Week 1 - Sentiment Analysis

For this week, we will use NN for sentiment analysis.

Structure of Our NN

The first layer will be an embedding layer.

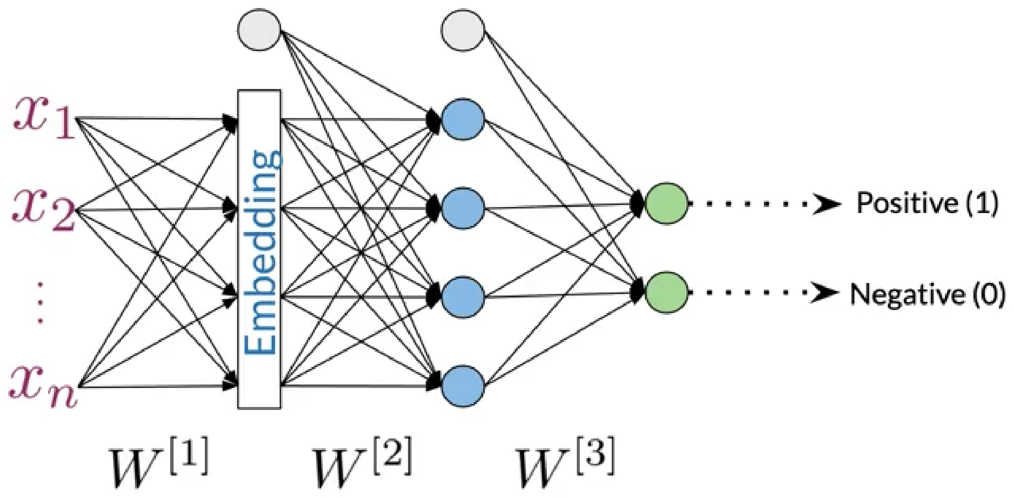


Figure 1

First, we will encode our vocabulary as integers. Then, the tweets will be encoded as a list of integers (of the corresponding words). They will be zero-padded to match the longest tweet.

For this week, the Trax framework will be used.

Importing layers

```
from trax import layers as tl
Model = tl.Serial(
        tl.Dense(4),
        tl.Sigmoid(),
        tl.Dense(4),
        tl.Sigmoid(),
        tl.Dense(3),
        tl.Softmax())
```

Why frameworks?

- Runs fast on CPUs, GPUs, and TPUs
- Parallel Computing
- Record algebraic computation for gradient evaluation

Classes in Python

The `__init__` method is called when the class is instantiated.

The `__call__` method is called when the instantiated class is called (again).

Week 2 - Recurrent Neural Networks

In n-gram models studied earlier, the probability of a sentence was derived from conditional probabilities of long sequences, which are often not found. Hence, the target sentence has a zero probability.

In short, n-gram models fail to capture dependencies between distant words. Also, they need a slot of memory.

RNNs use shared weights to reduce memory usage. Also, certain RNN models (e.g GRUs and LSTMs) easily capture long-distance dependencies.

Types of RNN Architecture

➤ One-to-One (like regression; not much useful)

➤ One-to-Many (e.g. Image captioning)

➤ Many-to-One (e.g. sentiment analysis)

➤ Many-to-Many (e.g. translation)

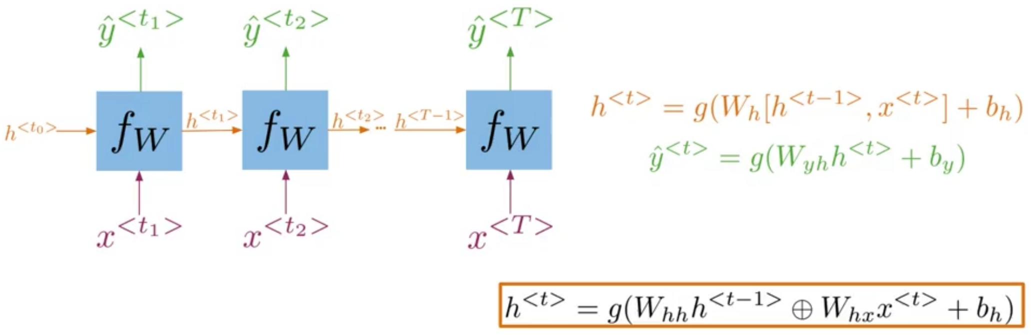

$\hat{y}^{<t_1>}$
$\hat{y}^{<t_2>}$
$\hat{y}^{<T>}$
$h^{<t_0>}$
f_W
$h^{<t_1>}$
$h^{<t_2>}$
$h^{<T-1>}$
$x^{<t_1>}$
$x^{<t_2>}$
$x^{<T>}$
$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$
$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$
$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$

