Linguística Computacional - MAC5725
Instituto de Matemática e Estatística
Universidade de São Paulo

Treinamento de Etiquetadores Morfossintáticos com LSTM bidirecionais

Fernando Leandro dos Santos

6 de Dezembro de 2018

Resumo

This paper describes an implementation of a bidirectional LSTM neural network for part-of-speech tagging on a portuguese dataset. A LSTM network was trained from a collection of portuguese sentences from the CETENFolha dataset. The embedding layer of the network was pre-trained with a 100 dimensions GloVe word embedding representation, provided by NILC. The LSTM network achived an average accuracy of 96.682% over a 5-fold cross validation iteration on the dataset. It outperformed previous methods applied and trained with the same dataset, a baseline model based on words frequency and a Viterbi implementation, which achieved the accuracy of 94.587% and 96.275%, respectively.

1 Introdução

O problema tratado neste artigo é a etiquetagem morfossintática de sentenças em português, ou como é mais conhecido, *part-of-speech tagging*. É apresentada uma solução utilizando redes neurais recorrentes, mais especificamente, uma rede de arquitetura conhecida por *Long Short-Term Memory*[4] (LSTM). Em outras palavras, essa rede neural aqui apresentada é capaz de classificar palavras do português em determinadas classes morfossintáticas. Nessa implementação são realizadas algumas simplificações, utilizamos uma quantidade fixa de etiquetas possíveis, uma quantidade fixa de palavras conhecidas e utilizamos um *corpus* já etiquetado para treinamento do modelo desenvolvido.

Para treinamento da rede neural foi necessária a utilização de um *corpus* de sentenças etiquetadas, isto é, uma grande variedade de sentenças do português com as respectivas classificações morfossintáticas para cada palavra. O *CETENFolha-1.0_jan2014.cg.gz*, utilizado neste trabalho, está disponível em `https://linguateca.pt/`.

Neste trabalho foi implementada uma rede neural LSTM bidirecional com representações de palavras (*word embeddings*) pré-treinadas com o modelo GloVe [7] especificamente para o português, disponibilizadas pelo NILC [1]. A acurácia final desse modelo foi então comparada com outras duas implementações de etiquetadores previamente desenvolvidas pelo autor em outro trabalho. A primeira delas é um modelo basal simples que faz a contagem de palavras

[1] `http://nilc.icmc.usp.br/embeddings`

e etiquetas para prever a etiqueta mais provável para cada palavra. O segundo etiquetador é uma implementação do algoritmo de Viterbi. Os detalhes dessas duas implementações estão disponíveis em outro trabalho e por isso não serão repetidas aqui. A métrica utilizada para comparação e validação dos três modelos foi a acurácia, ou seja, a quantidade de palavras em que o modelo faz a classificação correta da etiqueta sobre a quantidade total de palavras analisadas.

As seções seguintes contemplam uma breve introdução à redes neurais e também à redes neurais recorrentes. O capítulo seguinte entra em detalhes sobre a implementação realizada, seguido pelos resultados obtidos e a conclusão.

2 Redes neurais

Conceitos iniciais

Redes neurais artificiais [3] são basicamente conjuntos de unidades de processamento conectadas, onde cada unidade é responsável por receber uma entrada, realizar um processamento e passar uma saída para a unidade seguinte. Cada unidade de processamento consiste em uma regra de propagação e uma função de ativação, conforme a figura 1.

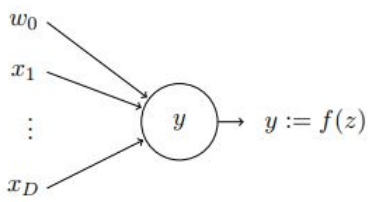

Figura 1: Unidade basica de processamento: $x_1...x_D$ representam as entradas de unidades anteriores, w_0 representa os pesos externos que multiplicam (ponderam) as entradas, f é a função de ativação aplicada à z, que é o produto entre $w0$ e $x_1...x_D$.

A rede neural feed-forward (FNN) é uma das primeiras arquiteturas de redes neurais desenvolvidas [3]. Ela é formada por uma camada de entrada (input layer), uma ou mais camadas escondidas (hidden layers) e uma camada de saída (output layer), conforme mostra a figura 2. Por camada, entende-se um conjunto interconectado de unidades de processamento que estão propagando valores da camada anterior para a camada seguinte, sempre passando por uma multiplicação de pesos (W) e de uma função de ativação.

