SMAI Project Report

“Similarity aware deep attentive model for clickbait detection”

Team 36
Project No: 23
Team Name: (ReLu)ctantly_leaky

Team Members:

- **T. H. Arjun, 2019111012**
- **Arvindh A, 2019111010**
- **Thota Gokul Vamsi, 2019111009**
- **Rakesh Mukkara, 2019101087**

Abstract

For this project, **we implemented two papers**, “Similarity aware deep attentive model for clickbait detection” [1], which is the main paper given for the project. We implemented “Learning Deep Structured Semantic Models for Web Search using Clickthrough Data” [2] as a **baseline** like the original paper to compare our model with. Then we **also experimented by replacing the GRU layer of the given by a more SOTA transformer-based model Bert[5] and then a superior model by using a sentence transformer[6].** By doing so, we **show the advantages of contextual embeddings, subword tokenization and overcoming length** issues by moving on to the sentence level and **show much improvement in performance.** While implementing these architectures we **learned about the concepts of RNNs, GRUs, Attention Mechanism, Hashing TF-IDF, transformers, multi-headed self-attention and explored the intuition behind every decision in the architecture** and why it helps in the prediction. We discuss these studies of ours in detail in the section on each architecture. We then experimented with the baseline, the given paper and the proposed sentence transformer architecture on a Dataset that we split into train, test, and validation by combining three major datasets on the task detailed in the experiments section. In our experiments, **we were able to achieve an accuracy of 94% with the proposed sentence transformer model compared with 92% of the given paper and 76% of the baseline on the unseen test set**. We analyze our experiments with plots. We also analysed the effects of the ideas of the paper like using similarity as a feature, attention, and bidirectionality by removing them, and training a simple GRU on the body only and testing it and proving the usefulness of the proposed

ideas of the paper by showing a drop in performance without them. **As part of the submission, we also submit the codebase in PyTorch for the proposed sentence transformer model, the given paper, Bert-based model, baseline, checkpoints, and dataset and some scripts for reproducing the results.**

Problem Statement

Given a set of titles H = {h1,h2,...,hN}, and their bodies B = {b1,b2,...,bN}, the goal is to predict a label Y = {y1,y2,...,yN} of these pairs, where yi = 1 if headline i is a clickbait. For executing this we implement papers [1] and [2].

Baseline

The baseline model we have implemented corresponds to the paper 'Learning Deep Structured Semantic Models for Web Search using Clickthrough Data'[2] which is as shown in Figure 1. This paper discusses exploiting deep neural networks to calculate the similarity between a given query and a set of documents it is matched against. This can be correlated with the problem we are dealing with, which is, exploiting the similarity between title and body for a given document to evaluate if it's clickbaity in nature.

For a given query and a set of documents, text associated with them is projected into the same semantic space. Raw text features (e.g., raw counts of terms in a query or a document without normalization) for each of the above pieces of text is processed by a method called Word-Hashing. This method is based on the letter n-gram. In this method, based on a given word (e.g. good), firstly word starting and ending marks are added (e.g. #good#). Then, the word is broken into letter n-grams (e.g. letter trigrams: #go, goo, ood, od#). Finally, the word is represented using a vector of the letter n-grams. This processing is applied for every word in a piece of text, the motivation is reducing the number of features required to represent a text by a vector (as fewer bi-grams or tri-grams are possible, in comparison with the possible number of words that could occur in a context). Thus, after calculating this vector for a given text, it is passed through a Deep Neural Network (DNN), which contains 3 dense layers, and eventually, cosine similarity is employed to calculate the similarity for a given query and its document, and the output is obtained after applying a softmax activation.