Figure 2

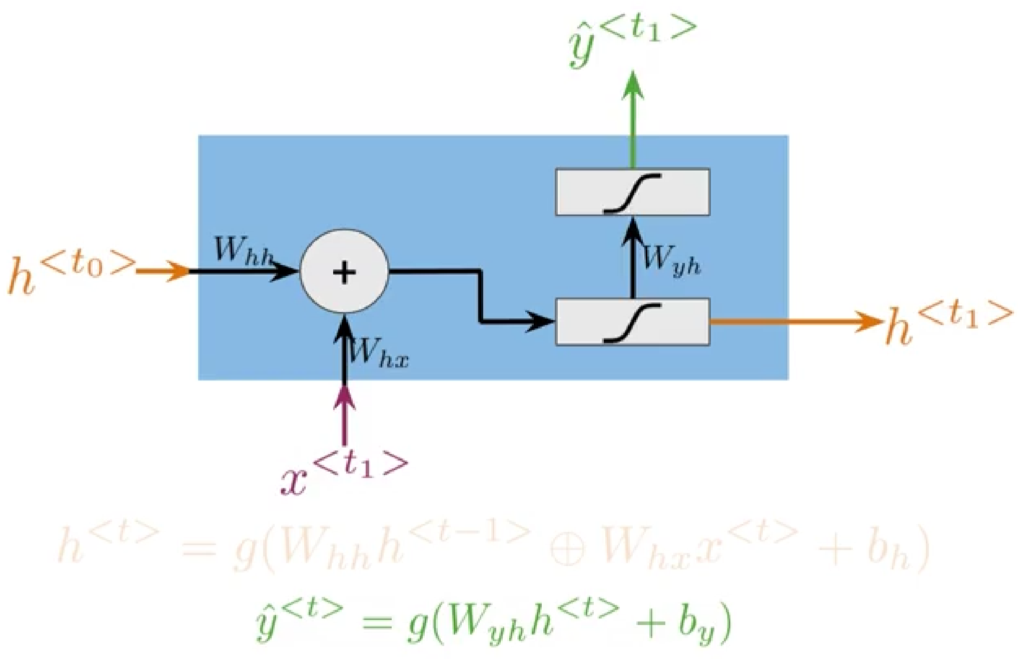

$\hat{y}^{<t_1>}$
$h^{<t_0>}$
W_{hh}
+
W_{yh}
W_{hx}
$h^{<t_1>}$
$x^{<t_1>}$
$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$
$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$

Figure 3

Loss Function

We will use the cross entropy loss function, given as

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{K} y_j \log \hat{y_j}$$

where the NN has K-classes or possibilities, and T is the 'number of steps' (or e.g. number of words in a sentence etc.)

To repeat over the T steps, we can use a scan function as follows initializer

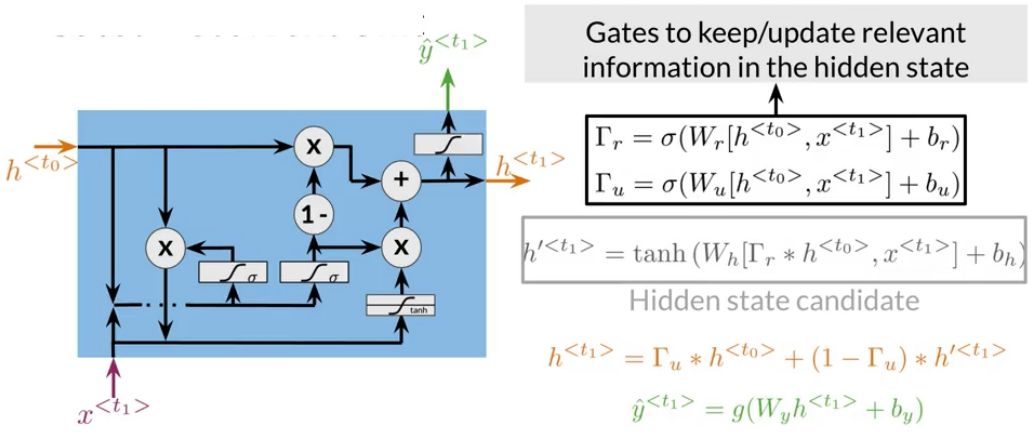
```python
def scan(fn, elems, initializer=None, ...):
    cur_value = initializer
    ys = []

    for x in elems:
        y, cur_value = fn(x, cur_value)
        ys.append(y)

    return ys, cur_value
```

