MAC5725 - Linguística Computacional
Relatorio EP3

Andre Barbosa-nUSP 7971751

22 de dezembro de 2020

Conteúdo

1 Introdução

O objetivo deste relatório é apresentar uma discussão do que foi discutido para a elaboração do terceiro Exercício Programa (EP) da disciplina MAC 5725.

2 Organização Geral do projeto e outras considerações

Os experimentos reportados neste relatório foram rodados em uma GPU NVIDIA Tesla V100-PCIE-16GB disponível no servidor **brutucuvi** do IME. Mais detalhes para reproduzir os experimentos estão no README do projeto. Além disso, no README também está apresentada a estrutura geral do projeto. A idéia foi ter um código modular e com as responsabilidades isoladas. Os principais resultados obtidos e experimentos conduzidos podem ser vistos pelos jupyter encontrados em *notebook/estudos-bert.ipynb* e *notebook/estudos-bilstm.ipynb* para encontrar os estudos conduzidos para a rede geradora de títulos usando o BERT e a BiLSTM, respectivamente.

É válido ressaltar, contúdo, que os notebooks foram gerados com a versão **reduzida** do dataset, enquanto os resultados finais foram baseados pelo dataset **inteiro**, gerados através da interface por linha de comando (**cli**) desenvolvida no script *ep3.py*

3 Pré Processamento dos dados

O pré processamento dos dados foi basicamente o mesmo considerando as duas redes. Detalhes de implementação estão dentro do código *code_utils/data_preprocessing.py*. O conjunto de dados foi dividido em 80% como treino e 20% para teste e computar as métricas finais. Além disso, considerando a parte do texto que funcionará como *decoder*, dois tokens especiais foram incluidos, um para indicar começo de sentença, chamado *xxstart* e outro para indicar fim de sentença, denominado *xxend*. Durante o treinamento, 10% dos dados foram reservados para validação.

Para fins de padding, foram computadas as estatísticas sobre os tamanhos das sequências afim de determinar os tamanhos de sequências máximas dos *reviews* e dos títulos. Com isso, fixou-se 65 como sequência máxima para a rede encoder e 10 como sequência máxima para a rede decoder.

Com relação ao pré processamento de texto, foi-se aplicado a *menor* quantidade possível, uma vez que foi utilizada a camada *TextVectorization* do Tensorflow e os tokenizadores da biblioteca Transformers ([Wol+20]).

4 Encoder-Decoder BiLSTM

Para a primeira parte do EP, codificou-se uma rede encoder-decoder LSTM com uma camada de atenção intermediária relacionando as partes da rede. Tanto

para o encoder quando para o decoder, foi utilizado uma LSTM de uma camada. A arquitetura geral da rede pode ser vista pela figura 1

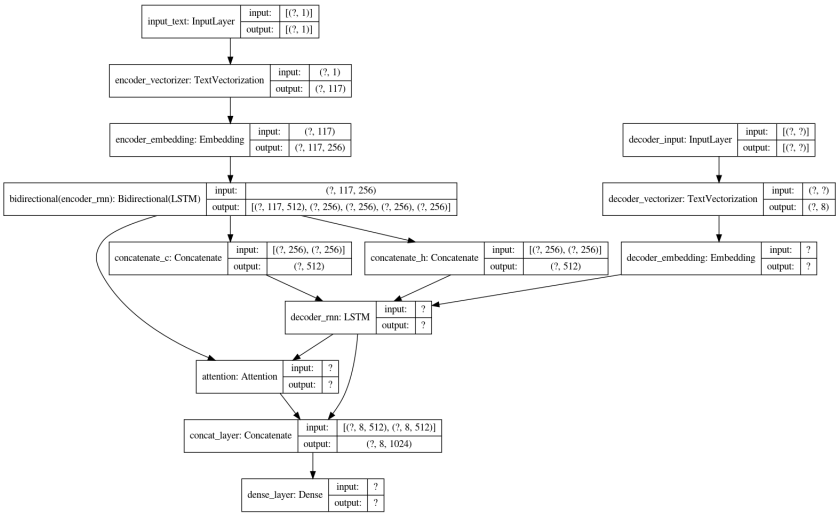

Figura 1: Modelo BiLSTM utilizado como Encoder Decoder

A figura 1 pode ser entendida da seguinte forma:

