A machine learning application to nourish incomplete datasets. Expanding from a traditional supervised classification task to Natural Language Processing approaches, this paper will examine the use of a traditional machine learning model and some of the more popular deep learning methods, such as word2vec and BERT, to predict the missing data values. The task is based on a Danish language text data.

By Yosuke Ueda

Abstract

This paper presents a use case for machine learning, deep learning, and natural language processing to predict missing values in a dataset. The main objective is to apply these technologies to solve a supervised classification task. The model is applied on job description texts to reinvent the job category, job type, and company names that are missing in the dataset. To complete this task Random Forest model, word2vec, recurrent neural network, and the transformer-based BERT are reviewed and applied. The different models show their capability and perform sufficiently on some tasks, while others need more improvement either in tuning parameters or the manipulations of the dataset. There are implications for deeper exploration in the Danish language area.

Purpose

The performance of predictive modeling is consistently dependent on the amount and quality of the dataset available. Suffering from insufficient or imperfect data will generate low performance from a machine learning (ML) model, especially if the missing data contain useful information for the training. This paper attempts to examine the use of ML and deep learning (DL) for a specific dataset containing different job description posted by various firms seeking qualified employees. The dataset is acquired by a third party every month, and the accumulated data from the first year 2019-

2020 consisted of missing values. The acquired dataset contains more than 330,000 rows and 14-16 columns.
The current problem is that the dataset contains more than 103,744 missing data values for the column company names, and 11,490 rows are missing in both job category and job types, which is a significant information loss that will affect the quality of data in the long term.

AdID	0
UserID	0
CompanyName	103745
Source	135948
CreatedDate	1
Zip	119690
Region	1240
AdURL	205
JobCategory	11490
JobType	11490
ApplicationDate	127270
Ansættelsesform	0
Headline	82
SourceDescription	3
language	0

The main objective is to apply ML and Deep Learning (DL) to predict the following relevant columns – company names, job category, job types based on job description.

To fill in the missing values, three approaches will be applied: traditional ML approach using Random Forest and DL methods with word2vec and BERT. The task is a simple supervised classification method which basically trains a model and predicts the classes of the missing data. It is not the overall aim to compare the performances of the three approaches, but to demonstrate their various abilities for the mentioned use cases and to lay out the progress in the paradigm of the natural language processing (NLP) field.

This paper will use the model score to evaluate model performance for both training and testing, further focusing on making prediction with real unseen data.
However, it will basically be most useful to apply metrics to model assessment, as it will give an overview of the accuracy and precision.

Dataset and preprocessing

The relevant columns to be targeted is job categories, job types, and company names, and will be based on the text column job description. These have been sorted to exclude other languages so to focus solely on the Danish job description. English and Scandinavian jobs are quite relevant for the whole Danish job industry, but when building a language model, it should concentrate on one language. The challenges faced with understanding language models is that, compared to English models, which are heavily researched by many, the Danish language model is still unexplored. (Obviously the Danish language is used predominantly just by Danes). Acknowledging that Danish is a low resource language in NLP implies that the task will involve challenging obstacles.

Following the procedure, the cleaning of the dataset is archived by removing special characters and converting all the words into lowercase.

An example of job description in the job category of “Informationsteknologi”

på vegne af vores kunde søger vi en it supporter. har du erfaring med at løse opgaver for kunder mens du taler me d dem i telefonen så er det nok dig vi leder efter. nogle af de opgaver som du har erfaring med og kan sætte x ved er vores forventninger til dig arbejdstider. du vil for det meste komme til at sidde på kontoret og løse opgaver for kunderne. der er ikke dage som er ens. i spidsbelastninger vil du også tage på kundebesøg for at levere og opsætte pc og netværk ved kunderne. de er tale om et længerevarende vikariat medhenblik på fastansættelse, vi afholder s amtaler løbende og ansætter når den rette kandidat er fundet.

Subsequently, the process of removing stopwords will be done using a list of Danish stopwords. A file containing Danish stopwords can be found at

https://gist.github.com/berteltorp/0cf8a0c7afea7f25ed754f24cfc2467b

Missing fields in Job category columns

The ML model will be trained on the preprocessed text, and it will predict the job category it belongs to. There are 10 different categories, and as ML models only take numeric inputs, these labels/target values will be converted to numeric labels using label encoder.

['Detail, Restauration og Hotel', 'Industri, håndværk og teknik',
'Informationsteknologi', 'Kommunikation, marketing, salg',
'Kontor, handel og service', 'Ledelse og personale',
'Sundhed og forskning', 'Undervisning', 'Økonomi og jura',
'Øvrige stillinger']

From the observation of job category labels below, it is shown that the dataset is imbalanced. This could lead to implications for ML learning as the model is sensitive and may result in bias towards specific classes. ML models perform well under conditions where the dataset is equally distributed among classes.

```
Sundhed og forskning              56322
Industri, håndværk og teknik      32896
Kontor, handel og service         31353
Detail, Restauration og Hotel     30044
Undervisning                      29710
Øvrige stillinger                 29328
Kommunikation, marketing, salg    22888
Ledelse og personale              20031
Økonomi og jura                   17634
Informationsteknologi              6818
Name: JobCategory, dtype: int64
```

An imbalanced dataset is a practical challenge, and alternatively, the dataset can be equally divided among all classes to create an even structure. The disadvantage is that it will operate on a smaller dataset and further drop relevant information if the selection is done randomly. With the considerations in mind, this case will proceed with the imbalanced dataset.

The relevant data will be divided in X and Y. X is the data to be trained on and Y is the labels to be predicted.

X = the cleaned preprocessed job description
Y = the numeric encoded labels (0 to 9)

Countvectorizer and Bag of words

The job description will be converted into numeric feature vectors by using countvectorizer. Applying countvectorizer will create a vocabulary of unique tokens and construct a feature vector from each document containing the count of words occurring in this document. This method is called bag of words.

The vocabulary is a python dictionary where the unique words are mapped to an integer, and each job description will count the frequency of the words if they exist in the vocabulary. It is called a sparse matrix because when words in the vocabulary do not occur in the text, it is simply labeled 0. If the word occurs the count of frequency will appear. The following code turns the text into numeric feature vectors.

```
vectorizer = CountVectorizer(stop_words=danishstopwords, max_df=0.10, max_features=5000)
x = vectorizer.fit_transform(X)
```

When the above code statement is executed, countvectorizer is converting words to lowercase and removing the stopwords. Adjusting the hyperparameters like “max df” and “max features” will yield a different result when training ML models, so trial and error experiments are necessary.