A função de ativação define o valor final de saída da unidade. Essa função pode ser algo mais simples como uma delimitação de limiar:

$$f(z) = \begin{cases} 1, & \text{if } z >= 0. \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

ou mais complexa como uma função logistíca:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

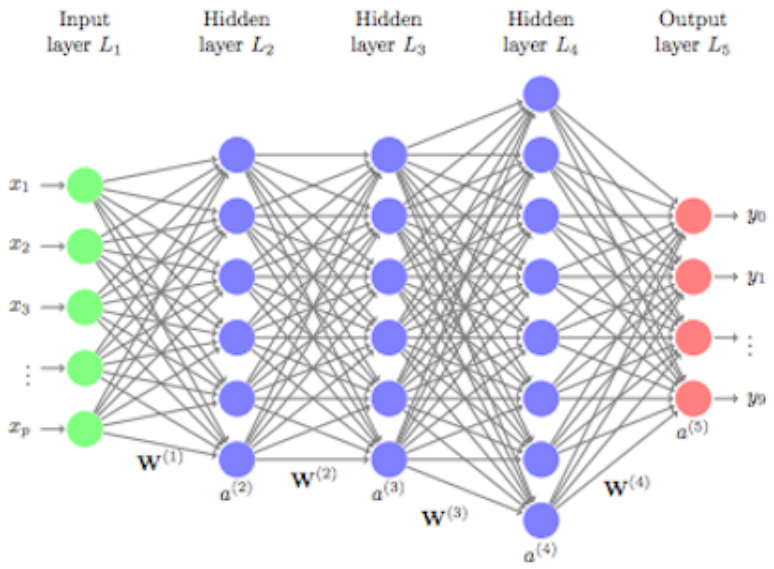

Figura 2: Exemplo de rede neural feed-forward: camadas de entrada, escondidas e de saída interconectadas, onde cada unidade está ligada à todas as unidades da camada seguinte.

Existem diversas outras funções de ativação tradicionalmente usadas em redes neurais profundas, como ReLu, LReLU, TanH, e diversas pesquisas apresentam comparações de performance entre elas, como mostrado em Dabal Pedamont [6].

Aprendizado

O aprendizado em uma rede neural ocorre através de um conceito chamado de *backpropagation*. O valor inicial das amostras de treinamento são recebidos pela camada de entrada, esses valores são propagados até a camada final (passando por multiplicações dos pesos e camadas de ativação) e assim temos, para cada unidade da camada final, um valor único, geralmente entre 0 e 1. Se pensarmos, por exemplo, em uma classificação binária, precisamos de somente um nó na camada final, que irá possuir um valor entre 0 e 1 (devido à aplicação da função de ativação logística), representando a classificação daquela amostra de treino. Esse valor então será comparado o seu valor real e esse erro L entre o valor real e o valor gerado pela rede será propagado de volta para todas as camadas da rede, fazendo com que os pesos de cada nó sejam atualizados proporcionalmente à sua contribuição para o erro.

Mais formalmente, temos uma função de erro L como a diferença entre o valor real e o valor gerado pela rede. A derivada parcial de L com relação aos pesos w de cada unidade retorna o gradiente, ou seja, a taxa de variação do erro com relação aos pesos. Em outras palavras, o gradiente diz o quanto devemos atualizar os pesos para que a essa função siga em direção ao seu ponto de mínimo local. De forma recursiva, calcula-se o gradiente para todos os nós de cada camada derivando parcialmente o seu valor de ativação com relação aos seus pesos, até chegar na camada de entrada. Feito isso, a próxima amostra de dados entra e o processo se repete continuamente, até a convergência do erro. Na prática, as atualizações realizadas nos pesos são feitas em pequenos passos, controlados por um parâmetro chamado de taxa de aprendizado (*learning rate*).

O processo completo de aprendizado de uma rede neural está bem representado na figura

3 [2]:

- **Passo 1:** Os pesos inicias de cada unidade são inicializados aleatoriamente;
- **Passo 2:** Os dados de entrada passam pela arquitetura da rede (multiplicação de pesos e funções de ativação) até ser calculado o valor da camada final (*output*);
- **Passo 3:** Cálculo da função de erro
- **Passo 4:** Cálculo da derivada parcial da função de erro com relação aos pesos da última camada
- **Passo 5:** Propagação do erro parcial para as camadas escondidas
- **Passo 6:** Atualização dos pesos seguindo um algoritmo de otimização e taxa de aprendizado definida

