EP 2: RNNs Bidirecionais, Overfitting, Underfitting

Entrega: 30/10/2020 (14h, antes da aula)

1 Motivação

O objetivo deste exercício é aumentar a familiarização dos alunos com redes neurais recorrentes bidirecionais, em arquiteturas LSTM e GRU e uma aplicação em análise de sentimentos. Além disso pretendemos abordar dois problemas muito comuns em aprendizado de máquina, a saber, o problema do subajuste (*underfitting*) e do sobreajuste (*overfitting*) e de técnicas para evitar o problema do sobreajuste, como a utilização de uma camada Dropout durante o treinamento e a divisão do córpus em três conjuntos: dados de treinamento, dados de validação e dados de teste.

Vamos começar por esses dois últimos problemas.

1.1 Subajuste e sobreajuste

O *subajuste* (*underfitting*) ocorre quando um modelo estatístico ou algoritmo de aprendizado de máquina não consegue capturar os padrões e tendências subjacentes aos dados. Em termos estatísticos, o subajuste ocorre quando o modelo ou o algoritmo mostra baixa variância, mas alto viés. Frequentemente, o subajuste é resultado de um modelo excessivamente simples, ou então é o resultado de um modelo muito rico que está sendo treinado com dados escassos.

Disto segue que a solução para o problema do subajuste passa por conseguir mais dados ou modificar o modelo para que ele capture melhor as nuances contidas nos dados.

O *sobreajuste* (*overfitting*) ocorre quando um modelo estatístico ou algoritmo de aprendizado de máquina captura o ruído dos dados. Intuitivamente, o sobreajuste ocorre quando o modelo ou algoritmo se ajusta aos dados bem demais, memorizando os dados de treinamento. Em termos estatísticos, o sobreajuste ocorre se o modelo ou algoritmo mostra baixo viés, mas alta variância. Em modelos estatísticos clássicos, o sobreajuste costuma ser resultado de um modelo excessivamente complicado. Uma forma de contornar este problema é a utilização de técnicas de *regularização*, que adiciona à função custo o valor dos parâmetros, o que tende a eliminar parâmetros de pouca importância. A *validação cruzada* (*k-fold cross validation*) produz o ajuste de vários modelos, em que testamos cada modelo em relação a uma parte reservada do conjunto de dados que não foi utilizada no treino do modelo em questão, para comparar suas precisões preditivas com estes dados de teste.

Outras vezes, o sobreajuste é resultado de um processo de treinamento que perdurou por tempo demais, devendo ter parado ainda quando o modelo possuía generalidade, antes de ter memorizado totalmente os dados de treinamento. Pode ser evitado ajustando-se vários modelos e usando a *validação*, um método a ser detalhado abaixo.

Tanto o sobreajuste quanto o subajuste levam a previsões ruins em novos conjuntos de dados.

1.2 Treinamento, validação e teste

Os dados usados para construir o modelo final geralmente vêm de vários conjuntos de dados (datasets). Em particular, três conjuntos de dados são comumente usados em diferentes estágios da criação do modelo: dados de treinamento, dados de validação e dados de teste.

O modelo é inicialmente ajustado em um conjunto de *dados de treinamento*, que é um conjunto de exemplos usados para ajustar o pesos do modelo. Periodicamente durante o treinamento, o modelo parcialmente treinado é usado para prever as respostas para as observações em um segundo conjunto de dados denominado *dados de validação*. Estes dados fornecem uma avaliação imparcial de um ajuste de modelo treinado até o momento, e podem ser usados para regularização para terminar antecipadamente o treinamento quando o erro (loss ou acurácia) no conjunto de dados de validação aumenta, pois isso é um sinal de ajuste excessivo do conjunto de dados de treinamento.

Decidir exatamente o ponto em que o processo de treinamento começou a sobreajustar é muito difícil, portanto na prática o que nós fazemos é armazenar o valor dos parâmetros do modelo parcial (*checkpointing*) e prosseguir com o treinamento. Quando houver certeza de que o sobreajuste está ocorrendo, retornamos aos parâmetros armazenados anteriores à ocorrência de sobreajuste. Este conjunto parâmetros sem sobreajuste serão considerados o modelo final.