1. As camadas *input_ layer* e *decoder_ input* que recebe um texto de entrada

2. As camadas *encoder_ vectorizer* e *decoder_ vectorizer*, pré processam os textos do encoder (texto do *review*) e do decoder (texto do título), respectivamente o texto de entrada, tokenizando e fazendo os pré processamentos necessários, como mapear as palavras para inteiros, eliminas excesso de espaços e remover acentos

3. Cada palavra é mapeada para uma matriz de Embeddings. No caso, então, existem dois embeddings, um para o encoder (*encoder_ embedding*) e outro para o decoder (*decoder_ embedding*). A escolha de dois embeddings diferentes é porquê a forma de escrever textos de *review* e títulos é diferente e, logo, pode a distribuição pode ser diferente

4. No caso do Encoder, os Embeddings servem de entrada para uma rede *BiLSTM* que, por ser **bidirecional**, gera 4 saídas: dois estados intermediários (o estado célula (c) e o estado oculto (h)), no sentido da esquerda para direita; e os estados **c** e **h** da direita para esquerda.

5. Os respectivos estados **c** e **h** são concatenados e, então, servem de incializadores para a camada *LSTM* do decoder, que também recebe o embedding *decoder_ embedding*

6. Para a camada de Atenção, aplicou-se a saída do encoder (encoder_rnn) com a saída do decoder (decoder_rnn).

7. A saída da camada de atenção foi concatenada com a saída do decoder_rnn e isso serviu de entrada para uma rede densa, cuja saída era o vocabulário

4.1 Configurações dos Experimentos

No caso desta rede, foi-se utilizado o otimizador foi o Adam [KB17], com o *callback* de *EarlyStopping* do *Keras* [Cho+15] com *patience=2* como critério de parada. Por conta disso, o número de épocas foi fixado em 50.

Como a saída é o vocabulário, dado que o objetivo é aprender qual é a próxima palavra, a função de loss utilizada foi a *Sparse Categorical Cross Entropy*. Para todos os experimentos utilizados, foram mantidas a mesma arquitetura geral da rede, otimizador, e hiperparâmetros.

4.1.1 Mecanismos de Atenção

Existem três implementações de mecanismos de atenção no Keras, a atenção de Luong ([LPM15]) e Bahdanau ([BCB16]), em que as diferenças entre ambos é dada basicamente pelo cálculo do vetor de contextos entre a entrada do decoder e as saidas do encoder. No modelo proposto por Bahdanau, os valores de atenção são obtidos por meio de uma soma ponderada (por isso ela também recebe o nome de Additive Attention), enquanto o de Luong, pode ser dada por três diferentes formas de produto interno (por isso, ela recebe o nome de Dot Product Attention). Além disso, como estamos lidando com um modelo de decoder, é preciso utilizar o parâmetro **causal** como **True**, que adiciona uma máscara em uma determinada sentença afim de evitar vazamento de informação para gerar a palavra em uma determinada posição. Os dois mecanismos foram testados **com os dados reduzidos e apenas com o modelo BiLSTM**. Os resultados são apresentados pela tabela 1 utilizando dados de validação **rodando após 5 épocas**.

4.1.2 Geração do Decoder

Para gerar as frases, o algoritmo utilizado foi totalmente guloso. Basicamente, verificá-se se a sentença atual atingiu o tamanho limite (10 palavras), ou se a palavra atual era igual ao token que indicava fim de sentença (*xxend*). Por conta disso, a geração de frases pelo modelo tinha uma complexidade $\mathcal{O}(km)$, com $k =$ número de dados de teste e $m = 10$, o tamanho máximo de sentenças. Por conta disso, as métricas que comparavam sentenças (BLEU, METEOR e NIST) foram computadas com 1000 amostras.

Mecanismo de Atenção	Validação Loss	Acurácia da Vailidação
Additive Attention	2.1922	0.6764
Dot Product Attention	2.2126	0.6775

Tabela 1: Comparação dos mecanismos de atenção nos dados de validação

Como os resultados foram próximos, optou-se por utilizar a atenção de Luong em todos os experimentos.