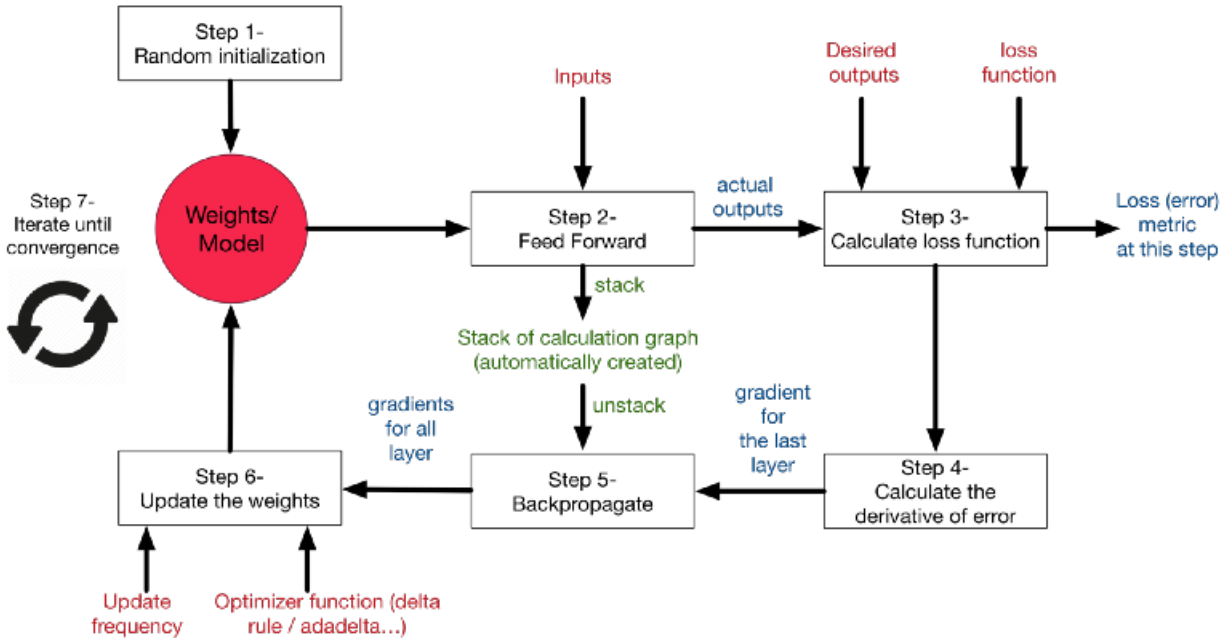

Figura 3: Processo de aprendizado da rede neural

3 Redes neurais recorrentes

3.1 Introdução

Redes neurais tradicionais são úteis para diversas tarefas de aprendizado de máquina. Entretanto, devido à sua própria estrutura, a rede não aprende nada sobre a ordem sequencial das amostras de entrada, a rede não guarda informação específica sobre a última ou sobre as últimas amostras vistas durante o treinamento. As redes neurais recorrentes[5] (RNNs) foram criadas justamente para tratar essa questão. Reconhecimento de fala, tradução de linguagem natural ou séries temporais são problemas onde a sequência dos dados é muito

[2] https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

importante e por isso precisam de um tratamento especial. Redes neurais recorrentes são arquiteturas complexas (geralmente com milhões de parâmetros) que conseguem armazenar informações sobre uma longa janela de contexto, criando uma "memória" sobre o que foi visto.

Entre as diversas arquiteturas de redes neurais recorrentes disponíveis e implementadas atualmente em pacotes de código aberto, foi escolhida para este trabalho a LSTM bidirecional. A arquitetura Long short-term memory [4] tem mostrado resultados superiores com relação à outras redes recorrentes em diversos modelos de linguagem, como etiquetagem, tradução de máquina e sumarização, principalmente pela sua característica de conseguir armazenar informação por longos contextos de dados. Além de mostrar bons resultados para modelos de linguagens, uma rede bidirecional leva em consideração as duas direções da sequência, e isso faz bastante sentido para um problema de etiquetagem morfossintática, onde tanto as palavra anteriores quanto as palavras seguintes podem trazer informação sobre a classe morfossintática da palavra atual.

Seguindo a ideia apresentada em Peilu Wang et. al. [8] e considerando a complexidade e o longo tempo de treinamento desse tipo de rede, foi decidido usar uma representação de palavras (*word embedding*) pré-treinada para o português. A representação de palavras são vetores n-dimensionais de valores reais que carregam alguma informação sintática e semântica sobre uma palavra. A rede é inicializada com essas representações no intuito de ganhar performance mais rapidamente, porém os pesos dessas representações não estão fixos, eles continuam a ser atualizados durante o treinamento da rede para o problema de etiquetagem. Foi utilizada uma representação pré-treinada do algoritmo GloVe [7], onde o treinamento é realizado sobre estatísticas de co-ocorrências de palavras em um corpus global.

A literatura apresenta diversas redes recorrentes para o problema de etiquetagem morfossintática. Por exemplo [1], abordagens mais simples como redes de uma camada, tratando palavras como vetores codificados *one-hot* e usando palavras anteriores e palavras posteriores para treinamento. Existem também abordagens não supervisionadas (sem a necessidade de uma base previamente etiquetada), como apresentado por Perez et. al. [1], onde foi treinada uma simples rede para prever a próxima palavra da sentença, e a representação da rede (os pesos) no momento após a leitura de determinada palavra era utilizada para agrupar as palavras em *clusters*, que seriam as etiquetas morfossintáticas.