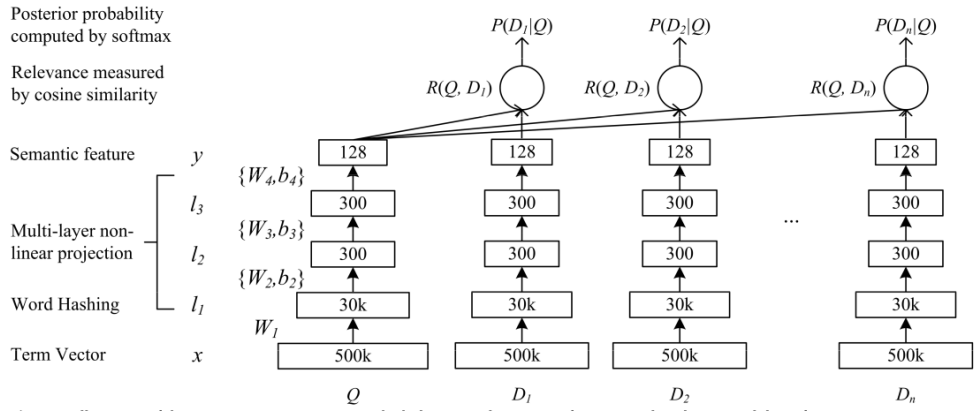

Figure 1: The architecture from [2] that we use as a baseline. (credits: the original paper)

Paper

The proposed architecture is from the paper, “Similarity aware deep attentive model for clickbait detection” [1] shown in Figure 2. The architecture is a similarity-aware attentive model (Bi-Directional GRU Model with attention) that captures and represents the similarities and differences between the misleading titles and the target content, with better expressiveness. The idea is to use similarity between heading and content to determine whether the content is clickbait or not. It uses learning local, global, and prediction learning where the problem is formulated in terms of these three losses.

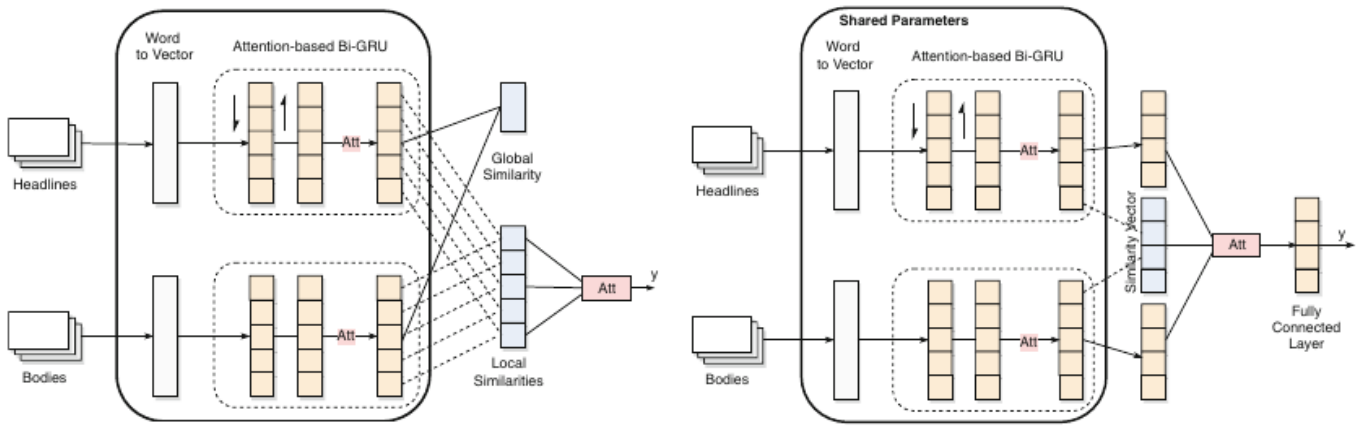

Figure 2: The Architecture from the paper given for the project (Credits: Original Paper)

Firstly the text is tokenized and passed through an Embedding Layer to get word embeddings. Thus for each token in heading and body, we get a word vector representation. These representations are then passed through a Bi-Directional GRU with shared parameters. Now the GRU Output for Each Token is concatenated as [Forward Output, Backward Output] and these vectors for each of the tokens for heading and body are passed through an attention layer. We take global similarity as the total similarity between these output sequences while we take the Local similarities and as the vector with similarities between corresponding chunks of the hidden layer representation of GRU. This is then passed through an attention layer. Both of these

outputs are used to predict y as y1, y2. We also concatenate the final layer representations of GRU along with a chunked similarity vector and then pass it through an attention layer followed by a fully connected layer resulting in prediction y3. We calculate Loss with respect to y1,y2,y3 which are termed as local, global, and learning losses. We backpropagate on the sum of these losses. During Prediction, we only take y3 as output like the way mentioned in the paper.