Max df is setting the maximum frequently occurring words. How many documents contain these words, and if it exceeds the stated maximum threshold, it is simply ignored. In this case the 10%

limit is set, which indicates that if words occur in more than 10% of the documents, they will be removed. The argument for this setting is to ignore the repeating sentences like “send os din ansøgning og cv” or “vi glæder os til at se dig”.

Max features represents the total vocabulary and is set to 5000 words. There exists more hyperparameter options to experiment like “min df” and “n-gram”. These will not be applied in this task.

In general, term frequency is said to be inadequate for text representation, as frequently occurring words do not typically contain any valuable information, and hence a more advanced form of term frequency called term “frequency-inverse document frequency” (tfidf) can be applied. The tfidf down-weights the frequently occurring words in the feature vectors. The tfidf was empirically tested in this task and did not yield any better performance, so this paper continues with the traditional countvectorizer.

Applying countvectorizer to the job ad text, the vocabulary can be seen as this dictionary form.

'grundigt': 1585,
'uddannelsesforløb': 4550,
'møde': 2883,
'succes': 4149,
'lagermedarbejder': 2422,
'pakning': 3230,
'omhyggelig': 3052,
'samarbejdsvillig': 3734,
'fysisk': 1472,
'hånden': 1803,
'fuldtid': 1442,
'rolle': 3656,
'salg': 3713,
'øget': 4976,
'kundetilfredshed': 2357,

Next is to split the datasets into training and testing with ratio 80% training and 20% testing.

x_train, x_test, y_train,y_test = train_test_split(x,y, test_size=0.2, random_state=42)

Random forest (RF)

Traditional ML lists models such as logistic regression, linear regression, bayes theorem, k-nearest neighbors, k-means, decision tree, and random forest. All dependent on the purpose and the type of

ML category, like supervised, unsupervised, and reinforcement learning. This paper will apply random forest, which is a powerful algorithm that can be applied to various of ML tasks.

RF uses the concept “wisdom of the crowd”, where it provides aggregated prediction from a group with an outcome that is regarded superior to predictions based on the individual. This is referred to ensemble learning, and RF is an ensemble of Decision Tree algorithm. RF will produce a prediction based on the voting from the best Decision Tree performance.

```
Model = RandomForestClassifier(random_state=0, n_estimators=1000)
Model.fit(x_train,y_train)
```

The RF model will be trained on 1000 trees, and again the process of tuning the hyperparameters is always a great option such as inspecting features as max depth, max features, and min samples leaf.

Results and limitations

The model has completed the training after approximately 2 days, and the score can be displayed for both training and test.

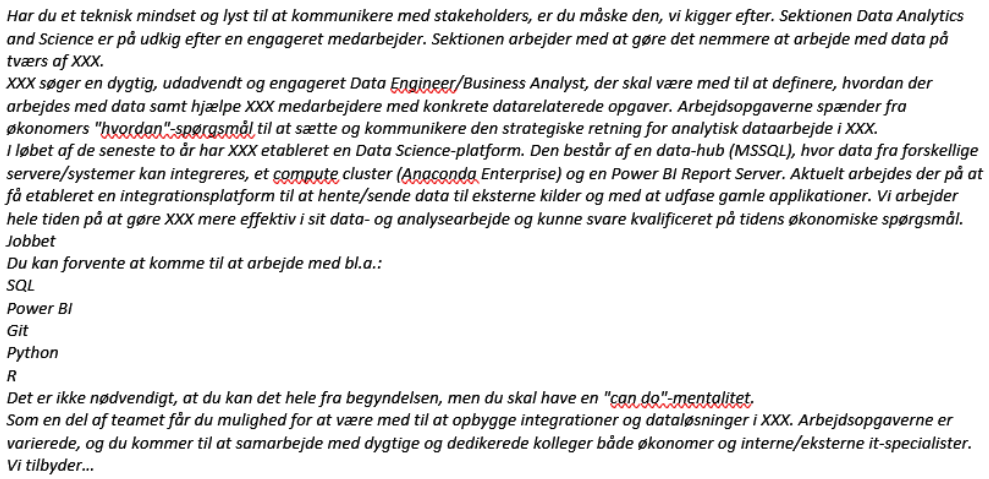
```
print(model.score(x_train,y_train))
predict = model.predict(x_test)
print(accuracy_score(predict,y_test))
```

```
0.9909574540089072
0.8079234726107752
```

The training score of 99% is high/too perfect compared to the testing score of 80%, which could indicate overfitting. In overfitting the model gets too used to the training data without been able to generalize well on test data. The issue in overfitting is a matter of a tradeoff between the bias-variance.

To test the usability of the trained model, it should try predicting with unseen job description. This job description is about software development. The result given is the correct prediction:

('Informationsteknologi', 31.88333333333333)



Har du et teknisk mindset og lyst til at kommunikere med stakeholders, er du måske den, vi kigger efter. Sektionen Data Analytics and Science er på udkig efter en engageret medarbejder. Sektionen arbejder med at gøre det nemmere at arbejde med data på tværs af XXX.
XXX søger en dygtig, udadvendt og engageret Data Engineer/Business Analyst, der skal være med til at definere, hvordan der arbejdes med data samt hjælpe XXX medarbejdere med konkrete datarelaterede opgaver. Arbejdsopgaverne spænder fra økonomers "hvordan"-spørgsmål til at sætte og kommunikere den strategiske retning for analytisk dataarbejde i XXX.
I løbet af de seneste to år har XXX etableret en Data Science-platform. Den består af en data-hub (MSSQL), hvor data fra forskellige servere/systemer kan integreres, et compute cluster (Anaconda Enterprise) og en Power BI Report Server. Aktuelt arbejdes der på at få etableret en integrationsplatform til at hente/sende data til eksterne kilder og med at udfase gamle applikationer. Vi arbejder hele tiden på at gøre XXX mere effektiv i sit data- og analysearbejde og kunne svare kvalificeret på tidens økonomiske spørgsmål.
Jobbet
Du kan forvente at komme til at arbejde med bl.a.:
SQL
Power BI
Git
Python
R
Det er ikke nødvendigt, at du kan det hele fra begyndelsen, men du skal have en "can do"-mentalitet.
Som en del af teamet får du mulighed for at være med til at opbygge integrationer og dataløsninger i XXX. Arbejdsopgaverne er varierede, og du kommer til at samarbejde med dygtige og dedikerede kolleger både økonomer og interne/eksterne it-specialister.
Vi tilbyder…

Nonetheless testing the model with a few new data, it distinctly demonstrates some misclassifications. There are examples indicating where the model classifies “Informationsteknologi” to “øvrige” or “Ledelse og personale”. It could have some issues predicting job description in the software development category with leadership capabilities. In case of excessiveness of mixed information in the description, it is a challenge for the model to perform correct prediction.

