Deep Learning

Homework 2

Group 71 members:

- Luis Jose De Macedo Guevara (95621)
- Vincent Jakl (108529)

Contributions of each member

- Luis Jose De Macedo Guevara (95621):
- Vincent Jakl (108529):

Question 1

1.

In order to determine the time complexity of performing the computation $\boldsymbol{Z} = \text{Softmax}\left(\boldsymbol{Q}\boldsymbol{K}^\top\right)\boldsymbol{V}$, where $\boldsymbol{Q} \in \mathbb{R}^{L\times D}$, $\boldsymbol{K} \in \mathbb{R}^{L\times D}$ and $\boldsymbol{V} \in \mathbb{R}^{L\times D}$ for a sequence length L and hidden size D, we consider three steps

1. The matrix multiplication $\boldsymbol{Q}\times\boldsymbol{K}^\top$, which has complexity $O\left(L \times D \times L\right) = O\left(L^2 D\right)$
2. The softmax transformation applied to matrix $\boldsymbol{Q}\boldsymbol{K}^\top$ (size $L \times L$), which has complexity $O\left(L^2\right)$
3. The matrix multiplication $\text{Softmax}\left(\boldsymbol{Q}\boldsymbol{K}^\top\right)\boldsymbol{V}$, which has complexity $O\left(L \times L \times D\right) = O\left(L^2 D\right)$

Resulting in a time complexity of $O\left(L^2 D + L^2 + L^2 D\right) = O\left(L^2 D\right)$. As we can see, this is a function of the sequence length (and to a lesser extent the hidden size), this means that the complexity grows quadratically as the sequence length grows.

2.

$$
\begin{aligned}
\exp\left(\boldsymbol{q}^\top\boldsymbol{k}\right) &\approx 1 + \boldsymbol{q}^\top\boldsymbol{k} + \frac{\left(\boldsymbol{q}^\top\boldsymbol{k}\right)^2}{2} \\
&= 1 + \boldsymbol{q}^\top\boldsymbol{k} + \frac{1}{2}\left(\sum_{i=1}^{D} q_i k_i\right)\left(\sum_{j=1}^{D} q_j k_j\right) \\
&= 1 + \boldsymbol{q}^\top\boldsymbol{k} + \frac{1}{2}\left(\sum_{i=1}^{D}\sum_{j=1}^{D} q_i q_j k_i k_j\right) \\
&= \underbrace{\begin{bmatrix} 1 & [q_i]_{i=1,\dots D}^\top & \frac{1}{\sqrt{2}}[q_i q_j]_{i,j=1,\dots D}^\top \end{bmatrix}}_{1\times M} \underbrace{\begin{bmatrix} 1 \\ [k_i]_{i=1,\dots D} \\ \frac{1}{\sqrt{2}}[k_i k_j]_{i,j=1,\dots D} \end{bmatrix}}_{M\times 1} \\
&= \phi\left(\boldsymbol{q}\right)^\top \phi\left(\boldsymbol{k}\right)
\end{aligned}
$$

Where $\phi : \mathbb{R}^D \to \mathbb{R}^M$ with $M = 1 + D + D^2$, if we used $K \geq 3$ terms from the MacLaurin series, we'd have $M = \sum_{i=0}^{K} D^i$.

3.

