ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Natural Language Processing
Spring 2021

Prof. Ryan Cotterell

Course Assignment

July 12, 2021

Sven Kohler
nethz Username: svkohler
Student ID: 15-731-524

Collaborators:
Karol Borkowski
Irfan Bunjaku

By submitting this work, I verify that it is my own. That is, I have written my own solutions to each problem for which I am submitting an answer. I have listed above all others with whom I have discussed these answers.

Question 1: Backpropagation

(a) Drawing the computational graph

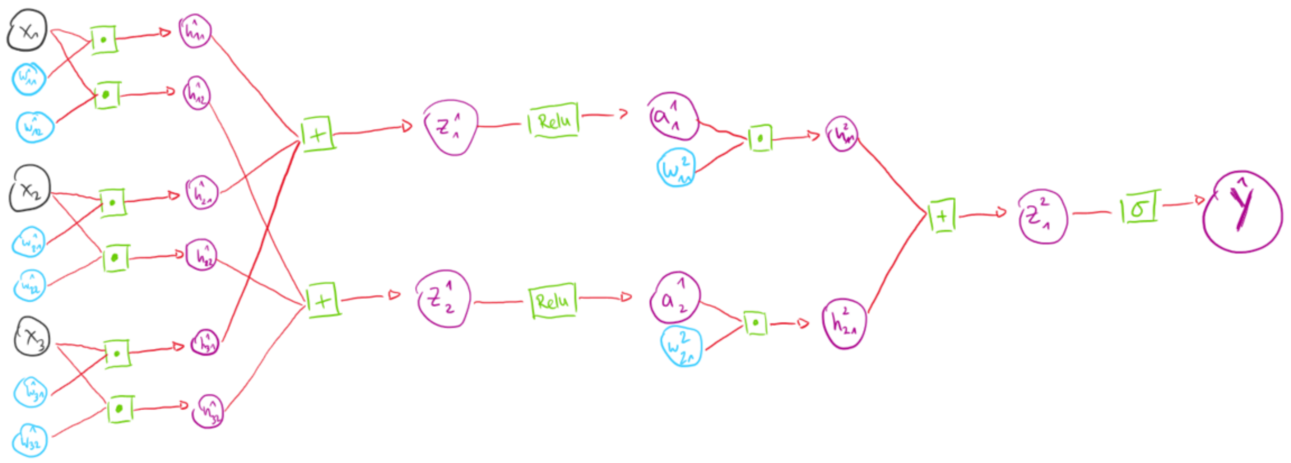


Figure 1: Corresponding computational graph to Neural Network f

(b) Computations using NN f, cross entropy loss and data-point $(x_0, y_0) = ((1,1,1), 0)$

(i) **Evaluate f at (x_0, y_0)**

We know that all the weights are initialized to be ones. Calculate all the intermediate result according to the computational graph:

$$\begin{array}{r|c|c}
h^1_{11} = 1 * 1 = 1 & z^1_1 = 1 + 1 + 1 = 3 & h^2_{11} = 3 * 1 = 3 \\
h^1_{12} = 1 * 1 = 1 & z^1_2 = 1 + 1 + 1 = 3 & h^2_{21} = 3 * 1 = 3 \\
h^1_{21} = 1 * 1 = 1 & a^1_1 = Relu(3) = max(3,0) = 3 & z^2_1 = 3 + 3 = 6 \\
h^1_{22} = 1 * 1 = 1 & a^1_2 = Relu(3) = max(3,0) = 3 & \hat{y} = \sigma(6) = 0.99753 \\
h^1_{31} = 1 * 1 = 1 & & \\
h^1_{32} = 1 * 1 = 1 & &
\end{array}$$

f evaluated at (x_0, y_0) is 0.99753.

(ii) Compute the partial derivatives of the network weights w^k_{ij}.

Start by computing **ALL** partial derivatives:

$$\begin{array}{cc|c|c}
\frac{\partial h^1_{11}}{\partial x_1} = w^1_{11} = 1 & \frac{\partial h^1_{11}}{\partial w^1_{11}} = x_1 = 1 & \frac{\partial z^1_1}{\partial h^1_{11}} = 1,\ \frac{\partial z^1_1}{\partial h^1_{21}} = 1,\ \frac{\partial z^1_1}{\partial h^1_{31}} = 1 & \frac{\partial h^2_{11}}{\partial a^1_1} = w^2_{11} = 1\ \frac{\partial h^2_{11}}{\partial w^2_{11}} = a^1_1 = 3 \\
\frac{\partial h^1_{12}}{\partial x_1} = w^1_{12} = 1 & \frac{\partial h^1_{12}}{\partial w^1_{12}} = x_1 = 1 & \frac{\partial z^1_2}{\partial h^1_{12}} = 1,\ \frac{\partial z^1_2}{\partial h^1_{22}} = 1,\ \frac{\partial z^1_2}{\partial h^1_{32}} = 1 & \frac{\partial h^2_{21}}{\partial a^1_2} = w^2_{21} = 1\ \frac{\partial h^2_{21}}{\partial w^2_{21}} = a^2_1 = 3 \\
\frac{\partial h^1_{21}}{\partial x_2} = w^1_{21} = 1 & \frac{\partial h^1_{21}}{\partial w^1_{21}} = x_2 = 1 & \frac{\partial a^1_1}{\partial z^1_1} = \begin{cases} 0 & \text{if } z^1_1 \leq 0 \\ 1 & \text{if } z^1_1 > 0 \end{cases} = 1 & \frac{\partial z^2_1}{\partial h^2_{11}} = 1,\ \frac{\partial z^2_1}{\partial h^2_{21}} = 1 \\
\frac{\partial h^1_{22}}{\partial x_2} = w^1_{22} = 1 & \frac{\partial h^1_{22}}{\partial w^1_{22}} = x_2 = 1 & \frac{\partial a^1_2}{\partial z^1_2} = \begin{cases} 0 & \text{if } z^1_2 \leq 0 \\ 1 & \text{if } z^1_2 > 0 \end{cases} = 1 & \frac{\partial \hat{y}}{\partial z^1_2} = \sigma(z^2_1)(1 - \sigma(z^2_1)) \\
\frac{\partial h^1_{31}}{\partial x_3} = w^1_{31} = 1 & \frac{\partial h^1_{31}}{\partial w^1_{31}} = x_3 = 1 & & = 0.99753 * 0.00247 = 0.00247 \\
\frac{\partial h^1_{32}}{\partial x_3} = w^1_{32} = 1 & \frac{\partial h^1_{32}}{\partial w^1_{32}} = x_3 = 1 & &
\end{array}$$