This type of abstraction is needed for frameworks to work properly.

GRUs

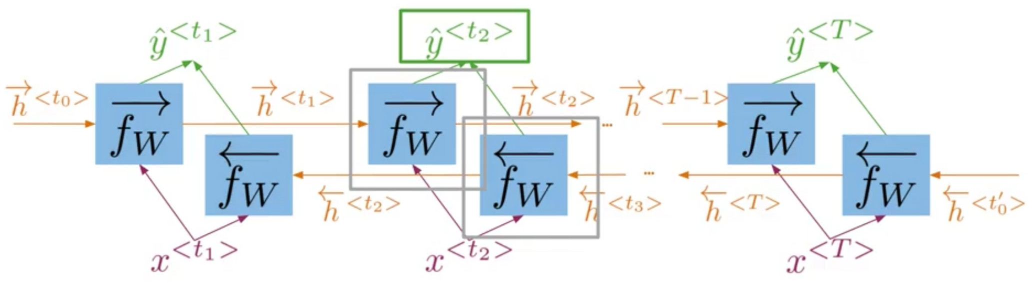


$\hat{y}^{<t_1>}$
Gates to keep/update relevant information in the hidden state
$h^{<t_0>}$
$h^{<t_1>}$
$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$
$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$
$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$
Hidden state candidate
$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$
$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$
$x^{<t_1>}$
x
+
1 -
tanh

Figure 4

Bi-directional GRUs

In BGRUs, information flows from the past and from the future independently.


$\hat{y}^{<t_1>}$
$\hat{y}^{<t_2>}$
$\hat{y}^{<T>}$
$\overrightarrow{h}^{<t_0>}$
$\overrightarrow{h}^{<t_1>}$
$\overrightarrow{h}^{<t_2>}$
$\overrightarrow{h}^{<T-1>}$
$\overrightarrow{fw}$
$\overleftarrow{fw}$
$\overleftarrow{h}^{<t_2>}$
$\overleftarrow{h}^{<t_3>}$
$\overleftarrow{h}^{<T>}$
$\overleftarrow{h}^{<t_0'>}$
$x^{<t_1>}$
$x^{<t_2>}$
$x^{<T>}$

$$\hat{y}^{<t>} = g(W_y[\overrightarrow{h}^{<t>}, \overleftarrow{h}^{<t>}] + b_y)$$

Figure 5

Week 3 - LSTMs

Vaniall RNNs do take up less RAM than traditional n-gram models, but they only capture short-range dependencies. They, however, fail to capture long-term dependencies and are also prone to vanishing/exploding gradients.

Vanishing/Exploding Gradients

Since a number of gradients are multiplied together in back-propagation, inputs earlier in the sequence may be multiplied with a 0 gradient - hence the name vanishing gradient. Such an input will not be updated and will not affect the output. Similarly, if one of the gradient 'explodes', the entire product will be infinity - hence the name exploding gradient.

Solutions

➤ Use RELU (or other activation functions with non-zero gradients)

➤ Gradient clipping - restrict gradient to a certain range

➤ Use skip-connections to override specific layers

LSTMs

LSTMs enforce memory by 'learning' what to remember and what to forget.

Anatomy

➤ A cell state

➤ A hidden state (with three gates)
  - Forget Gate
  - Input Gate
  - Output Gate

➤ Loops back again at the end of each time step

The gates allow information to flow unchanged.

