSTMCell | 1603.05118.pdf | Intro_to_Attention_Mechanism | Attention (machine learning) -

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

+ Code + Text Connect

```
[ ]  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()
```

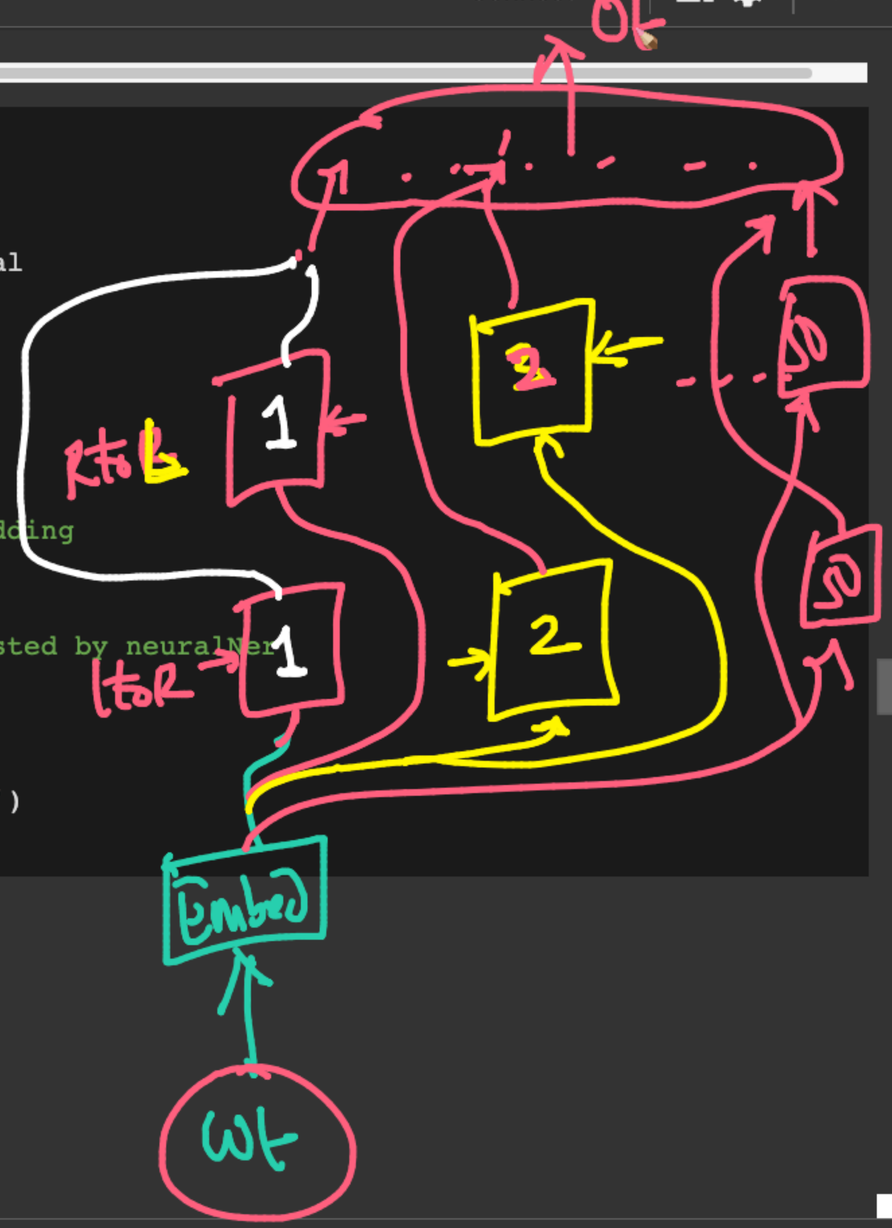



```
Model: "model"
_________________________________________________________________
```

10 / 11

```
[ ] # 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)           28400
```



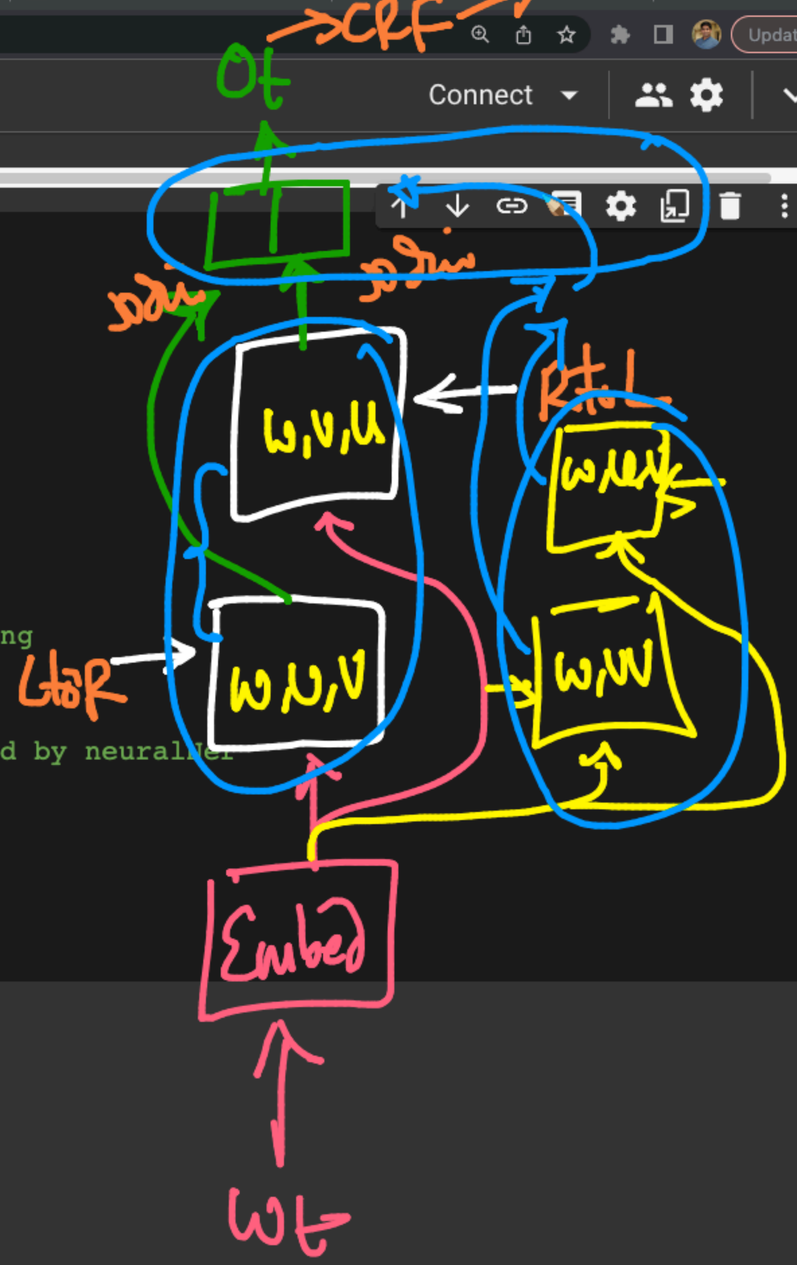
```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
#from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# 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)          28400
```

+ Code + Text

Connect

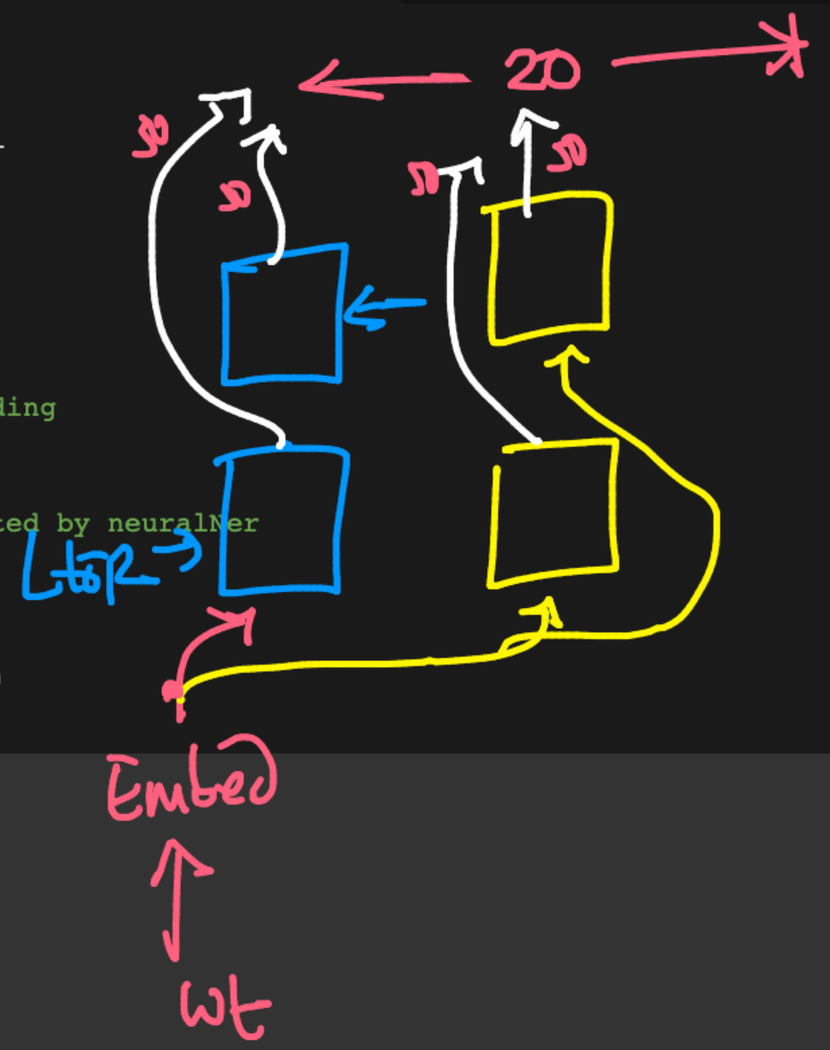
```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
#from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# 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 (None, 75, 100)           28400
```