3.2 Etiquetagem morfossintática

Dada uma sentença $w_1...w_n$ com etiquetas $y_1...y_n$, a rede é utilizada para prever a distribuição de probabilidade de etiquetas y para cada palavra w. A arquitetura da rede neural utilizada neste trabalho é composta por uma camada inicial de representação de palavras, uma camada bidirecional LSTM e por fim uma camada linear que passa por uma função de ativação *Softmax*, responsável por converter o resultado da camada final em uma distribuição de probabilidade. A figura 4 representa o fluxo dessa rede. Cada palavra é convertida em sua representação, nesse caso, a representação vetorial GloVe. Essa representação segue para a célula LSTM e cada célula envia duas saídas, uma representação final de *output* e uma representação intermediária que será enviada à célula seguinte. O *output* das camadas LSTM passam por uma camada linear e então pela função Softmax. A etiqueta com maior probabilidade nessa distribuição final é então a etiqueta classificada para essa palavra. Veja que a bidirecionalidade da rede faz com que com tenhamos dois fluxos em paralelo, duas camadas de LSTM, uma olhando para a sentença na ordem normal e outra olhando para a sentença

em ordem reversa. Essas duas representações (normal e reversa) são então concatenadas no fim do processo para que seja feita a previsão final (conversão da camada última camada linear pela função Softmax) e atualização dos pesos seguindo o algoritmo de *backpropagation*. Vale ressaltar que o fim do ciclo só ocorre ao final de toda a sentença. Cada célula é responsável por fazer a previsão de uma palavra e a concatenação final dos outputs de todas as previsões é passado para a camada linear. Também é possível passar somente o *output* da última palavra para a camada linear, uma vez que esse output final contém informações (representações) sobre toda a sentença. Essa é justamente a principal característica da LSTM, a dependência de longo termo, isto é, mesmo após passar por toda a sentença, o *output* da última célula ainda contém informações sobre todas as palavras, inclusive as palavras vistas pelas primeiras células.

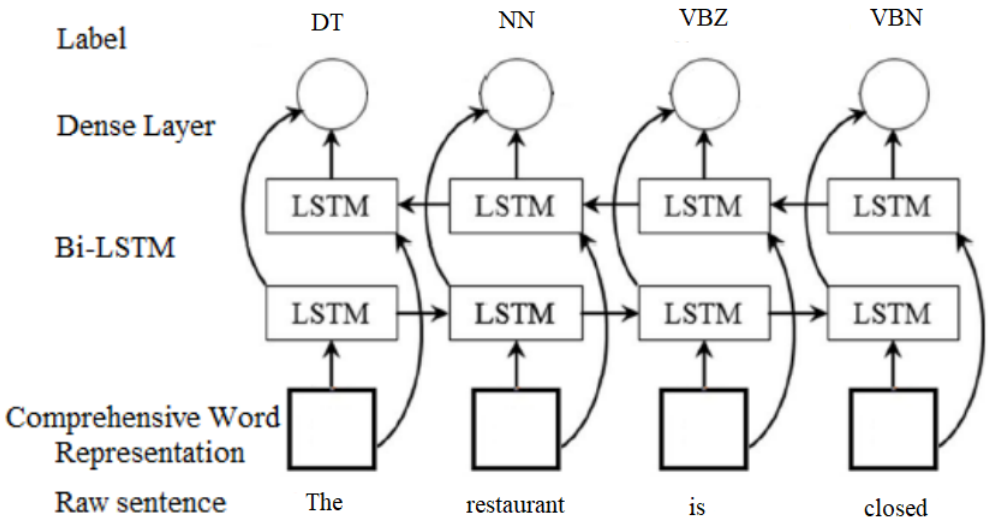

Figura 4: Arquitetura da LSTM Bidirecional para etiquetagem morfossintática

A grande complexidade dessa rede está na célula LSTM. A figura 5 ilustra o fluxo de dados dentro dessa estrutura. Falaremos aqui brevemente dos principais pontos dessa arquitetura. A linha superior C_t representa o estado da célula, é por onde passa principalmente toda a informação de uma célula para outra. O quanto de informação passa por esse estado é controlado pelos *gates*, estruturas compostas por uma camada linear de rede neural com ativação logística e uma operação de multiplicação de matrizes. A primeira caixa laranja é o chamado *forget gate*. Ele olha para o input x_t e para o output da célula anterior h_{t-1} e decide o quanto dessa informação deve ser "esquecida". O próximo passo é decidir qual informação nova será enviada para o estado da célula C_t. A segunda caixa laranja chamada de *input gate* decide quais valores serão atualizados enquanto a terceira caixa laranja "tanH" cria um vetor de novos valores candidatos à serem adicionados no estado C_t. No passo seguinte esses valores são combinados para decidir qual informação nova será atualizada no estado C_t, isto é, esse processo está definindo qual informação deve ser "lembrada". Como último passo, a última caixa laranja chamada de *output gate* é responsável por definir, através da aplicação de outra camada logística sobre a $tanh(C_t)$, o output h_t da célula para a célula seguinte. Os gates são controles de "memória" das células. São eles quem decidem o quanto uma

informação deve ser esquecida ou lembrada em cada célula. Vale ressaltar que os *gates* são também redes neurais que aprendem exatamente o que esquecer e o que lembrar de acordo com a resposta que a rede encontra no fim de cada ciclo de *backpropagation*.