$$
\begin{aligned}
\boldsymbol{Z} &= \operatorname{Softmax}\left(\boldsymbol{Q}\boldsymbol{K}^{\top}\right)\boldsymbol{V} \\
&= \operatorname{Softmax}\left(\begin{bmatrix} \boldsymbol{q}_1^{\top}\boldsymbol{k}_1 & \boldsymbol{q}_1^{\top}\boldsymbol{k}_2 & \cdots & \boldsymbol{q}_1^{\top}\boldsymbol{k}_L \\ \boldsymbol{q}_2^{\top}\boldsymbol{k}_1 & \boldsymbol{q}_2^{\top}\boldsymbol{k}_2 & \cdots & \boldsymbol{q}_2^{\top}\boldsymbol{k}_L \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{q}_L^{\top}\boldsymbol{k}_1 & \boldsymbol{q}_L^{\top}\boldsymbol{k}_2 & \cdots & \boldsymbol{q}_L^{\top}\boldsymbol{k}_L \end{bmatrix}\right)\boldsymbol{V} \\
&= \begin{bmatrix} \frac{\exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_1\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_j\right)} & \frac{\exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_2\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_j\right)} & \cdots & \frac{\exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_L\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_j\right)} \\ \frac{\exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_1\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_j\right)} & \frac{\exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_2\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_j\right)} & \cdots & \frac{\exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_L\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_j\right)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_1\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_j\right)} & \frac{\exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_2\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_j\right)} & \cdots & \frac{\exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_L\right)}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_j\right)} \end{bmatrix}\boldsymbol{V} \\
&= \operatorname{Diag}\left(\begin{bmatrix} \frac{1}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_j\right)} \\ \vdots \\ \frac{1}{\sum_{j=1}^{L}\exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_j\right)} \end{bmatrix}\right)\begin{bmatrix} \exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_1\right) & \exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_2\right) & \cdots & \exp\left(\boldsymbol{q}_1^{\top}\boldsymbol{k}_L\right) \\ \exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_1\right) & \exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_2\right) & \cdots & \exp\left(\boldsymbol{q}_2^{\top}\boldsymbol{k}_L\right) \\ \vdots & \vdots & \ddots & \vdots \\ \exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_1\right) & \exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_2\right) & \cdots & \exp\left(\boldsymbol{q}_L^{\top}\boldsymbol{k}_L\right) \end{bmatrix}\boldsymbol{V} \\
&\approx \operatorname{Diag}\left(\begin{bmatrix} \frac{1}{\sum_{j=1}^{L}\phi(\boldsymbol{q}_1)^{\top}\phi(\boldsymbol{k}_j)} \\ \vdots \\ \frac{1}{\sum_{j=1}^{L}\phi(\boldsymbol{q}_L)^{\top}\phi(\boldsymbol{k}_j)} \end{bmatrix}\right)\Phi\left(\boldsymbol{Q}\right)\Phi\left(\boldsymbol{K}\right)^{\top}\boldsymbol{V} \\
&= \left(\operatorname{Diag}\left(\begin{bmatrix} \sum_{j=1}^{L}\phi\left(\boldsymbol{q}_1\right)^{\top}\phi\left(\boldsymbol{k}_j\right) \\ \vdots \\ \sum_{j=1}^{L}\phi\left(\boldsymbol{q}_L\right)^{\top}\phi\left(\boldsymbol{k}_j\right) \end{bmatrix}\right)\right)^{-1}\Phi\left(\boldsymbol{Q}\right)\Phi\left(\boldsymbol{K}\right)^{\top}\boldsymbol{V} \\
&= \left(\operatorname{Diag}\left(\begin{bmatrix} \phi\left(\boldsymbol{q}_1\right)^{\top}\phi\left(\boldsymbol{k}_1\right) & \cdots & \phi\left(\boldsymbol{q}_1\right)^{\top}\phi\left(\boldsymbol{k}_L\right) \\ \vdots & \ddots & \vdots \\ \phi\left(\boldsymbol{q}_L\right)^{\top}\phi\left(\boldsymbol{k}_1\right) & \cdots & \phi\left(\boldsymbol{q}_L\right)^{\top}\phi\left(\boldsymbol{k}_L\right) \end{bmatrix}\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}\right)\right)^{-1}\Phi\left(\boldsymbol{Q}\right)\Phi\left(\boldsymbol{K}\right)^{\top}\boldsymbol{V} \\
&= \left(\operatorname{Diag}\left(\Phi\left(\boldsymbol{Q}\right)\Phi\left(\boldsymbol{K}\right)^{\top}\mathbf{1}_L\right)\right)^{-1}\Phi\left(\boldsymbol{Q}\right)\Phi\left(\boldsymbol{K}\right)^{\top}\boldsymbol{V} \\
&= \boldsymbol{D}^{-1}\Phi\left(\boldsymbol{Q}\right)\Phi\left(\boldsymbol{K}\right)^{\top}\boldsymbol{V}
\end{aligned}
$$

4.