✓ Functional API ✓

```
# 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)           28400
```

```
# 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()
```

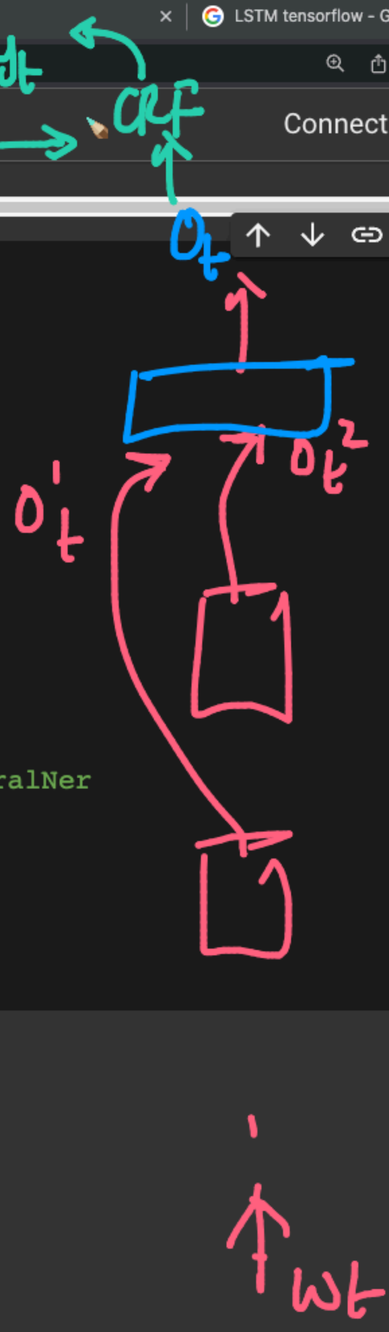
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 75)]              0

 embedding (Embedding)       (None, 75, 20)            703600

 bidirectional (Bidirectiona  (None, 75, 100)          28400
```

```
# 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)          28400
```

```
# 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)           28400
```



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.**

Sentence — NER

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

```
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

	Disease		
	S800	s800	SPECIES
Clinical	i2b2-2010	i2b2	PROBLEM, TEST, TREATMENT
	Radiology	radiology	ANATOMY, OBSERVATION, ANATOMY_MODIFIER, OBSERVATION_MODIFIER, UNCERTAINTY

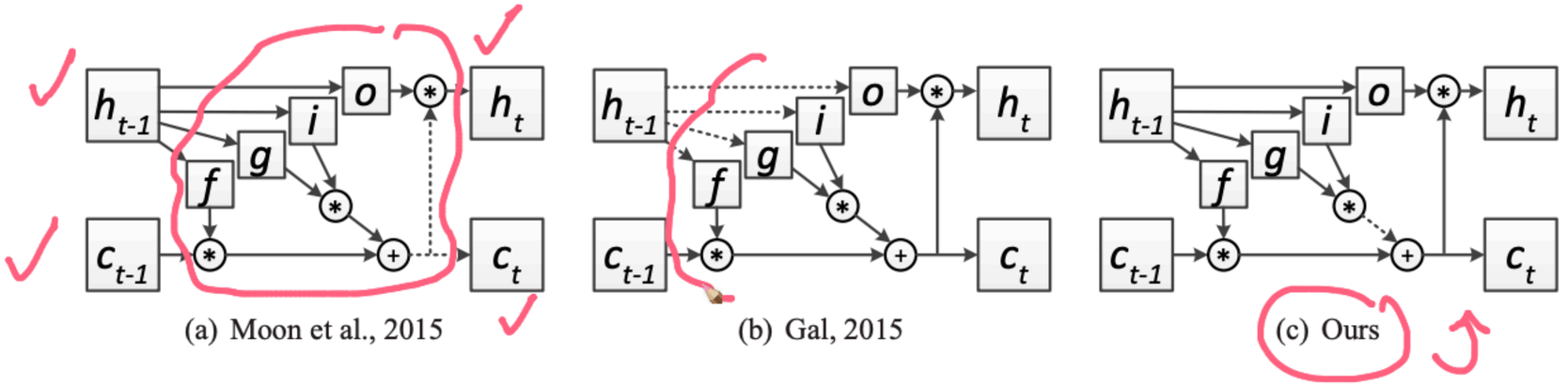
The 16 entity types in the BioNLP13CG model include, some of which are:

- 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)
     |                                | 691 kB 6.9 MB/s
```



(a) Moon et al., 2015 (b) Gal, 2015 (c) Ours

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.

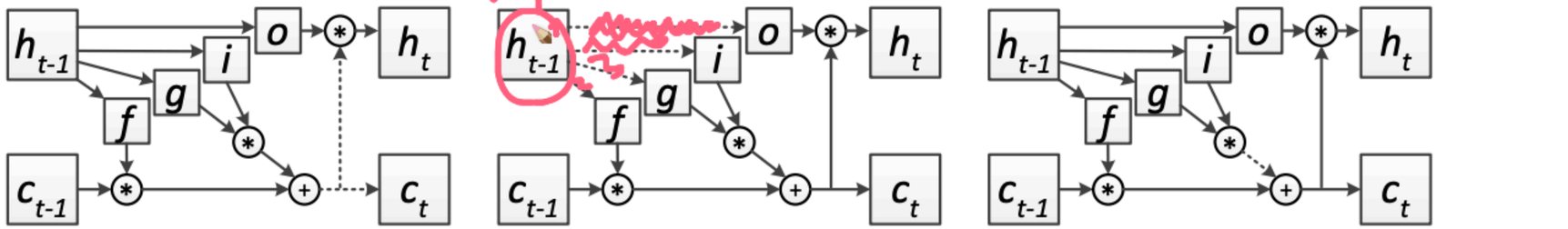
$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propoose 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[illegible] behave as binary switches. vious hidden states directly as proposed in The fact that 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)



(a) Moon et al., 2015 (b) Gal, 2015 (c) Ours

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.

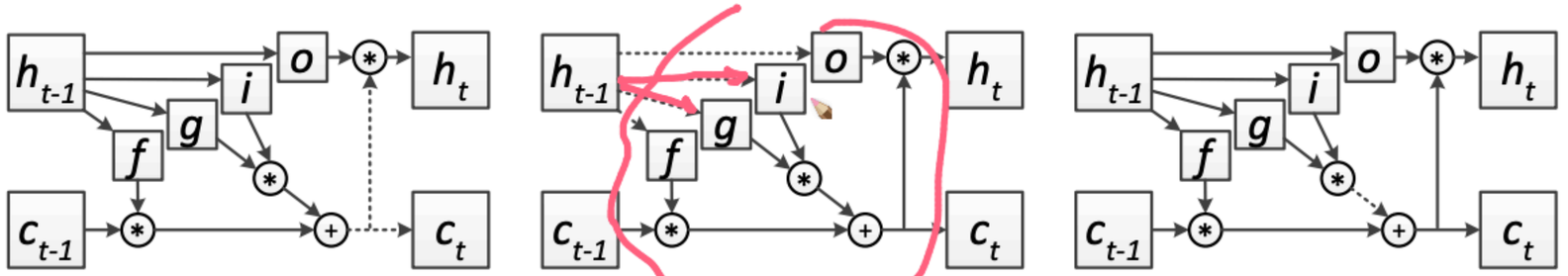
$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propoose 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[illegible] behave as binary switches. vious hidden states directly as proposed in The fact that 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)



(a) Moon et al., 2015 (b) Gal, 2015 (c) Ours

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.

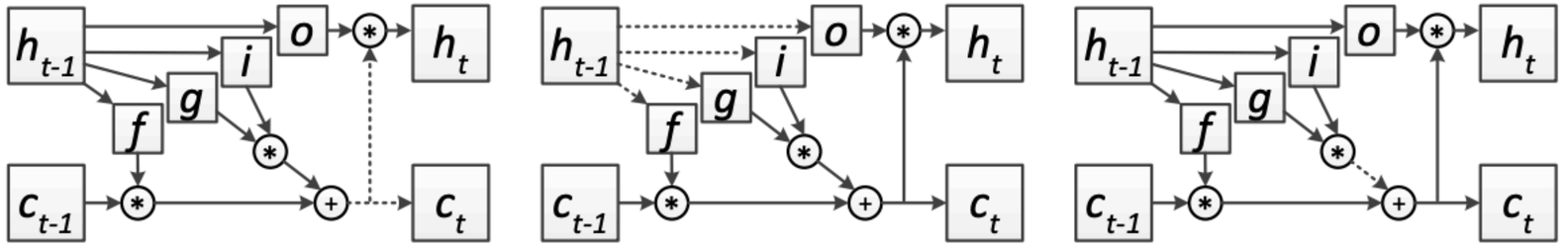
$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propoose 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[illegible] behave as binary switches. vious hidden states directly as proposed in The fact that 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)



(a) Moon et al., 2015 (b) Gal, 2015 (c) Ours

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.

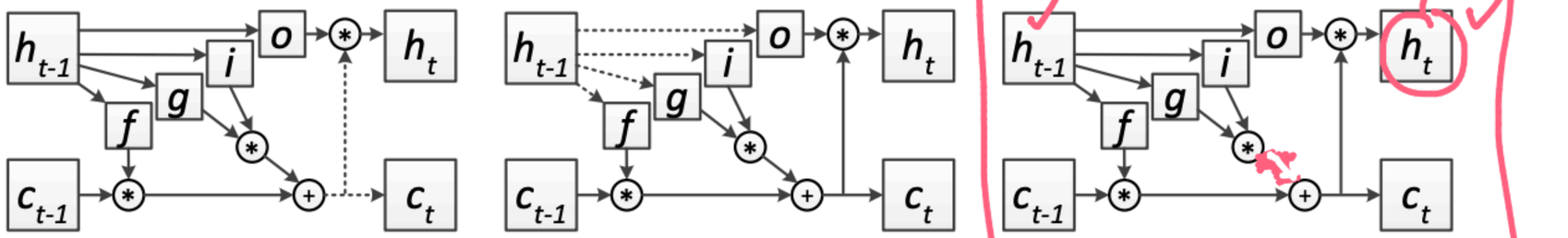
$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \mathbf{g}_t \quad (12)$$

Similarly to the LSTMs, we propoose 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[illegible] behave as binary switches. vious hidden states directly as proposed in The fact that 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)



(a) Moon et al., 2015 (b) Gal, 2015 (c) Ours

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 propoose to apply dropout to the hidden state updates vector $\mathbf{g}_t$:

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

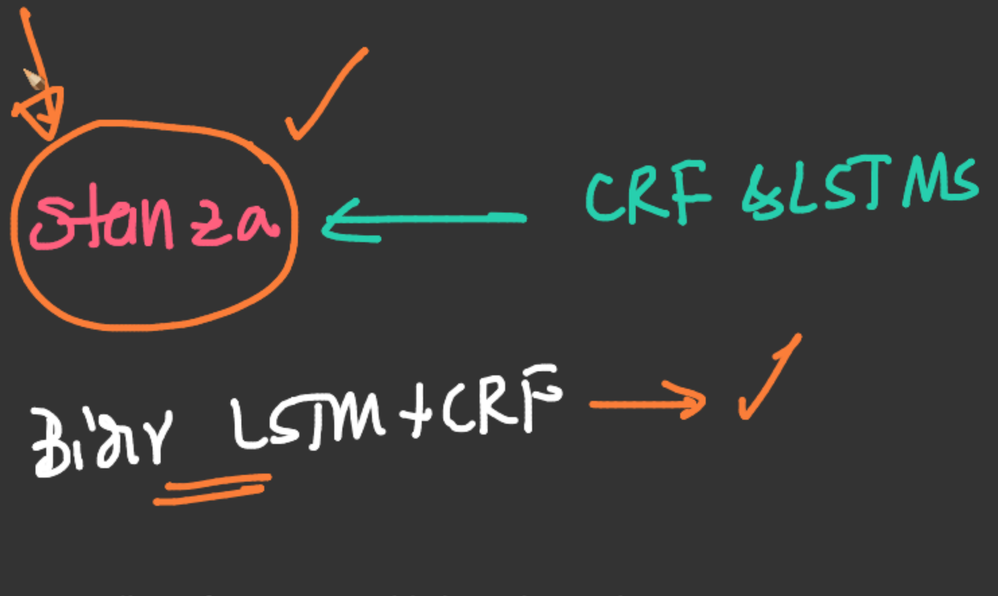
We found that an[illegible] behave as binary switches. vious hidden states directly as proposed in The fact that Moon et al. (2015) have achieved an

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)

```
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:

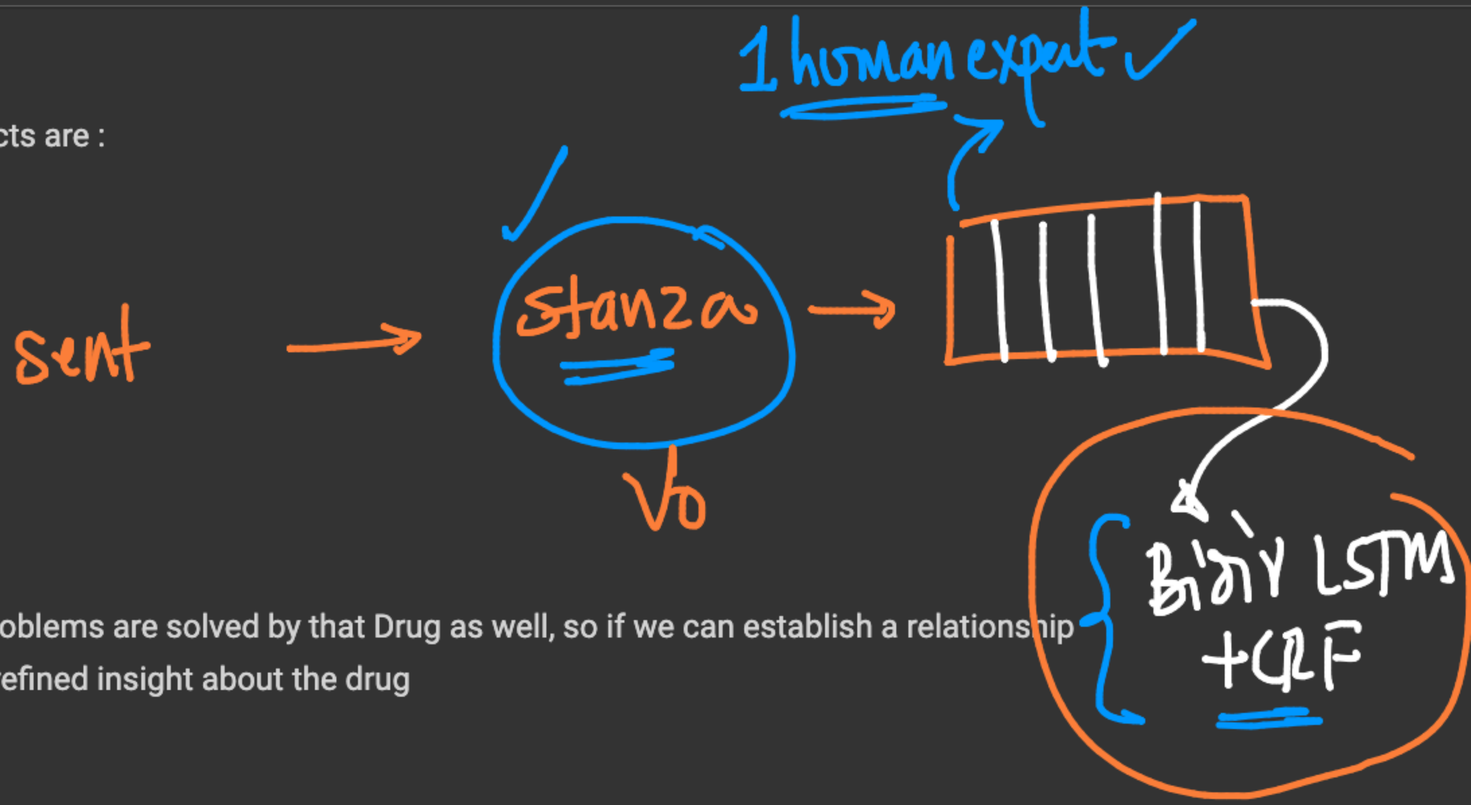
https://www.persistent.com/blogs/building-named-entity-recognition-models-for-healthcare/

