A Context-Aware Deep Learning Model for Turkish Word Sense Disambiguation

Ege Yenen Huseyin Bora Baran

March 2025

Abstract

Word Sense Disambiguation (WSD) is the automatic choice of an appropriate meaning (sense) of a word in context. The purpose of this study is to enhance Turkish WSD research by introducing an approach based on deep learning, in addition to offering a beneficial annotated corpus, thereby facilitating future advancement in processing Turkish semantics.

Keywords: Natural Language Processing, Word Sense Disambiguation, Lexical Sample, BERT, Deep Learning, Context Awareness, Turkish NLP, Annotated Dataset.

1 Introduction

Our primary objectives are:

1. To construct a new lexical sample dataset for 10 common polysemous Turkish words.
2. To develop and implement a modern, context-aware WSD model utilizing BERT [2].
3. So as to get the proposed model trained and tested on the created dataset.
4. To contribute to the existing resources and methodologies for Turkish WSD.

2 Dataset Creation and Preprocessing

2.1 Target Word Selection

Ten common Turkish words were selected among all words. The selected target words, together with their primary senses, are listed in Table 1.

2.2 Data Collection and Annotation Methodology

For every sense, around 50 sample sentences were collected. Annotation was based on the "one-sense-per-sample" rule [1]. A sample from our annotated corpus (`labeled_sentences.csv`) is given below (Turkish column titles and content intact):

```
Kelime,Anlam No,Anlam,Örnek Cümle
yüz,0,surat (yüz organı),kısa kütük gibi bir bedeni...bir yüzü vardı.
yüz,1,sayı (100 rakamı),makarna iki küçük kabak...yüz elli gr.
yüz,2,yüzmek (suda ilerlemek),yüksek tempoda yürüyüş...koşmak idealdir
```

Table 1: **Target Words and Their Senses**

Word	Sense 1	Sense 2	Sense 3
yüz	surat (yüz organı)	sayı (100 rakamı)	yüzmek (suda ilerlemek)
çal	hırsızlık yapmak	enstrüman çalmak	kapıya vurmak
yaz	mevsim (yaz ayı)	yazmak (kalemle yazı yazmak)	-
dal	ağaç parçası (dal)	belirli bir konu veya alan	suya atlamak (dalmak)
kara	siyah (renk)	toprak parçası (kara parçası)	kötü haber (kara haber)
at	hayvan (at)	fırlatmak (topu atmak)	-
gül	çiçek türü (gül çiçeği)	gülmek (kahkaha atmak)	-
kaz	hayvan (ördek benzeri)	kazmak (toprağı kazmak)	-
kır	arazi (kır alanı)	kırmak (bir nesneyi parçalamak)	beyaz saç rengi (kır saç)
dolu	boş olmayan	hava olayı	-

2.3 Data Preprocessing for Model Input

By concatenating the 'word' column with its original sense-number identifier (originally called 'Anlam No', then internally mapped), a `composite_id` (e.g., `yüz_0`) was generated. Later these `composite_ids` were mapped into integer labels in sequential order (referred to as the `sense_id` in the code and used as target labels in the model) for purposes of training and evaluation. This ensures that each unique word-sense pair is considered by the model as a separate class.

3 Model Architecture and Implementation

3.1 Data Representation: `ContextAwareWSDDataset`

The `ContextAwareWSDDataset` is a custom PyTorch `Dataset` class. Its function is to take a row from the dataset–containing the target word, the sentence, and the target `sense_id`–tokenize it in the way appropriate for BERT, and locate the target word in this tokenized sequence. The important steps in the `__getitem__` method are:

1. Create a `marked_sentence` as `f"{word} [SEP] {sentence}"`, explicitly indicating the target word to the model.
2. Tokenize the `marked_sentence` with the BERT tokenizer, producing `input_ids` and `attention_mask`.
3. Locate the target word tokens within `input_ids` (after the `[SEP]` token).
4. Generate a `target_mask` (binary tensor) having 1s at the target word token positions and 0 elsewhere. This mask will be used by the model to later isolate embeddings of the target word.
5. Return a dictionary containing `input_ids`, `attention_mask`, `target_mask`, padded `target_positions`, and the numerical labels.

This provides a form of input whereby the model can distinguish between the target word and context.