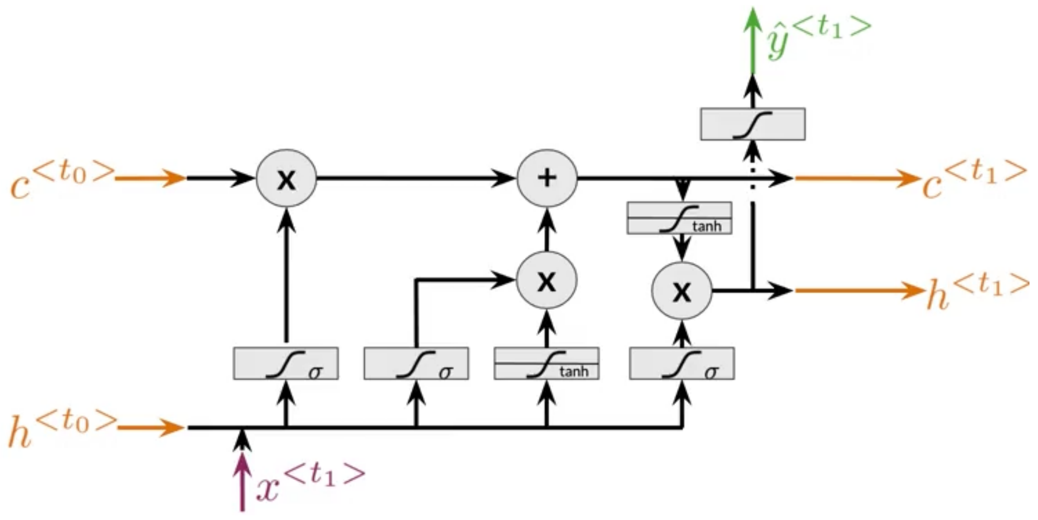


Figure 6

Named Entity Recognition

This task involves identifying words in a sentence as a named entity e.g. place, time, name, etc.

Week 4 - Siamese Networks

Siamese Networks can be used to detect duplicate statements.

They consist of two 'identical' networks - hence only one of them is trained, and then replicated.

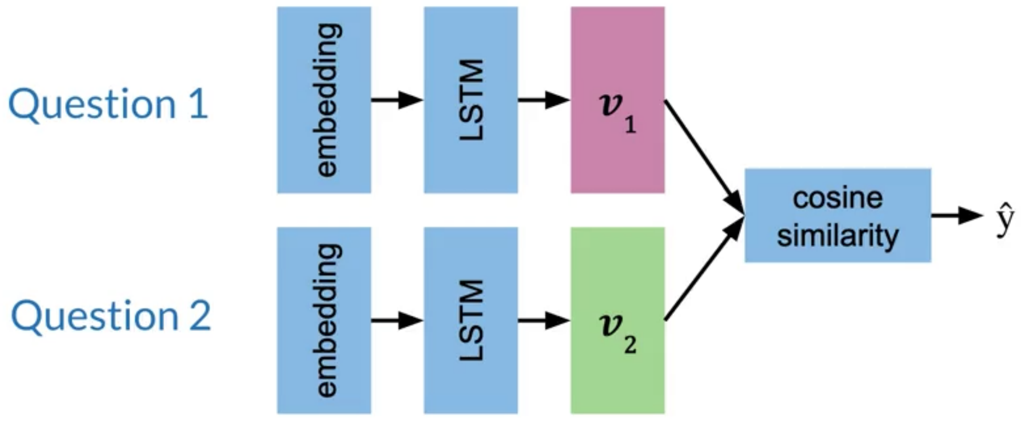


Figure 7

Note that -1 $\leq \hat{y} \leq$ 1. The questions are categorized as follows

$$\hat{y} = \begin{cases} \leq \tau & \text{different} \\ > \tau & \text{same} \end{cases}$$

where τ is the threshold (a hyperparameter)

Cost Function

(In the following context, positive/negative means that the question has the same/opposite meaning; it has nothing to do with the sentiment)

The cosine function is used to find the similarity between an anchor and its counterpart.

$$s(v_1, v_2) = \cos(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \|v_2\|}$$