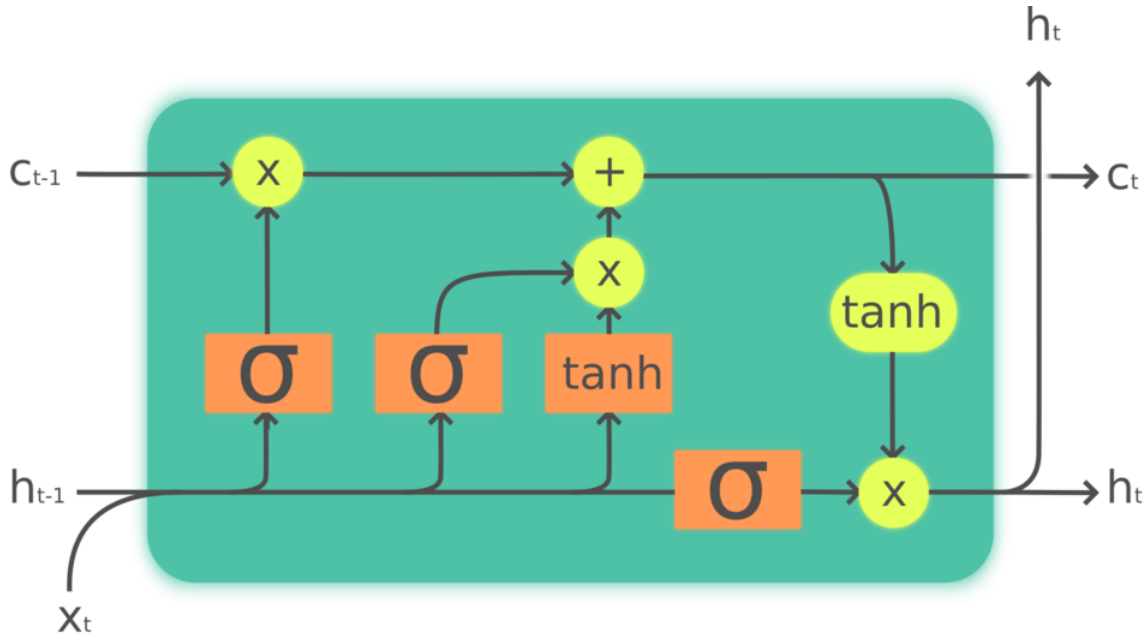

Figura 5: Diagrama de fluxo em uma LSTM: cada linha representa a transferência de um vetor de valores reais de um nó para outro, os círculos representam operações matriciais entre os vetores, linhas se unindo representam concatenação de vetores enquanto linhas se separando representam a duplicação de vetores para outputs diferentes. As caixas são os *gates*.

4 Implementação

O desenvolvimento deste trabalho foi realizado com a linguagem Python (v3.6) e o pacote *PyTorch* para implementação da rede neural. O sistema operacional utilizado para desenvolvimento foi o Windows 10, com 16GB de memória RAM e um processador Intel Core i7 2.20GHz. Entretanto, uma vez que a estrutura da rede foi bem definida e implementada, foi utilizada uma máquina virtual do serviço de cloud AWS[3] com acesso à placas de vídeo GPU, para possibilitar o treinamento mais rápido dos pesos da rede. Para esse treinamento, foi utilizada a máquina virtual modelo p2.xlarge, que possui uma GPU NVIDIA K80.

Pré-processamento das sentenças

A primeira parte da implementação consiste em ler o *corpus CETENFolha* e separá-lo em sentenças, além de identificar corretamente a etiqueta morfossintática de cada palavra das sentenças. Mais detalhes de como esse pré-processamento foi realizado encontram-se no artigo anterior à esse.

[3] https://aws.amazon.com/pt/blogs/machine-learning/get-started-with-deep-learning-using-the-aws-deep-learning-ami/

Seleção de etiquetas

Após a limpeza e separação das sentenças, o próximo passo foi a seleção de etiquetas. As etiquetas finais utilizadas estão descritas abaixo, e novamente, mais detalhes para a definições dessas etiquetas encontram-se no artigo anterior à esse. A única diferença aqui, é a adição da tag <PAD>, que é utilizada para completar sentenças de tamanhos diferentes durante o treinamento em lote de sentenças.

- '<PAD>', 'N', 'DET' ,'PRP', 'V', 'PROP', 'ADJ' , 'ADV', 'NUM', 'KC','SPEC','PERS', 'KS', 'IN', 'EC', 'PRON', '__PUNCT__'

Palavras raras

Durante a separação das bases para treino, validação e teste, palavras na base de treino com frequência total menor ou igual a 5 foram convertidas para a marcação __RARE__. De forma análoga, palavras na validação e no teste que nunca foram vistas na base de treino são convertidas para a marcação __RARE__. Dessa maneira, é possível simular a aparição de palavras pouco comuns na base de treino para que seja feita uma previsão para palavras não vistas nas bases de validação e teste.