Though overfitting can be some part of the explanation, these misclassifications also do demonstrate the significant problem in capturing the semantic meaning from the text.

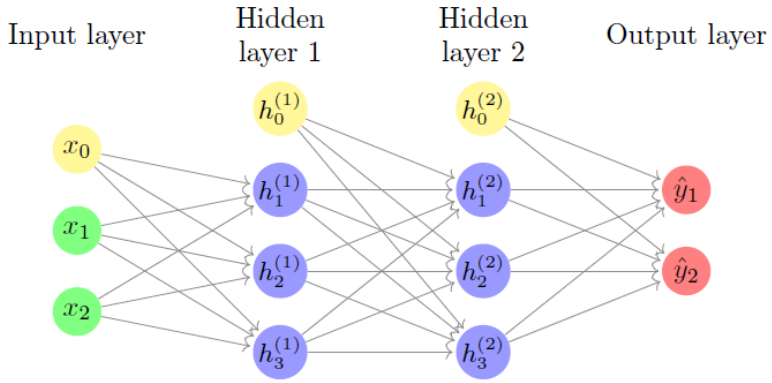
The next section attempts to highlight neural network and NLP approaches to seek more extensive insights into handling language modeling, also with the purpose of exploiting the semantic problem.

Neural network (NN)

Derived from the concept of how the human brain works when attempting to solve a complex problem, an artificial neural network is interconnected layers of neurons where predictions are generated from multiple calculations and pattern recognition. The network consists of input, hidden, and output layers. Every single neuron is associated with a numerical weight, which is basically initiated randomly, and through the forward propagation these weights will be feedforwarded to the next layer and finally at the output layer. The neurons connect to the next layers through an activation function, which is specified from the threshold settings dependent on the activation function selected (Sigmoid, ReLU, Tanh, Softmax).

The aggregated weights are compared with the correct output. The error size indicates the difference between the predicted and the actual value, and this will then be adjusted through backpropagation. The weights will be updated and the corresponding cost function which represents the error is minimized. Mathematically this minimizing error is referred to as gradient descent, and its function endeavors to locate the global minimum with the combination of the learning rate.

The process of NN illustrates an example of a connected network with 2 outputs:



Source: google

The implications for NN in NLP fields

To extract meaningful insights from a sentence requires an understanding of how the sentence is constructed. The order in which the words appear will have an important effect in the structure of the sentence. Unlike a typical supervised ML algorithm that handles data independently of each other, the process of words in NN must be implemented in sequences. The sequential method assembles words in some order, making every sequence unique. The problem with standard NN is that it does not have the structure to process non-independent data, and thus does not have the capability to carry information from the past.

The memory issue in sequence modeling is improved by recurrent neural network (RNN) and long-short-term-memory (LSTM). RNN implements a self-directed loop where it keeps the memory in the data flow so that it retains previous information in the sequence. However, RNN encounters the increasing number of information, and this will lead to a vanishing gradient problem. The problem occurs when gradient value become so small that learning become absent.
LSTM introduce gates to allowing the model to remember and forget, and this will give the vanishing gradient the chance to flow unchanged and avoid diminishing gradient value.

Bidirectional information flows independently in both directions. This will allow information to flow from past to future and from future to past. When both independent directions are computed it then propagates the preserved information in the sequence like RNN and is capable of predict next coming or previous missing words.

*"I tried to call _?_ many times. I finally was able to talk to **Jan** 2 hours later. "*

This sentence exemplifies bidirectional flow as the information in the second sentence contains the missing word in the first sentence – in this case Jan.

The concept of NN will serve as a building block for word2vec and BERT, and the tuning of the hyperparameters will involve parameters like number of layers, the learning rate, number of epochs, batch size, selecting activation functions… etc.

Furthermore, the weights which will be multiplied with neurons and updated through backpropagation are essential for the NN mechanism, as this is the process where learning does occur.

The learning process in NN and ML is based on the previously mentioned optimization algorithm gradient descent. It seeks to minimize the cost functions by locating the direction of gradient. The slope given by the derivatives will find the steepest spot on the curve and eventually locate the global minimum, indicating the point where the cost function is at minimum. The learning rate parameter is crucial to identify this optimal point, as the size of the steps must be utilized so it avoids missing it. The size of the steps cannot be too small or too big.

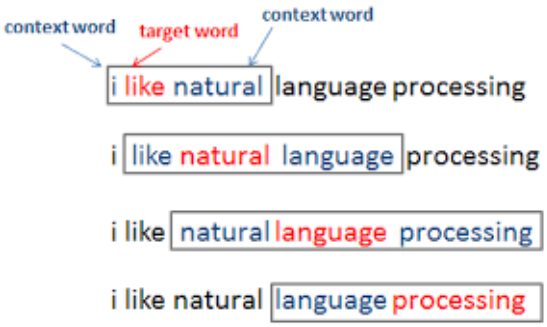
Word embedding with word2vec

The objective of word embedding is to transform words into word vectors that have contextual relationships among words in the vector space, so that similar words have closer distance to each other and vice versa. Word embedding consists of floating-point vectors and is trained in a shallow neural network. The operation is performed in a combination of supervised and unsupervised learning, as input corpus is un-labeled, but simultaneously as the context gets created by data, this will become the actual label.

In contrast to countvectorizer and bag of word method shown earlier, word embedding is more efficient in both capturing the semantic and the computation of the feature dimensionality.

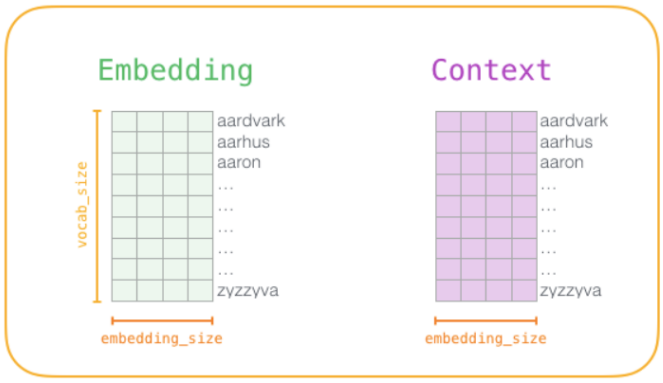
The process is to reconstruct the language based on the words originating from the corpus. The Word2vec model, published in 2013 by Tomas Mikolov at Google, can be trained relatively easily to learn word embedding from text corpus.