Given $\boldsymbol{Q} \in \mathbb{R}^{L \times D}$, $\boldsymbol{K} \in \mathbb{R}^{L \times D}$ and $\boldsymbol{V} \in \mathbb{R}^{L \times D}$, to compute $\boldsymbol{Z} \approx \boldsymbol{D}^{-1} \Phi(\boldsymbol{Q}) \Phi(\boldsymbol{K})^{\top} \boldsymbol{V}$

1. We compute $\Phi(\boldsymbol{Q})$ and $\Phi(\boldsymbol{K})$, both can be computed with a complexity of $O(LDM)$
2. We compute $\Phi(\boldsymbol{K})^{\top} \times \boldsymbol{V}$ corresponding to a complexity of $O(MLD)$
3. We compute $\Phi(\boldsymbol{Q}) \times \Phi(\boldsymbol{K})^{\top} \boldsymbol{V}$ corresponding to a complexity of $O(LMD)$
4. To normalize the scores, this is, $\boldsymbol{D}^{-1} \times \Phi(\boldsymbol{Q}) \Phi(\boldsymbol{K})^{\top} \boldsymbol{V}$, we need the sums of the rows of $\Phi(\boldsymbol{Q}) \Phi(\boldsymbol{K})^{\top}$.

   We know that $\left(\Phi(\boldsymbol{Q}) \Phi(\boldsymbol{K})^{\top}\right)_{i,j} = \phi(\boldsymbol{q}_i)^{\top} \phi(\boldsymbol{k}_j)$ therefore, the sum of row i is given by $\sum_{j=1}^{L} \phi(\boldsymbol{q}_i)^{\top} \phi(\boldsymbol{k}_j) = \phi(\boldsymbol{q}_i)^{\top} \sum_{j=1}^{L} \phi(\boldsymbol{k}_j)$.

   (a) $\sum_{j=1}^{L} \phi(\boldsymbol{k}_j)$ can be computed in $O(LM)$ and stored.

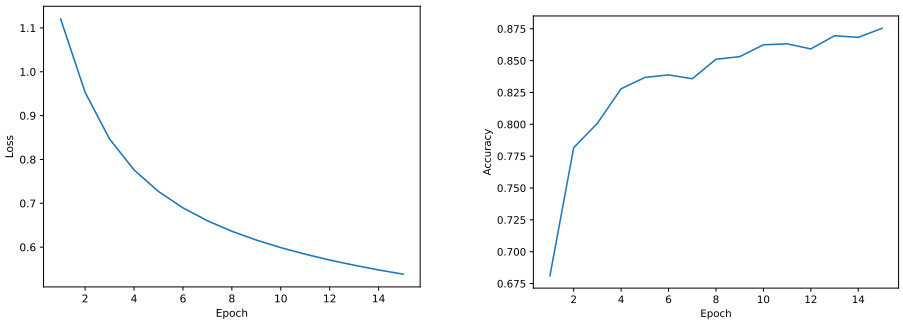
   (b) We reuse $\sum_{j=1}^{L} \phi(\boldsymbol{k}_j)$ to compute $\phi(\boldsymbol{q}_i)^{\top} \sum_{j=1}^{L} \phi(\boldsymbol{k}_j)$ for each row i, this is done in $O(M)$, leading to a complexity of $O(LM)$ for the L rows.
5. Having the sums of the rows, what remains is normalizing the scores, this is, dividing each element of $\Phi(\boldsymbol{Q}) \Phi(\boldsymbol{K})^{\top} \boldsymbol{V}$ $\left(\in \mathbb{R}^{L \times D}\right)$ by a scalar, leading to a complexity of $O(LD)$

This leads us to a complexity of $O(LDM + LM + LD) = O(LDM)$

Question 2

1.