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

```
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:

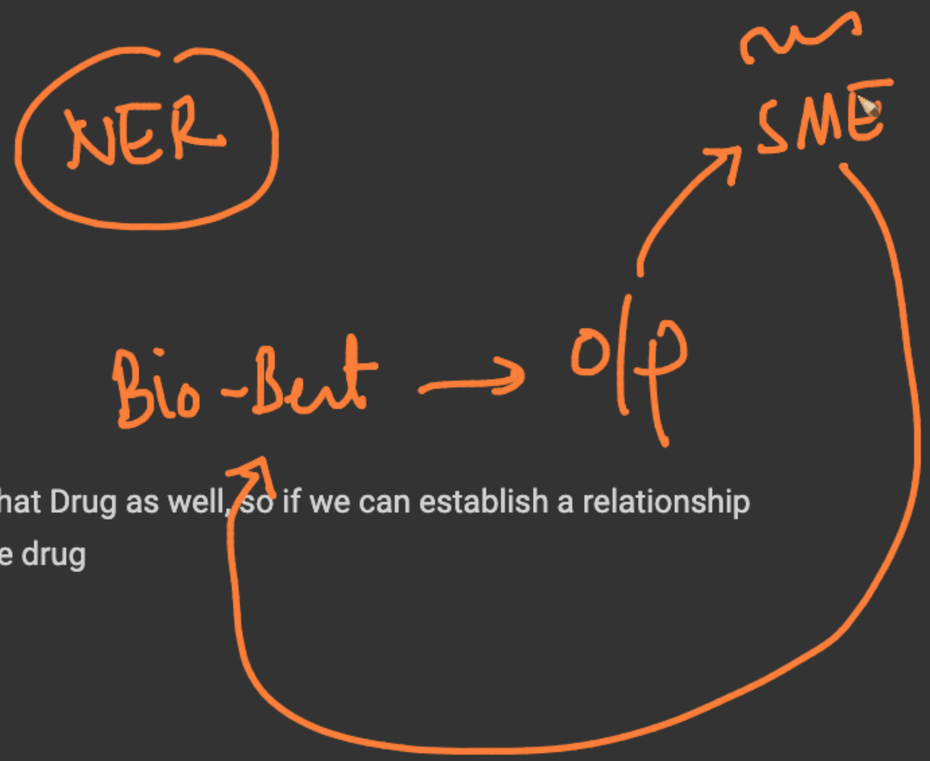
https://www.persistent.com/blogs/building-named-entity-recognition-models-for-healthcare/

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

```
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:

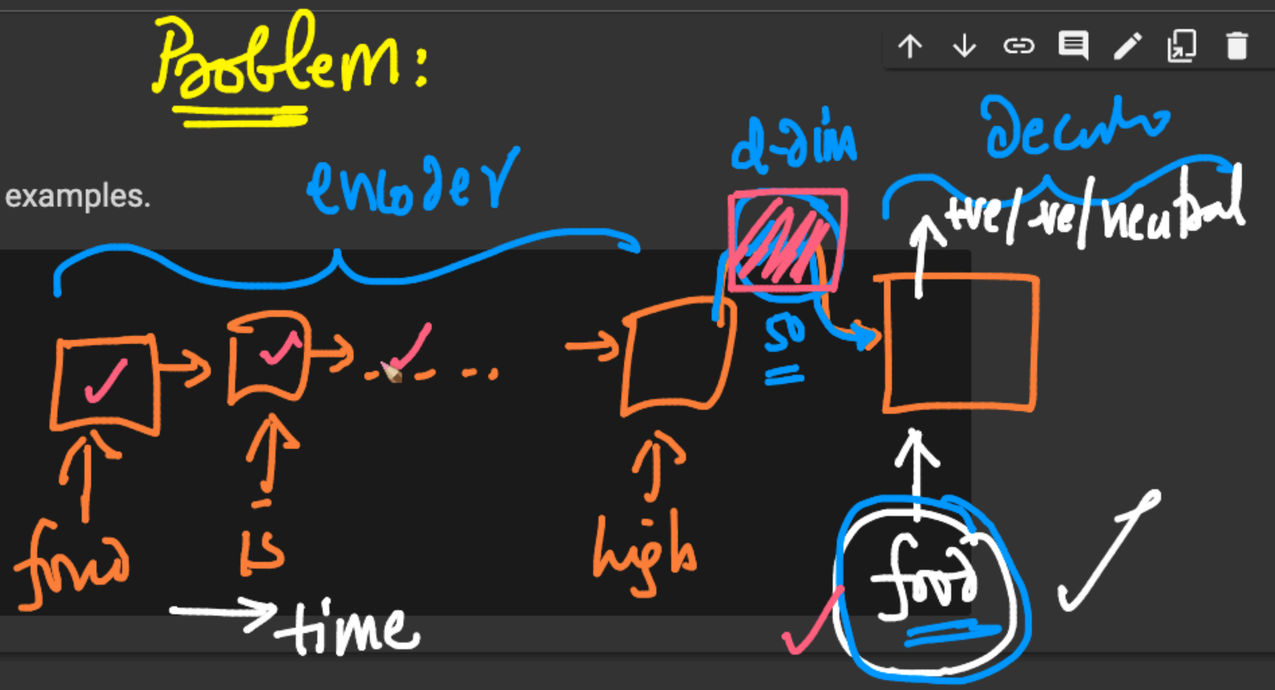
https://www.persistent.com/blogs/building-named-entity-recognition-models-for-healthcare/

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'
              }
```



```
[ ] #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)
```

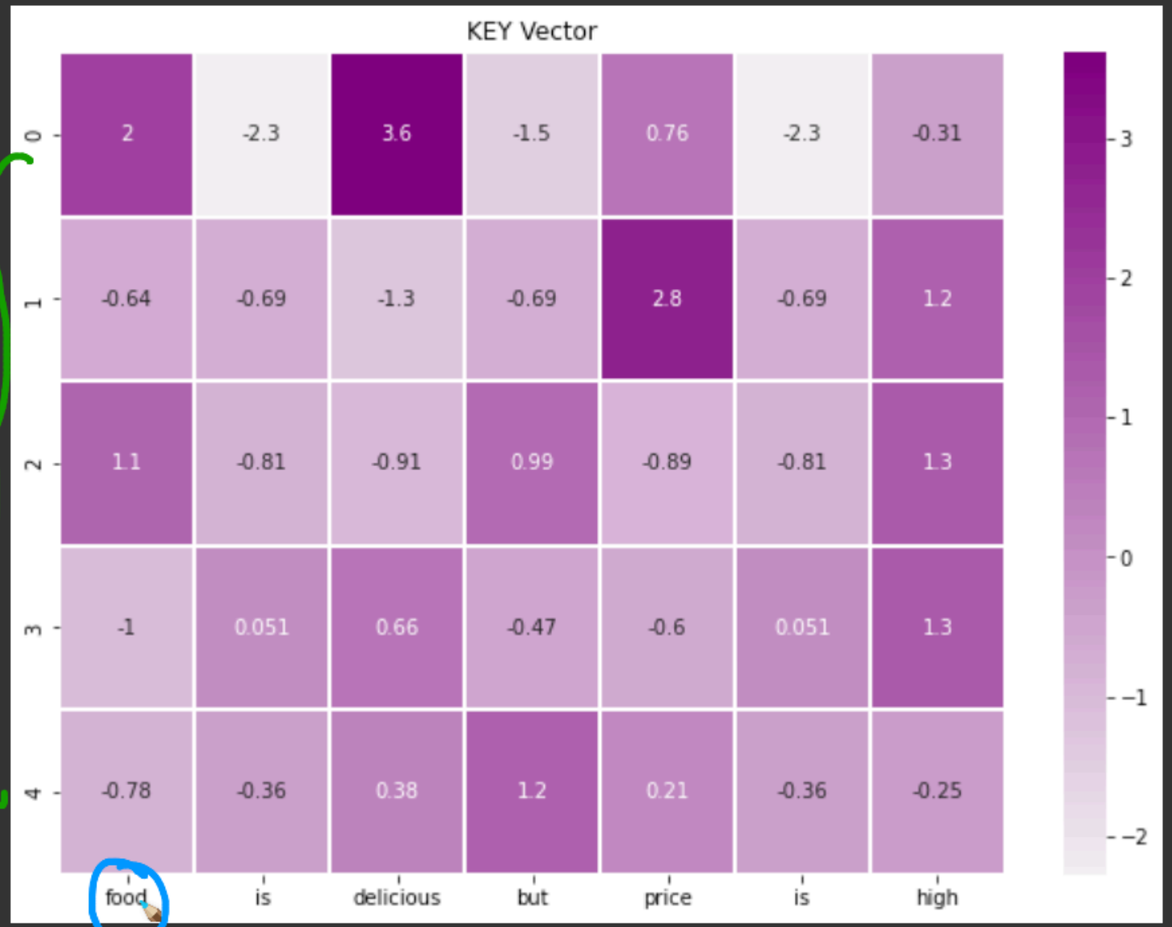
```
[ ] #load glove word embedding vectors
    glove_vectors = gensim.downloader.load('glove-twitter-25')
```

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)
```



5-dim of each w_i

↓

PCA (dim-red) on w2vec

2. Get Query

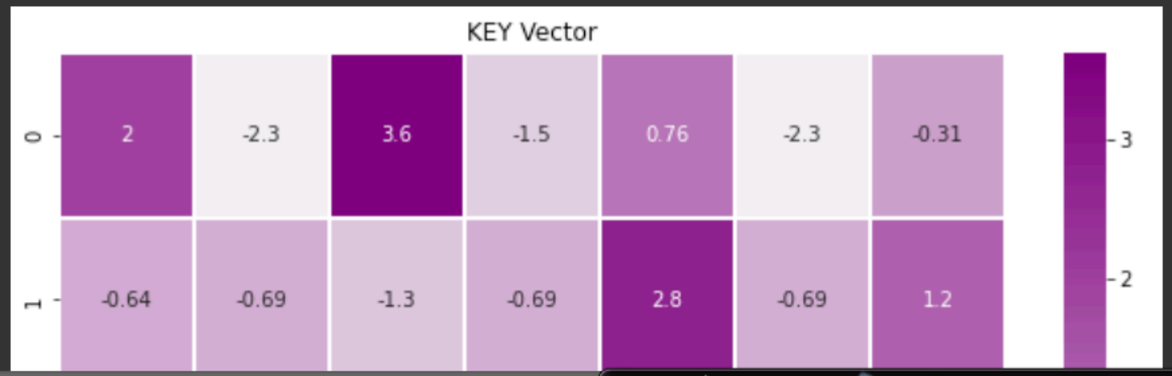
target = [tasty, cost]