Derive the derivative of the sigmoid function:

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+exp(-x)} = [1+exp(-x)]^{-1} \\
\frac{\partial \sigma(x)}{\partial x} &= -1 * [1+exp(-x)]^{-2} * exp(-x) * -1 \\
&= \frac{exp(-x)}{1+exp(-x)} * \frac{1}{1+exp(-x)} \\
&= \sigma(x) * \left[\frac{1+exp(-x)-1}{1+exp(-x)}\right] \\
&= \sigma(x) * \left[1 - \frac{1}{1+exp(-x)}\right] \\
&= \sigma(x) * (1-\sigma(x))
\end{aligned}$$

Now we use the chain rule to get the partial derivatives for the network weights:

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial w^1_{11}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{11}} * \frac{\partial h^2_{11}}{\partial a^1_1} * \frac{\partial a^1_1}{\partial z^1_1} * \frac{\partial z^1_1}{\partial h^1_{11}} * \frac{\partial h^1_{11}}{\partial w^1_{11}} = 0.00247 * 1 * 1 * 1 * 1 * 1 = 0.00247 \\
\frac{\partial \hat{y}}{\partial w^1_{21}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{11}} * \frac{\partial h^2_{11}}{\partial a^1_1} * \frac{\partial a^1_1}{\partial z^1_1} * \frac{\partial z^1_1}{\partial h^1_{21}} * \frac{\partial h^1_{21}}{\partial w^1_{21}} = 0.00247 * 1 * 1 * 1 * 1 * 1 = 0.00247 \\
\frac{\partial \hat{y}}{\partial w^1_{31}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{11}} * \frac{\partial h^2_{11}}{\partial a^1_1} * \frac{\partial a^1_1}{\partial z^1_1} * \frac{\partial z^1_1}{\partial h^1_{31}} * \frac{\partial h^1_{31}}{\partial w^1_{31}} = 0.00247 * 1 * 1 * 1 * 1 * 1 = 0.00247 \\
\frac{\partial \hat{y}}{\partial w^1_{12}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{21}} * \frac{\partial h^2_{21}}{\partial a^1_2} * \frac{\partial a^1_2}{\partial z^1_2} * \frac{\partial z^1_2}{\partial h^1_{12}} * \frac{\partial h^1_{12}}{\partial w^1_{12}} = 0.00247 * 1 * 1 * 1 * 1 * 1 = 0.00247 \\
\frac{\partial \hat{y}}{\partial w^1_{22}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{21}} * \frac{\partial h^2_{21}}{\partial a^1_2} * \frac{\partial a^1_2}{\partial z^1_2} * \frac{\partial z^1_2}{\partial h^1_{22}} * \frac{\partial h^1_{22}}{\partial w^1_{22}} = 0.00247 * 1 * 1 * 1 * 1 * 1 = 0.00247 \\
\frac{\partial \hat{y}}{\partial w^1_{32}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{21}} * \frac{\partial h^2_{21}}{\partial a^1_2} * \frac{\partial a^1_2}{\partial z^1_2} * \frac{\partial z^1_2}{\partial h^1_{32}} * \frac{\partial h^1_{32}}{\partial w^1_{32}} = 0.00247 * 1 * 1 * 1 * 1 * 1 = 0.00247 \\
\frac{\partial \hat{y}}{\partial w^2_{11}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{11}} * \frac{\partial h^2_{11}}{\partial w^2_{11}} = 0.00247 * 1 * 3 = 0.0073995 \\
\frac{\partial \hat{y}}{\partial w^2_{21}} &= \frac{\partial \hat{y}}{\partial z^2_1} * \frac{\partial z^2_1}{\partial h^2_{21}} * \frac{\partial h^2_{21}}{\partial w^2_{21}} = 0.00247 * 1 * 3 = 0.0073995
\end{aligned}$$

(iii) Compute L_{BCE} and its derivatrive w.r.t. $\hat{y}$ using the natural logarithm.

$$\begin{aligned}
L_{BCE} &= -(y * log(\hat{y}) + (1-y) * log(1-\hat{y})) = -(0 * log(0.99753) + (1-0) * log(1-0.99753)) = 6.002476 \\
\frac{\partial L_{BCE}}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}} \left(-[y * log(\hat{y}) + (1-y) * log(1-\hat{y})]\right) \\
&= -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} * (-1) = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}} \\
&= \frac{1}{1-0.99753} = 404.42879
\end{aligned}$$

(iv) Perform a gradient update step:
In general the rule is as follows:

$$w_{new} \leftarrow w_{old} - \eta * \nabla_x L(x)$$

In our case:

$$w^1_{11,new} \leftarrow 1 - 0.1 * 404.42879 * 0.00247 = 0.90025$$

Computations, learning rates, weights, gradients are all the same for the weights on the first layer.

$$w^2_{11,new} \leftarrow 1 - 0.1 * 404.42879 * 0.0073995 = 0.70074$$

Computations, learning rates, weights, gradients are all the same for the weights on the second layer.

(c) Revised/optimized computational graph

One can see that a lot of gradients and intermediate results are the same. They stays the same even after a couple of gradient updates. This stems from the trivial initialization of the weights. In this setting one could reduce the hidden layer from two to one neuron and would not lose expressivity. This would only lead to a re-weighting of the parameter. This results in better memory usage and faster training/test time since the model is reduced.

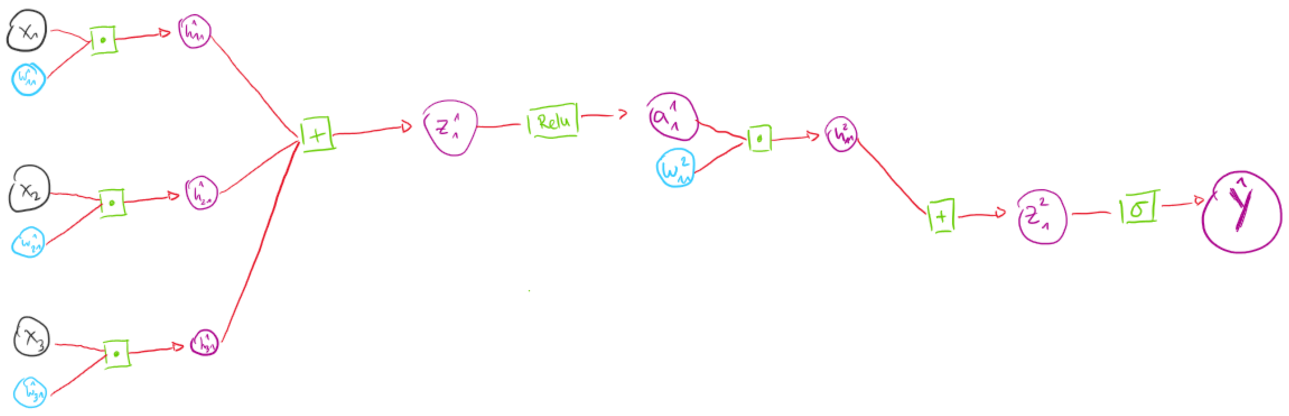


Figure 2: Revised Computational Graph

Question 2: Log-linear models

See Jupyter Notebook for solution.

