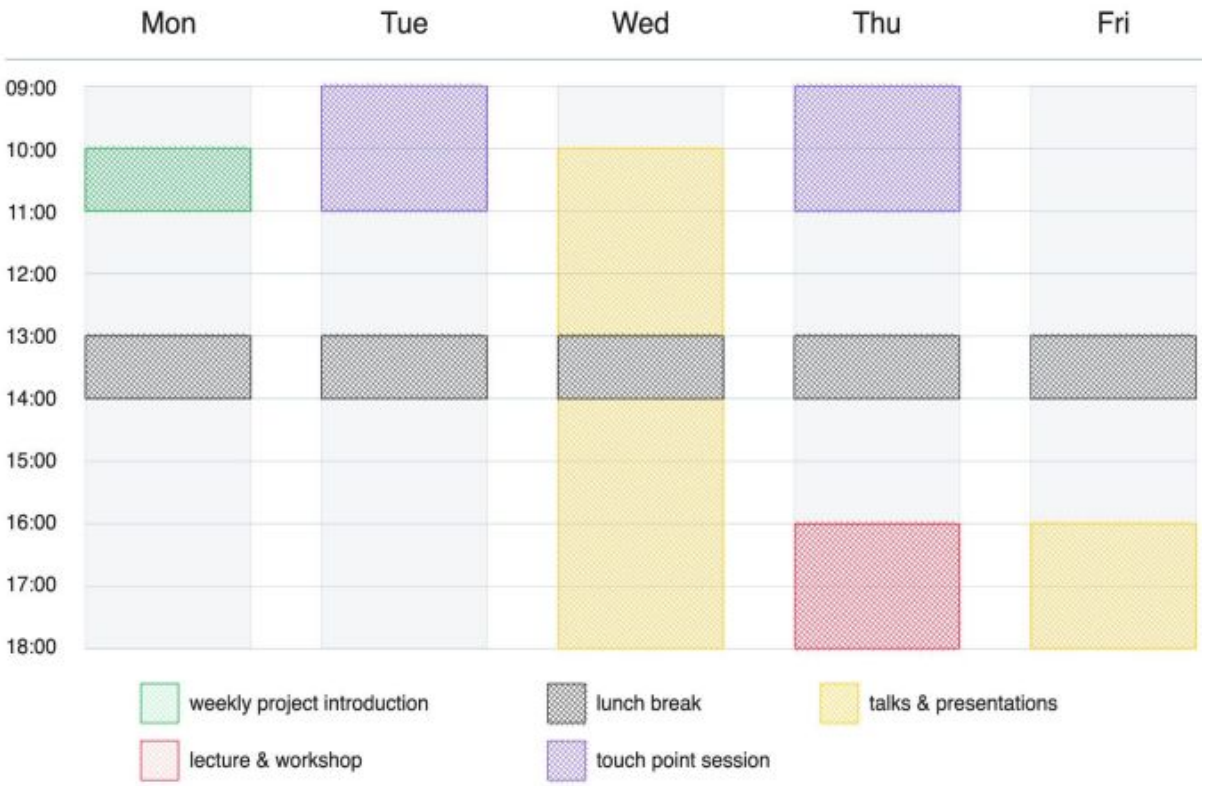
Becoming a Machine Learning Engineer

Week 1: Natural Language Processing Systems

Objectives

- Understand the main concepts of Natural Language Processing Systems
- Familiarise with the most prominent architecture and techniques
- Deploy an end to end language detection model