Running 15 epochs with learning rates 0.1, 0.01 and 0.001 we get that the CNN performs best with learning rate 0.01 with a final validation accuracy of 0.8754 and final test accuracy of 0.8318.



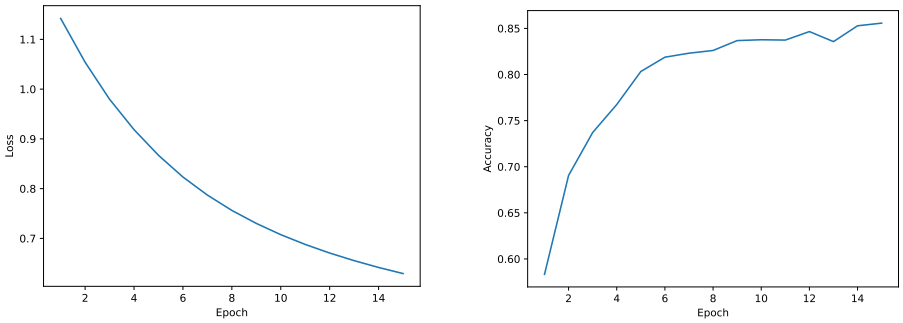
(a) Training loss per epoch (b) Validation accuracy per epoch

Figure 1: Training metrics for the CNN with the max pooling layers

As can be seen in Figure 1, both the training loss and the validation accuracy consistently improve with each epoch, suggesting that perhaps more epochs would result in a better performance.

2.

Running 15 epochs with the same optimal hyperparameters from Question 2.1 but without max pooling layers and slightly different convolution layers as specified, we get a final validation accuracy of 0.8557 and final test accuracy of 0.8110.



(a) Training loss per epoch (b) Validation accuracy per epoch

Figure 2: Training metrics for the CNN **without** the max pooling layers

As can be seen in Figure 2 and as was the case in Question 2.1 both the training loss and the validation accuracy improve with each epoch, suggesting that more epochs could result in a better performance.

3.

Through the function get_number_trainable_params we get that the CNNs from Question 2.1 and 2.2 have the same number of trainable parameters, this is, 224892, yet the CNN with max pooling layers (the one from Question 2.1) reaches a better performance for the same hyperparameters and number of epochs. This could be because the max pooling layers help to detect the desired features from the images by discarding less relevant features, and introducing translation, rotation and scale invariance into the network, making it more robust to these types of transformations in the input data.

Question 3

1.