Question 3: Skip-gram

See Jupyter Notebook for solution.

Question 4: Language Modelling

Setting: samples $x_1, x_2, x_3...$ are drawn uniformly at random from a corpus V.

(a) Mary draws n tokens

(i) Expected number of distinct tokens?
Define

$$N_k \equiv \# \text{ distinct tokens after k draws}$$

Then the probability to draw a new token at time $k+1$ is: $\frac{|V|-N_k}{|V|}$
Now we can derive the expected number of distinct tokens after $k+1$ draws:

$$E[N_{k+1}] = E[N_k] + \frac{|V| - E[N_k]}{|V|} = 1 + \underbrace{(1 - \frac{1}{|V|})}_{a} E[N_k] = 1 + aE[N_k]$$

$$= 1 + \underbrace{(1 - \frac{1}{|V|})}_{a} *(1 + \underbrace{(1 - \frac{1}{|V|})}_{a} *E[N_{k-1}]) = 1 + a(1 + aE[N_{k-1}])$$

$$= 1 + a + a^2E[N_{k-1}] = 1 + a + a^2(1 + aE[N_{k-2}]) = 1 + a + a^2 + a^3E[N_{k-2}]$$

$$= 1 + a + a^2 + ... + a^{k-1} + a^k \underbrace{E[N_0]}_{\overset{!}{=}0}$$

$$= 1 + a + a^2 + ... + a^{k-1} = \sum_{i=0}^{k-1} a^i$$

Take the right-hand side of the above equation and multiply by sides by a:

$$a + a^2 + ... + a^k = a\sum_{i=0}^{k-1} a^i$$

$$(\sum_{i=0}^{k-1} a^i) - 1 + a^k = a\sum_{i=0}^{k-1} a^i$$

Rearrange terms and you get:

$$\sum_{i=0}^{k-1} a^i[1-a] - 1 + a^k = 0$$

$$\sum_{i=0}^{k-1} a^i = \frac{1-a^k}{1-a} = \frac{1-(1-\frac{1}{|V|})^k}{1-(1-\frac{1}{|V|})} = |V|(1-(1-\frac{1}{|V|}))^k$$

For our setting we simply replace k by n.

(ii) Probability that all of V is drawn in n draws. Apporach via the coupon collector problem. The solution can be obtained via the following steps:

- Number of possible outcomes of n consecutive draws: $|V|^n$
- Now look at the number of possible combinations of tokens such that in n draws all of V is present.
  $\rightarrow$ do via the inclusion-exclusion principle

- In the end take the fraction of the two.

In the following the second step is explained more thoroughly:

- We start the inclusion-exclusion principle by assuming that all possible outcomes of n consecutive draws ($|V|^n$) are valid.
- However, this is an overcount, since the the set of all possible draws trivially also includes those which do not cover all of V.
- We now compensate and subtract for every token in V the amount $(|V|-1)^n$. Hereby, $(|V|-1)^n$ equals the number of draws which do not include a particular token. Basically we simulate a vocabulary which is smaller by one token.
  This leaves is with $|V|^n - |V|(|V|-1)^n$ draws after this iteration.
- However, we subtracted to much. For every token we subtracted the number of draws where said token was not present. E.g. for token "X" the subtracted draws also include draws where e.g. token "Y" is not present. The problem is that these draws where in addition to "X" also "Y" is missing get double counted when we subtract the draws where "Y" is not present.
  Hence, for each pair of tokens (there are $\binom{|V|}{2}$ pairs in a vocabulary of size $|V|$) we have to add $(|V|-2)^n$ draws. This represents the number of draws where neither token of the pair is present.
  Thus, we remain with $|V|^n - |V|(|V|-1)^n + \binom{|V|}{2}(|V|-2)^n$ draws after this iteration.
- However, this results in an overcount again (account for the occurence of triples) and we have to subtract.
- This goes on until we reach the $|V|$'th iteration and we are left with

$$\sum_{k=0}^{|V|}(-1)^k\binom{|V|}{k}(|V|-k)^n$$

  draws.

Now, we only must take the fraction of the valid draws and all draws, which leaves us with the probability that all tokens of the vocabulary have appeared:

$$\frac{\sum_{k=0}^{|V|}(-1)^k\binom{|V|}{k}(|V|-k)^n}{|V|^n}$$

(b) Bigram "work hard" has to appear in the corpus.

(i) What is the expected number of draws until bigram "work hard" appears.
Define

$$\text{K} \equiv \#\text{ draws until "hard" is drawn after "work"}$$

$$E[K] = \frac{|V|-1}{|V|}(E[K]+1) + \frac{1}{|V|} * \frac{|V|-1}{|V|}(E[K]+2) + \frac{1}{|V|^2} * 2$$

- In the first term $\frac{|V|-1}{|V|}$ stands for the probability to draw anything else than "work" and $(E[K]+1)$ means that you have to start fresh with drawing and increment the expected number of draws by one.

- In the second term $\frac{1}{|V|}$ stands for the probability to draw "work", followed by $\frac{|V|-1}{|V|}$ which stands for the probability to draw anything else than "hard". $(E[K]+2)$ means that you have to start fresh and "wasted" 2 draws in the process.
- The third term composes of $\frac{1}{|V|^2}$ which stands for the probability that one draws "work" and "hard" consecutively and the factor 2 meaning that you used 2 draws to get there.