The process of constructing word embedding is to select one of the following methods – continuous bag of words (CBOW) or skip-gram. CBOWs functionality is to predict a missing word based on the surrounding words, where skip-gram seeks to predict the neighboring words based on a target word. Advancing from word to word, the window size can be set to determine how many words in the sentence will be processed. Progressing with the selected algorithm, the training data set will be generated with context words and the predictions. Illustration of CBOW with window size 2



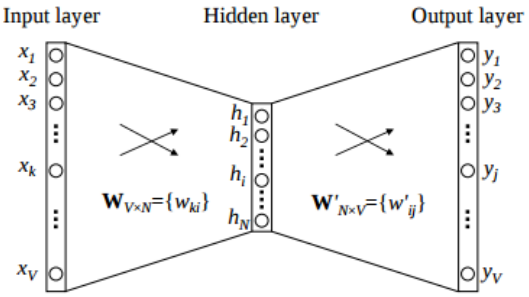
Source: google

The target word will become labels in training data, and learning will be facilitated through the network where backpropagation will be adjusting the weights. For the training process, the input words are transformed into one-hot vectors before being passed into the network, and words are converted into vectors based on some feature representation. Each word in the vocabulary is represented as feature dimensions, and these features are adding characteristics and helps to define each word. To determine the number of these dimensions is a hyperparameter that can be tuned by experimentation.



Source: google

The word similarity is derived by dot product, and the activation function will convert them to probability. The highest score is compared to the actual label and the error margin can then be reduced from the training process.



word2vec model architecture

Source: google

When word2vec training is completed, every word will be mapped to the adjusted word vector holding the distinct quality that will be used again.

Applying word2vec and RNN to predict the missing job types

To extend the classification task from the previous use case, word2vec will create word embedding from the job description and predict the job types, which consist of 133 categories. The job types are more detailed than the previous job categories. The main objective is still to classify the job types where the prediction algorithm is trained from the job description, so the task is to employ the word vector from word2vec and use it as the initial embedding layer in the RNN model.

The word2vec model itself will not be used but only the word vector will be used as weights. The embedding layer is the initial layer in RNN, and the purpose is to learn the weights, so for this case the weights are used and not initialized randomly. The job type labels are shown below.

['Advokatsekretær', 'Afdelingsleder', 'Akademisk medarbejder',
'Apoteker', 'Arkitekt', 'Bager', 'Bibliotekar', 'Biolog',
'Blomsterhandler', 'Bud', 'Butik', 'Butikschef', 'Byggeledelse',
'Bygningsarbejde', 'Bygningsingeniør', 'Bygningskonstruktør',
'Børnepasning', 'Café', 'Data management', 'Direktionssekretær',
'Direktør', 'Diætist', 'Driftsledelse', 'Ejendomsadministrator',
'Ejendomsmægler', 'Ejendomsservice', 'Elektriker',
'Elektroingeniør', 'Ergoterapeut', 'Finans',
'Forretningsudvikling', 'Forsikring og pension', 'Forsker',
'Frisør', 'Fritidsjobs', 'Frivillig', 'Frontendudvikling',
'Fysioterapeut', 'Gartner', 'Geografi og geologi',
'Grafisk design', 'HR og personale', 'HR-ledelse', 'Hardware',
'Hotel', 'IT ledelse', 'IT undervisning', 'IT-sikkerhed',
'IT-support', 'Indkøber', 'Jordemoder', 'Journalistik', 'Jurist',
'Karriererådgiver', 'Kemiingeniør', 'Kemiker', 'Kiropraktor',
'Kok', 'Kommunikation', 'Konsulent', 'Kontor',
'Kreativ medarbejder', 'Kultur', 'Kundeservice', 'Kvalitetschef',
'Kvalitetskoordinator', 'Køkken og kantine', 'Laborant', 'Lager',
'Landbrug', 'Levnedsmiddelteknik', 'Logistik', 'Læge',
'Lægesekretær', 'Lærer', 'Lønspecialist', 'Maler', 'Maritim',

'Marketing', 'Maskiningeniør', 'Maskinteknik', 'Mekaniker',
'Murer', 'Naturarbejde', 'Netværk', 'Personlig træner',
'Produktchef', 'Produktionschef', 'Produktionsteknologi',
'Produktspecialist', 'Projektledelse', 'Psykologi',
'Public Relations', 'Public affairs', 'Pædagog',
'Pædagogmedhjælper', 'Radiograf', 'Receptionist', 'Regnskab',
'Rengøring', 'Restaurationsledelse', 'Revisor', 'SOSU-assistent',
'SOSU-hjælper', 'Salg', 'Salgschef', 'Sekretær', 'Sikkerhed',
'Skoleledelse', 'Skønhed', 'Slagter', 'Smed', 'Socialrådgiver',
'Softwareudvikling', 'Supply Chain Management', 'Sygeplejerske',
'Tandlæge', 'Tandplejer og klinikassistent', 'Teknisk design',
'Teknisk sundhedsarbejde', 'Tjener', 'Tolk', 'Transport', 'Tømrer',
'Uddannelsesvejleder', 'Ufaglært', 'Ungarbejder', 'VVS',
'Veterinær', 'Voksenundervisning', 'Økonom', 'Økonomidirektør',
'Øvrige']

The procedure for job type classification will be accomplished with the following steps:

- Preprocess the input text to word2vec format (tokenized)
- Train the word2vec model with the tokenized data
- Utilize the weights/vector as the initial embedding layer
- Train the RNN and predict the class

Setting parameters for word2vec:

Word2Vec(sentences=wordvector, size=200, window=5, min_count=1, sg=0)

After experimenting with different parameters, the word2vec model is trained on the tokenized text input (referred to wordvector), with a window size of 5, the selected method is CBOW (sg=0), and minimum frequency of 1, and feature dimensions of 200.

