TÉCNICO LISBOA

Aprendizagem Profunda

MEEC

Homework 2

Autores:

André Alexandre Costa Santos - 96152 andresantos1.alf@gmail.com
João Bernardo Vieira Pinto dos Santos - 96237 joaobsantos2001@tecnico.ulisboa.pt

Grupo 3

2022/2023 – 1º Semestre, P2

Conteúdo

1 Introdução

Neste *homework* foi proposto o desenvolvimento de redes neuronais mais complexas, com um maior ênfase em questões teóricas e abordando um desenvolvimento a um nível mais elevado comparativamente com o trabalho anterior.

2 Questão 1

Questão 1.1. a) Dimensão de z

Para calcular o tamanho de z, podemos recorrer à seguinte formula:

$$\textbf{\textit{Output width}} = \frac{\textbf{\textit{input width}} - \textbf{\textit{kernel width}} + 2 \times \textbf{\textit{padding width}}}{\textbf{\textit{stride}}} + 1$$

Figura 1: Expressão utilizada para calcular a largura de z

Para o cálculo da altura, utiliza-se uma forma equivalente, substituindo a largura por altura (*width* por *height*).

Substituindo os valores do problema na equação apresentada, obtem-se:

$$\begin{aligned} &\frac{W - N + 2 \times 0}{1} + 1 \\ &= W - N + 1 \end{aligned} \tag{1}$$

Conclui-se assim que a *width* de z é de *W-N+1*. O cálculo da altura é muito semelhante:

$$\begin{aligned} &\frac{H - M + 2 \times 0}{1} + 1 \\ &= H - M + 1 \end{aligned} \tag{2}$$

Concluindo-se assim que a *height* de z é de *H-M+1*

Questão 1.1. b) Expressão de elementos de M

Dado que a dimensão de x é de H x W e a dimensão de z é de *(H - M + 1)* x *(W - N + 1)*, as dimensões de *x'* e de *z'* serão respetivamente *HW* x *1* e *H'M'* x *1*, com *H' = H - M + 1* e *W' = W - N + 1*.

Como tal, para que *z'=Mx'*, a dimensão de M terá de ser *H'W'* x *HW*.

Cada elemento da matriz $\boldsymbol{M}$ pode tomar ou o valor de zero ou de um determinado peso W_{ab}, sendo este um elemento da matriz $\boldsymbol{W}$ cujas coordenadas são dadas por a e b. Caso a seja maior que M, ou seja, maior que a altura do filtro, ou caso j seja maior que N, ou seja, maior que a largura do filtro, obtemos um zero na posição *(a,b)*.

$$x_{ij}^{\ell} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} y_{(i+a)(j+b)^*}^{\ell-1} \tag{3}$$

Sendo que a fórmula da convolução é dada pela equação 3, podemos afirmar que a posição de um peso W_{ab} é dada pela seguinte equação em função de *(i,j)*.

$$\begin{aligned} \mathrm{W}_{ab} \Rightarrow a &= j - (b-1)H - (i-1) - \left(\left\lceil \frac{i}{H'} \right\rceil - 1 \right)(M-1). \\ b &= \left\lceil \frac{j}{H} \right\rceil - \left\lceil \frac{i}{H'} \right\rceil + 1 \end{aligned} \tag{4}$$

Ambas as expressões podem ser decompostas em parcelas. Uma vez que estamos a percorrer as linhas da matriz $\boldsymbol{M}$ podemos dizer que 'a' será dependente de j.

Cada linha da matriz $\boldsymbol{M}$ é composta por uma sequência de pesos, seguida de uma interrupção por uma sequência de zeros, sendo esta seguida por uma nova sequência de pesos. O início desta sequência de pesos pode ser traduzida em função de 'a' pela parcela (b-1)H.

Este padrão repete-se na linha seguinte, mas começa uma posição à frente da linha anterior, ou seja, existe um zero na linha antes dos *weights*, sendo assim representado pela parcela *(i-1)*.

A última parcela representa um *shift* maior do que o descrito anteriormente. Este ocorre após a linha H', ou seja, na linha $H'+1$. Este *shift* ocorre um total de *M-1* vezes, sendo M o tamanho vertical do *kernel*.

Quanto à segunda equação, existem apenas duas parcelas. A primeira destas é análoga à ultima parcela da equação a, mas desta vez referente à segunda coordenada do peso da matriz $\boldsymbol{W}$.

Por último, a parcela $\left\lceil \frac{i}{H'} \right\rceil + 1$ é dada pela alteração da posição de cada elemento de W_{ab} ao longo da linha, estando dependente da vetorização da matriz x.