3.2 Model Architecture: `ContextAwareWSDModel`

The `ContextAwareWSDModel` (see Listing 1 for the forward pass outline) processes the input generated by the dataset class.

```
class ContextAwareWSDModel(torch.nn.Module):
  def __init__(...):
      self.bert = BertModel.from_pretrained('bert-base-uncased')
      self.attention = torch.nn.Linear(hidden_size, 1) # Custom attention
      # ... other layers (dropout, context_fusion, classifier) ...

  def forward(self, input_ids, attention_mask, target_mask=None):
      # 1. Get BERT's last hidden state
      bert_outputs = self.bert(input_ids, attention_mask)
      last_hidden_state = bert_outputs.last_hidden_state # [B, SeqLen,
         HiddenSize]

      # 2. Target Word Embedding Extraction
      # Uses target_mask to select and average embeddings for the target word
      # from last_hidden_state. Fallback to [CLS] if target_mask is None.
      # Result: target_embedding of shape [B, HiddenSize]

      # 3. Context Embedding (Custom Attention)
      # Applies a linear layer and softmax to last_hidden_state to get
      # attention_weights.
      # context_embedding = weighted sum of last_hidden_state using these
         weights.
      # Result: context_embedding of shape [B, HiddenSize]

      # 4. Fusion and Classification
      combined_embedding = torch.cat([target_embedding, context_embedding],
         dim=1)
      # Pass through fusion layers (Linear, Tanh, Dropout)
      fused_embedding =
         self.dropout(torch.tanh(self.context_fusion(combined_embedding)))

      logits = self.classifier(fused_embedding)
      return logits
```

Listing 1: Simplified forward pass logic of the ContextAwareWSDModel.

First, the model obtains contextualized embeddings from BERT. Then, a specialized representation for the `target word` is computed by using the `target_mask`, and a representation for the `context` is generated using an ad hoc-attention mechanism over all token embeddings. These two representations are concatenated, merged, and sent to a classifier to predict the sense.

3.3 Training Regimen

The model trained for 5 epochs with the AdamW optimizer instantiated with a learning rate set to 2e-5, and CrossEntropyLoss as the loss function. Standard training and evaluation procedures were implemented during which the loss and accuracy metrics were gathered. The training function proceeds with the following operations: transfer of data to the device, stating that gradients need to be reset for all optimizers, calculations on forward and backward passes, and finally, optimizer update. The evaluation proceeds inside the context of `torch.no_grad()`. In the end, the last epoch will generate detailed classification reports and confusion matrices.

4 Results

4.1 Evaluation Metrics

Accuracy, Precision, Recall, F1-Score (per-class and macro/weighted averages), and Confusion Matrix were used.

4.2 Training Performance

Training and Test Loss (Left) and Test Accuracy (Right) over Epochs are represented in Figure 1.