Once the training is completed the model can display the most similar word vectors for words like “corona” and “danmark”. The model contains 592,333 words.

```
1 word2vecmodel.wv.most_similar('corona')
```

```
[('coronasituationen', 0.8045446872711182),
 ('covid', 0.7696050405502319),
 ('coronarestriktioner', 0.7334851622581482),
 ('restriktioner', 0.6985650062561035),
 ('coronakrisen', 0.6858429312705994),
 ('covidsituationen', 0.6744838953018188),
 ('coronavirus', 0.6641621589660645),
 ('restriktionerne', 0.6590617895126343),
 ('coronapandemien', 0.6399785876274109),
 ('langtidssygemelding', 0.6300688982009888)]
```

```
1 word2vecmodel.wv.most_similar('danmark')
```

```
[('europa', 0.6579431295394897),
 ('norden', 0.6525466442108154),
 ('skandinavien', 0.6235536336898804),
 ('landet', 0.6086030006408691),
 ('nordeuropa', 0.5927224159240723),
 ('islanddanmark', 0.5468878746032715),
 ('tyrkiet', 0.5362533926963806),
 ('europæiske', 0.5304189324378967),
 ('japan', 0.5292078256607056),
 ('usa', 0.5253511667251587)]
```

As stated previously the model will not be used, but the vectors will be used as initial weights in the network model, hence the vectors will be converted into dictionary form.

```
wording_dict = {}
for a,b in word2vecmodel.wv.vocab.items():
    wording_dict[a] = word2vecmodel.wv.get_vector(a)
```

This python dictionary form can look up a word like “danmark” and see all the 200 vectors in the dimensions predetermined earlier.

```
array([ 3.89480919e-01,  7.66826212e-01, -2.15563250e+00,  3.20349455e+00,
        1.84833515e+00, -5.84987342e-01,  1.64099383e+00, -2.51924467e+00,
        1.17218482e+00,  7.03320801e-01, -2.49689460e-01, -3.82811689e+00,
       -2.86499953e+00, -8.96033823e-01, -1.42322624e+00,  1.30785608e+00,
        1.65277553e+00,  4.47939783e-01,  6.30625963e-01,  2.77941346e+00,
        1.31427801e+00,  3.93645525e+00, -1.05418193e+00, -1.18048227e+00,
        1.93878567e+00, -3.09116989e-01,  2.28545189e+00,  5.22925496e-01,
        2.45018268e+00, -6.89611495e-01, -2.47026706e+00, -2.19085979e+00,
       -1.30784595e+00,  9.59302709e-02, -1.43356466e+00,  9.43020761e-01,
        9.88909662e-01, -1.48519075e+00,  2.76782489e+00, -6.16285145e-01,
        2.72636819e+00, -1.66198862e+00, -2.73071098e+00,  1.19054186e+00,
       -3.39746785e+00,  2.24377012e+00,  1.02908802e+00, -1.40479386e+00,
        1.20341635e+00, -1.94520223e+00,  5.72276637e-02,  7.85961390e-01,
        1.10183501e+00, -9.86267149e-01, -4.11034763e-01, -3.26417351e+00
```

The vectors altered from the model represent the weights in an embedding matrix that will be input to the embedding. To proceed with the classification, the dataset which consists of the job description and the 133 job type labels will have to be preprocessed preliminary to be trained in the RNN model. This will be done with Keras, which is the library built on Tensorflow and is well suited for applying neural network. The steps, in this procedure is as follows:

- Creating an instance of tokenizer with a vocabulary size of – 10000
- Apply tokenizer.fit_on_texts() and tokenizer.texts_to_sequences()
- x = pad_sequences(x, padding='post', maxlen=200)
- y = to_categorical(y)
- x_train, x_test, y_train,y_test = train_test_split(x,y, test_size=0.2, random_state=42)

Similar, to the case of countvectorizer the vocabulary will be set. In this case it will be set to 10,000 words. The entire job ad text data will be tokenized, indexed, and converted to numeric sequences. Each sentence is given a maximum length of 200 words, and padding will be applied post for words shorter than 200 words. The network requires all sentences to be same length. The illustration of numeric representation of tokens with padding will appear as below.

```
array([ 735,  129, 3008,  658,  460,    1, 1171,   14, 1878, 2408,   69,
        345,   61,   19, 7095, 2471, 2262,  535, 5114,  397,  252,    2,
       1153,   24,  565,  383,   37,  416, 1127,  328,  207,   28,  364,
        113,  289, 1124,  261,  367,   79,  449,    7,   20,   13, 2894,
         44,    8, 2591, 1828, 1685,  231,    6,    8,   75,   25,  458,
       5548,   66,   17,  119,  318,    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,
```

The 133 labels will be categorized numerically, and the data will be split into a train and test set with, ratio of 80% and 20%. The data is ready to be fed into the network model. However, the model must first initialize the embedding layer containing the weights.

To implement the embedding layer an empty matrix will be constructed with the required shape (592333,200), and it will be filled with vectors from the word2vec model. The following code constructs the embedding matrix with zeros and subsequently fill it with the vectors from the dictionary created earlier.

```
embedding_dimensions = 200
embeddingmatrix = np.zeros((vocabsize, embedding_dimensions))

for word, index in wordindex.items():
    vector = wording_dict.get(word)
    if vector is not None:
        embeddingmatrix[index] = vector
```

The filled embedding matrix will be applied in the embedding layer when initiating the RNN parameters. A glimpse of the embedding matrix.

```
array([ 1.25813115, -0.66763872, -0.11685624,  3.42604256,  4.83187962,
        2.33321428, -0.95239645,  0.33823931, -0.23795471,  0.50706518,
       -1.34465086,  0.04918016,  0.13173234,  1.05961049,  3.09759569,
       -1.58487678,  0.32252765,  3.46163297,  0.66593349,  0.384877  ,
       -0.20404212,  3.02310848,  2.11522698,  0.71630359, -2.55743241
```

The parameters for the embedding layer will be set as follows: the input dimensions to 592333, embedding dimension to 200, weights equal to the embedding matrix, length to 200, and the weights containing the values of pretrained word vectors which shall not be retrained again (trainable = False).

```
embeddinglayer = Embedding(input_dim=vocabsize,
                           output_dim=embedding_dimensions,
                           weights=[embeddingmatrix],
                           input_length=maxlen,
                           trainable = False)
```

Building the RNN model starts initiates with implementing the defined embedding layer, and subsequently adding layers containing bidirectional, LSTM, dropout, and the activation function softmax. Dropout is a regularization method to prevent overfitting.

```
model = Sequential()
model.add(embeddinglayer)
model.add(layers.Bidirectional(layers.LSTM(200, dropout=0.2, recurrent_dropout=0.2, return_sequences=True)))
model.add(layers.Bidirectional(layers.LSTM(200)))
model.add(layers.Dense(133, activation='softmax'))

opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

In output layer the number of classes to predict is 133 and “softmax” is set to the activation function. The optimizer selected is “Adam” with learning rate 0.001, and the loss is set to categorical cross-entropy since this task is a multiclass classification with 133 labels. The model will be trained on 5 epochs which means the number of passes/iterations through the training set.