Week Structure

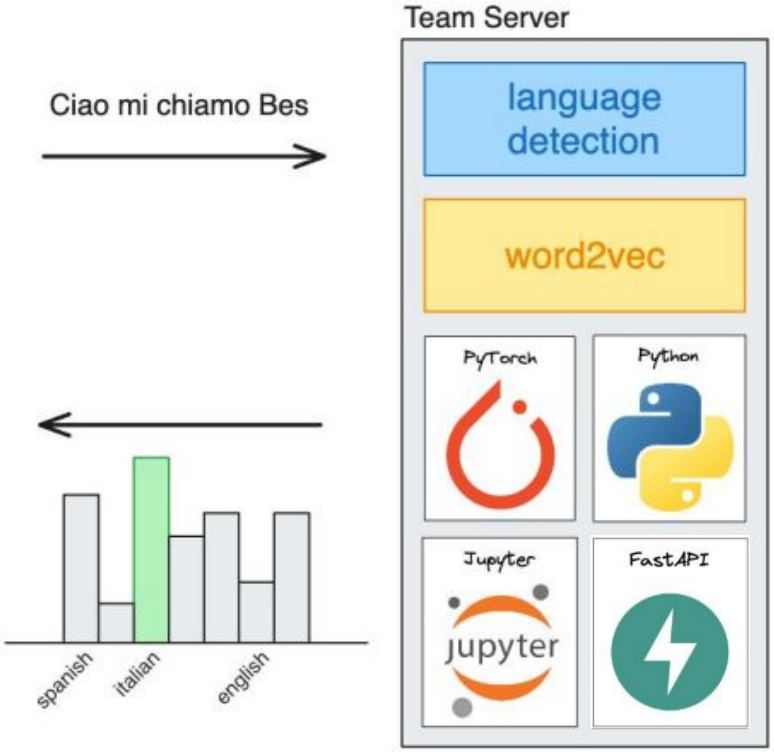


FOUNDERS AND CODERS

First challenge

Machine Learning Institute

- German
- Esperanto
- French
- Italian
- Spanish
- Turkish
- English



- Introduces word2vec to produce word embeddings
- Demonstrate how these vectors capture syntactic and semantic word relationships efficiently with less computational resources.

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

Goals of the paper

- Develop new model architectures that preserve the linear regularities among words.

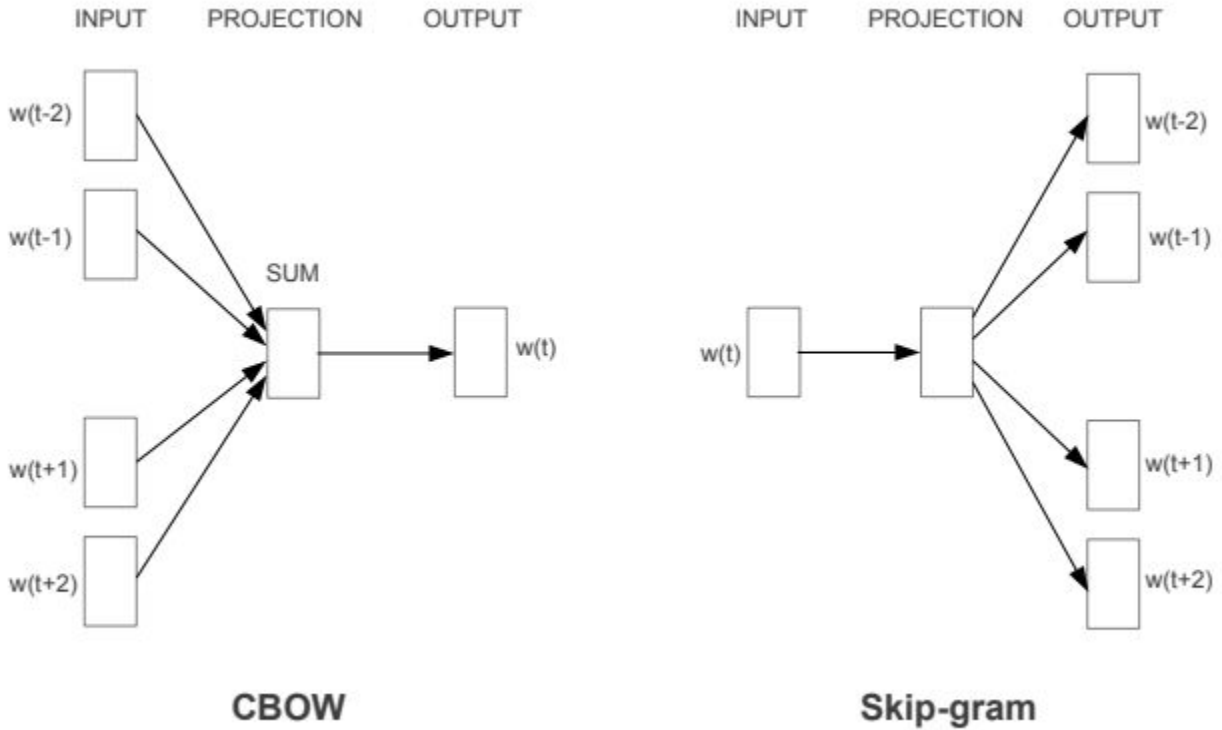
  vector(”King”) - vector(”Man”) + vector(”Woman”) = vector(“Queen”)

- Design a new comprehensive test set for measuring both syntactic and semantic regularities