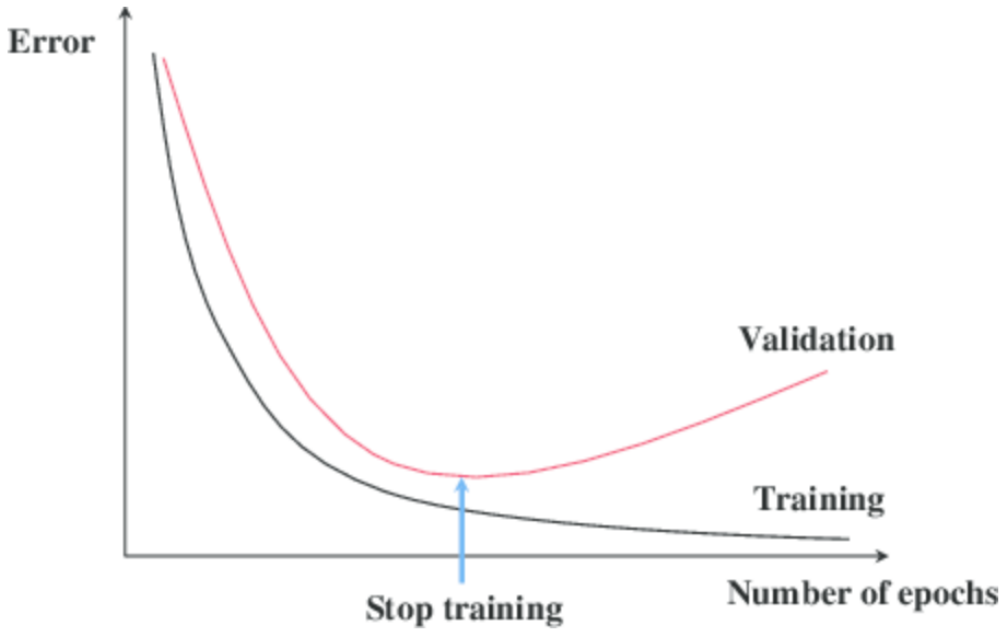

Figura 1: Grafo de erro do córpus de validação ao longo das épocas de treinamento, determina o ponto em que deve-se parar o treinamento para evitar sobreajuste.

A Figura 1 ilustra como o erro computado com os dados de validação permitem detectar o ponto em que o sobreajuste começa a ocorrer. A figura mostra que o erro nos dados de treinamento (linha preta) continua a cair ao longo de todo o processo de treinamento, no entanto existe um ponto em que o erro do córpus de validação (linha vermelha) deixa de cair e começa a subir; este erro pode ser medido como o valor de *loss* sobre os dados de validação, ou então pode ser medido como a perda de acurácia (ou o aumento de $1 - acc$, onde acc é acurácia) medida no córpus de validação.

Neste trabalho vocês deverão levantar um gráfico semelhante a este com os dados de treinamento e validação particionados a partir do dataset inicial.

Finalmente, o conjunto de *dados de teste* é um conjunto de dados usado para fornecer uma avaliação imparcial do modelo final. Os dados no conjunto de dados de teste não podem ser usados no treinamento ou na validação. O conjunto de dados deve ser particionado em treinamento, validação e teste antes do início do treinamento. Os dados de teste são usados uma única vez, para medir a acurácia oficial do modelo treinado.

1.3 Diluição/Dropout

No caso de redes neurais, técnicas de *diluição* (dropout) podem retardar o momento no treinamento em que o sobreajuste ocorre e desta forma aumentar a capacidade de generalizar de um modelo, utilizando o mesmo conjunto de dados de treino. O dropout é uma técnica de regularização que não altera diretamente a função de custo (loss), mas que visa reduzir a complexidade do modelo com o objetivo de evitar sobreajuste.[1]

Usando dropout, desativa-se aleatoriamente certas unidades (neurônios) em uma camada com uma certa probabilidade p de uma distribuição de Bernoulli (normalmente 50%, mas este é outro hiperparâmetro a ser ajustado). Então, se você definir metade das ativações de uma camada para zero, a rede neural não será capaz de utilizar um caminho específico em uma determinada passagem durante o treinamento. Como consequência, a rede neural aprenderá representações diferentes e redundantes; a rede não pode contar com a presença de neurônios específicos e da combinação (ou interação) deles. Outro bom efeito colateral é que o treinamento será mais rápido. O dropout só é aplicado durante o treinamento e você precisa redimensionar as ativações de neurônios restantes. Quando o treinamento tiver terminado, usa-se a rede completa para teste (ou seja, define-se a probabilidade de dropout como 0). O dropout pode ser utilizado como uma camada do Keras a ser adicionada à saíde uma camada à qual se deseja evitar sobreajuste, em geral uma rede recorrente.

[1] Ver o artigo que introduziu esta técnica: Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). *Dropout: A simple way to prevent neural networks from overfitting*. The Journal of Machine Learning Research, 15(1), pp. 1929–1958.

```
tf.keras.layers.Dropout(
        rate, noise_shape=None, seed=None, **kwargs
)
```

1.4 Batches (Lotes)

Este item não tem a ver diretamente com sobreajuste mas tem a ver com tempo e espaço usados no treinamento.