The motivation behind the architecture:

- The motivation behind using a Bi-Directional GRU:
    - RNNs like GRU are good at processing sequences (here text). They can aggregate meaningful representations for a sequence.
    - Bi-Directionality is motivated to aggregate meaning from the right and from the left because the meaning of a word depends on its context. For example:
        - S1: “I went to the bank to get on a boat”
          S2: “I went to the bank to get some money”
          In both these sentences, the word bank has a different meaning and our model will be able to capture this difference as the best way to do this is to look from the back and front of the sentence and aggregate meaning in both directions. As they say “You are defined by the company you keep” likewise words derive their meaning from the surrounding words. Thus motivating the bidirectionality.
- The motivation behind Attention:
    - By using Attention on the sequence, the model learns which words in the sequence to give importance to. Thus it disregards the representations of words that do not provide any usefulness to the final prediction.
- The motivation behind the three losses:
    - *Global Similarity Loss:* Here the model tries to learn the similarities between the representation of the whole text in Heading and Body. Thus learning it at a higher level
    - *Local Similarity Loss:* Here the model tries to learn the similarities at a granular level that is which parts of the text chunks are resulting in the prediction thus telling the sentence level contributions.
    - *Predictive learning:* Here the model tries to learn from a mix of representations (meaning of heading and body) and also from the similarity.
- The motivation behind using word2vec vectors for initial Embedding is that they have shown great performance in RNN based models in the literature.

Data

For Dataset we concatenate three famous datasets on the task [Clickbait Challenge](), [FNC dataset](), [Kaggle Clickbait Challenge](). We clean the text by preprocessing, lemmatizing, and then split it into test, train & validation sets. We have linked the dataset in CSV format along with the code. The training dataset contains 93,652 heading-body pairs, the test dataset contains 11,707 and the validation dataset contains 11,706 such pairs (an approximate 80-10-10 split was performed).

Hyperparameters

For training all the models, we use AdamW optimizer with a learning rate of 1e-5. We use the cross-entropy loss as the loss function. We add the cross-entropy loss of the three predictions in the case of the models using the ideas from the project paper. We also save the best model by using the validation which we submit. For the number of epochs to train the models for we trained each of the models until the validation losses plateaued as shown in Figure 3.