Previous work

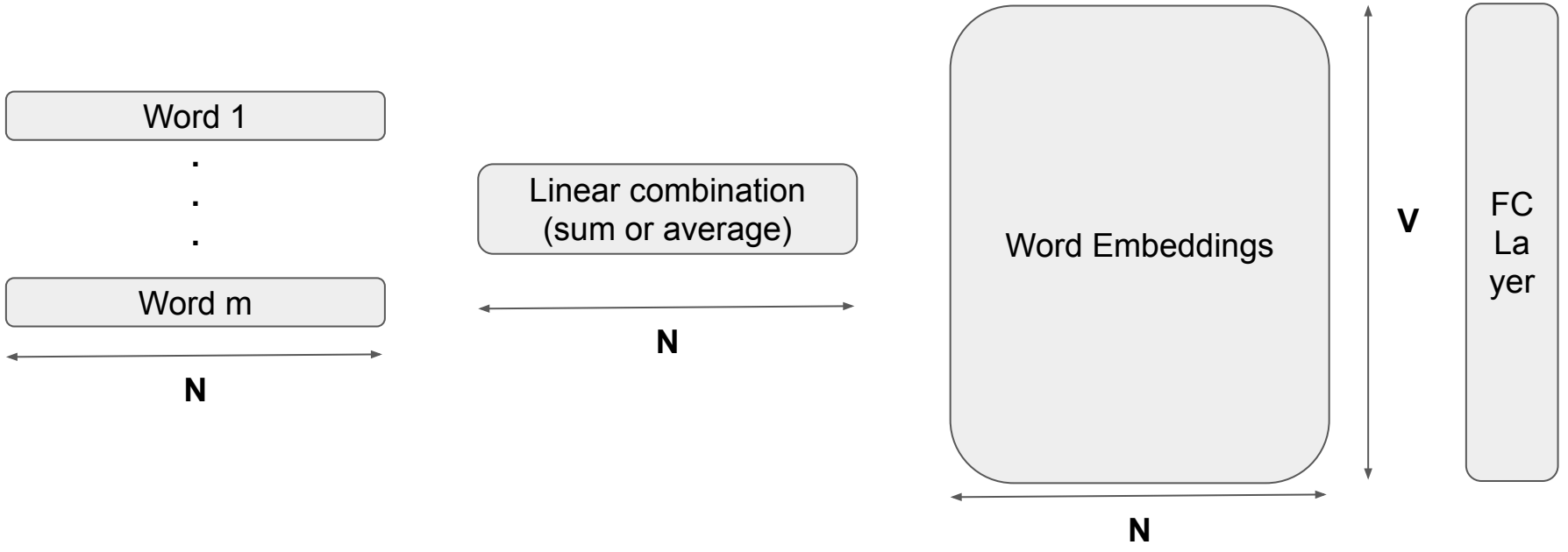
- Complexity is caused by the non-linear hidden layer in the model.
- Explore simpler models that might not be able to represent the data as precisely as neural networks, but can possibly be trained on much more data efficiently.
- NNLM can be successfully trained in two steps:
  - Continuous word vectors are learned using simple model
  - N-gram NNLM is trained on top of these distributed representations of words.

CBOW and Skip-gram



CBOW

- Uses a simple linear combination of word embeddings without applying a non-linear activation function in its hidden layer,