As redes neurais são treinadas usando gradiente descendente, onde a estimativa do erro usada para atualizar os pesos é calculada com base em um subconjunto do conjunto de dados de treinamento.

Um *batch* (lote) é um conjunto de exemplos do conjunto de dados de treinamento usado na estimativa do gradiente.

O tamanho do lote controla a precisão da estimativa do gradiente ao treinar redes neurais. Existe uma tensão entre o tamanho do lote e a velocidade e estabilidade do processo de aprendizagem.

Gradiente descendente em lote, estocástico e minibatch são os três tipos principais do algoritmo de aprendizagem.

- Gradiente descendente estocástico em Keras: batch de tamanho 1

```
...
model.fit(trainX, trainY, batch_size=1)
```

- Gradiente descendente em um único lote em Keras: batch de tamanho da entrada (não usar!!!)

```
...
model.fit(trainX, trainY, batch_size=len(trainX) )
```

- Gradiente descendente em minibatches em Keras: fixar o tamanho TAM do lote (default Keras, TAM= 32). Necessita gerar dados em lotes de tamanho TAM. Usar minibatches em gerar.

```
...
model.fit(trainX, trainY, batch_size=TAM )
```

Outros fatores determinam o tamanho do lote, como a quantidade disponível de memória e o tempo gasto no treinamento. O tamanho do lote é um híper-parâmetro, que em geral é uma potência de 2.

1.5 Acurácia

A *acurácia* é o a razão do número de previsões corretas sobre o numero total de exemplos.

Podemos ter a acurácia de treinamento (de pouco uso), acurácia de validação (que pode ser usada em vez do erro de validação para a determinação da época em que começa haver sobreajuste, e a acurácia de teste (que é a acurácia oficial do modelo).

2 Pré-processamento de dados

Vamos criar um **pipeline** que automaticamente extrai os dados de uma planilha csv, gera os corpos de Treinamento, validação e teste, cria os iteradores que vão compor os batches que serão alimentados na entrada do sistema, codificando as palavras de acordo com algum esquema de vetorialização (ou seja, esquema de transformação de uma palavra em um vetor de valores no espaço d-dimensional).

Nós utilizaremos o corpus de avaliações da B2W, e estaremos interessados em apenas duas colunas: `review_text`, que será a referida daqui para frente como `texto`; e `overall_rating`, Que será referido apenas como `rótulo`.

Este pipeline deve conter os seguintes passos:

filtro Filtrar todas as linhas cujo rótulo não é um valor numérico entre 0 e 5.

partilha Recebe o nome do arquivo csv contendo o dataset de entrada, e recebe também o nome do diretório de saída, e as proporções dos datasets de avaliação e de teste. A partir dos dados filtrados, altera aleatoriamente a sua ordem e gera as planilhas csv de treinamento, validação e teste. Note que essas planilhas só devem conter as colunas de texto e de rótulo. As proporções sugeridas são as seguintes (não é obrigatório, é só uma sugestão):

- Treinamento: 65%
- Validação: 10%
- Teste: 25%

Se a quantidade de dados for pequena, pode-se aumentar a quantidade de dados de treinamento, com proporções como 75-10-15, ou 80-10-10. É usual, mas não obrigatório, que o modelo seja salvo após o cálculo do erro de validação.

Dica: Não é obrigatório fazer, mas recomenda-se usar a biblioteca `pandas` com o extrator de dados de um DataFrame `iloc`; o embaralhador aleatório `numpy.random.shuffle()` para o qual podemos fixar uma semente; por fim utilize a função do pandas `to_csv` para escrever Data Frame num arquivo csv.

codifica Utilizar um codificador de palavras para vetor de dados de d-dimensional. Pode ser o word2vec que vocês implementaram, pode ser o pacote para compilado do Nilc, pode ser uma rede neural do tipo `Embedding`, a ser treinada com os dados de treinamento.

3 Treinamento

O treinamento se inicia verificando que o número de palavras da entrada não excede um valor máximo (`tammax`), que é um hiperparâmetro do modelo. As sentenças devem ser truncadas caso a entrada exceda este valor máximo. Supondo uma ordem aleatória das sentenças do córpus de treinamento, o conjunto de sentenças devem ser organizadas em batches (lotes) de tamanho fixo, onde o tamanho de cada lote de treinamento (*batch_size*) também é um hiperparâmetro. Tipicamente esses valores são potências de 2, e são limitados pelo tamanho da memória do modelo e, no caso de estar se usando uma GPU, pela quantidade de memória da GPU. Todas as sentenças no lote devem ter o mesmo tamanho, e portanto deve-se encontrar a maior sentença e completar as sentenças menores com caracteres de *padding* <PAD>.