$Tokenized$ = [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)
ax = sns.heatmap(key_df, annot=True, cmap=sns.light_palette("purple", as_cmap=True), linewidths=1, ).set_title('KEY Vector')
```



target : ---

ABSA

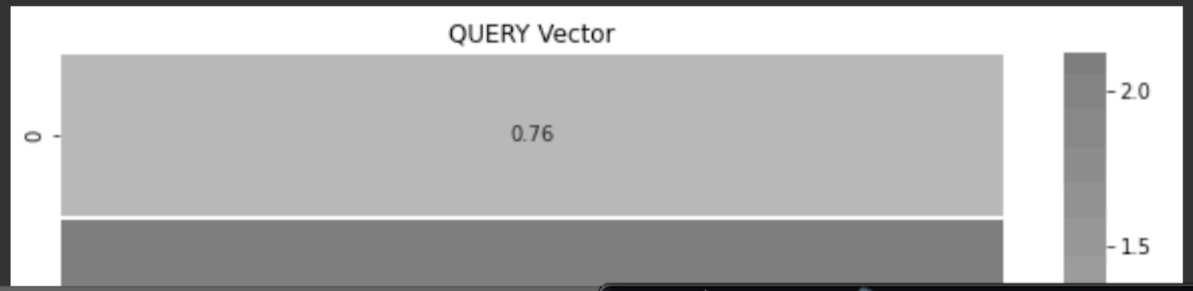
2. Get Query

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', )
```

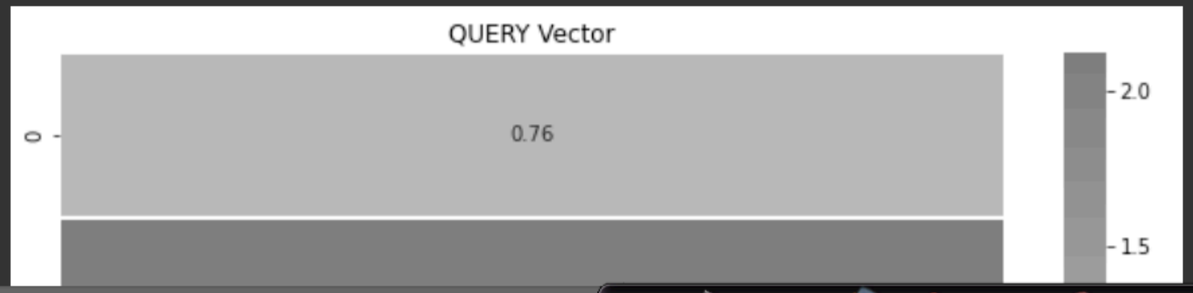


2. Get Query

> *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', )
```



```
attn_df = pd.DataFrame(ATTN_WEIGHTS, index = sentences, columns = target)

plt.figure(figsize=(14.5, 7.5))
ax = sns.heatmap(attn_df , annot=True, cmap=sns.light_palette("green", as_cmap=True), linewidths=1).set_title('ATTN_WEIGHTS', )
```

K dxm ← sentence. length
5 = 7

Q dxm ← #aspects
5 = 1

Atten fn: dot

Energy/Attn score = $K^T \cdot Q$
nxd dxm

A = nxm = 1
7

$Softmax(K^T \cdot Q)$

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

K_{dxn} , Q_{dxM}

7×1

$ATTN\text{-}WEIGHTS = Softmax(K^T_{nxd} Q_{dxM})$

$7 \times M \times 1$

$V = key = V = K_{dxn}$

V^T_{7x5}

$V_{nxd=5}$

$= 7$

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

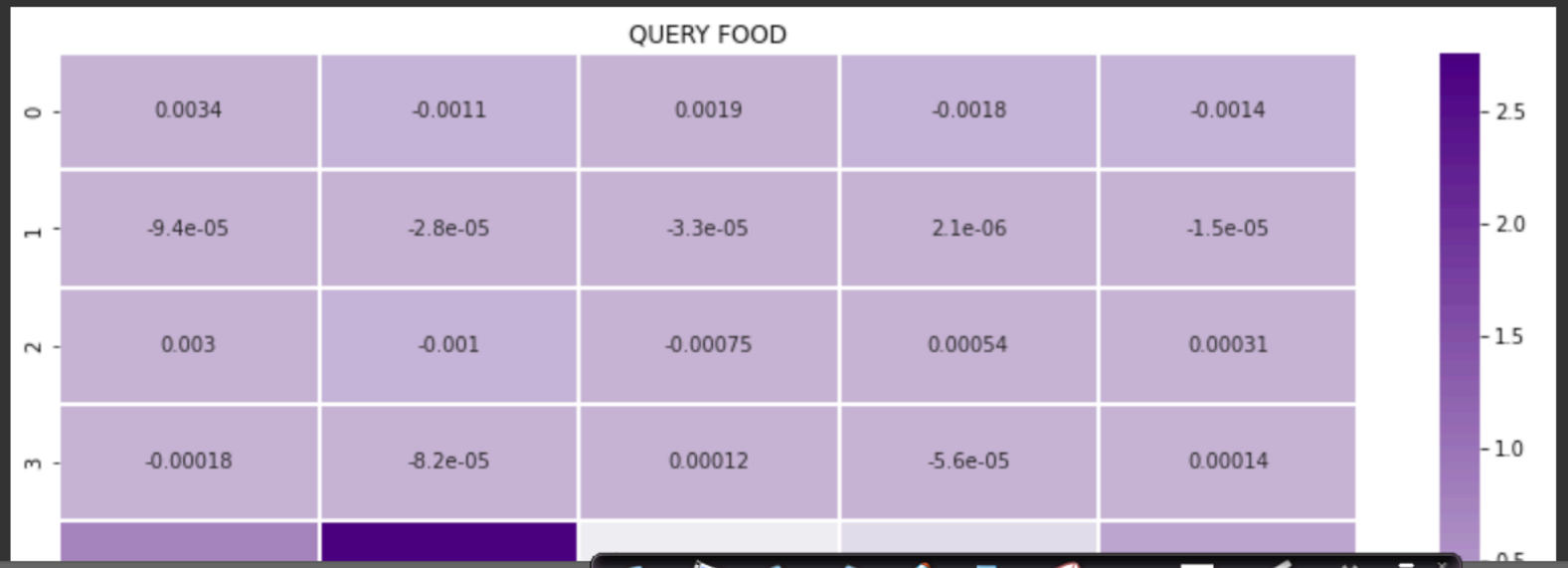
36 / 38

6. Context Vectors

$V^T_{7\times5}$

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

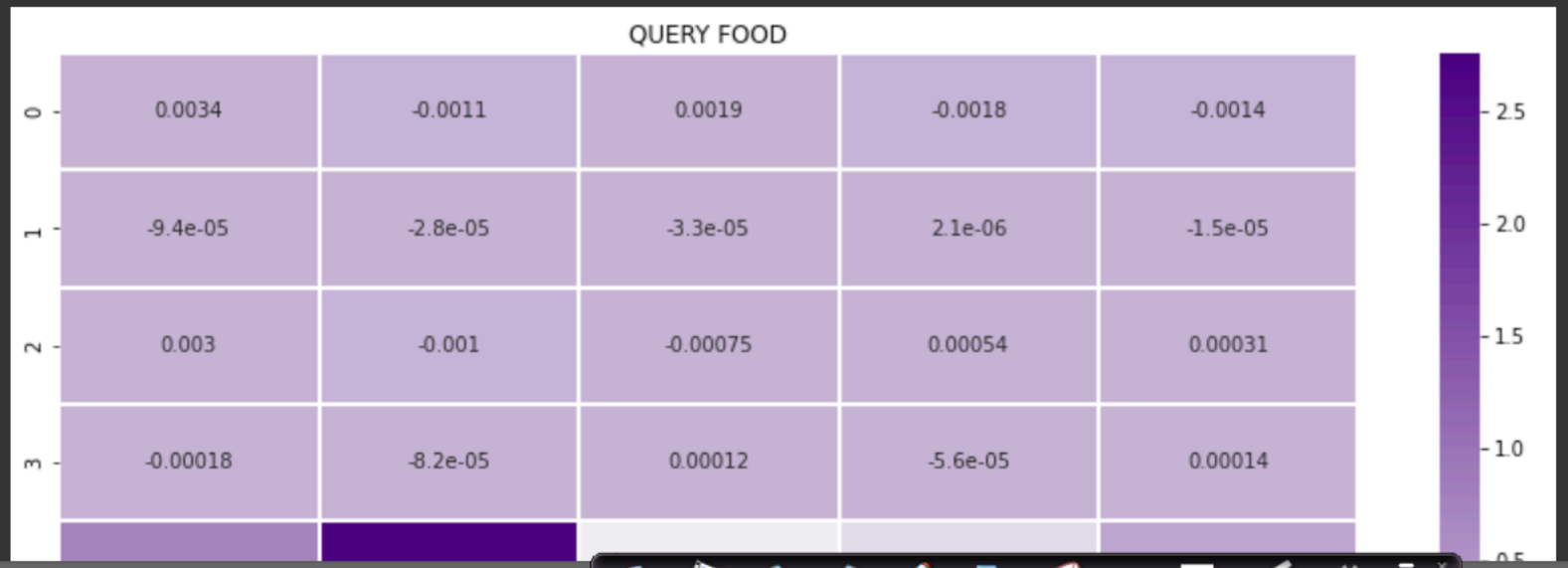
```
def apply_attention_scores(attention_weights, annotations):
 # Multiple the annotations by their attention weights
 return  [  annotations[:, i] * attention_weights[i] for i in range(len(attention_weights))]