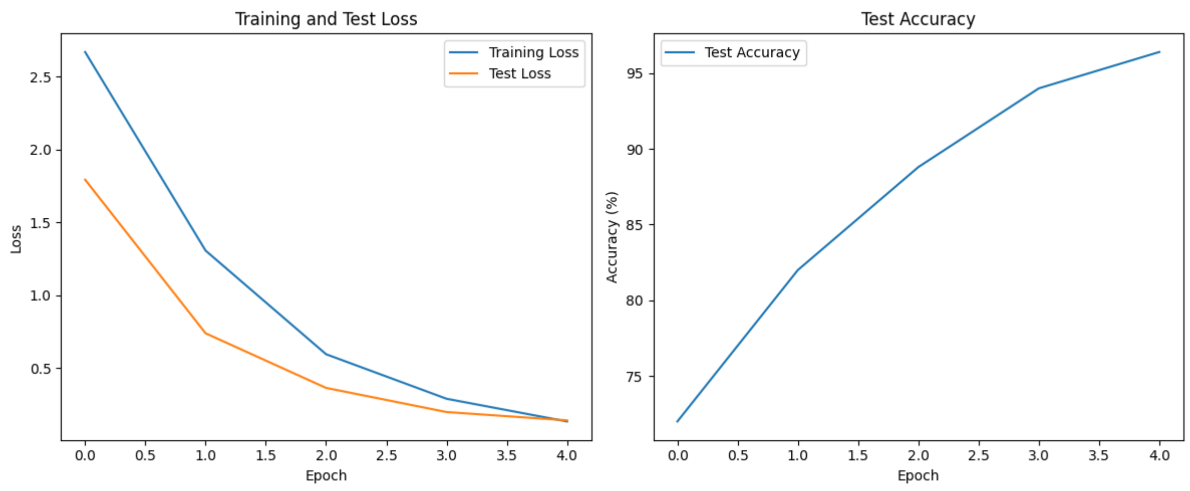

Figure 1

4.3 Classification Performance Analysis

4.3.1 Overall Accuracy and Class-wise Metrics

The classification report from the script's output provides per-sense performance.

```
# Example of expected output format from the script
Classification Report:
                 precision  recall   f1-score   support

        at_1       1.00       1.00      1.00        10
        at_2       1.00       1.00      1.00        10
        dal_1      1.00       1.00      1.00        10
        dal_2      1.00       1.00      1.00        10
        dal_3      1.00       1.00      1.00        10
        dolu_1     1.00       1.00      1.00        10
        dolu_2     1.00       1.00      1.00        10
        gül_1      1.00       1.00      1.00        10
        gül_2      1.00       1.00      1.00        10
        kara_1     1.00       1.00      1.00        10
        kara_2     1.00       1.00      1.00        10
        kara_3     1.00       1.00      1.00        10
        kaz_1      1.00       1.00      1.00        10
        kaz_2      1.00       1.00      1.00        10
        kır_1      1.00       1.00      1.00        10
        kır_2      1.00       1.00      1.00        10
        kır_3      1.00       1.00      1.00        10
        yaz_1      0.91       1.00      0.95        10
        yaz_2      1.00       1.00      1.00        10
        yüz_1      0.59       1.00      0.74        10
        yüz_2      1.00       0.90      0.95        10
```

```
       yüz_3       1.00      0.40      0.57        10
       çal_1       1.00      0.90      0.95        10
       çal_2       0.90      0.90      0.90        10
       çal_3       1.00      1.00      1.00        10

    accuracy                           0.96       250
   macro avg       0.98      0.96      0.96       250
weighted avg       0.98      0.96      0.96       250
```

.

4.3.2 Confusion Matrix Analysis

The confusion matrix, plotted in Figure 2, offers a deeper visualization detailing the layered classification abilities of the model, showing actual versus predicted sense labels. The diagonal carries the number of instances correctly predicted for each sense, and off-diagonal entries describe misclassification.

A detailed projection of the confusion matrix will give the following insights:

- **Commonly Confused Senses:** Having a look at those highly-counted off-diagonal boxes, one notes pairs or groups of senses that the model confuses with each other quite frequently. For instance, there would be an obvious lack of distinction if 'yüz_0' (surat) is more often confused with 'yüz_1' (sayı), or vice versa. Such confusions generally occur between semantically neighboring senses or between those that tend to appear within similar contexts.

- **Asymmetric Confusion:** Confusions are also, at times, simply asymmetric. For example, sense A is most of the time misclassified as sense B, whilst sense B is hardly ever misclassified as sense A. This might be an indicator that sense B is somewhat more generic or possesses some contextual clues which have a stronger effect than those from sense A.

- **Impact of Sense Frequency:** Senses with few examples to train on (i.e., lower support in the classification report) might be wrongly classified to common-generic-senses more often, or their correct classifications might be smaller. The confusion matrix can present a nice view of whether the less common senses are always 'outweighed' by the more common ones.

- **Well-Separated Senses:** Meaning instances with very few off-diagonal entries are highly separated by the model, meaning that their contextual indicators are heterogeneous and well acquired.

- **Systematic Errors:** Confusion patterns may suggest systematic flaws in the model or ambiguity in the data. For example, when a cluster of senses for a particular metaphorical application is conflated by the model, it probably fares poorly at the non-literal meaning for that word.

Looking at the produced confusion matrix (Figure 2), some observations pertinent to this research would be that yüz_3 gets confused with yüz_1. Analysis such as these is necessary if one wishes to examine the subtle behavior of the model and provide the basis for future design improvements to either the model architecture or the data set.