Question	Type	Similarity
How old are you?	Anchor	-
What is your age?	Positive	s(A,P) $\approx$ 1
Where are you from?	Negative	s(A,N) $\approx$ -1

$$\text{diff} = s(A, N) - s(A, P)$$

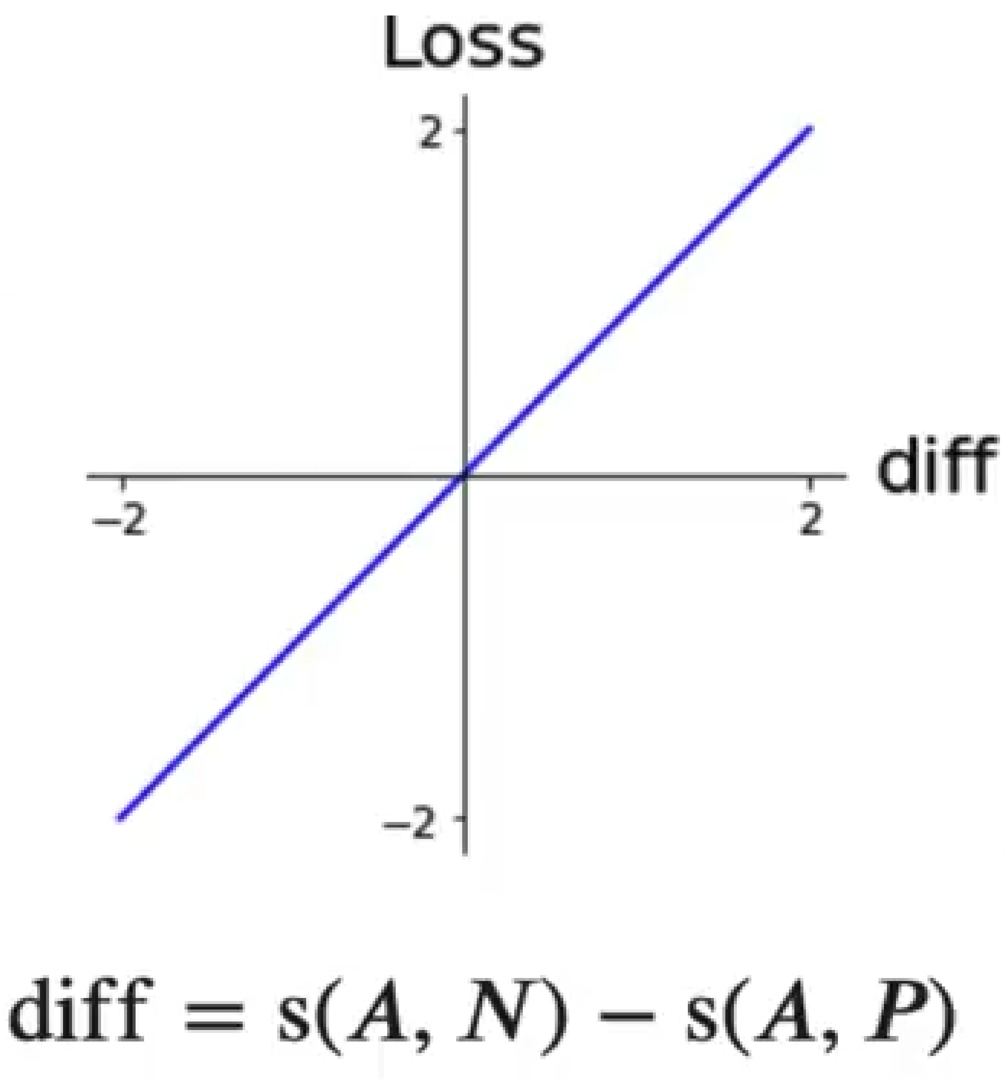
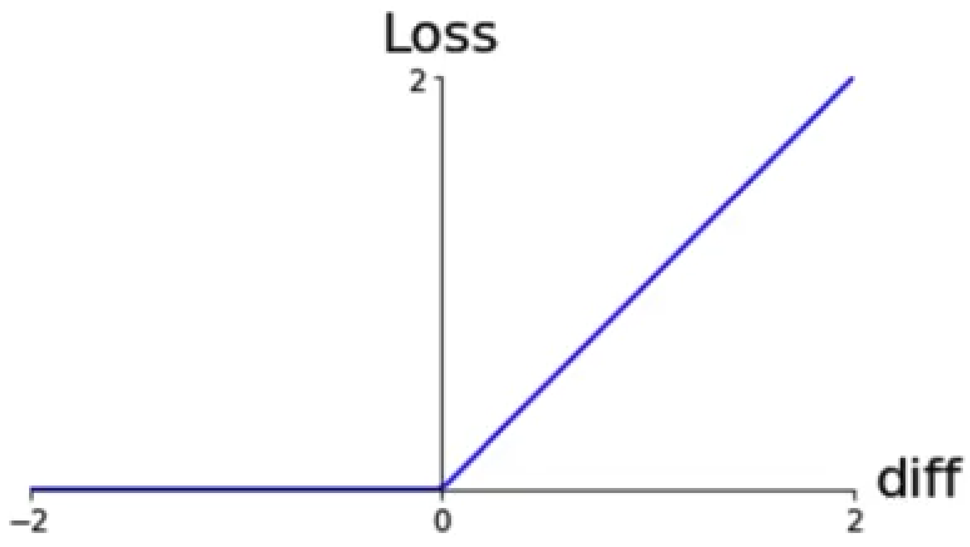