Now, let's start manipulating the above equation.

$$\begin{aligned}
E[K] &= \frac{|V|-1}{|V|}E[K] + \frac{|V|-1}{|V|} + \frac{|V|-1}{|V|^2}E[K] + 2\frac{|V|-1}{|V|^2} + 2\frac{1}{|V|^2} \\
&= E[K](\frac{|V|-1}{|V|} + \frac{|V|-1}{|V|^2}) + \frac{|V|-1}{|V|} + \frac{2}{|V|^2}(|V|-1+1) \\
&= E[K](\frac{|V|^2-|V|+|V|-1}{|V|^2}) + \frac{|V|-1}{|V|} + \frac{2}{|V|} \\
&= E[K](\frac{|V|^2-1}{|V|^2}) + \frac{|V|+1}{|V|} \\
E[K](1-\frac{|V|^2-1}{|V|^2}) &= \frac{|V|+1}{|V|} \\
E[K](\frac{|V|^2-|V|^2+1}{|V|^2}) &= \frac{|V|+1}{|V|} \\
E[K] &= \frac{|V|^2(|V|+1)}{|V|} \\
E[K] &= |V|^2+|V|
\end{aligned}$$

(ii) Expected number of draws to draw token "work" with 95% probability

One can look at the complementary probability. This means we want to limit the probability that "work" is not drawn at 5%. This can be reformulated in the following equation since we know that the probability of drawing anything else than "work" is $\frac{|V|-1}{|V|}$:

$$\left(\frac{|V|-1}{|V|}\right)^m < 0.05$$

Now solve for m (the number of draws) and we get

$$m > \frac{ln(0.05)}{ln\left(\frac{|V|-1}{|V|}\right)}$$

(c) Mary dislikes when when the same token appears next to each other

(i) Expected number of draws before the same token appears next to each other

Here we take the same approach as with the "work hard" bigram, however, there is no special need to start with a certain word.

$$\text{K} \equiv \#\text{ draws until two of the same tokens appear next to each other}$$

$$E[K] = 1 * \frac{n-1}{n}(E[K]+1) + 1 * \frac{1}{n} * 2 \tag{1}$$

- In the first term 1 stands for the probability to draw anything and $\frac{n-1}{n}$ represents the probability to draw anything else than the first word. $(E[K]+1)$ means that you have to start fresh with drawing and increment the expected number of draws by one.
- In the second term 1 again stands for the probability to draw anything, followed by $\frac{1}{n}$ which stands for the probability to draw the previous word. The trailing factor of 2 means that we used two draws to get there.

Now again, let's start manipulating the above equation.

$$\begin{aligned}
E[K] &= \frac{n-1}{n}E[K] + \frac{n-1}{n} + \frac{2}{n} \\
E[K](1 - \frac{n-1}{n}) &= +\frac{n+1}{n} \\
E[K](\frac{n-n+1}{n}) &= +\frac{n+1}{n} \\
E[K] &= n\frac{n+1}{n} \\
E[K] &= n+1
\end{aligned}$$

(ii) Network for bigram detection

Formally we have

$$\begin{aligned}
h_t^0 &= f(w_0 x_{t-1} + w_1 x_t + w_2 h_{t-1}^0 + b_0) \\
y_t &= g(w_3 h_t^0 + b_1)
\end{aligned}$$

The unknowns are chosen to be:

$$\begin{aligned}
f(x) &= -abs(x) \\
w_0 &= 1 \\
w_1 &= -1 \\
w_2 &= 0 \\
b_0 &= 0 \\
w_3 &= 1 \\
b_1 &= 0 \\
g(x) &= \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}
\end{aligned}$$

Activation function f is chosen as a non-linear function. In my opinion a linear function is not suited to solve this problem.

(iii) Network for detection of number of bigrams so far

Formally we have

$$\begin{aligned}
h_t^0 &= f(w_0 x_{t-1} + w_1 x_t + w_2 h_{t-1}^0 + b_0) \\
h_t^1 &= g(w_3 h_{t-1}^1 + w_4 h_t^0 + b_1) \\
y_t &= h(w_5 h_t^1 + b_2)
\end{aligned}$$

The unknowns are chosen to be:

$$\begin{aligned}
w_0 &= 1 \\
w_1 &= -1 \\
w_2 &= 0 \\
b_0 &= 0 \\
w_3 &= 1 \\
w_4 &= 1 \\
b_1 &= 0 \\
w_5 &= 1 \\
b_2 &= 0 \\
f(x) &= \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x \neq 0 \end{cases} \\
g(x) &= x \\
h(x) &= x
\end{aligned}$$

(iv) Non-uniform unigram distribution

Mary's idea is not worth considering. Deviating from the uniform weights for the words increases the chance of sequentially drawing the same token.

Question 5: Dijkstra's algorithm

(a) Decoding Conditional Random Field (CRF) using Dijkstra's algorithm

Hereby the output scores are over sequences in the output space $\mathcal{Y}$ of fixed length N. In this problem we consider the problem of POS-tagging.

(i) Define the graph and write pseudo-code using a prioritized version of Dijkstra's

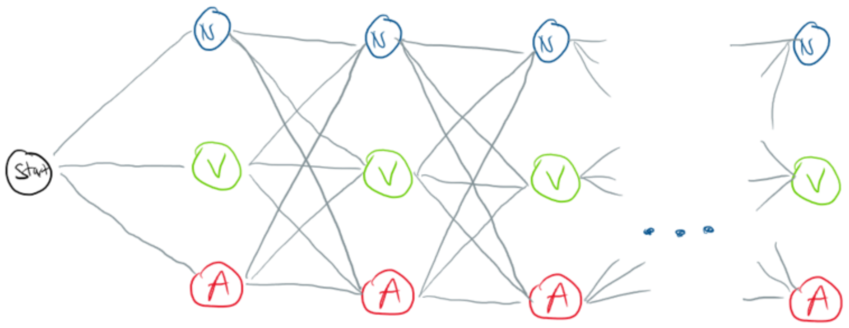


Figure 3: Sketch of the graph of the POS-tagging problem

- Vertices V: POS tags, $\forall V \in Y$
- Edges E: represent pairs of POS tags, $\langle y_{n-1}, y_n \rangle$
- Weights W: These are the scores for the pair of POS tags, score($\langle y_{n-1}, y_n \rangle$, x), where x is the input sentence.

✗ The obstacle of applying standard Dijkstra's to this decoding problem is that trivial Dijkstra's finds the shortest path in a graph. However, we are interested in the longest path, i.e. highest scoring path, in the graph.

✓ The solution is to transform the edge weights and make use of the constant length of the relevant paths through the graph.

Following is a sketch of a proof which shows that the transformation preserves the relative ordering of path scores.