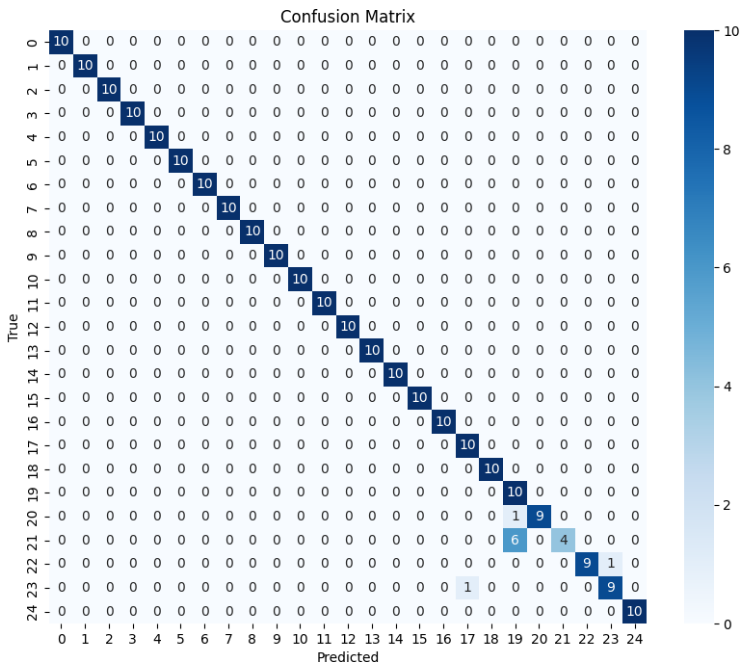
Figure 2: Confusion Matrix for test set predictions.

4.4 Qualitative Analysis: Example Prediction and Attention Visualization

In order to have qualitative knowledge of the model behavior, predictions for some specific cases were examined and the internal attention mechanism was visualized.

For an example sentence like `"hırsız bilgisayarımı çalmış."` with the target word `"çal"`, the `predict_sense` function is employed. The expected output format from the script would be:

```
# Example of expected output format from the script
Prediction for new sentence:
Sentence: 'hırsız bilgisayarımı çalmış.'
Target word: 'çal'
Predicted label: 22
Predicted sense ID: 1
Predicted meaning: 'hırsızlık yapmak'
```

Also, the `visualize_attention` function creates an attention plot where weights are assigned to the input tokens by the classifier's custom attentional layer. Such visualizations help figure out which bits of the sentence context the model considers most important in generating its context representation for disambiguation. Below is one example output of token-level attention data from the script:

```
# Example of expected token attention output
Token attention analysis:
[CLS]: 0.0139
cal: 0.0564
[SEP]: 0.0004
h: 0.0815
```

```
##ı: 0.0693
##rs: 0.0606
##ı: 0.0608
##z: 0.0596
bi: 0.0571
##l: 0.0540
##gis: 0.0505
##aya: 0.0493
##r: 0.0479
##ı: 0.0497
##m: 0.0620
##ı: 0.0561
calm: 0.0401
##ı: 0.0682
##s: 0.0446
.: 0.0177
[SEP]: 0.0004
```

5 Conclusion

This project was successful in developing a context-aware deep learning model for Word Sense Disambiguation in Turkish. One major contribution has been the development of a novel lexical sample dataset of 10 Turkish ambiguous words annotated with senses. The model, having a BERT encoder and employing special mechanisms for both target word and context representation, did very well in disambiguating the correct meanings of these words. The article discusses the potential use of Transformer-based models for solving tough semantic tasks in highly inflected languages like Turkish. The paper furnishes the design direction and concrete material to build upon for further studies pertaining to this problem, clearly indicating how the opposing information from the target word and the general context needs to be balanced for WSD. Introduction of `ContextAwareWSDDataset` and `ContextAwareWSDModel` controls the lack to tell the model which word in the input needs to be disambiguated, a critical component for lexical sample tasks.

References

[1] B. Ilgen, E. Adalı, and A. C. Tantuğ, "Building up lexical sample dataset for Turkish word sense disambiguation," in *2012 International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*, pp. 1–5, IEEE, 2012.

[2] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, pp. 4171-4186, 2019.

[3] A. Vaswani et al., "Attention is all you need," in *Advances in NIPS*, pp. 5998-6008, 2017.

[4] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *ICLR*, 2019.

[5] S. Schweter, "BERTurk - BERT for Turkish," GitHub repository, 2020.

[6] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. EMNLP (Demos)*, pp. 38-45, 2020.