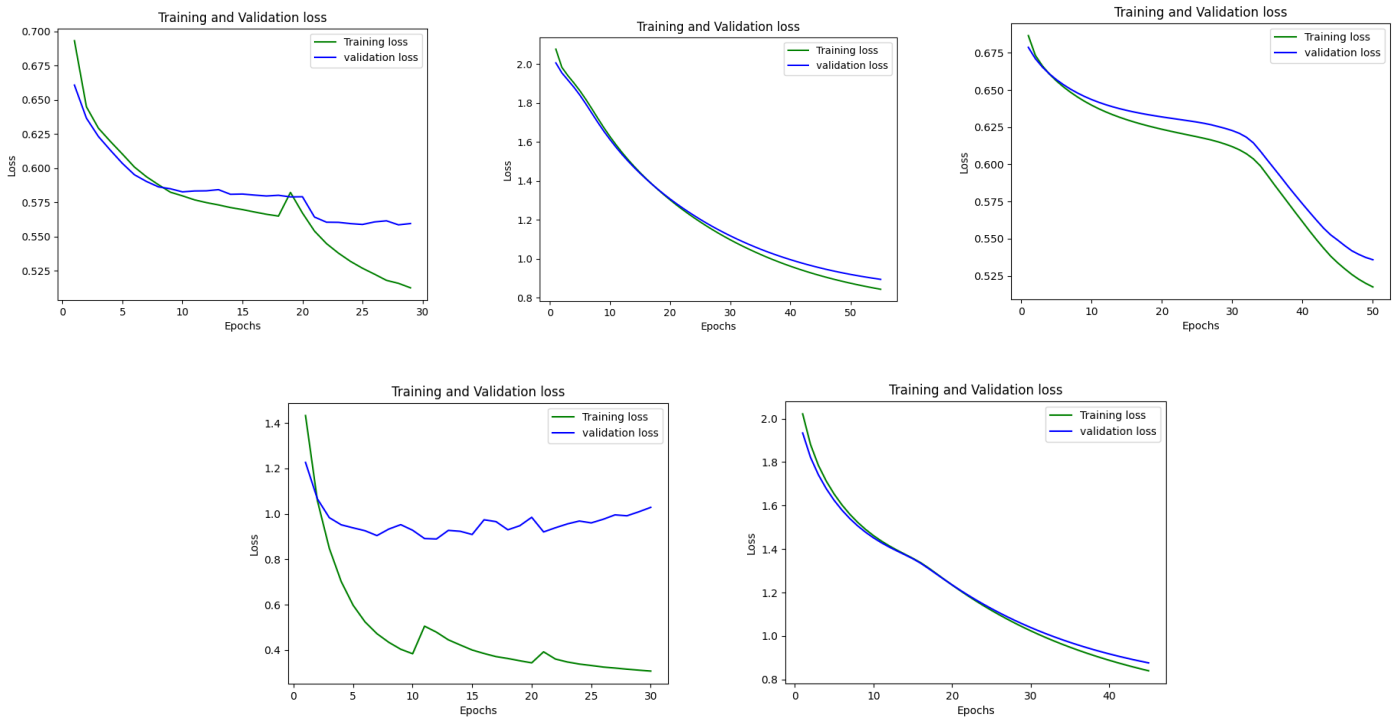

Figure 3: We train the models until the validation losses plateaued. The figures show validation losses and train losses vs Epochs for baseline, paper model, GRU only experiment, Bert-based, sentence-transformer model respectively (left to right, top to bottom)

For the word embeddings, we trained a word2vec model from scratch on the corpus for 200 epochs using the Gensim library as we were getting a lot of OOV words initially with pre-trained word embeddings on the Google News Corpus. For the implementation of the models, we use Pytorch.

Experiments

Model	*Accuracy*	*F1*	*Precision*	*Recall*
Baseline Model	0.7375	0.7289	0.7463	0.7283
GRU (only body)	0.7516	0.7493	0.7508	0.7485
Transformers	0.9166	0.9162	0.9159	0.9166
Paper Implementation	**0.9269**	**0.9267**	**0.9262**	**0.9283**
Sentence Transformers	**0.9370**	**0.9368**	**0.9365**	**0.9388**

Table 1: The Performance metrics of baseline, actual model and other experiments

How much do the ideas from the given paper help in the prediction? An Ablation study

To perform an Ablation study on the usefulness of the architectural features of the model proposed in the given paper as explained above, we trained a GRU model only on the body of the training set after removing the features of the architecture such as bi-directionality, attention and using the three learning from similarities from heading and body. In our experiments, we show that this model performs worse compared to other models on the test set as shown in Table 1. Thus proving the usefulness of these features to the model prediction. We establish that the ideas introduced in the paper are revolutionary for the boost in performance, clearly visible in Table 1.

Improving on top of the Paper Model- Introducing the Sentence transformer

To improve the model given in the paper, we experimented with two other models which helped us overcome the difficulties we faced.

Bert instead of GRU

To overcome the issue of the OOV we wanted to try and replace the GRU layer of the model with a SOTA transformer-based model Bert [5] because these words use subword tokenization and perform better than GRUs in terms of generating contextual embeddings because of multi-headed attention mechanism. They provide all the

usefulness of the bidirectional GRU but with much better performance. So we replaced the GRU with a Bert and used the last hidden state of Bert for each of the 512 tokens (Bert has this token limit) instead of the GRU. In our experiments, we see that the Bert does not improve the performance and is very similar to GRU or worse. Our hypothesis in this case for the failure of the Bert Model is as follows. As we mention, there is a limit of the number of tokens that can be given to a Bert model and the model we used from Google has a limit of 512 tokens. This limit is set on these transformer-based models because the self-attention mechanism scales quadratically in the sequence length. Here Bert uses subword tokenization and hence the 512 tokens of the really unclean text we have is not going to cover a lot of the context and hence due to truncation, it misses out a lot of information.

So to overcome this issue, we wanted to get the advantages of subword tokenization to overcome OOV but this was hindered due to the larger size of the text. Hence we experimented with the architecture mentioned below.

Sentence Encoder - Encoding at sentence level instead of the word and using sentence embeddings instead of word2vec.

Recently, [6] introduced an architecture based upon Bert that can create sentence level embeddings similar to doc2vec [7] but contextual. We thus use this architecture to pass it to our model. We tokenize at the sentence level instead of word-level and pass these sentences to the sentence encoder and get the sentence embeddings and use the sentence embeddings instead of word embeddings in the original paper model as shown in Figure 4. This provides two advantages. One is that since the sentence transformer uses subword tokenization, we can overcome OOV and since each sentence is not expected to exceed 512 tokens we don't lose any information. The second is that since the average text has about 30 sentences, we won't lose any information due to truncation in GRU also. This hypothesis is proved in our experiments.