4.2 Resultados e Discussão

4.2.1 Métricas de Treino e Validação

Afim de validar a presença de overfitting ou underfitting, foram computadas a loss do modelo. As métricas foram comparadas entre **treino** e **validação**. Os resultados são apresentados nas figuras 2. Além disso, a acurácia obtida entre treino e validação está apresentada na tabela 2.

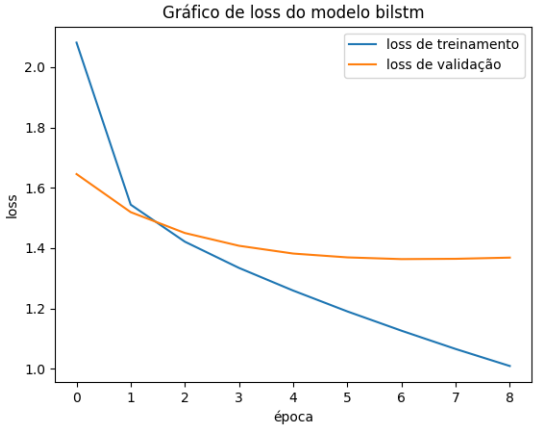

Figura 2: Curva de loss do modelo BiLSTM

Acurácia no Treino	Acurácia em Validação
0.7814	0.7627

Tabela 2: Acurácia de Treino e Validação

É possível ver que os resultados ficaram parecidos, o que é positivo.

4.2.2 Métricas de Teste

Para avaliar a qualidade do modelo, foram utilizados métricas comuns em sistemas de tradução: Bleu ([Pap+02]), METEOR ([BL05]) e NIST ([Dod02]). No caso, a implementação do NLTK do Bleu pode ser entendida como uma espécie de intersecção de *n-gramas* (com n entre 1 e 4) normalizada pelo tamanho da frase e aplicando uma penalidade se o resultado tem um tamanho muito curto. A métrica final, então, é um número entre 0 (não tem nenhuma intersecção) e 1 (intersecção total entre o termo gerado e o original). Contudo, o Bleu possui algumas falhas, da qual segundo o artigo [COK06], podemos citar as seguintes:

1. Ela não leva em consideração o significado (o NIST tenta corrigir isso)
2. Ela não considera diretamente a estrutura da sentença
3. Ela não mapeia muito bem quando comparada com julgamento humano (o METEOR tenta corrigir isso)

Como alternativas em relação à esses problemas apresentados pelo BLEU, duas novas métricas foram propostas: o METEOR e o NIST. O NIST é baseado diretamente no BLEU, tendo como principais diferenças o fato de que a métrica de penalização é diferente e, além disso, palavras mais raras tem um peso maior e, enquanto o BLEU utiliza usa uma média geométrica para determinar as intersecções dos *n-gramas*, o NIST usa média aritimética. O METEOR, por sua vez, utiliza apenas unigrama e não tem a abordagem de n-gramas que as outras métricas, mas também leva em consideração stemming e sinônimos. Além disso, o METEOR usa um F1 parametrizado para ter uma maior correlação com o julgamento humano, enquanto o BLEU e NIST usam são mais orientados para precisão. Os valores do BLEU e METEOR variam entre 0 e 1, em que 1 indica um valor perfeito. No caso do NIST, o valor é positivo e quanto maior, melhor. As métricas utilizadas em **todos** os experimentos foram a respeito do corpus. No caso do BLEU e NIST, usou-se o módulo *corpus_ bleu* e *corpus_ nist*. No caso do METEOR, por conta da ausência de um método direto, foi aplicado a média do score obtido com base em todas as sentenças, porquê essa é exatamente a mesma implementação aplicada pela biblioteca datasets do HuggingFace [Wol+20] Os resultados dos experimentos com relação a rede BiLSTM estão presentes na tabela 3

Acurácia no Teste	BLEU	METEOR	NIST
0.771	0.2656	1.7228	0.2120

Tabela 3: Métricas de Teste

Apesar da acurácia do modelo ter sido razoável, o BLEU indica que o resultado do modelo foi um pouco medíocre, uma vez que a métrica aponta que um modelo ideal tem resultado perto de 1.