Aqui podemos observar algumas linhas do caso genérico da matriz $\boldsymbol{W}$:

$$\begin{pmatrix} W_{11} & \cdots & W_{M1} & 0 & \cdots & 0 & W_{12} & \cdots & W_{M2} & 0 & \cdots & 0 \\ 0 & W_{11} & \cdots & W_{M1} & 0 & \cdots & 0 & W_{12} & \cdots & W_{M2} & 0 & \cdots \\ \vdots & & \ddots & & & & & & & & & \vdots \\ 0 & \cdots & 0 & W_{11} & \cdots & W_{M1} & 0 & \cdots & 0 & W_{12} & \cdots & W_{M2} \end{pmatrix}$$

Questão 1.1. c) Número de parâmetros na rede

É necessário calcular o número de parâmetros da *convolutional layer* e da *output layer*, dado que a *max pooling* layer não apresenta qualquer parâmetros livre.

Quanto à *convolutional layer*, o tamanho do seu resultado é dado por:

$$dim(h1) = (H - M + 1) \times (W - N + 1) \tag{5}$$

No caso do output layer, é necessário saber o número de channels à saída da *max pooling layer*. Dado que esta apenas realiza uma operação de *down-sampling*, este valor é igual ao utilizado anteriormente. Multiplica-se então este valor pelo número de classes à saída, valor dado no enunciado, e obtem-se o resultado seguinte:

$$dim(MaxPooling) = \frac{dim(h1)}{2} \tag{6}$$

Depois de aplicar o (flatten) obtemos:

$$dim(h) = dim(MaxPooling) \times 3 \tag{7}$$

É possível então concluir que existem ((H-M+1) x (W-N+1))/2 x 3 + MN parâmetros livres na rede.

Uma *fully connected layer* tem, normalmente, muitos mais parâmetros do que um *convolutional layer*, dado que na *fully connected layer*, cada output está ligado a todos os inputs, enquanto que na *convolutional layer*, cada output está ligado apenas a uma região de inputs. O número de parâmetros será então H x W x *h*.

Questão 1.2.

Dadas as seguintes matrizes Q, K e V:

$$\boldsymbol{Q} = \boldsymbol{X'W}_Q = vect(x) \cdot \begin{bmatrix} 1 \end{bmatrix} = vect(x)$$
$$\boldsymbol{K} = \boldsymbol{X'W}_K = vect(x) \cdot \begin{bmatrix} 1 \end{bmatrix} = vect(x)$$
$$\boldsymbol{V} = \boldsymbol{X'W}_V = vect(x) \cdot \begin{bmatrix} 1 \end{bmatrix} = vect(x)$$

Podemos calcular as *attention probablities* $\boldsymbol{P}$ através da seguinte fórmula:

$$\boldsymbol{P} = \text{softmax}\left(\frac{\boldsymbol{QK}^\top}{\sqrt{1}}\right) = \text{softmax}\left(\frac{\boldsymbol{vect(x)} \cdot \boldsymbol{vect(x)}^\top}{\sqrt{1}}\right)$$

Sendo então o output dado por:

$$\boldsymbol{Z} = \boldsymbol{PV} = \text{softmax}\left(\frac{\boldsymbol{vect(x)} \cdot \boldsymbol{vect(x)}^\top}{\sqrt{1}}\right) \cdot \boldsymbol{vect(x)}$$

3 Questão 2

Questão 2.1

Uma fully connected network precisa de *biases* e *weights* diferentes para cada ligação entre neurónios de diferentes níveis, sendo assim o número de pesos proporcional ao número de neurónios da camada inicial multiplicado pelo número de neurónios da camada final. Numa CNN, os *weights* e *biases* são partilhados por todos os neurónios, reduzindo assim o número de parâmetros e facilitando a generalização do modelo.

Questão 2.2

Uma CNN terá um desempenho melhor do que uma *Fully-Connected Network* devido à sua arquitetura. Esta permite que o tempo de execução seja significativamente menor do que um FCN porque existem menos parâmetros. No entanto, este atributo não é a única vantagem, uma vez que a CNN tem em conta toda a imagem, estando cada um dos seus pixeis interligados com os "vizinhos". Assim existe uma invariância de translação quando são aplicados os filtros resultando no reconhecimento de objetos iguais em diferentes partes da imagem com, ou não, diferentes escalas. Podemos assim concluir que, utilizar uma CNN é menos dispendioso

temporalmente, além de também produzir melhores resultados para classificação de imagens ou padrões como letras ou números.