Parâmetros da LSTM

A LSTM implementada nesse projeto foi definida com um tamanho de camada escondida de 200 unidades. A camada de entrada da LSTM é de tamanho 100, e foi previamente inicializada com as representações de palavras GloVe 100. A implementação da rede bidirecional no PyTorch é basicamente a alteração de um parâmetro na definição da rede juntamente com a duplicação da dimensão das camadas escondidas (com a rede bidirecional, temos o dobro de parâmetros à serem treinados). A etapa mais complexa da implementação é a definição de como a rede faz a passagem do *input* para a camada final (*feed-forward*), é um processo de multiplicação de diversas matrizes (*inputs* e pesos). Outro parâmetro que deve ser definido é a quantidade de camadas de LSTM. A figura 4 mostra, por exemplo, uma rede com uma única camada bidirecional LSTM, mas redes podem possuir n camadas até chegar numa camada linear final. A rede implementada aqui possui somente 1 camada bidirecional, assim como mostra a imagem.

- Camadas LSTM: 1 (bidirecional)
- Dimensão da camada escondida da LSTM: 200
- Dimensão da camada de representação das palavras: 100

Parâmetros do treinamento

Durante o treinamento da rede, é possível realizar o ciclo de *feed-forward* e *backpropagation* para cada sentença. Porém esse processo pode ser computacionalmente caro se o número de sentenças for muito alto e também pode interferir na performance final do modelo, uma vez que estaremos atualizando os pesos da rede com uma frequência muito maior. Por isso trabalhamos com treinamento em lote (*batch training*), onde passamos para a rede n sentenças, e o processo de *backpropagation* só ocorre após a avaliação dessas n sentenças.

Esse é o princípio do gradiente descendente estocástico. Entretanto, o processo de passar múltiplas sentenças pela rede de uma só vez é mais complexo, uma vez que todas as sentenças precisam possuir o mesmo tamanho. Nesse caso, concatenamos ao fim da sentença uma série de palavras <PAD>até que todas as sentenças do mesmo lote possuam o mesmo tamanho. Essa abordagem pode trazer perdas de performance, uma vez que agora a rede vai tentar prever também a palavra <PAD>, por isso, utilizamos as funções do *Pytorch* *torch.nn.utils.rnn.pack_padded_sequence* e *torch.nn.utils.rnn.pad_packed_sequence* para comprimir as sentenças removendo as tags <PAD>exclusivamente para a passagem nas células LSTM. Durante o treinamento foram realizados testes com diferentes tamanhos de lote 1, 2, 4, 8, e 16.

Outro parâmetro definido para o treinamento é taxa de aprendizagem (*learning rate*). Esse parâmetro define o quão fortemente os pesos são atualizados a cada iteração do *back-propagation*. Se esse valor estiver muito alto, os pesos podem ser atualizados mais do que o necessário em cada iteração e nunca convergir. Se for muito baixo, o erro pode demorar muito para chegar à um ponto de mínimo e também não é o ideal. Na prática, diferentes valores devem ser testados para essa taxa afim de encontrar uma taxa intermediária. Nesse caso, 0.1 foi o valor utilizado para esse parâmetro.

Por limitações computacionais, não foi possível realizar o treinamento da rede em toda a base. O tempo necessário para realizar o treinamento da rede é muito alto, dada a sua complexidade e quantidade de parâmetros (pesos). Após a realização de vários testes, foi utilizada uma amostra de 100.000 sentenças para realizar o treinamento da rede. Entretanto, como é mostrado nos resultados a seguir, mesmo utilizando aproximadamente só 6% da base (a base total possui 1.693.101 de sentenças) , conseguimos resultados melhores do que o algoritmo de Viterbi sendo treinado em toda a base. O treinamento final foi realizado em GPU na máquina virtual da AWS com as configurações descritas acima e por 3 epochs, isto é, o treinamento foi repetido 3 vezes por toda a base de treino. O tempo para treinamento dessas 3 epochs foi de 59 minutos e 11 segundos. Enquanto o treinamento era realizado, foram feitas avaliações periódicas em uma base de validação de 2.000 sentenças. A figura 6 mostra a evolução da acurácia na base de treino corrente e na base de validação durante a primeira epoch, enquanto a figura 7 mostra a continuação desse treinamento na terceira epoch. Percebe-se que o treinamento ainda poderia continuar por mais epochs ou até mesmo incluir mais dados, pois a acurácia na base de validação ainda está menor que na de treinamento. Entretanto, para os fins de experimentação o treinamento com 3 epochs foi suficiente.

Avaliação

No intuito de comparar os resultados da rede LSTM com a implementação anterior deste autor, desenvolvida através do algoritmo de Viterbi e redes de Markov, a avaliação foi realizada exatamente na mesma base de teste, utilizando-se das 5 pastas de teste de validações cruzadas. Isto é, o treinamento do modelo foi feito exclusivamente nas 100.000 sentenças de treino, independentemente da validação cruzada. Já a avaliação e os resultados aqui apresentados foram realizados nas mesmas amostras de validações cruzadas na qual também foram avaliados o modelo de Viterbi e o modelo basal de contagem de palavras. Mais detalhes de como essa divisão com validação cruzada foi realizada estão presentes no artigo anterior à este.