5 BERT como Encoder

Seguindo o enunciado do EP, a implementação desta rede foi **diferente** da proposta pelo enunciado. Sua estrutura será detalhada a seguir,mas pode ser resumida pela figura 3

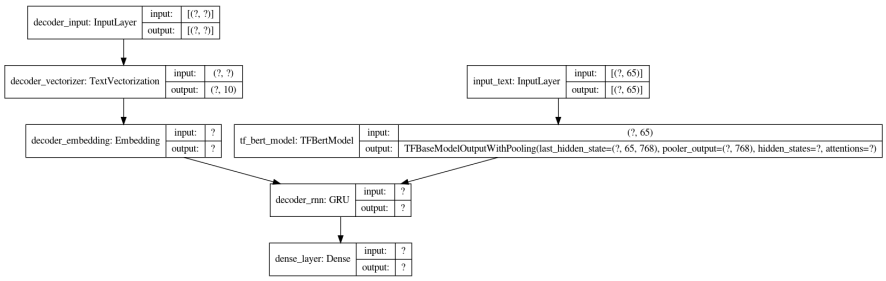

Figura 3: Modelo BERT utilizado como Encoder

A figura 3 pode ser entendida da seguinte forma:

1. As camadas *decoder_ input* recebe um texto de entrada enquanto a camada *input_ text* recebe um texto já tokenizado pelo tokenizador *BERTTokenizerFast* da biblioteca HuggingFace [Wol+20], utilizando como base o modelo BERTimbau [SNL20]

2. As camadas *decoder_ vectorizer*, pré processam os textos dodecoder (texto do título), tokenizando e fazendo os pré processamentos necessários, como mapear as palavras para inteiros, eliminas excesso de espaços e remover acentos

3. Cada palavra do decoder é mapeada para uma matriz de Embeddings, *decoder_ embedding*.

4. O *input_ text* é usado de entrada para o modelo BERT, treinado pelo modelo de linguagem do BERTimbau, na versão base (o modelo com 768 dimensões).

5. Com relação ao modelo BERT, a saída *pooler*, que é uma camada linear seguida de uma *tanh* aplicada em cima do primeiro token do modelo é usada como estado inicial da camada *GRU* aplicada como decoder. [1] [2]

6. A saída do GRU servem de entrada para uma camada de densa, assim como o modelo apresentado na sessão 4

[1] Também foi feito um experimento em que era validada a possibilidade de aplicar um *AverageGlobalPooling1D* nos pesos da ultima camada e usar o resultado como entrada para a rede densa, mas os resultados não mudaram

[2] também testou-se utilizar mais uma camada de Atenção, recebendo como entrada os pesos da ultima camada do BERT junto com a saída da GRU, mas isso resultava em um overfitting intenso e, como opção, essa camada foi removida

A principal vantagem de utilizar a rede desta forma é que, diferente do proposto no enunciado como forma opcional, com esta implementação de arquitetura, não haveria necessidade de modificar os dados de treinamento da rede.

5.1 Configurações dos Experimentos

Para os experimentos, mantiveram-se os mesmos hiper parâmetros que os citados na sessão 4.1. Com relação a geração das sequências para validação das métricas, foi empregado o mesmo algoritmo que o descrito na sessão 4.1.2

5.2 Resultados e Discussão

5.2.1 Métricas de Treino e Validação

Seguindo o que foi apresentado na sessão 4.2.1, os resultados são apresentados pelas figuras 4 para a função de custo e 4 no caso de acurácia.

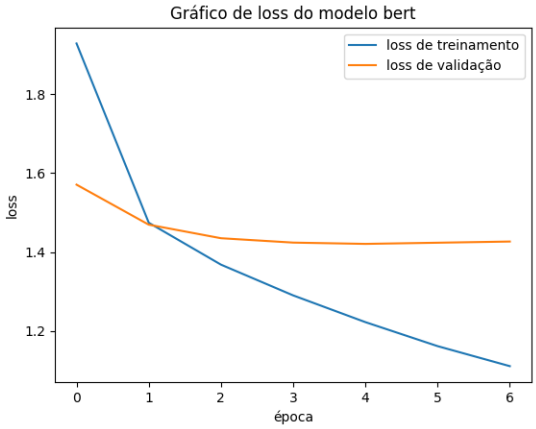

Figura 4: Modelo BERT utilizado como Encoder

Acurácia no Treino	Acurácia em Validação
0.7730	0.7517

Tabela 4: Acurácia de Treino e Validação

5.2.2 Métricas de Teste

As métricas foram as mesmas discutidas na sessão 4.2.2 e apresentados na tabela 5