$G(V, E, c)$	where c is the original scoring function
$p_1, p_2 \subseteq E$	all paths are a subset of the edges
$\hat{e}_{max} \in E$	is the highest scoring edge in G
$\lvert p_1 \rvert = \lvert p_2 \rvert$	all relevant paths are equally long. (every word in the input sentence x needs a POS-tag)
$\hat{c}(e) = c(\hat{e}_{max}) - c(e)$	Transformation of edge-scores
$c(p_1) = \sum_{e \in p_1} c(e)$	score of path p_1
$c(p_2) = \sum_{e \in p_2} c(e)$	score of path p_2

- Case 1: $c(p_1) > c(p_2)$

  Now suppose that the paths following the transformation would behave as follows:

$$\begin{aligned}
\hat{c}(p_1) = \sum_{e \in p_1} c(\hat{e}_{max}) - c(e) &< \sum_{e \in p_2} c(\hat{e}_{max}) - c(e) = \hat{c}(p_2) \\
|p_1| c(\hat{e}_{max}) - \sum_{e \in p_1} c(e) &< |p_2| c(\hat{e}_{max}) - \sum_{e \in p_2} c(e) \\
\sum_{e \in p_1} c(e) &> \sum_{e \in p_2} c(e) \\
c(p_1) &> c(p_2)
\end{aligned}$$

  This shows that if p_1 is shorter after the transformation than p_2 then it must hold hold that before the transformation p_1 was longer than p_2.
  This holds for any pair of paths (e.g. p_{max} and arbitrary comparison path).

- The same can be shown for Case 2 ($c(p_1) < c(p_2)$) and Case 3 ($c(p_1) = c(p_2)$).

Now that we showed that after this proposed transformation we can apply Dijkstra's, the pseudo-code is presented:

Algorithm 1: Dijkstra's algorithm with priority queue

```
Result: Find the shortest path between source node and all other nodes in a graph G
create dict for distances: dist;
$dist[\langle Start \rangle] \longleftarrow 0$;
create dict for predecessors: pred;
for all vertices $V \setminus \langle Start \rangle$ in $G$ do
    $dist[V] \longleftarrow \infty$;
    $pred[V] \longleftarrow None$;
end
create priority queue: queue;
queue.add($\langle Start \rangle, dist[\langle Start \rangle]$);
while queue is non-empty do
    $x \longleftarrow queue.return\_min()$;
    for all neighbors n of x do
        if $dist[n] > dist[x] + len(x, n)$ then
            dist[n] = dist[x]+len(x,n);
            pred[n] = x ;
            queue.update(n, dist[n]);
        end
    end
end
return dist, pred
```

(ii) Dijkstra's with a min-priority queue runs in $O((V+E)logV)$, whereas the Viterbi algorithm runs in $O(P^2L) = O(|E|)$, with P being the different POS-tags and L being the length of the sentence. One can not say that any algorithm is strictly better than the other.

(b) Dynamic programming and semi-rings for Dijkstra's

(i) Rewrite the pseudo-code from above in semiring notation

We use the tropical semi-ring: $\langle \mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1} \rangle = \langle \mathbb{R}^{+}_{0,\infty}, min, +, \infty, 0 \rangle$
Pseudo-code in semi-ring notation:

Algorithm 2: Dijkstra's algorithm with priority queue in semi-ring notation

```
Result: Find the shortest path between source node and all other nodes in a graph G
create dict for distances: dist;
$dist[\langle Start \rangle] \longleftarrow \overline{1}$;
create dict for predecessors: pred;
for all vertices $V \setminus \langle Start \rangle$ in G do
    $dist[V] \longleftarrow \overline{0}$;
    $pred[V] \longleftarrow None$;
end
create priority queue: queue;
queue.add($\langle Start \rangle, dist[\langle Start \rangle]$);
while queue is non-empty do
    $x \longleftarrow queue.return\_min()$;
    for all neighbors n of x do
        $dist[n] \longleftarrow dist[n] \oplus (dist[x] \otimes len(x, n))$;
        pred[n] = x ;
        queue.update(n, dist[n]);
    end
end
return dist, pred
```

(ii) $\langle \mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1} \rangle = \langle \mathbb{R}^{-}_{-\infty,0}, max, +, -\infty, 0 \rangle$

(iii) Widest path problem with semiringified Dijkstra's We use the semi-ring: $\langle \mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1} \rangle = \langle \mathbb{R}^{+}_{-\infty,\infty}, max, min, -\infty, \infty \rangle$

Algorithm 3: Dijkstra's algorithm with priority queue in semi-ring notation for widest path problem

```
Result: Find the shortest path between source node and all other nodes in a graph G
create dict for distances: dist;
$dist[\langle Start \rangle] \longleftarrow \overline{1}$;
create dict for predecessors: pred;
for all vertices $V \setminus \langle Start \rangle$ in G do
    $dist[V] \longleftarrow \overline{0}$;
    $pred[V] \longleftarrow None$;
end
$createpriorityqueue : queue$;
$queue.add(\langle Start \rangle, dist[\langle Start \rangle])$;
while queue is non-empty do
    $x \longleftarrow queue.return\_min()$;
    for all neighbors n of x do
        $dist[n] \longleftarrow dist[n] \oplus (dist[x] \otimes len(x, n))$;
        pred[n] = x ;
        queue.update(n, dist[n]);
    end
end
return dist, pred
```

Part II: Question 1

(a) Define the Context Free Grammar (CFG)

$S \longrightarrow NP, VP$	$NP \longrightarrow I/glasses$
$VP \longrightarrow VP, PP$	$V \longrightarrow draw/hit$
$VP \longrightarrow V, NP$	$Det \longrightarrow a/an$
$PP \longrightarrow P, NP$	$N \longrightarrow man/pencil/ball/umbrella$
$NP \longrightarrow Det, N$	$P \longrightarrow with$
$NP \longrightarrow NP, PP$	

(b) Assign probabilities to the production rules (PCFG)

$S \longrightarrow NP, VP$	$\frac{4}{4} = 1$	$NP \longrightarrow I/glasses$	$\frac{4}{14} = \frac{2}{7}/\frac{1}{14}$
$VP \longrightarrow VP, PP$	$\frac{2}{6} = \frac{1}{3}$	$V \longrightarrow draw/hit$	$\frac{2}{4} = \frac{1}{2}/\frac{2}{4} = \frac{1}{2}$
$VP \longrightarrow V, NP$	$\frac{4}{6} = \frac{2}{3}$	$Det \longrightarrow a/an$	$\frac{6}{7}/\frac{1}{7}$
$PP \longrightarrow P, NP$	$\frac{4}{4} = 1$	$N \longrightarrow man/pencil/ball/umbrella$	$\frac{4}{7}/\frac{1}{7}/\frac{1}{7}/\frac{1}{7}$
$NP \longrightarrow Det, N$	$\frac{7}{14} = \frac{1}{2}$	$P \longrightarrow with$	$\frac{4}{4} = 1$
$NP \longrightarrow NP, PP$	$\frac{2}{14} = \frac{1}{7}$		