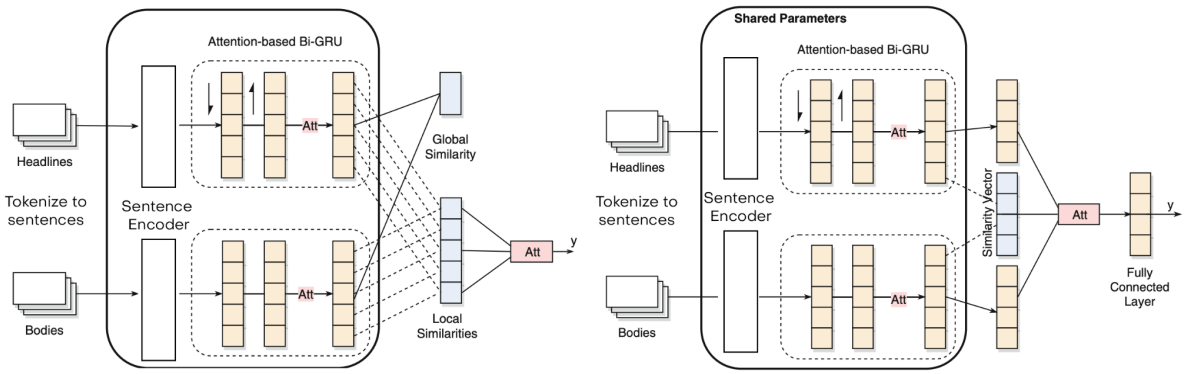

Figure 4: The Architecture from the paper modified with the sentence transformer which improves performance (Credits: Original Paper edited)

In our experiments, we were able to achieve **an accuracy of 93.70% with the proposed sentence encoder model compared with 92.69% of the model proposed by the given paper and 73.7% of the baseline on the unseen test set**. We plot the confusion matrices for both models as shown in Figure 5.