Podemos limitar o tamanho do vocabulário em um número fixo de palavras mais frequentes, por exemplo 20.000, transformando as demais palavras em palavras desconhecidas. O primeiro passo do treinamento consiste no embedding das palavras num espaço vetorial de tamanho fixo. Neste passo você pode utilizar os valores calculados pelo programa do ep1, ou então os valores para calculados pelo NILC; uma terceira possibilidade é o uso de uma camada específica de embedding, mas nesse exercício Estamos dando preferência para o uso de algum mapeamento pré-calculado.

O treinamento se inicia verificando que o número de palavras da entrada não é sede um valor máximo (tammax), que é um hiperparâmetro do modelo. As sentenças devem ser truncadas caso a entrada exceda este valor máximo. Supondo uma ordem aleatória das sentenças do córpus de treinamento, as sentenças devem ser organizadas em batches (lotes) de tamanho fixo, onde o tamanho de cada lote de treinamento também é um hiperparâmetro. tipicamente esses valores são potências de 2, e são limitados pelo tamanho da memória do modelo, e no caso de estar se usando uma GPU, pela quantidade de memória da GPU. Todas as sentenças no lote devem ter o mesmo tamanho, e portanto deve-se encontrar a maior sentença e completar as sentenças menores com caracteres de padding <PAD>.

Podemos limitar o tamanho do vocabulário em um número fixo de palavras mais frequentes, por exemplo 20.000, transformando as demais palavras em palavras desconhecidas. O primeiro passo do treinamento consiste no embedding das palavras num espaço vetorial de tamanho fixo. Neste passo você pode utilizar os valores calculados pelo programa do ep1, ou então os valores para calculados pelo NILC; uma terceira possibilidade é o uso de uma camada específica de embedding, mas nesse exercício Estamos dando preferência para o uso de algum mapeamento pré-calculado.

Cada lote é submetido durante o treinamento a uma rede neural recorrente formada por elementos LSTM. Vamos experimentar dois casos. No primeiro caso utilizaremos um Encoder unidirecional e no segundo caso utilizaremos um Encoder bidirecional. Em ambos os casos, a saída do Encoder (o código) gerado após toda a entrada ser lida deve ser conectada em uma rede linear densa cujo número de saídas é igual ao número de classes que desejamos classificar a entrada.

Ao final desta rede solicitamos inserir uma camada de dropout para melhorar o desempenho no caso de sobre-ajuste. Vamos experimentar com três valores de porcentagem de caminhos eliminados no dropout: 0% (sem dropout), 25% e 50%.

Ou seja, teremos seis experimentos variando os parâmetros de rede uni/bi- direcional e os valores de dropout de treinamento.

4 Validação

Vamos rodar para cada experimento um treinamento que consiste de 50 a 100 épocas. O valor exato deve ser determinada pela detecção do ponto em que começa a ver sobre ajuste, ou seja se já houver aumento no erro do córpus de validação nas primeiras 50 épocas não é necessário continuar o treinamento.

A cada cinco épocas de treinamento, devemos avaliar o erro computado no córpus de validação. Recomenda-se computar também o valor da acurácia do Corpus de validação. Esses valores devem ser armazenados para gerar um gráfico como o da Figura 1 para cada um dos seis experimentos. Deve-se também armazenar o erro (loss) do córpus de treinamento e verificar que este sempre diminui.

É recomendado que se armazene os parâmetros do modelo a cada vez que se faça uma rotina de validação.

5 Teste

Ao final, uma vez que se detectou qual o melhor conjunto de parâmetros do modelo antes do treinamento começar a sobreajustar, deve-se carregar o melhor modelo treinado e executar uma vez o córpus de teste, medindo sua acurácia final. Não há nenhum interesse em se medir o erro do córpus de teste. Uma única tabela deve reunir o resultado dos seis testes finais dos seis modelos e concluir qual modelo apresenta a melhor acurácia.

Instruções entrega

Entregar um zip no e-Disciplinas (moodle) contendo os seguintes elementos:

1. Um diretório `src` com os seus arquivos em python3.
2. Um arquivo `README` com as instruções de pré-processamento, treinamento e teste
3. Um arquivo `relatorio-ep2.pdf` com os seguintes conteúdos:
   (a) Uma descrição das configurações e dos híper-parâmetros utilizados.
   (b) Os seis gráficos de validação gerados, apontando o número de épocas usados para gerar o modelo sem sobreajuste.
   (c) Uma tabela com as seis acurácias de teste.
   (d) Uma discussão dos seus resultados.

Não incluir os dados de entrada, basta apenas indicar no `README` onde os dados de entrada devem estar, ou como selecionar este arquivo na linha de comando de execução.