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
```



6. Context Vectors

$softmax(K^T \cdot Q) \cdot V$

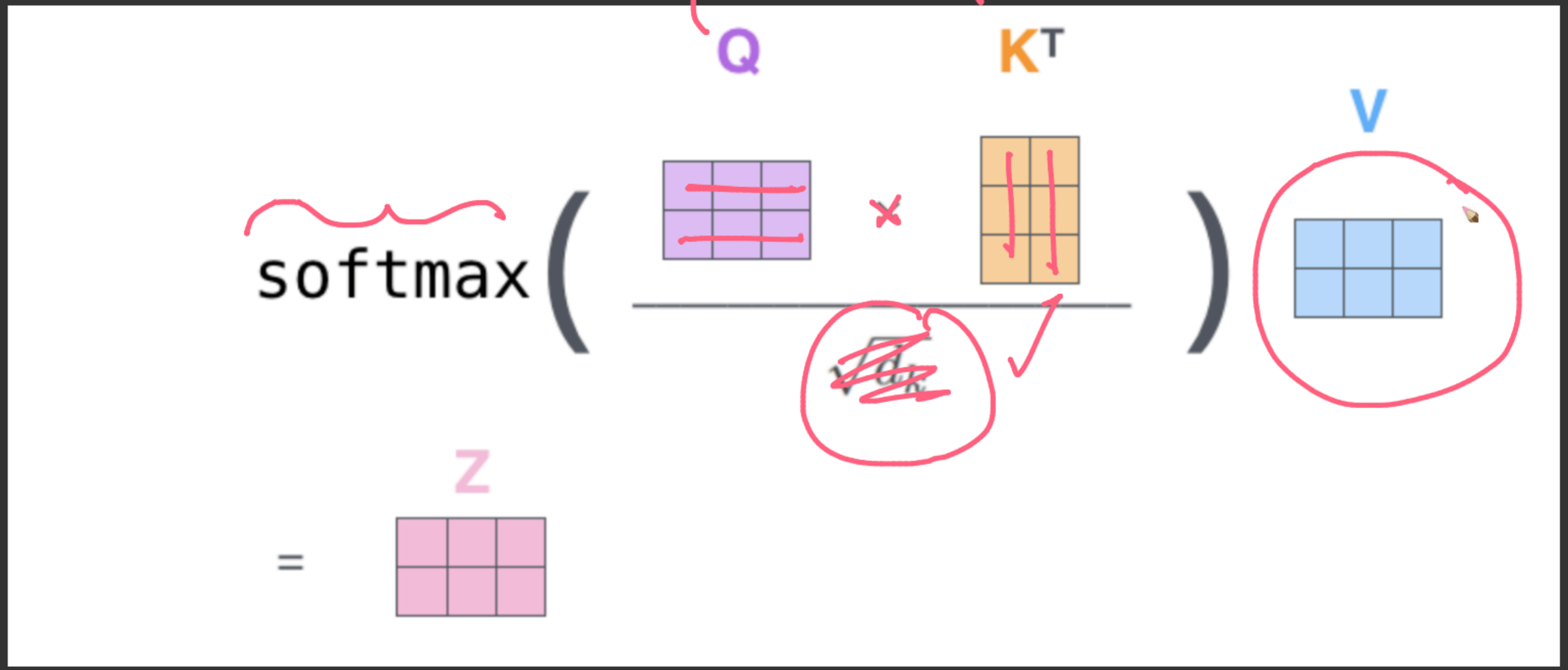
```
def apply_attention_scores(attention_weights, annotations):
 # Multiple the annotations by their attention weights
 return  [  annotations[:, i] * attention_weights[i] for i in range(len(attention_weights))]


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
```



38 / 40

$K = Key,$
$V = Value,$
$where \ \ K == Q == V$

K=V

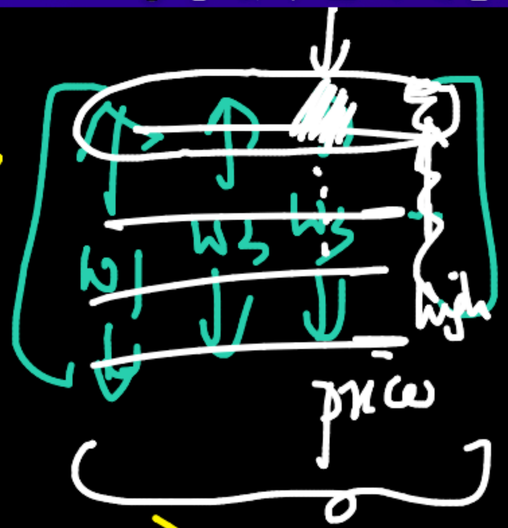