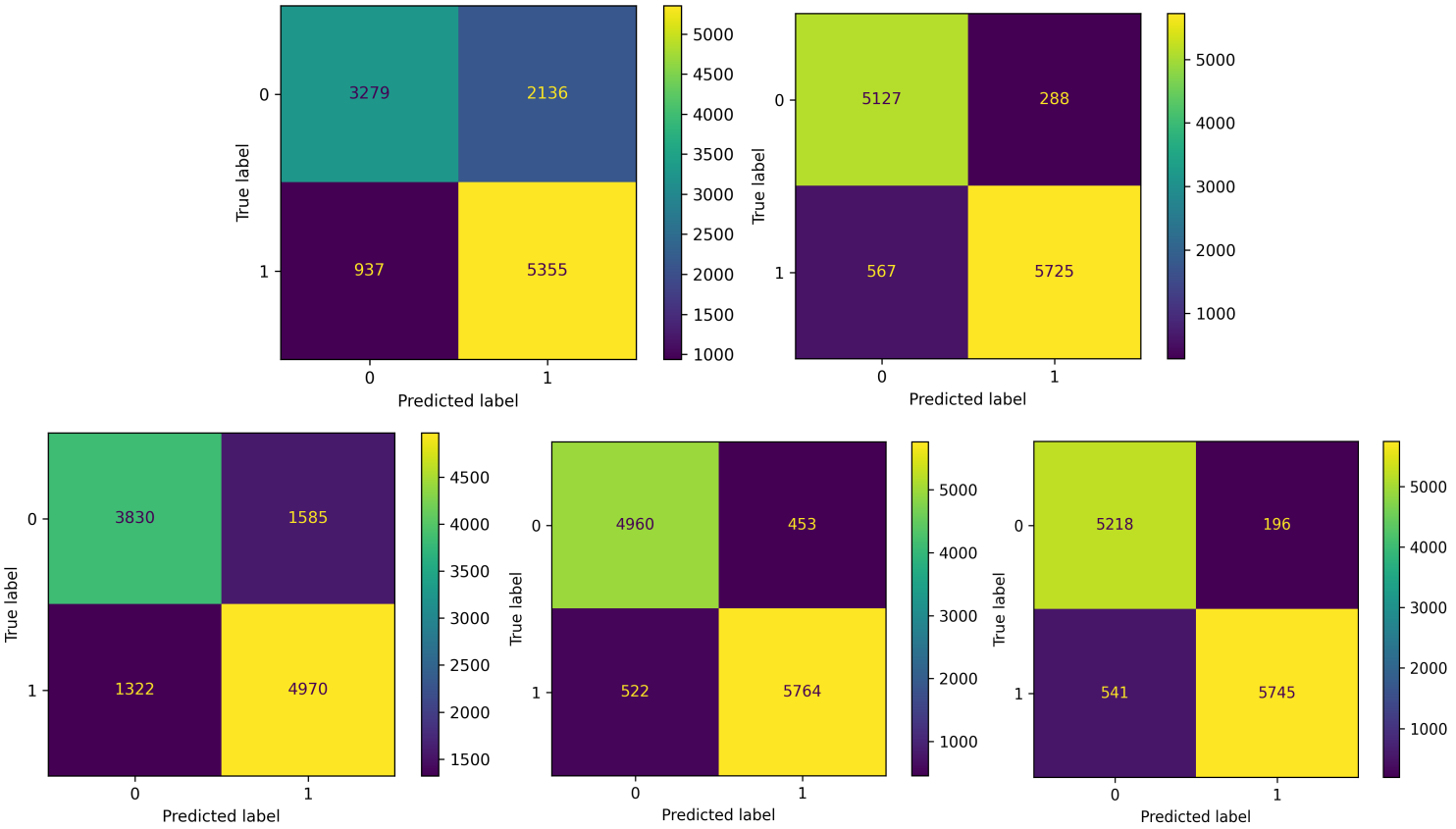

Figure 5: Confusion Matrices for baseline, paper model, GRU only experiment, Bert-based, sentence-transformer model respectively (left to right, top to bottom)

We show that the architecture from the paper outperforms the baseline by a big margin providing competitive metrics which we improve thereby a bigger margin using our proposed architecture as shown in Table 1. This we infer is due to the advantages these later models gain due to the intuitions we discussed before.

Deliverables Delivered

- Dataset and implementation corresponding to its pre-processing
- Complete implementation of one of the baselines mentioned in the paper (described above), and the relevant outputs obtained on training (train and validation losses), saved model checkpoint.
- Implementation of the model discussed in the paper - and the relevant outputs obtained on training (train and validation losses), saved model checkpoint.

- Implementation of different approaches, by experimenting with architectures and text embeddings.
    - Bi-directional GRU: Considering only body text from body-heading pairs, excluding attention mechanism and cosine similarity used in the original paper.
    - Sentence Transformers: Using sentence transformer for obtaining text embeddings instead of word2vec (which was trained in the previous phase).
    - BERT: Using transformer architecture, along with cosine similarity and attention mechanism for the whole classification task.

  Relevant outputs such as losses calculated, plots, confusion matrix are also logged in their respective folders. Model checkpoints are also saved and stored separately.

Challenges Faced

- On using the initial dataset while implementing the paper, it was observed that the classification performance was sub-optimal, primarily because of most words in the dataset were identified to be out of vocabulary (OOV) words with respect to the pre-trained word embeddings from Google News Corpus. Hence, to combat this challenge, a word2vec model was trained from scratch.
- On training the baseline model on a subset of data (significant size) for only a few epochs, it was observed that it predicted only one class for every data sample. This problem was addressed with the help of extensive training for more epochs on the whole dataset, and it was observed that the model started effective learning only after 3 epochs, where it started predicting both classes. This issue can be attributed to the nature of the text in the dataset (which also caused challenges with OOV words as discussed above).
- It was also a challenge to arrive at the best model checkpoint while training the model in the actual paper as well as the various experiments, because of lack of clarity up-front about the architecture's performance. To ensure that the loss values plateaued fairly well, we had to execute training scripts multiple times for multiple epochs, which varied based on the type of model.