(c) PCFG updated for subject NP expansions and object NP expansions

Differentiate the noun phrase (NP):

- object NP gets own non-terminal → ONP
- subject NP gets own non-terminal → SNP

This leads to the following PCFG:

$S \longrightarrow SNP, VP$	$\frac{4}{4} = 1$	$SNP \longrightarrow I$	$\frac{4}{4} = 1$
$VP \longrightarrow VP, PP$	$\frac{2}{6} = \frac{1}{3}$	$V \longrightarrow draw/hit$	$\frac{2}{4} = \frac{1}{2}/\frac{2}{4} = \frac{1}{2}$
$VP \longrightarrow V, NP$	$\frac{2}{6} = \frac{1}{3}$	$Det \longrightarrow a/an$	$\frac{6}{7}/\frac{1}{7}$
$PP \longrightarrow P, NP$	$\frac{4}{4} = 1$	$N \longrightarrow man/pencil/ball/umbrella$	$\frac{4}{7}/\frac{1}{7}/\frac{1}{7}/\frac{1}{7}$
$NP \longrightarrow Det, N$	$\frac{7}{8}$	$P \longrightarrow with$	$\frac{4}{4} = 1$
$NP \longrightarrow NP, PP$	$\frac{2}{14} = \frac{1}{7}$	$NP \longrightarrow glasses$	$\frac{1}{8}$
$ONP \longrightarrow NP, PP$	$\frac{2}{2} = 1$		

(d) Parse tree for the sentence “I hit a man with glasses” with lexicalized rules

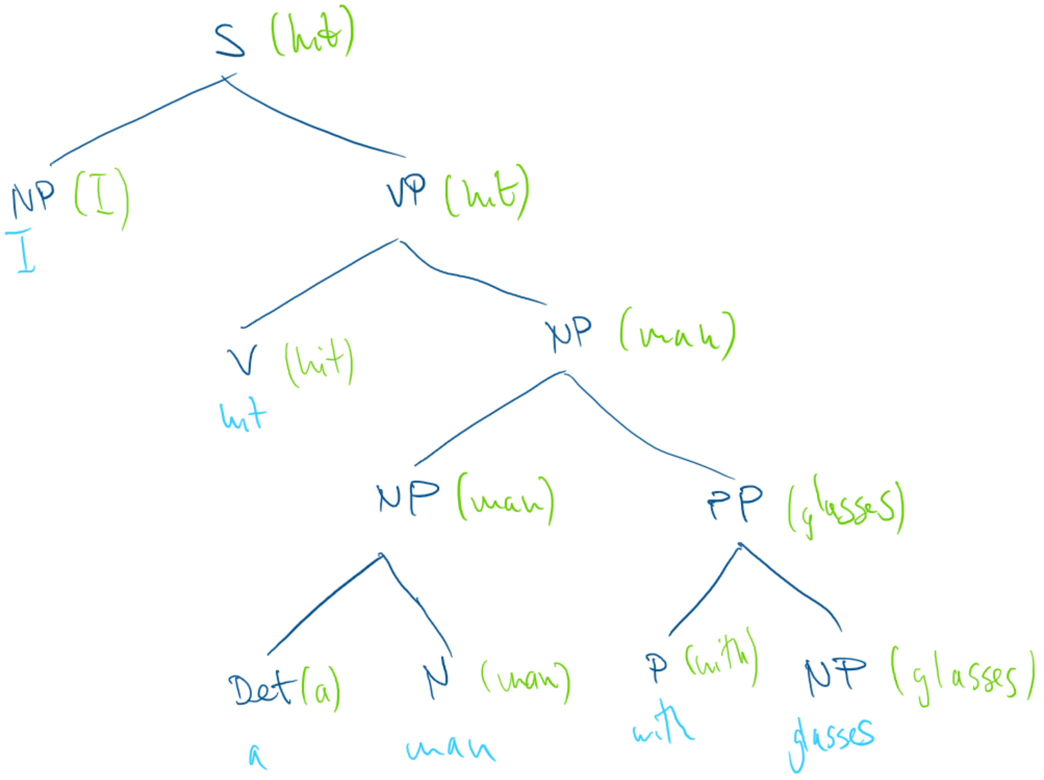


Figure 4: Lexicalized Parse Tree

(e) How do the modifications affect the runtime of the CKY algorithm

Runtime of the standard CKY algorithm is

$$O(n^3 * |G|)$$

where n represents the length of the sentence and $|G|$ stands for the size of the grammar.

⇒ the modifications have an impact on $|G|$ as they introduce new production rules.

⇒ however, these changes are constant factor which have no influence on the Big-O notation.

⇒ Nevertheless, they have an impact on runtime since the space of production rules can potentially blow up.

Space complexity of the CKY algorithm is

$$O(n^2)$$

where n is again the length of the string.

⇒ the modifications have no effect on the Big-O notation.

Part II: Question 2

(a) To make a projective dependency tree out of a lexicalized constituency tree, you have to draw arcs between heads of constituents and their parents.

$\Rightarrow$ convert arc scores into scores for lexicalized production

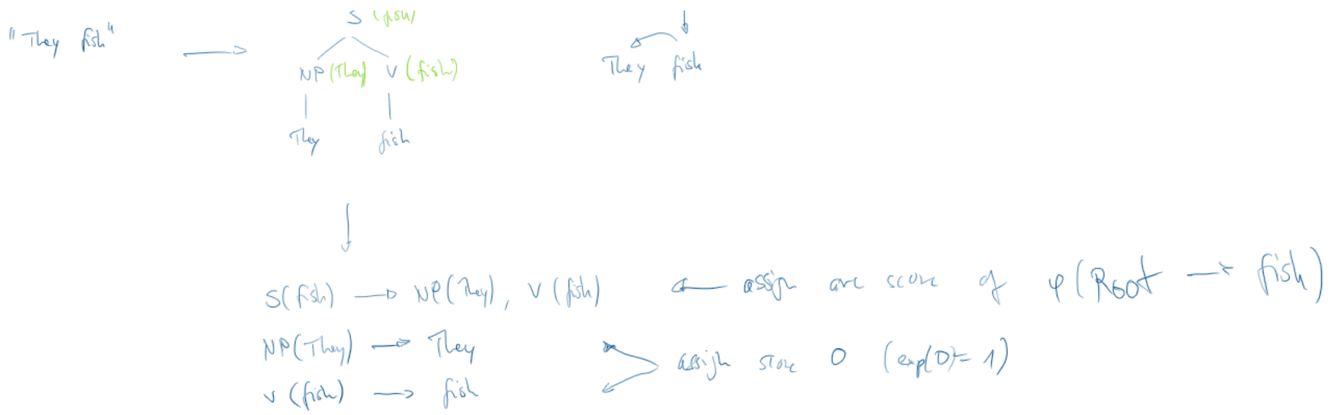
Given the set of unlabelled arc scores

$$\psi(i \to j)_{i,j=1}^{M} \bigcup \psi(Root \to j)_{j=1}^{M}$$

do the following:

- travers input and note all the lexicalized production rules
- identify the root production $S(\sigma) \to b(\beta), c(\sigma)$ and assign score $\psi(Root \to \sigma)$
- for arbitrary production rule $a(\alpha) \to b(\beta), c(\alpha)$ assign arc score $\psi(\alpha \to \beta)$
- for all production rules involving terminal nodes, e.g. $o(\Omega) \to d(\delta)$ assign the score 0.

(b) Verify for the sentence "They fish"



"They fish"
S (fish)
NP (They)
V (fish)
They
fish
They fish
S(fish) → NP(They), V(fish)
NP(They) → They
V(fish) → fish
assign arc score of φ(Root → fish)
assign score 0 (exp(0) = 1)

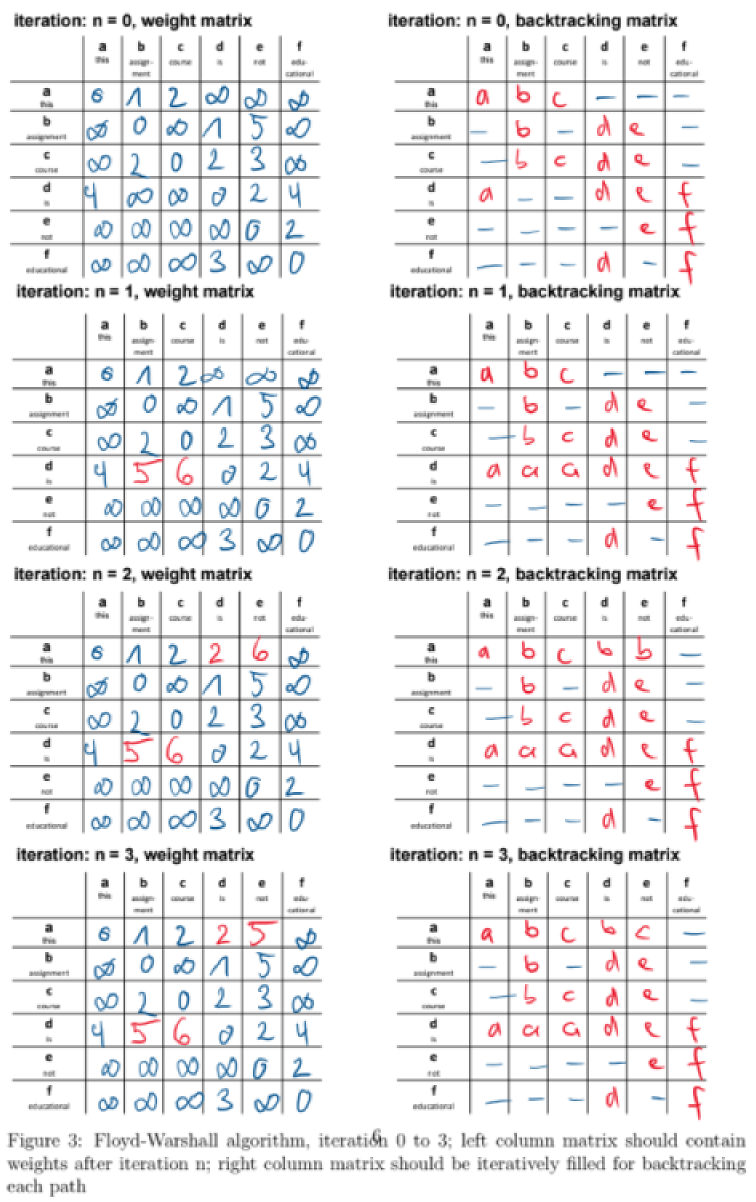
Figure 5: Verification

Part II: Question 4

(a) Acceptance under WSFA

	accepted	weight
1	NO	-
2	NO	-
3	YES	$2+3+2+2+3=12$
4	YES	$1+1+2+2+3=9$
5	NO	-
6	NO	-
7	YES	$4+1+5+2+3=15$
8	YES	$1+5+2=8$
9	YES	$2+2+1+2+2+3=12$
10	YES	$2+2+2+10+2+3=21$
11	Yes	$2+3+4+1=10$
12	YES	$2+1+2+2+3=10$
13	NO	
14	YES	$10+10+2+3=25$
14	YES	$4+2+2+5+2+3=18$
15	YES	$2+1+4+1=8$
16	NO	
17	YES	$2+2+5+2+3=14$

(b) Floyd-Warshall steps



iteration: n = 0, weight matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	0	1	2	∞	∞	∞
b assignment	∞	0	∞	1	5	∞
c course	∞	2	0	2	3	∞
d is	4	∞	∞	0	2	4
e not	∞	∞	∞	∞	0	2
f educational	∞	∞	∞	3	∞	0

iteration: n = 0, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	a	b	c	–	–	–
b assignment	–	b	–	d	e	–
c course	–	b	c	d	e	–
d is	a	–	–	d	e	f
e not	–	–	–	–	e	f
f educational	–	–	–	d	–	f

iteration: n = 1, weight matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	0	1	2	∞	∞	∞
b assignment	∞	0	∞	1	5	∞
c course	∞	2	0	2	3	∞
d is	4	5	6	0	2	4
e not	∞	∞	∞	∞	0	2
f educational	∞	∞	∞	3	∞	0

iteration: n = 1, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	a	b	c	–	–	–
b assignment	–	b	–	d	e	–
c course	–	b	c	d	e	–
d is	a	a	a	d	e	f
e not	–	–	–	–	e	f
f educational	–	–	–	d	–	f

iteration: n = 2, weight matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	0	1	2	2	6	∞
b assignment	∞	0	∞	1	5	∞
c course	∞	2	0	2	3	∞
d is	4	5	6	0	2	4
e not	∞	∞	∞	∞	0	2
f educational	∞	∞	∞	3	∞	0