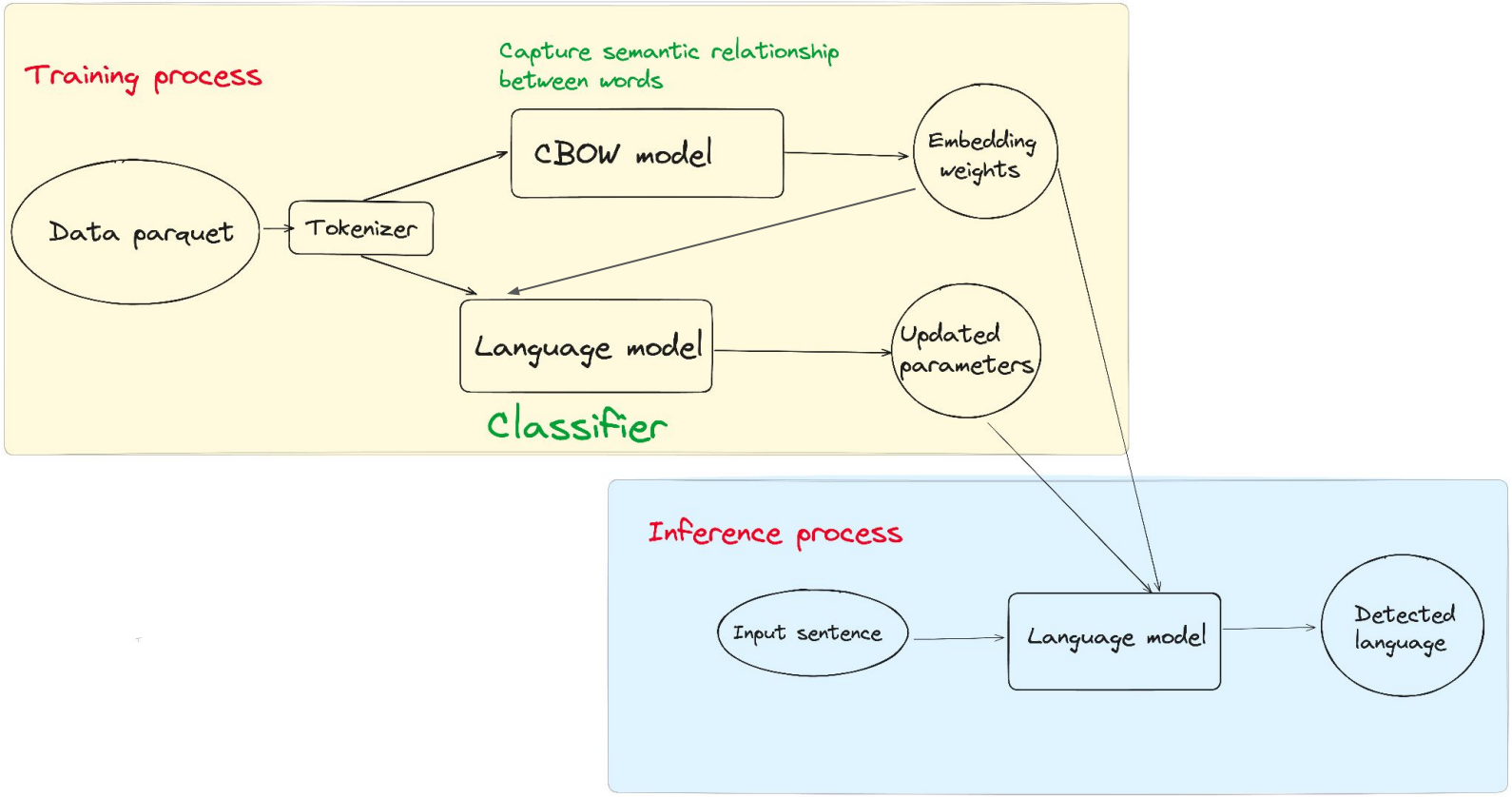


Results

Table 2: *Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k words are used.*

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

What language-detection look like:



Users > basha > Downloads > theera_processed > theera > train_cbow.py > ...

train_cbow.py

```python
import torch
import dataset
import pandas
import model
import tqdm


data = pandas.read_parquet('./Flores7Lang.parquet')
long_format = data.melt(value_vars=data.columns)
corpus = long_format['value'].tolist()
ds = dataset.W2VData(corpus, 3)
dl = torch.utils.data.DataLoader(ds, batch_size=4, shuffle=True)


cbow = model.CBOW(len(ds.tokenizer.vocab), 50)
loss_function = torch.nn.NLLLoss()
optimizer = torch.optim.SGD(cbow.parameters(), lr=0.001)


for epoch in range(3):
    total_loss = 0
    for context, target in tqdm.tqdm(dl, desc=f"Epoch {epoch+1}/10", unit="batch"):
        optimizer.zero_grad()
        log_probs = cbow(context)
        loss = loss_function(log_probs, target)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1}/10, Loss: {total_loss}")
    torch.save(cbow.state_dict(), f"./cbow_epoch_{epoch+1}.pt")
```

Create the dataset (Tokenize)

Define the Model

Train the model -> Embeddings

train_lang.py

```
import torc
import data
import pand
import model


data = pandas.read_parquet('./Flores7Lang.parquet')
ds = dataset.LangData(data)
dl = torch.utils.data.DataLoader(ds, batch_size=1, shuffle=True)


vocab_size = len(ds.tknz.vocab)
loaded_embeddings = torch.load('cbow_epoch_3.pt')
loaded_embeddings = loaded_embeddings['embeddings.weight']
lang = model.Language(loaded_embeddings, 7)
loss_function = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(lang.parameters(), lr=0.003)
torch.save(lang.state_dict(), f"./lang_epoch_0.pt")


for epoch in range(20):
  for sentence, target, _ in dl:
    optimizer.zero_grad()
    log_probs = lang(sentence)
    loss = loss_function(log_probs, target)
    loss.backward()
    optimizer.step()
    print(f"Epoch {epoch+1}/5, Loss: {loss.item()}")
  torch.save(lang.state_dict(), f"./lang_epoch_{epoch+1}.pt")
```