Acurácia no Teste	BLEU	METEOR	NIST
0.7624	0.3299	3.3336	0.19621

Tabela 5: Métricas de Teste

6 Discussão Geral

É interessante ver que, apesar do BERT ser muito mais pesado e levar muito mais tempo para rodar, os modelos tiveram resultados muito próximos, sendo que o modelo que possuia o BERT como encoder teve um resultado *levemente* melhor quando comparado ao modelo BiLSTM. Contudo, vale ressaltar que analisando a curva de loss, o modelo com BERT teve uma tendência a overfitting maior, o que indica que com mais dados de treinamento, o modelo poderia ter uma convergência melhor. Porém, avaliando a qualidade das sentenças geradas, assim como interpretando as métricas BLEU, METEOR e NIST, é possível perceber que o modelo gerador de títulos ainda está muito aquém do ideal.

7 Miscelânea

Por conta da baixa qualidade dos dados, um dos maiores desafios desse exercício programa foi ter *certeza* se a rede estava implementada corretaemente. Afim de validar isso, criou-se um conjunto de dados *fake* que recebia como arquitetura de rede **exatamente** a mesma rede utilizada em cada um dos experimentos. Os dados por sua vez, também seguiam uma estrutura para alimentar um *encoder* e *decoder*. No caso, porém, o encoder era uma sequência de 20 de números em ordem aleatória e o decoder era um subconjunto da lista, contendo 5 números, em ordem invertida. A hipótese aplicada nesse experimento era que se a rede conseguisse aprender essa tarefa, o problema muito provavelmente estaria relacionado aos dados do EP em si.

Os resultados desses experimentos estão no caminho *notebooks/debbuger_ bert.ipynb* e *notebooks/debbuger_ bilstm.ipynb* para os experimentos feitos com o BERT e a BiLSTM, respectivamente.

Os resultados foram bem sucedidos, o que indica que o maior desafio muito provavelmente se dá pela baixa qualidade dos dados deste EP. Além disso, foi interessante ver que nessas configurações, a rede BERT também teve uma *leve* vantagem

8 Bibliografia

[Dod02] George Doddington. «Automatic Evaluation of Machine Translation Quality Using N-Gram Co-Occurrence Statistics». Em: *Proceedings of the Second International Conference on Human Language Tech-*

nology Research. HLT '02. San Diego, California: Morgan Kaufmann Publishers Inc., 2002, pp. 138–145.

[Pap+02] Kishore Papineni et al. «Bleu: a Method for Automatic Evaluation of Machine Translation». Em: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, jul. de 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://www.aclweb.org/anthology/P02-1040.

[BL05] Satanjeev Banerjee e Alon Lavie. «METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments». Em: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ann Arbor, Michigan: Association for Computational Linguistics, jun. de 2005, pp. 65–72. URL: https://www.aclweb.org/anthology/W05-0909.

[COK06] Chris Callison-Burch, Miles Osborne e Philipp Koehn. «Re-evaluating the Role of Bleu in Machine Translation Research». Em: *11th Conference of the European Chapter of the Association for Computational Linguistics*. Trento, Italy: Association for Computational Linguistics, abr. de 2006. URL: https://www.aclweb.org/anthology/E06-1032.

[Cho+15] François Chollet et al. *Keras*. https://keras.io. 2015.

[LPM15] Minh-Thang Luong, Hieu Pham e Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL].

[BCB16] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[KB17] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[SNL20] Fábio Souza, Rodrigo Nogueira e Roberto Lotufo. «BERTimbau: pretrained BERT models for Brazilian Portuguese». Em: *9th Brazilian Conference on Intelligent Systems, BRACIS, Rio Grande do Sul, Brazil, October 20-23 (to appear)*. 2020.

[Wol+20] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. arXiv: 1910.03771 [cs.CL].