iteration: n = 2, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	a	b	c	b	b	–
b assignment	–	b	–	d	e	–
c course	–	b	c	d	e	–
d is	a	a	a	d	e	f
e not	–	–	–	–	e	f
f educational	–	–	–	d	–	f

iteration: n = 3, weight matrix

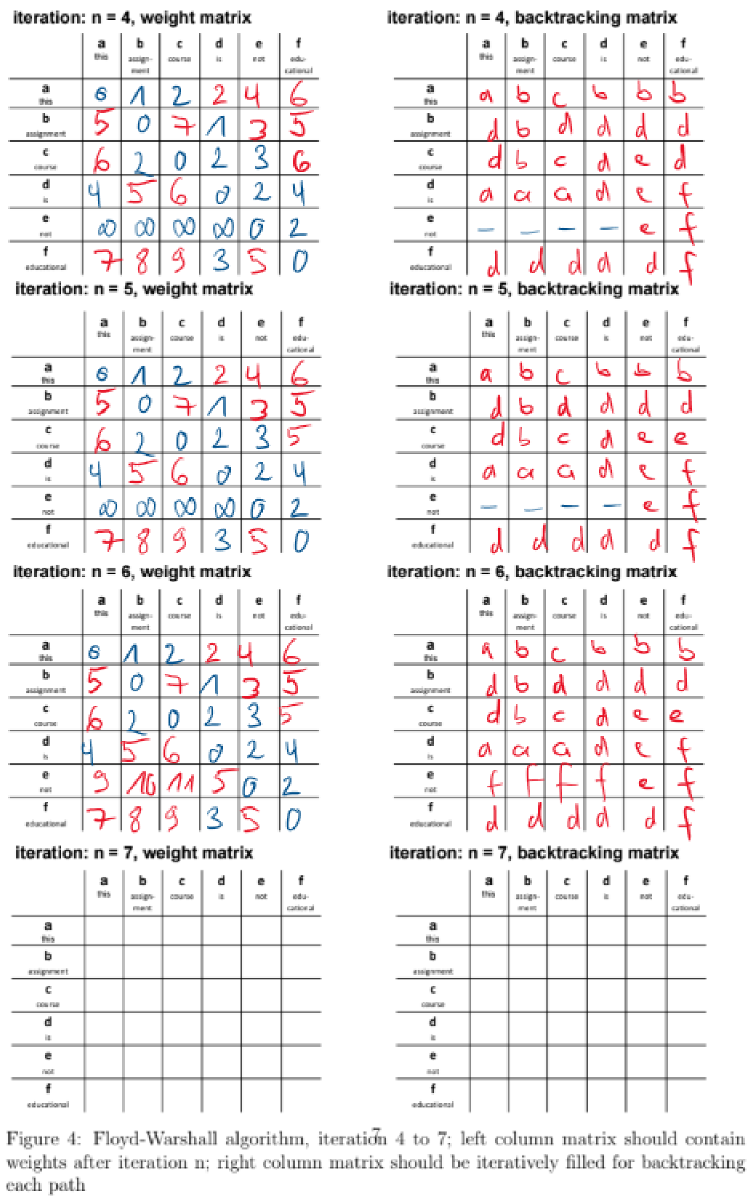
	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	0	1	2	2	5	∞
b assignment	∞	0	∞	1	5	∞
c course	∞	2	0	2	3	∞
d is	4	5	6	0	2	4
e not	∞	∞	∞	∞	0	2
f educational	∞	∞	∞	3	∞	0

iteration: n = 3, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	a	b	c	b	c	–
b assignment	–	b	–	d	e	–
c course	–	b	c	d	e	–
d is	a	a	a	d	e	f
e not	–	–	–	–	e	f
f educational	–	–	–	d	–	f

Figure 3: Floyd-Warshall algorithm, iteration 0 to 3; left column matrix should contain weights after iteration n; right column matrix should be iteratively filled for backtracking each path

Figure 6: Floyd-Warshall algorithm Part 1



iteration: n = 4, weight matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	6	1	2	2	4	6
b assignment	5	0	7	1	3	5
c course	6	2	0	2	3	6
d is	4	5	6	0	2	4
e not	∞	∞	∞	∞	0	2
f educational	7	8	9	3	5	0

iteration: n = 4, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	a	b	c	b	b	b
b assignment	d	b	d	d	d	d
c course	d	b	c	d	e	d
d is	a	a	a	d	e	f
e not	–	–	–	–	e	f
f educational	d	d	d	d	d	f

iteration: n = 5, weight matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	6	1	2	2	4	6
b assignment	5	0	7	1	3	5
c course	6	2	0	2	3	5
d is	4	5	6	0	2	4
e not	∞	∞	∞	∞	0	2
f educational	7	8	9	3	5	0

iteration: n = 5, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	a	b	c	b	b	b
b assignment	d	b	d	d	d	d
c course	d	b	c	d	e	e
d is	a	a	a	d	e	f
e not	–	–	–	–	e	f
f educational	d	d	d	d	d	f

iteration: n = 6, weight matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	6	1	2	2	4	6
b assignment	5	0	7	1	3	5
c course	6	2	0	2	3	5
d is	4	5	6	0	2	4
e not	9	10	11	5	0	2
f educational	7	8	9	3	5	0

iteration: n = 6, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this	a	b	c	b	b	b
b assignment	d	b	d	d	d	d
c course	d	b	c	d	e	e
d is	a	a	a	d	e	f
e not	f	f	f	f	e	f
f educational	d	d	d	d	d	f

iteration: n = 7, weight matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this						
b assignment						
c course						
d is						
e not						
f educational						

iteration: n = 7, backtracking matrix

	a this	b assign-ment	c course	d is	e not	f edu-cational
a this						
b assignment						
c course						
d is						
e not						
f educational						

Figure 4: Floyd-Warshall algorithm, iteration 4 to 7; left column matrix should contain weights after iteration n; right column matrix should be iteratively filled for backtracking each path

Figure 7: Floyd-Warshall algorithm Part 2

(c) The number of iterations is bounded by the number of vertices. In our case this is $n = 6$.

(d) Complexities

- time complexity for Floyd-Warshall with backtracking matrix: $O(n^3)$
- space complexity for Floyd-Warshall with backtracking matrix: $O(n^2)$ (constant factor of 2 can be neglected in Big-O notation)
- max. time complexity for backtracking lowest weight: $O(n)$.