Conforme apresentado na introdução, a métrica utilizada para avaliação do modelo foi

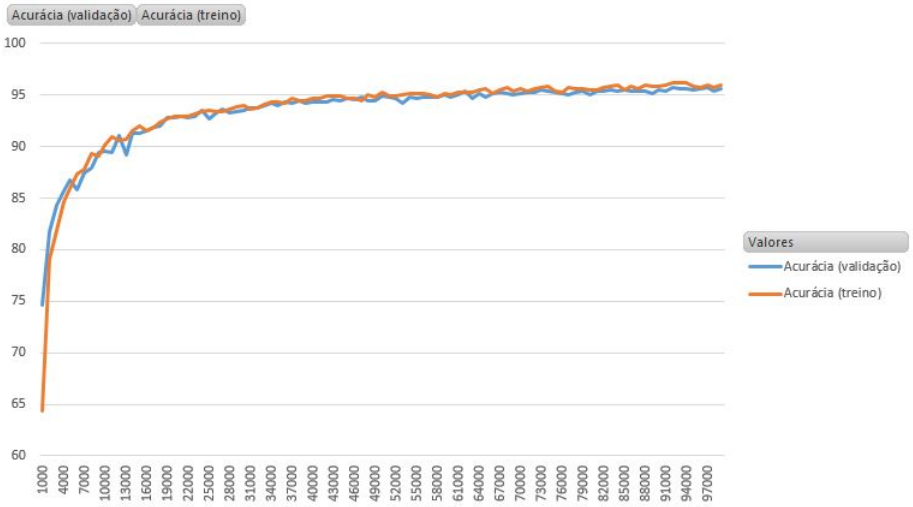

Figura 6: Acurácia na base de treino e validação durante a *epoch 1*

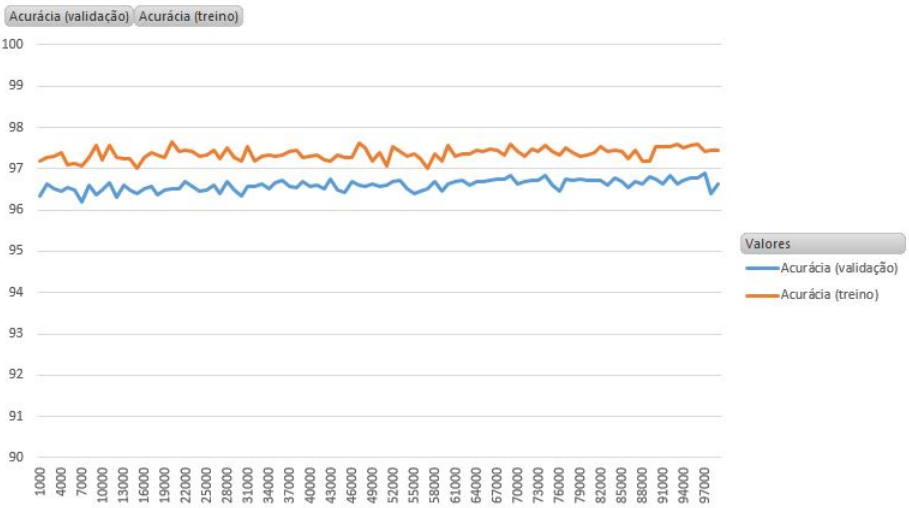

Figura 7: Acurácia na base de treino e validação durante a *epoch 3*

a acurácia. Foi calculado o número de palavras classificadas corretamente pela rede LSTM sobre o total de palavras.

5 Resultados

Apresentamos na tabela abaixo o resultado final comparativo dos dois modelos implementados anteriormente e também do novo modelo LSTM apresentado neste artigo. É possível perceber valores bem similares entre as diferentes rodadas de execução, o que indica boa consistência dos dados e dos modelos de forma geral, indicando que estão robustos à variações de palavras em diferentes partes do *corpus*.

Percebemos que o modelo LSTM superou a acurácia do modelo de Viterbi para todas as pastas da validação cruzada, chegando à acurácia média de 96.682%. Vemos que, utilizando

uma GPU, o tempo necessário para treinamento desse modelo foi de aproximadamente 1 hora. Por outro lado, o modelo de Viterbi possui um curto tempo de treinamento, pois faz basicamente contagem de palavras e n-gramas, porém, levou 25 horas para avaliar toda essa base de teste enquanto esse tempo de avaliação para a LSTM foi de 20 minutos somente.