Appendix

1. We release the dataset splits here.
2. We have uploaded the model checkpoints here
3. We have submitted all codes/scripts with a README for reproducibility
4. Shown below are some snippets of our implementation. You can refer to the codebase with the README for further details.

```python
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        w2v = KeyedVectors.load_word2vec_format("./models/word2vec.bin", binary=True)
        index_to_key = w2v.index_to_key
        key_to_index = w2v.key_to_index
        index_to_key.append("<oov>")
        index_to_key.append("<pad>")
        key_to_index["<oov>"] = index_to_key.index("<oov>")
        key_to_index["<pad>"] = index_to_key.index("<pad>")
        weights = w2v.vectors
        weights = np.append(weights,np.array([[0]*50,[0]*50]),axis=0)
        self.tokenizer = Tokenizer(key_to_index)
        self.embedding = nn.Embedding.from_pretrained(torch.FloatTensor(weights),padding_idx=key_to_index["<pad>"],freeze=True)
        self.h_atten = Attention(128,512)
        self.b_atten = Attention(128, 512)
        self.sim_atten = Attention(1,4)
        self.biGRU = nn.GRU(50, 64, bidirectional=True, batch_first=True)
        self.cos = torch.nn.CosineSimilarity(dim=1, eps=1e-08)
        self.pred =  nn.Linear(260 , 2)
        self.local =  nn.Linear(1 , 2)
    def forward(self,h,b,hm,bm):
        # Headine..
        hemb = self.embedding(h)
        gru_out, _ = self.biGRU(hemb)
        h_att = self.h_atten(gru_out,hm)

        # Body
        bemb = self.embedding(b)
        gru_out, _ = self.biGRU(bemb)
        b_att = self.b_atten(gru_out,bm)

        #Global Prediction
        glob_sim = self.cos(h_att,b_att).unsqueeze(-1)
        global_logits = torch.cat((glob_sim, 1-glob_sim),1)

        # Similarity Vector
        h_chunks = torch.chunk(h_att,4,dim=1)
        b_chunks = torch.chunk(b_att,4,dim=1)
        sim_vector = torch.cat([self.cos(hc,bc).unsqueeze(-1) for hc , bc in zip(h_chunks,b_chunks)],dim=1)

        # Local Prediction
        local_pred = self.sim_atten(sim_vector.unsqueeze(-1))
        local_logits = self.local(local_pred)

        # Prediction
        final_vector = torch.cat((h_att,sim_vector,b_att),dim=1)
        prediction_logits = self.pred(final_vector)
        return local_logits , global_logits , prediction_logits
```