```
[ ] #softmaxing
```

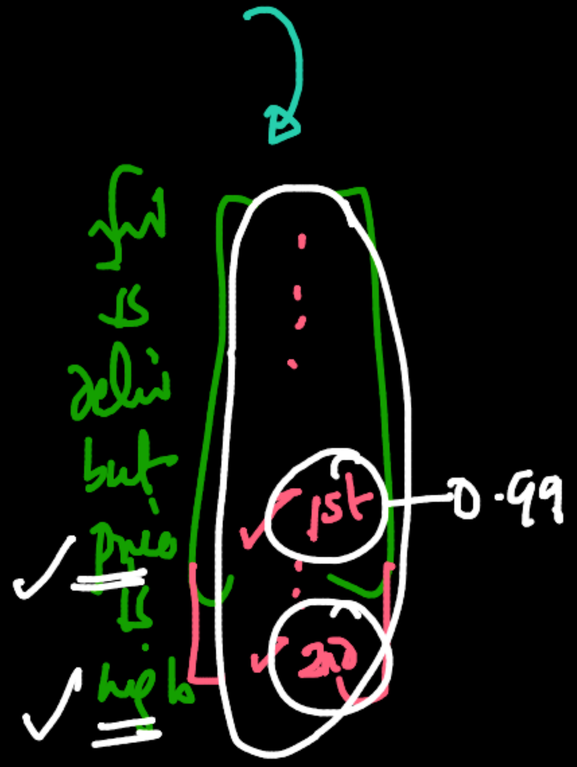
ABSA

aspects

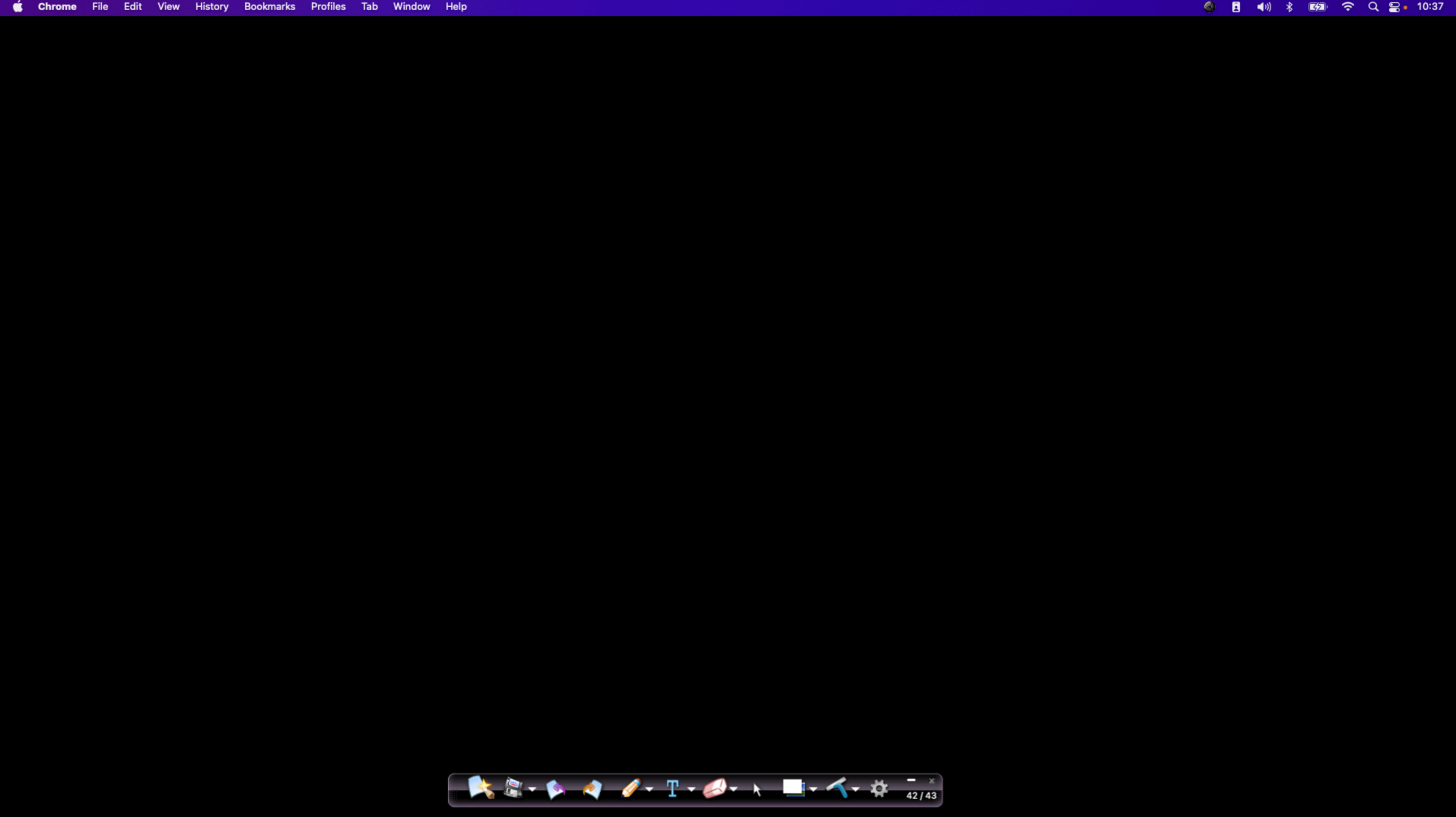
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d}}\right) \cdot V$$

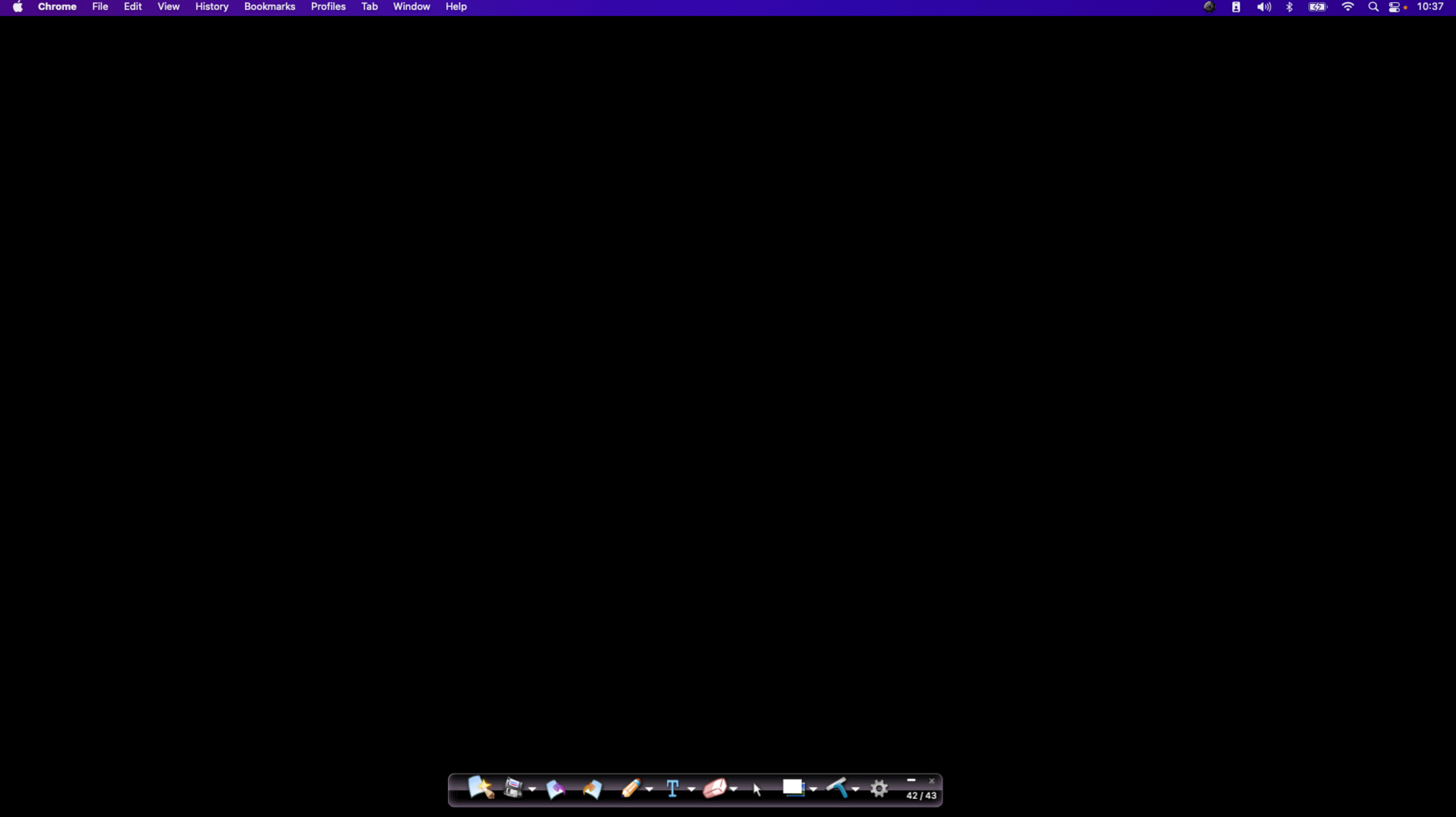


vec. rep of each word in my Sent



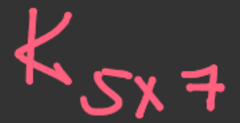
fr every aspect, which words in the Sent are most similar

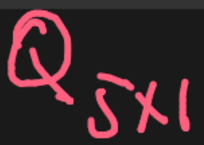

Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:37
42 / 43

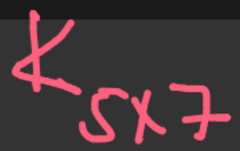

Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:37
42 / 43

3. Compute Energy Function.

> Here, Energy Function is *Dot Product*.







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

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

```
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)
```

6. Context Vectors

```
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
```

food way del but pic way high

Q: Cost

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

food →
good →
1. pizza →
2 high →

K
V
Softmax $(K^T Q)$ · V
Cost ✓

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

46 / 46

- 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.

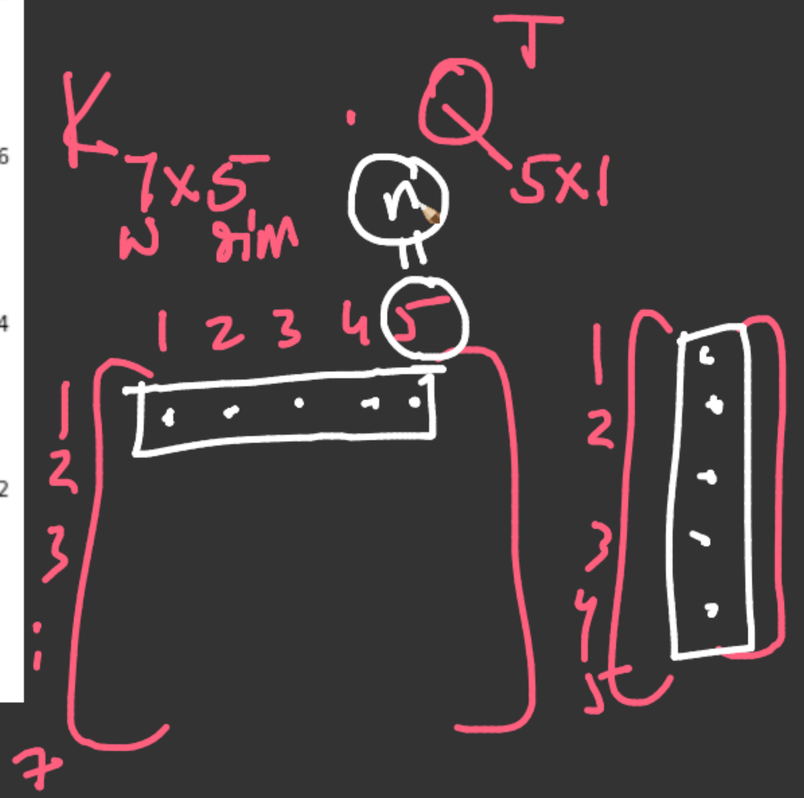
$Attention\_Function$	$Equation\ f(Q, K)$
$Similarity/(e.g.\ CosineSim)$	$Sim(Q, K)$
$DotProduct/Multiplicative$	$Q^T K$
$Scaled\_DotProduct$	$\frac{Q^T K}{\sqrt{n}}$

$\frac{Q^T k}{\sqrt{n}}$ or $k^T Q$

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)
```

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.

$Attention\_Function$	$Equation\ f(Q, K)$
$Similarity/(e.g.\ CosineSim)$	$Sim(Q, K)$
$Dot\ Product/Multiplicative$	$Q^T K$
$Scaled\_Dot\ Product$	$\frac{Q^T K}{\sqrt{n}}$

2. General alignment function

1. **General alignment**
2. **Biased general alignment**

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.