Load the learned Embeddings

Define Lang Model

Train Lang Model -> Model parameters

```
    vocab = list(                                                    reshold})
    return vocab

def save_vocab(
    with open(pat
        for word in self.vocab: f.write(word + '\n')
    return self

def load_vocab(self, path):
    with open(path, 'r') as f: self.vocab = [line.strip() for line in f.readlines()]
    self.word2idx = {word: idx for idx, word in enumerate(self.vocab)}
    self.idx2word = {idx: word for idx, word in enumerate(self.vocab)}
    return self

def encode(self, sentence):
    words = sentence.split()
    words = [self.clean_word(word) for word in words if word]  # Clean and filter out empty words
    return [self.word2idx[word] for word in words if word in self.word2idx]

def decode(self, indices):
    return ' '.join(self.idx2word[idx] for idx in indices if idx in self.idx2word)

@staticmethod
def clean_word(word):
    word = word.lower()
    word = ''.join(char for char in word if char not in string.punctuation)
    word = ''.join(char for char in word if not char.isdigit())
    return word.strip()


if __name__ == '__main__':
    import pandas as pd
    data = pd.read_parquet('./Flores7Lang.parquet')
    print(data.columns)
    long_format = data.melt(value_vars=data.columns)
    corpus = long_format['value'].tolist()
    tknz = Tokenizer(corpus)
    tknz.save_vocab('./vocab.txt')
    tknz.load_vocab('./vocab.txt')
    print(len(tknz.vocab))
    print(tknz.vocab[90:100])
```

tokenizer.py

Tokenize the dataset -> Vocab List

server.py

```python
@app.on_event("startup")
async def startup_event():
    app.state.maybe_model = "Attach my model to some variables"
    app.state.tknz = (tokenizer.Tokenizer()).load_vocab("./vocab.txt")
    loaded_embeddings = torch.load('cbow_epoch.pt')
    loaded_embeddings = loaded_embeddings['embeddings.weight']
    app.state.lang = model.Language(loaded_embeddings, 7)
    app.state.lang.load_state_dict(torch.load("./lang_epoch.pt"))
    app.state.langs = ["German", "Esperanto", "French", "Italian", "Spanish", "Turkish", "English"]
    app.state.lang.eval()
```

Import Vocab List

Import learned Embeddings

Import learned Lang Model Param

If server is in use -> Ask Stathis

uvicorn server:app --host 0.0.0.0 --port 8080 --reload

Theera

Bk

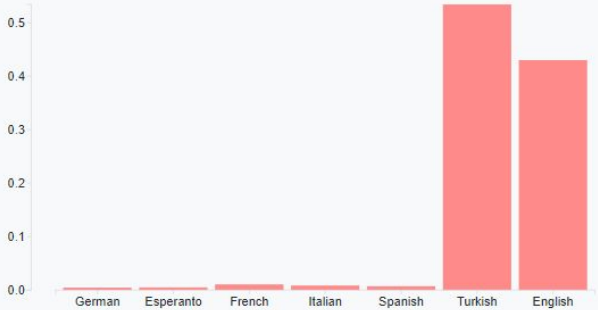
http://recurrent.mlx.institute:8080

what_language_is_this

```
1  Which nationality would you give Bes?
```

Exec Request

Recurrent Rebels



CBOW

- CBOW Model: In the CBOW model of Word2Vec, there is a non-linear hidden layer that maps the input context words (represented as word embeddings) to a hidden layer. However, unlike the Feedforward Neural Network Language Model (NNLM), there is no non-linear activation function applied in this hidden layer. Instead, it simply sums or averages the context word embeddings linearly to produce the hidden layer representation.
- Feedforward NNLM: In the Feedforward Neural Network Language Model (NNLM), there is a non-linear hidden layer that applies a non-linear activation function (typically a hyperbolic tangent or sigmoid function) to the linear combination of the input context word embeddings. This non-linear layer introduces non-linearity into the model.

The key difference is that CBOW uses a simple linear combination of word embeddings without applying a non-linear activation function in its hidden layer, whereas the Feedforward NNLM does use a non-linear activation function.

Previous work

- The main observation from the previous section was that most of the complexity is caused by the non-linear hidden layer in the model. While this is what makes neural networks so attractive, we decided to explore simpler models that might not be able to represent the data as precisely as neural networks, but can possibly be trained on much more data efficiently.
- where it was found that neural network language model can be successfully trained in two steps: first, continuous word vectors are learned using simple model, and then the N-gram NNLM is trained on top of these distributed representations of words.