```
history = model.fit(x_train, y_train,
                    epochs=5,
                    verbose=1,
                    validation_data=(x_test, y_test),
                    batch_size=128)
```

The result from the model training has indication of overfitting and is not merely impressing.

Training Accuracy: 0.7587
Testing Accuracy: 0.6984

However, testing the usability of the model with unseen data provides a fair result.
Prediction of job description for nurses is given the correct label.

'Sygeplejerske'

```
Afdeling 3 søger en sygeplejerske til et års vikariat i dagvagt på hverdage uden weekendvagter.
Årsagen er, at afdelingens ene sygeplejerske i denne periode skal varetage opgaverne med den
sygeplejefaglige dokumentation på plejecentret. Vi tilbyder dig en målrettet og grundig introduktion,
tilpasset netop dine erfaringer og kompetencer. Vi er et velrenommeret ældrecenter,
kendt for at være visionære og fagligt dygtige. Vi får ros for en åben.
```

Another correct label prediction is a job description of frontend development.

'Frontendudvikling'

```
Vi søger en dygtig frontend-udvikler, som skal stå for at udvikle brugerfladen til vores produkt.
Du vil blive en del af vores udviklingsteam,
som består af dygtige folk med kompetencer inden for bl.a. machine learning-modeller, NLP og databaser.
Til at supplere vores team har vi brug for:
Teknisk frontend kompetencer, så vi kan omsætte vores vision og idéer til en skarp brugerflade (web-app).
En person, som har styr på sine frontend 'frameworks', så vi hurtigt kan komme i gang med at teste vores idéer på brugere.
Kompetencer inden for brugerinddragelse, test og UX-design i praksis.
Hvad forventer vi?
```

Result of word2vec and RNN

The result from the prediction indicates that for predicting the job types, word2vec and RNN model delivers an acceptable performance. However, the relatively known shortcomings of word2vec resides in the inability to manage the word ambiguity. Word2vec is a single vector representation meaning only one numeric vector for one word. Thus, the words are context independent and limit the capabilities, as it does not operate on sub word level. The same meaning for a particular word will be used in different context, which is a major disadvantage.

Another drawback is when a specific word does not exist in the vocabulary word2vec cannot make any interpretation of that word. (“out-of-vocabulary”)

Next section will deal with the transformer approach and the transfer learning with the BERT model. The transformer-based approach attempts to manage these shortcomings.

Transformers and attention

Sequential model like RNN is based on sequential computation, which implies that all elements in the sequence are dependents on the sequential order. This is computational time consuming since, in order to obtain the last word, an entire sequence must be processed.

The sequence model is composed of an encoder, hidden state, and decoder handling a word vector from the word embedding algorithm as input. The nature of the RNN mechanism updates the hidden states based on its input and previous input, and this identifies problems with longer sequences, which will result in information loss and a vanishing gradient.

A technique referred to attention which is highly efficient to improve machine translation system solves this issue by focusing on the most important part of the sequence provided by the similarity score. In contrast to the sequence approach, the attention can process far more data from the encoder to decoder.

The transformer architecture is based on attention and is supporting the parallelization, so sequences can be processed simultaneously. Transformers can be applied to text-summarization, auto-complete, named entity recognition (NER), question-answering, translation, chat-bot, and other NLP tasks such as sentiment analysis, classification, market intelligence… etc.

How transformers work

Transformers inherit the principles of processing stacked encoders and decoders, and each of the decoders contains a self-attention layer and a feed-forward neural network.

Input text is converted to vectors using word embedding with 512 dimensions and the entire sequence get passed simultaneously to the self-attention layer.

Self-attention is basically processing words and position in the same input sentence so to improve the encoding of words by looking at various relationship among the words. It bears a resemblance to the concept of RNNs bidirectional flow
For example:

“The animal didn't cross the street because it was too tired”

The self-attention will relate the word “it” to “animal”. It constructs matrices for query, keys, and values associated with the corresponding word vectors. The matrices are initialized with random weights and the score will be calculated by dot product, and this will be given a probability score by the softmax activation function.

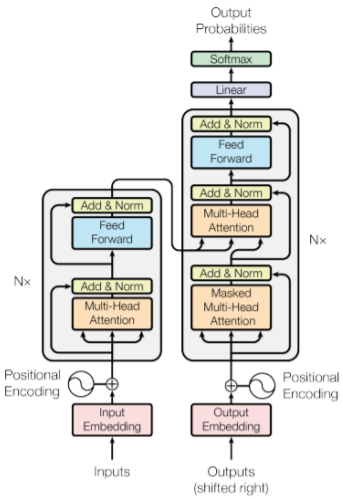


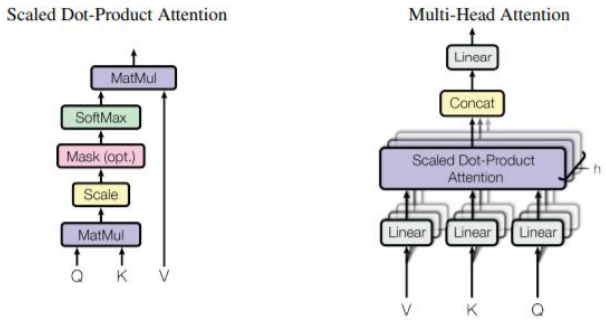
Figure 1: The Transformer - model architecture.

Source: google

Multi-head attention and positional encoding

The multi-head attention is allowing same words to have different representation in the context it is used. Each head applies different linear transformation (subspace) to represent a word, and thus the different heads can learn different relationships between words. Having many heads will perform a final concatenation of the heads resulting from scaled dot product attention and merging of one final vector corresponding to each word.
Given the animal sentence example applying the multi-head attention, one head will focus on “it” and “animal” where another head will focus on “it” and “tired”.

The transformer architecture does not follow the sequential approach like in RNN, and hence the model must alternatively learn about the specific pattern in the sequence. Positional encoding is adding a vector to the embedding, so it can keep track of the distance and position in the sequence.



Source: google

Transfer learning with BERT (Bidirectional encoder representation from transformer)

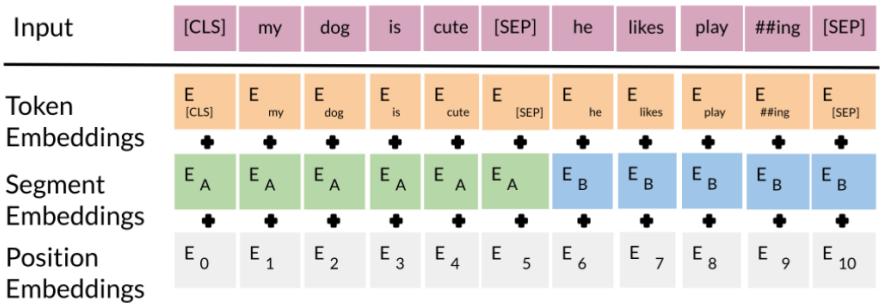
The transformer architecture can process input in parallel and learn the contextual relationships between words in a text. BERT is based on the transformer architecture and uses bidirectional learning simultaneously. It uses only the encoder as the model only needs to learn the language representation that subsequently can be adjusted to a specific task.

The BERT architecture distinguishes the training steps between pre-trained and fine-tuning steps. The pre-training process involves training with unlabeled data, while fine-tuning is applying the pre-trained models and its trained parameter to new label data depending on the downstream tasks. BERT provides various downstream tasks in NLP, like classification, NER, language translation, etc.

Instead of predicting the next word in sentence, BERT architecture applies masked language modeling (MLM) and next sentence prediction (NSP).

In MLM some input tokens are masked (replaced by [MASK]) so the model can predict the correct word behind the masked token based on the context from the remaining word.

NSP attempts to learn the coherent relationships between the sentences by predicting the subsequent sentence based on the first sentence.



Source: google

Inputs are processed in 3 levels, token embeddings, segment embeddings, and position embeddings. Token embeddings insert [CLS] at the beginning of every sentence and [SEP] as sentence separators. Segment embeddings are created to distinguishes between sentence A and B. They provide information about the sentence and the tokens which belongs to them. The position embeddings deliver information about the position of the words in the input sequence. When BERT tokenizes the input sentence the tokens are split into sub words and characters.

The BERT architecture is passing the inputs to the transformer and trains both MLM and NSP with the purpose of minimizing the cost function.

Pretrained BERT can be applied easily by transfer learning, and the implementation can be done relatively simple with a few lines of python code.

Applying BERT for classification of job types

To proceed with the same size of dataset containing the same job types will require a huge amount of time to fine-tune this pretrained BERT model. Considering the resource and time available, this pretrained model will be trained on only 20,000 rows of data.

The pretrained model can be applied directly from the Hugging Face site, which is an open-source provider of natural language processing (NLP) technologies. To build and train the required dataset for BERT, the ktrain library is used. Ktrain library is an outstanding tool for managing TensorFlow and Keras operations, with the intent to debug, train, build, and deploy neural network and ML models.