$Attention\_Function$	$Equation\ f(Q, K)$
$Similarity/(e.g.\ CosineSim)$	$Sim(Q, K)$
$Dot\ Product/Multiplicative$	$Q^T K$
$Scaled\_Dot\ Product$	$\frac{Q^T K}{\sqrt{n}}$

2. General alignment function

1. **General alignment**
2. **Biased general alignment**

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

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

- 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.

$Attention\_Function$	$Equation\ f(Q, K)$
$General\_alignment$	$Q^T W K$
$Biased\_general\_alignment$	$K^T(W \cdot Q + b)$
$Activated\_general\_alignment$	$activation(Q^T W K + b)$

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

3. Additive alignment function

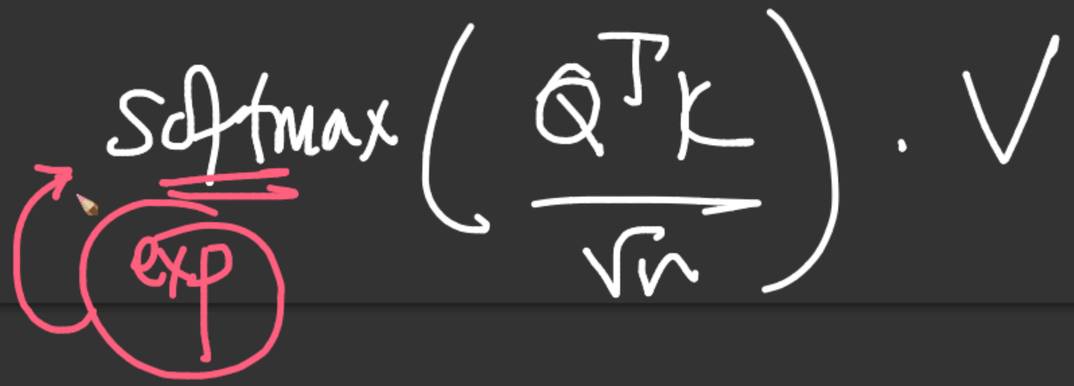
1. **Concat**
2. **Additive Alignment**
3. **Deep alignment**

- combine keys and query to form a joint representation.

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.

$Attention\_Function$	$Equation\ f(Q, K)$
$Similarity/(e.g.\ CosineSim)$	$Sim(Q, K)$
$DotProduct/Multiplicative$	$Q^T K$
$Scaled\_DotProduct$	$\frac{Q^T K}{\sqrt{n}}$



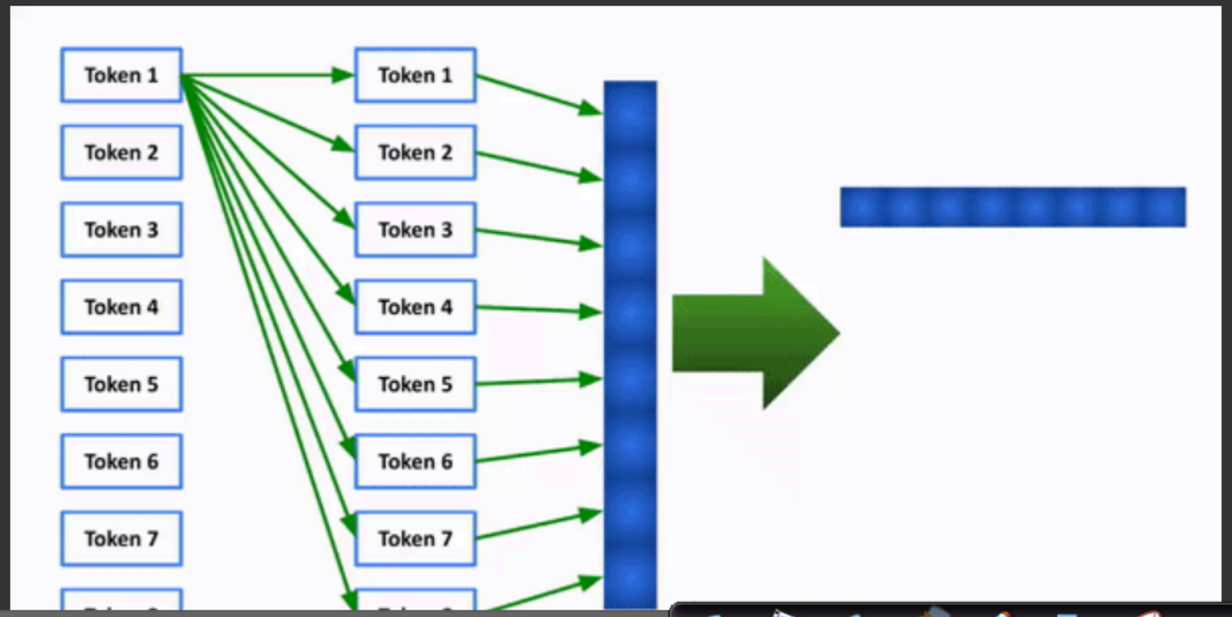
2. General alignment function

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

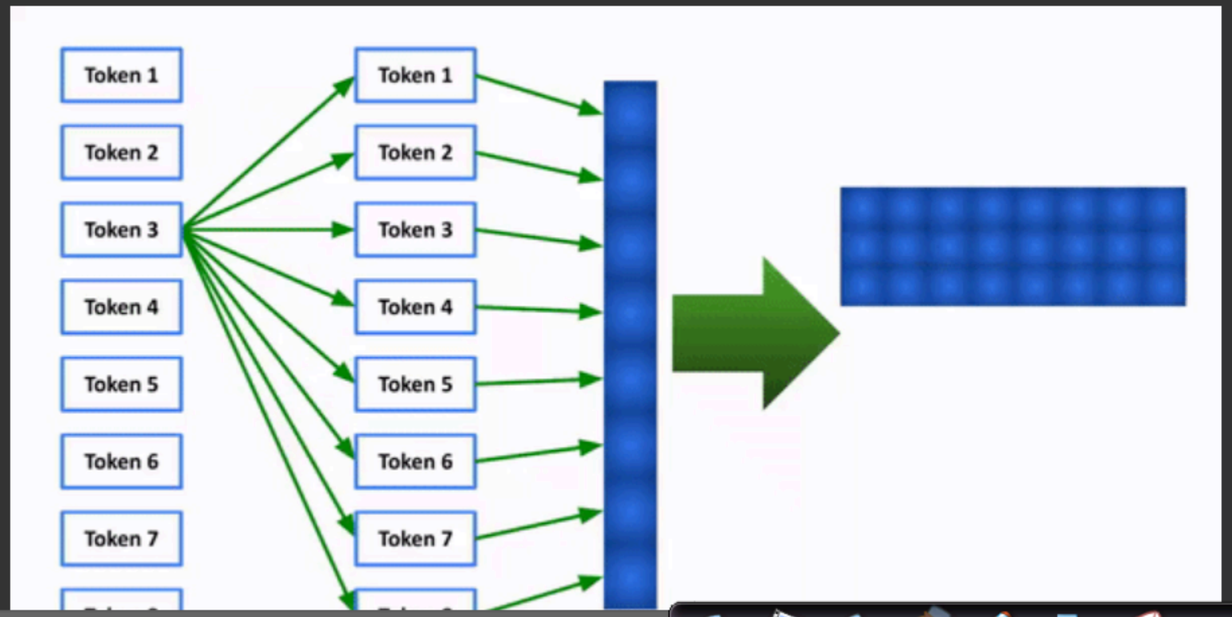
2. Self Attention AM

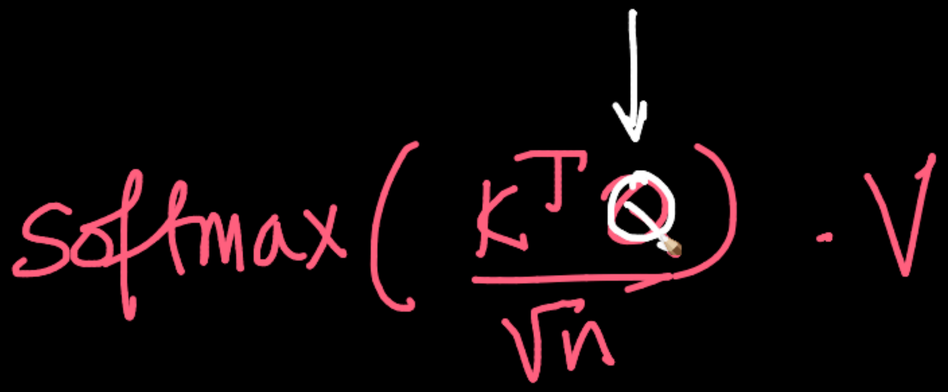
- In Self-Attention both KEY and QUERY sequence is Same.
- Basically, QUERY = KEY = VALUE
- **How to Compute Self AM?**
  - Compute KEY, QUERY and VALUE Vectors
  - Compute Energy function
  - Compute Attention Score



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





Intro_to_Attention_Mechanisms_V1.ipynb

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