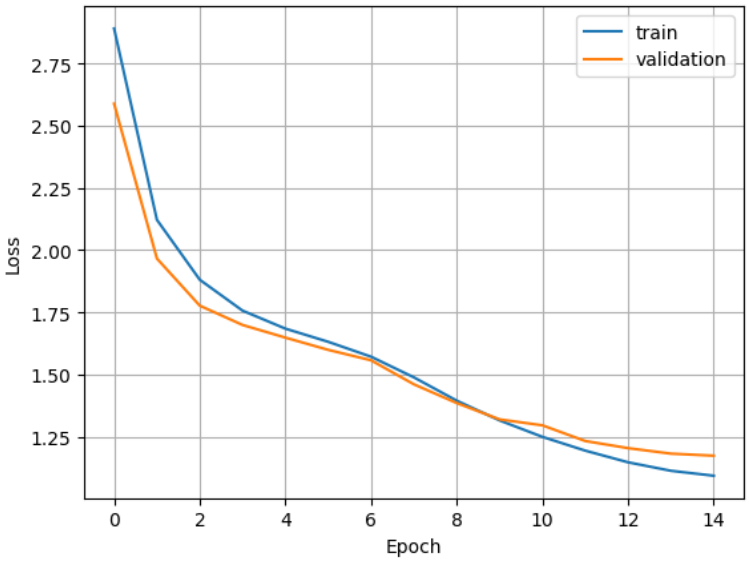
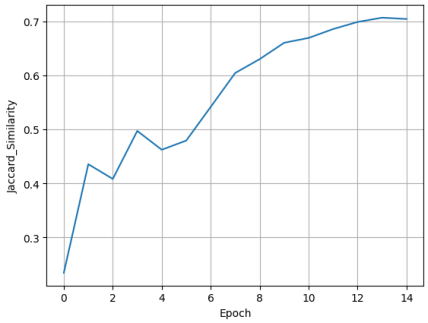
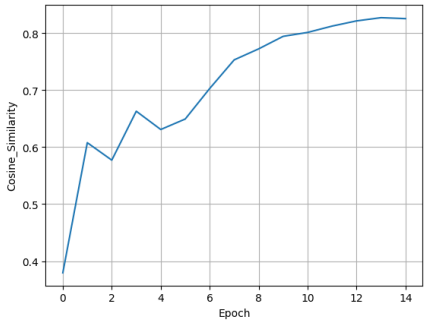


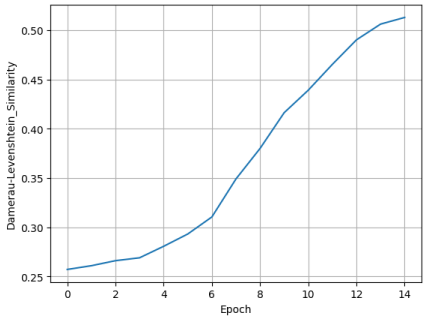
Figure 3: Training and validation loss for the RNN model.



(a) Training Jaccard similarity per epoch



(b) Training cosine similarity per epoch



(c) Training Damerau-Levenstein similarity per epoch

Figure 4: Training similarity metrics per epoch for the RNN model.

The final testing metrics for the RNN were as follows:

- Jaccard similarity: 0.7149
- Cosine similarity: 0.8324
- Damerau-Levenstein similarity: 0.5087
- Loss: 1.1828

2.

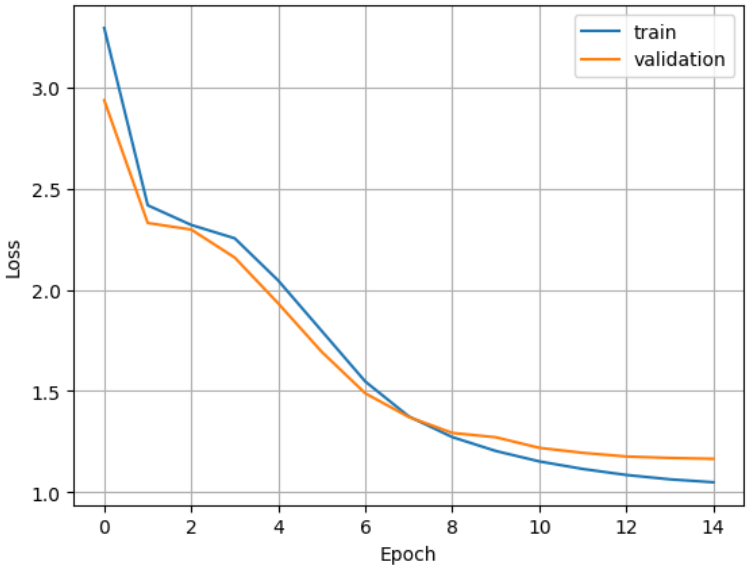
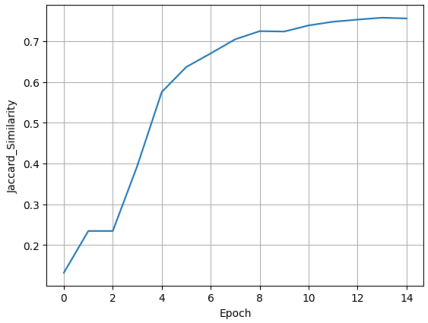
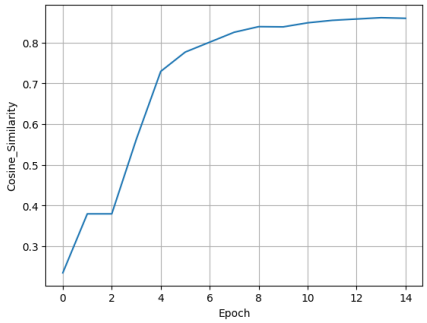