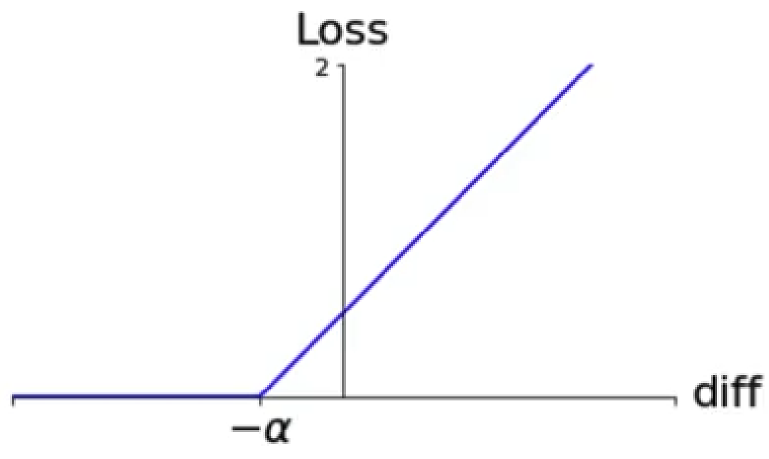
Figure 8

But, for loss < 0, we don't really want the network to learn much because it is OK... Therefore, our loss function will be modified to one of the following.



$$\mathcal{L} = \begin{cases} 0; & \text{if } diff \leq 0 \\ diff; & \text{if } diff > 0 \end{cases}$$

Figure 9



$$\mathcal{L} = \begin{cases} 0; & \text{if } diff + \alpha \leq 0 \\ diff + \alpha; & \text{if } diff + \alpha > 0 \end{cases}$$

Figure 10

$$\mathscr{L} = \max(s(A, N) - s(A, P) + \alpha, 0)$$

$$\mathscr{J} = \sum_{i=1}^{m} \mathscr{L}$$

Hard triplets are ones where s(A,P) $\approx$ s(A,N). They are hard to train, but will result in a well-trained network.

Preparing Data

- ➤ Prepare batches (of size b)with mutually exclusive questions
- ➤ Generate embedding matrices for each batch
- ➤ Compute the loss function

Few Concepts to train hard triplets

Mean negative: mean of off-diagonal values in each row

Closest negative: off-diagonal value closest to (but less than) the value on diagonal for each row.

The latter ensures that the similarity functions are close in value (despite being different in meaning)

$$\mathscr{L}_1 = \max(mean\_neg - s(A,P) + \alpha, 0)$$

Now, instead of training on n-1 (= 3 in this case) examples, it just trains using the mean. This reduces noise during training.

$$\mathscr{L}_2 = \max(closest\_neg - s(A,P) + \alpha, 0)$$

This uses the example with the smallest difference between the cosine similarities.

$$\mathscr{L}_{new} = \mathscr{L}_1 + \mathscr{L}_2$$

$$\mathscr{J} = \sum_{i=1}^{m} \mathscr{L}_{new}$$