Figure 6: The main paper model code snippet

```python
class Attention(nn.Module):
    def __init__(self, feature_dim, step_dim, bias=True, **kwargs):
        super(Attention, self).__init__(**kwargs)

        self.supports_masking = True

        self.bias = bias
        self.feature_dim = feature_dim
        self.step_dim = step_dim
        self.features_dim = 0

        weight = torch.zeros(feature_dim, 1)
        nn.init.kaiming_uniform_(weight)
        self.weight = nn.Parameter(weight)

        if bias:
            self.b = nn.Parameter(torch.zeros(step_dim))

    def forward(self, x, mask=None):
        feature_dim = self.feature_dim
        step_dim = self.step_dim

        eij = torch.mm(
            x.contiguous().view(-1, feature_dim),
            self.weight
        ).view(-1, step_dim)

        if self.bias:
            eij = eij + self.b

        eij = torch.tanh(eij)
        a = torch.exp(eij)

        if mask is not None:
            a = a * mask

        a = a / (torch.sum(a, 1, keepdim=True) + 1e-10)

        weighted_input = x * torch.unsqueeze(a, -1)
        return torch.sum(weighted_input, 1)
```

```python
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.l1 =  nn.Linear(VOCAB_SIZE ,WORD_DIM)
        self.l2 =  nn.Linear(WORD_DIM ,WORD_DIM)
        self.l3 =  nn.Linear(WORD_DIM ,FINAL_DIM)
        self.tanh = nn.Tanh()
        self.do = nn.Dropout(p=0.2)
        self.cos = torch.nn.CosineSimilarity(dim=1, eps=
1e-08)
        self.l4 = nn.Linear(1,2)
    def forward(self,h,b):
        h = h.squeeze(1)
        b = b.squeeze(1)
        h = self.l1(h)
        h = self.tanh(h)
        h = self.l2(h)
        h = self.tanh(h)
        h = self.l3(h)
        h = self.tanh(h)
        b = self.l1(b)
        b = self.tanh(b)
        b = self.l2(b)
        b = self.tanh(b)
        b = self.l3(b)
        b = self.tanh(b)
        h = self.do(h)
        b = self.do(b)
        c = self.cos(h,b).unsqueeze(-1)
        out = self.l4(c)
        return out
```

```python
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.bert = AutoModel.from_pretrained("bert-base-uncased")
        self.h_atten = Attention(768,512)
        self.b_atten = Attention(768,512)
        self.sim_atten = Attention(1,4)
        self.cos = torch.nn.CosineSimilarity(dim=1, eps=1e-08)
        self.pred =  nn.Linear(1540 , 2)
        self.local =  nn.Linear(1 , 2)
    def forward(self,hi,hm,ht,bi,bm,bt):
        # Headine.
        houtputs = self.bert(hi, attention_mask=hm, token_type_ids=ht)
        hhidden , _ = houtputs[0], houtputs[1]
        h_att = self.h_atten(hhidden,hm)

        # Body
        boutputs = self.bert(bi, attention_mask=bm, token_type_ids=bt)
        bhidden,_= boutputs[0], boutputs[1]
        b_att = self.b_atten(bhidden,bm)

        #Global Prediction
        glob_sim = self.cos(h_att,b_att).unsqueeze(-1)
        global_logits = torch.cat((glob_sim, 1-glob_sim),1)

        # Similarity Vector
        h_chunks = torch.chunk(h_att,4,dim=1)
        b_chunks = torch.chunk(b_att,4,dim=1)
        sim_vector = torch.cat([self.cos(hc,bc).unsqueeze(-1) for
 hc , bc in zip(h_chunks,b_chunks)],dim=1)

        # Local Prediction
        local_pred = self.sim_atten(sim_vector.unsqueeze(-1))
        local_logits = self.local(local_pred)

        # Prediction
        final_vector = torch.cat((h_att,sim_vector,b_att),dim=1)
        #print(final_vector.shape)
        prediction_logits = self.pred(final_vector)
        return local_logits , global_logits , prediction_logits
```

```python
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.h_atten = Attention(128,8)
        self.b_atten = Attention(128,30)
        self.biGRU = nn.GRU(768, 64, bidirectional=True,
batch_first=True)
        self.sim_atten = Attention(1,4)
        self.cos = torch.nn.CosineSimilarity(dim=1, eps=1e-08)
        self.pred =  nn.Linear(260 , 2)
        self.local =  nn.Linear(1 , 2)
    def forward(self,h,hm,b,bm,l):
        # Heading
        hgru_out, _ = self.biGRU(h)
        h_att = self.h_atten(hgru_out,hm)
        # Body
        bgru_out, _ = self.biGRU(b)
        b_att = self.b_atten(bgru_out,bm)
        #Global Prediction
        glob_sim = self.cos(h_att,b_att).unsqueeze(-1)
        global_logits = torch.cat((glob_sim, 1-glob_sim),1)
        # Similarity Vector
        h_chunks = torch.chunk(h_att,4,dim=1)
        b_chunks = torch.chunk(b_att,4,dim=1)
        sim_vector = torch.cat([self.cos(hc,bc).unsqueeze(-1)
for hc , bc in zip(h_chunks,b_chunks)],dim=1)
        # Local Prediction
        local_pred = self.sim_atten(sim_vector.unsqueeze(-1))
        local_logits = self.local(local_pred)
        # Prediction
        final_vector = torch.cat((h_att,sim_vector,b_att),dim=1)
        #print(final_vector.shape)
        prediction_logits = self.pred(final_vector)
        return local_logits , global_logits , prediction_logits
```

Figure 7: Attention Layer, Baseline model, Bert, Sentence Encoder+GRU code snippets(from left to right, top to bottom)

References

1. Dong, Manqing & Yao, Lina & Wang, Xianzhi & Benatallah, Boualem & Huang, Chaoran. (2019). *Similarity-Aware Deep Attentive Model for Clickbait Detection.*
2. Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. *Learning deep structured semantic models for web search using clickthrough data*. In Proceedings of the 22nd ACM international conference on Information & Knowledge Management (CIKM '13). Association for Computing Machinery, New York, NY, USA, 2333–2338.
3. Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations.
4. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need.
5. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
6. Thakur, N., Reimers, N., Daxenberger, J., & Gurevych, I. (2020). Augmented SBERT: Data Augmentation Method for Improving Bi-Encoders for Pairwise Sentence Scoring Tasks. CoRR
7. Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14). JMLR.org, II–1188–II–1196.