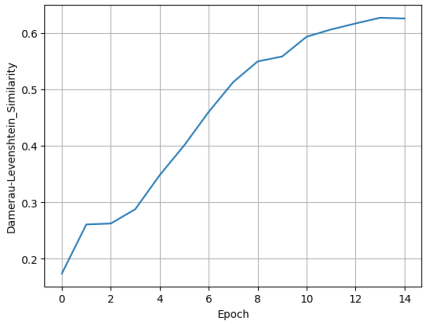
Figure 5: Training and validation loss for the RNN model.



(a) Training Jaccard similarity per epoch



(b) Training cosine similarity per epoch



(c) Training Damerau-Levenstein similarity per epoch

Figure 6: Training similarity metrics per epoch for the transformer model.

The final testing metrics for the transformer were as follows:

- Jaccard similarity: 0.7648
- Cosine similarity: 0.8655
- Damerau-Levenstein similarity: 0.6344
- Loss: 1.1596

3.

The LSTM decoder in question 1 uses a RNN to process the text input sequentially. It maintains a hidden state and a cell state that encode the previous output and the context vector from the encoder. The LSTM decoder generates the next output token based on the current input token, the

hidden state, and the cell state. It doesn't have direct access to the whole encoder output nor to the decoder outputs.

The attention decoder in question 2 uses a transformer to process the text input in parallel, without relying on recurrence. It uses an attention mechanism to learn the relevance between the encoder output and the decoder input, and to produce a weighted context vector for each decoder input token. The attention decoder generates the next output token based on the current input token, the context vector, and the previous decoder outputs. It has direct access to the entire encoder output and the previous decoder outputs. The non-reliance on recurrence allows the transformer to be run in parallel, which would explain the speed of training of 15 epochs of the transformer being about 10 minutes faster than the RNN.

The attention decoder in question 2 generally achieves better performance than the LSTM decoder in question 1. This is because the attention decoder can capture the long-range dependencies and the global structure of the text input more effectively than the LSTM decoder. The LSTM decoder can suffer from the vanishing gradient problem.

4.

Jaccard similarity is a measure of similarity between two sets. It is the size of the intersection of the two sets divided by the size of the union of them.

Cosine similarity is a measure of similarity between two vectors. It basically represents the angle between the two vectors. The higher the cosine similarity, the smaller the angle and the more similar the two vectors are.

Damerau-Levenstein similarity is a measure of similarity between two strings. It looks at the number of operations (inserion, deletion, transposition, substitution) needed to transform one string into the other. It is calculated by subtracting the edit distance from the maximum possible distance, and dividing by the maximum possible distance.

All the similarity measures listed above can be applied to the text input and the text output of the model. They differ in the way they measure the similarity. The Jaccard similarity only compares the presence and non-presence of tokens. It does not take into account the order and the frequency of the tokens. The cosine similarity takes only into account the direction of the vectors but not their magnitude. Whereas the Damerau-Levenstein similarity focuses on the amount of operations needed to transform one string into another. Each of the similarity measures then differs in the way they are calculated and their respective values. They can all be used depending on the application and the desired similarity measure.