Outro ponto a favor da LSTM é que, como mostrado na figura 7, ainda seria possível treinar a base por mais epochs sem perder acurácia na base de validação, pois vemos que as curvas ainda avançam paralelamente, ou seja, o treino podería continuar até que a acurácia de validação começasse a cair. Vale ressaltar que a LSTM foi treinada somente com 100.000 sentenças, e é esperado que adicionando mais sentenças teríamos uma melhora na performance final do modelo, entretanto não foi possível realizar esse teste por longas horas devido ao custo para utilização de máquina virtual com GPU na AWS.

Rodada	*Acertos LSTM*	*Acurácia LSTM*	*Acertos Viterbi*	*Acurácia Viterbi*	*Acertos Basal*	*Acurácia Basal*	*Total de palavras*
$k = 1$	5.317.961	96.64%	5.297.854	96.27%	5.204.619	94.58%	5.502.802
$k = 2$	5.325.037	96.83%	5.294.017	96.27%	5.201.878	94.59%	5.499.002
$k = 3$	5.338.313	96.65%	5.317.937	96.28%	5.224.428	94.59%	5.523.073
$k = 4$	5.333.106	96.65%	5.313.382	96.29%	5.219.220	94.58%	5.517.860
$k = 5$	5.313.108	96.64%	5.292.300	96.26%	5.199.382	94.57%	5.497.557
Média		**96.682%**		**96.275%**		**94.587%**	**27.540.294**

6 Conclusão

Foi apresentado nesse artigo o desenvolvimento de uma solução para o problema de etiquetamento morfossintático para sentenças do português. Foi desenvolvido um modelo de linguagem utilizando uma rede neural com arquitetura LSTM Bidirecional, um tipo de arquitetura que considera a ordem sequencial dos dados, o que faz total sentido para entender a classificação morfossintática das palavras. A performance desse modelo foi comparada com outros dois modelos desenvolvidos anteriormente, um modelo basal que faz a contagem de palavras e etiquetas e classifica palavras pela etiqueta mais provável e também o modelo seguindo a implementação do algoritmo de Viterbi que utiliza um modelo oculto de Markov de ordem 2 para encontrar a sequência de etiquetas mais provável para uma determinada sentença. O modelo LSTM superou a performance dos outros dois modelos, mesmo sendo treinado por um tempo limitado e com uma quantidade reduzida da base de treino. Fica comprovada aqui a boa performance de redes neurais recorrentes para a tarefa de etiquetagem morfossintática.

A implementação foi realizada em Python e a biblioteca Pytorch foi utilizada para implementação da rede neural. Ficam ainda várias possibilidades de melhorias para o modelo atual. Como informação de entrada de cada palavra, foi utilizada somente a representação de palavras (*word embedding* GloVe). Entretanto, vemos na litetura [2] a possibilidade de criar paralelamente outra rede LSTM que olha para cada palavra à nível de caracteres, isto é, uma representação de caracteres. Dessa forma, a aparição de letras específicas, como por exemplo,

letras acentuadas, letras maiúsculas, finalização de palavras com a letra "R" (forte indicação de um verbo no infinitivo), estariam sendo levadas em consideração. Outra possibilidade de melhoria para a arquitetura da rede seria trabalhar com mecanismos de atenção. Ao invés de utilizar o output completo da rede, é possível utilizar outra camada para aprender qual parte do output deve ser mais considerada.

Por fim, o desenvolvimento deste trabalho trouxe bastante aprendizado para o autor. Considero que as principais dificuldades dessa implementação estiveram relacionadas com a implementação da rede neural, mais especificamente, com o tratamento de dimensões de matrizes e como os pesos e dados de entrada se relacionam para passar pelo processo de *feed-forward* e *backpropagation*.

Referências

[1] Juan Antonio Perez-ortiz e Mikel L. Forcada. "Part-of-Speech Tagging with Recurrent Neural Networks". Em: (out. de 2001).

[2] Cıcero Nogueira Dos Santos e Bianca Zadrozny. "Learning Character-level Representations for Part-of-speech Tagging". Em: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, pp. II-1818–II-1826. URL: http://dl.acm.org/citation.cfm?id=3044805.3045095.

[3] Thomas Epelbaum. "Deep learning: Technical introduction". Em: (2017). URL: https://arxiv.org/abs/1709.01412.

[4] Sepp Hochreiter e Jürgen Schmidhuber. "Long Short-Term Memory". Em: *Neural Comput.* 9.8 (nov. de 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[5] Zachary Chase Lipton. "A Critical Review of Recurrent Neural Networks for Sequence Learning". Em: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019. URL: http://arxiv.org/abs/1506.00019.

[6] Dabal Pedamonti. "Comparison of non-linear activation functions for deep neural networks on MNIST classification task". Em: *CoRR* abs/1804.02763 (2018). arXiv: 1804.02763. URL: http://arxiv.org/abs/1804.02763.

[7] Jeffrey Pennington, Richard Socher e Christopher D. Manning. "GloVe: Global Vectors for Word Representation". Em: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[8] Peilu Wang et al. "Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network". Em: *CoRR* abs/1510.06168 (2015). arXiv: 1510.06168. URL: http://arxiv.org/abs/1510.06168.