```
model_name = "distilbert-base-cased"
```

```
transformer_model = text.Transformer(model_name = model_name,
                                     maxlen=512,
                                     class_names=labels)
```

Applying the distilbert-base-cased model will be faster and cheaper to train, as this is a distilled version of BERT, and the max length is set to 512, referring to the maximum tokens in each document. Documents shorter than 512 will be padded, and the number of tokens greater than 512 will be truncated. Max length is also a parameter that can be experimented with dependent on the dataset. Labels are the jobtype labels.

The tools provided by ktrain support the building, training, and evaluation of the model. This process allows the ktrain to preprocess the training data and provides an overview of the pretrained model to be used for finetuning.

```
training = transformer_model.preprocess_train(x_train, y_train)
testing = transformer_model.preprocess_test(x_test, y_test)
```

```
Is Multi-Label? False
preprocessing test...
language: da
test sequence lengths:
        mean : 416
        95percentile : 738
        99percentile : 986
```

When inspection of the training and model has been completed, it can be observed that the type of classification is not “multi-label” and the language is Danish. Advancing further with the correct setting a classifier can be defined to the pretrained model and the finetuning can be initiated.

```
1 bertmodel = transformer_model.get_classifier()
```

```
1 learning = ktrain.get_learner(bertmodel, train_data = training, val_data = testing, batch_size=6)
```

The model will be wrapped in Ktrain’s object Learner, which allows an effortless accessibility to manage train, inspect, and predict with the model. The Learner abstraction from Ktrain allows for an exceptional function to find the optimal learning rate. The “lr_find” function will generate an excellent learning rate estimation. However, for the sake of time saving this is omitted, and a standard learning rate 2e-5 is applied.
The finetuning is launched with 2 epochs.

```
1 learning.fit_onecycle(lr = 2e-5, epochs =2)
```

```
begin training using onecycle policy with max lr of 2e-05...
Epoch 1/2
2667/2667 [==============================] - 98654s 37s/step - loss: 3.7473 - accuracy: 0.2181 - val_loss: 2.0083 - val_accurac
y: 0.5228
Epoch 2/2
2667/2667 [==============================] - 62418s 23s/step - loss: 1.8151 - accuracy: 0.5685 - val_loss: 1.5226 - val_accurac
y: 0.6208
```

The result of 57% and 62% is not impressive. This is most likely caused by the reduction in data to 20,000, and the labels have not been divided equally which could be indicating strong bias.

Furthermore, the tuning parameter such as increasing the number of epochs could also have great impact on the performance. If the same model was trained on a full dataset of 330,000, the performance would probably have been preferable.

The result of prediction has no sign of improvement, as several job description within "Softwareudvikling" are classified as "Regnskab", "Økonom", and "Finans". These descriptions do contain work function within "Regnskab" or "Finans", but the overall job type belongs to "Softwareudvikling".

Overall, the finetuning needs greater improvement and training on a full dataset instead of a subset will yield a significantly better result.

Applying BERT to identify company names with Named Entity Recognition (NER)

NER is another useful application that BERT can provide from the pretrained model. Given a specific document, NER can identify entities such as person, location, and organization and hereby detect important information for the purpose.

To retrieve the 103,744 rows of missing company names NER will attempt to detect if a company name is listed in the text. The Hugging Face site lists pretrained BERT models supporting 104 languages, and these can be applied for various tasks such as NER.
To this mission a pretrained BERT with NER in Danish linguistic can be retrieved from either https://huggingface.co/Maltehb/danish-bert-botxo-ner-dane or https://github.com/ebanalyse/NERDA
The first link in Hugging Face delivers a pretrained NER model that has been finetuned on Danish dataset which has been provided by Alexandra institute. The second option is to finetune a pretrained NERDA model created by Ekstra Bladet. NERDA is dispatched as a python package, and it is straightforward to be finetuned in different languages. Alternatively, on equal stance, there is also Ælæctra, which is a major tool within NLP area in Danish linguistic. This task will examine both the already finetuned NER taken directly from Hugging Face and NERDA.

NER from Hugging Face can be directly applied following few lines of code:

```
1 from transformers import AutoTokenizer, AutoModelForTokenClassification
2
3 tokenizer = AutoTokenizer.from_pretrained("Maltehb/danish-bert-botxo-ner-dane")
4 model = AutoModelForTokenClassification.from_pretrained("Maltehb/danish-bert-botxo-ner-dane")
```

```
Downloading: 100% 1.16k/1.16k [00:03<00:00, 345B/s]

Downloading: 100% 72.0/72.0 [00:00<00:00, 497B/s]
```

When the model is downloaded it can be implemented in a pipeline together with the tokenizer, so the instantiated object will process the necessary stages to predict new input text and return the result.

```
from transformers import pipeline
```

```
nlp = pipeline("ner", model=model, tokenizer=tokenizer)
```

A self-manufactured sentence processed in the pipeline will give following result.

Peter Hansen er i dag ansat i LEO Pharma som data science.
Før var han beskæftiget hos i Carlsberg som softwareudvikler.
Efterfølgende blev han tiltrukket af Novo Nordisk,
og fik stillingen som systemadminstrator før han skiftede stilling igen

```
{'word': 'peter', 'score': 0.9977080225944519, 'entity': 'B-PER', 'index': 1, 'start': 0, 'end': 5}
{'word': 'hansen', 'score': 0.9955257773399353, 'entity': 'I-PER', 'index': 2, 'start': 6, 'end': 12}
{'word': 'leo', 'score': 0.9952142238616943, 'entity': 'B-ORG', 'index': 8, 'start': 30, 'end': 33}
{'word': 'ph', 'score': 0.9794678688049316, 'entity': 'I-ORG', 'index': 9, 'start': 34, 'end': 36}
{'word': '##arm', 'score': 0.9828788042068481, 'entity': 'I-ORG', 'index': 10, 'start': 36, 'end': 39}
{'word': '##a', 'score': 0.9841528534889221, 'entity': 'I-ORG', 'index': 11, 'start': 39, 'end': 40}
{'word': 'carlsberg', 'score': 0.976718544960022, 'entity': 'B-ORG', 'index': 22, 'start': 90, 'end': 99}
{'word': 'novo', 'score': 0.9925857186317444, 'entity': 'B-ORG', 'index': 33, 'start': 159, 'end': 163}
{'word': 'nordisk', 'score': 0.9722078442573547, 'entity': 'I-ORG', 'index': 34, 'start': 164, 'end': 171}
```

As the example illustrates, the NER predicts Peter Hansen as a person and LEO Pharma, Carlsberg, and Novo Nordisk as companies (ORG).

The second approach is finetuning the NERDA with an included Danish dataset from the package. The time used for finetuning added up to 2 full days, and the usability compared to the finetuned NER with respect to identifying company names seems not to differ substantial. Thus, NERDA gives a flexibility to finetune specific datasets, while the condition is to format your own dataset in entity annotation.

NERDA identifies Telenor from the job description.

['Motiveres', 'du', 'af', 'at', 'være', 'den', 'finansielle', 'sparringspartner', 'for', 'Telenors', 'Technology', 'division', 'Vil', 'du', 'være', 'med', 'til', 'at', 'påvirke', 'væsentlige', 'beslutninger', ',', 'komme', 'med', 'de', 'rette', 'anbefalinger', 'og', 'udvikle', 'Technology', 'organisationen', 'Kommer', 'du', 'med', 'ledelseserfaring', ',', 'som', 'Business', 'Partner', 'el.lign', '.'],

['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-ORG', 'I-ORG', 'I-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

According to the observation both models are performing correct predictions of company names but with some misprediction as well. Both models, depending on the context, do identify Bilka as either a location or person, though it is a name of shopping place. In a different context Bilka is correctly identified as an organization.

Implications for further practice

Applying the three approaches in different classification tasks did not highlight any comparison of the models, but simply emphasized the efficiency and usefulness. Predicting missing values might not be the most complex tasks for ML, DL, and NLP but it gave implications for further exploitation with models like BERT and how this can help finetune a tailored model in Danish. Considering it is a low-resource language, the existing pretrained models for the Danish vocabulary, like NERDA and NER, work astonishingly well thanks to the people whose efforts have expanded the usability of the models to the Danish language.

Conclusion

This paper attempted to apply ML, DL, and NLP on specific datasets containing various of job descriptions on the Danish job market, with the purpose of reinventing the missing values in job category, job types and company names. Applying the Random Forest model to predict job category resulted in overfitting, and further the imbalanced dataset could be the reasons for some of the misclassifications. There is plenty of room for improvement, also with regards to hyperparameter experimentation. Word2vec and RNN did demonstrate correct predictions for job types, and considering the affordable scope of this task, the model seemed adequate. However, when facing more challenging tasks that require more comprehensive sentence construction with a focus on contextual semantics and the issue related to word ambiguity, raises questions concerning the capability of word2ved and RNN. The transformer-based BERT was applied to both classification of job types and detecting company name with NER. The classification task provided by BERT was not an improvement, which most likely was due to the massive reduction in dataset, and the lack of experimenting with hyper parameter tuning such as number of epochs. NER delivered an acceptable performance in identifying company names from job description, but clearly demonstrates complications in the contextual sense.

References:

https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

https://jalammar.github.io/illustrated-transformer/

https://jalammar.github.io/illustrated-word2vec/

https://towardsdatascience.com/text-classification-with-hugging-face-transformers-in-tensorflow-2-without-tears-ee50e4f3e7ed

https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/

https://israelg99.github.io/2017-03-23-Word2Vec-Explained/

https://towardsdatascience.com/easy-fine-tuning-of-transformers-for-named-entity-recognition-d72f2b5340e3

https://www.youtube.com/watch?v=KRNsGjfSdIU

https://github.com/Lukecn1/Danish_summarizarion

https://medium.com/@zafaralibagh6/simple-tutorial-on-word-embedding-and-word2vec-43d477624b6d

https://gist.github.com/berteltorp/0cf8a0c7afea7f25ed754f24cfc2467b