Questão 2.3

Uma vez que um dos maiores atributos da CNN é a sua capacidade de reconhecer facilmente padrões a partir da deslocação de *Kernels* ao longo das imagens, se não existir nenhuma conexão entre elementos espaciais, o resultado final será bastante pior. Deste modo, a deteção de certos elementos em imagens utilizando *Fully-Connected Networks*, ou outro tipo de redes neuronais, torna-se mais preciso do que utilizando uma CNN, devido ao facto de que as FCN conseguem detetar padrões, embora que menos rigorosamente do que uma CNN normal.

Questão 2.4

Após implementar a *network* como descrito, obtiveram-se os seguintes gráficos:

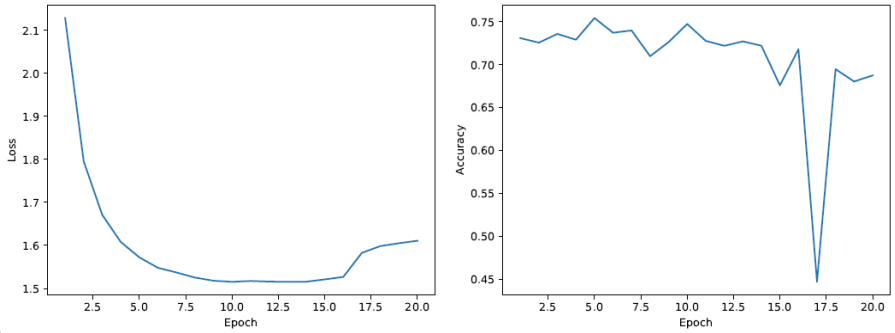

Figura 2: Gráficos de *loss* e *accuracy* para *learning rate = 0.01*

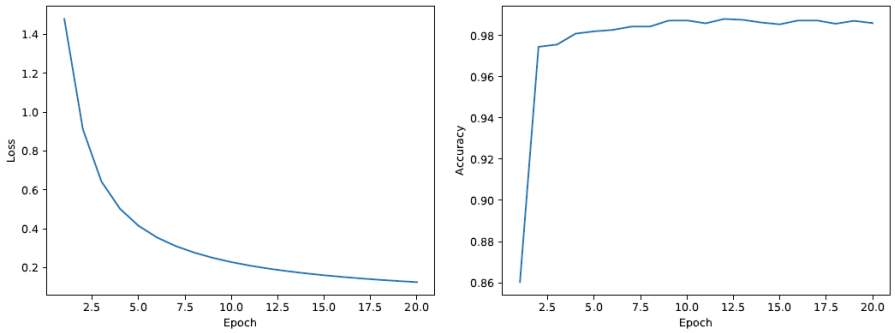

Figura 3: Gráficos de *loss* e *accuracy* para *learning rate = 0.0005*

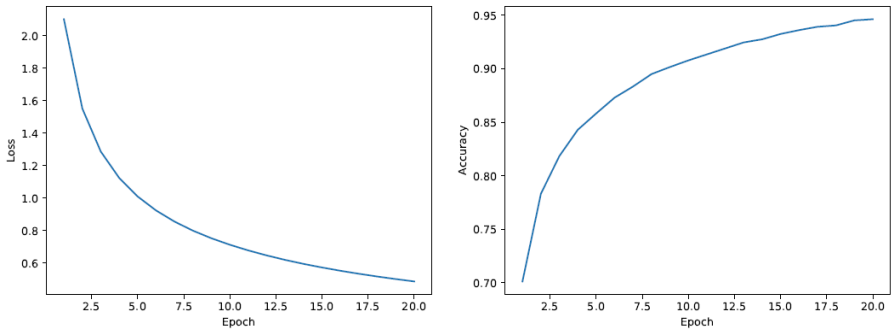

Figura 4: Gráficos de *loss* e *accuracy* para *learning rate = 0.00001*

Como é possível observar, o melhor resultado obtem-se para *learning rate = 0.0005*, tendo obtido 0.9523 para a métrica *Final test accuracy*.

Questão 2.5

Para a seguinte imagem:

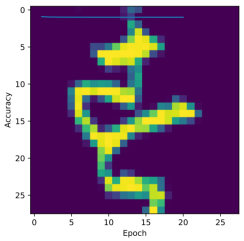

Figura 5: Imagem Original

Obtiveram-se os seguintes activation maps:

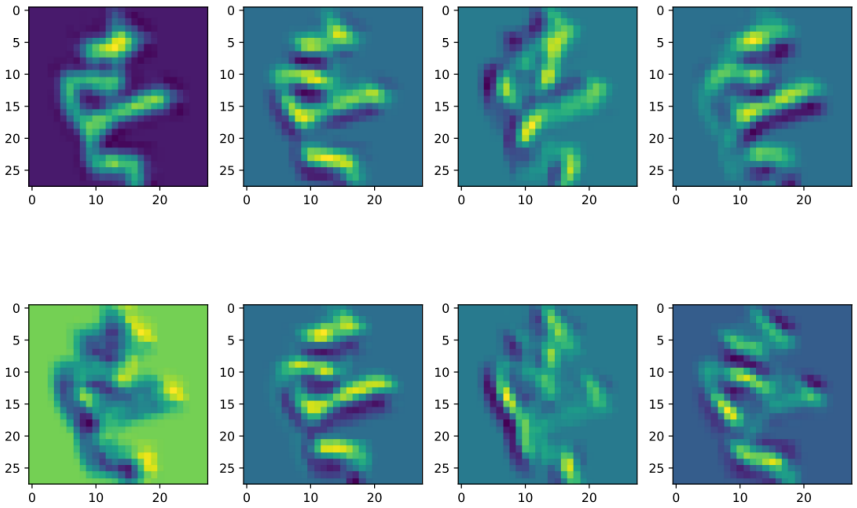

Figura 6: Activation Maps

É possível ver que se encontram realçados os contornos do símbolo, sendo assim possível concluir que cada uma das imagens corresponde a um *output channel* da primeira camada de convolução.

4 Questão 3

Nesta questão foi proposto o desenvolvimento de uma rede capaz de traduzir texto de uma lingua para outra.

Questão 3.1. a)

Procedeu-se então à implementação do algoritmo pedido, de modo a realizar a tradução de texto de espanhol para inglês. Foi necessário realizar a implementação do método *forward* para ambos o *encoder* e o *decoder*.

O método *forward* do *encoder* têm a seguinte estrutura:

Algorithm 1 Forward do Encoder

$embedded \leftarrow embedding(source)$
$embedded \leftarrow dropout(embedded)$
$packed \leftarrow packed(embedded,\ lengths,\ batch\_first = True,\ enforce\_sorted = False)$
$encoder\_outputs, hidden\_n \leftarrow lstm(packed)$
$enc\_output, \_ \leftarrow padded(encoder\_ouputs,\ batch\_first = True)$
$final\_hidden \leftarrow reshape(hidden\_n)$

Paralelamente, aqui temos o *forward* do *decoder*:

Algorithm 2 Forward do Decoder

if $dec\_state[0].shape[0] == 2$ **then**
    $dec\_state = reshape\_state(dec\_state)$
if $target.size(1) > 1$ **then**
    $dec\_state = reshape\_state(dec\_state)$
$embedded \leftarrow embedding(target)$
$embedded \leftarrow dropout(embedded)$
$outputs,\ dec\_state \leftarrow lstm(embedded,\ dec\_state)$
if $attention\ is\ Not\ None$ **then**
    $outputs = attention(outputs,\ encoder\_outputs\ , src\_lengths\ )$
$outputs \leftarrow dropout(outputs)$

É possível então observar o resultado no seguinte gráfico, que representa o *error rate* sem a utilização de atenção:

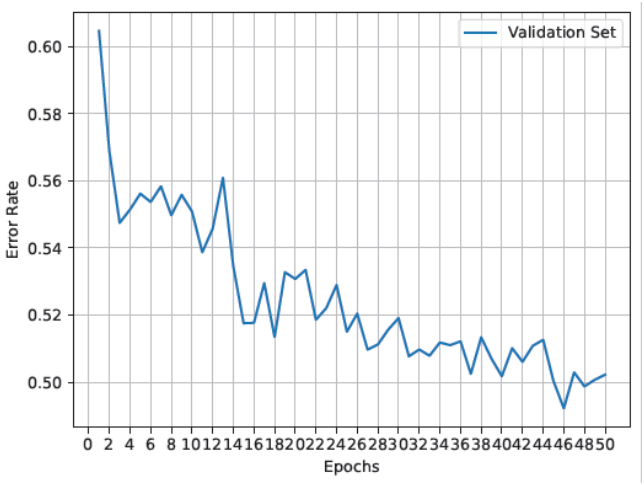

Figura 7: Error rate sem a utilização de *Attention*

Obtemos assim os seguintes resultados:

Attention	Test Error Rate	Final Validation Error Rate
False	50.23%	50.18%

Questão 3.1. b)

De seguida, prosseguiu-se para a implementação de um mecanismo de atenção. Para tal, foi necessário implementar o método *forward* da classe *Attention* para que este realizasse as operações descritas no enunciado.

Foi então calculado o score através da seguinte expressão:

$$S_i = W_q^\top \cdot q \cdot h_i \tag{8}$$

Operação implementada através da seguinte linha de código:

```
scores = torch.bmm((self.linear_in(query)) , torch.transpose(encoder_outputs,1,2))
```

Prossegue-se então para o *masking*, que é realizado da seguinte forma:

```
src_seq_mask = ~self.sequence_mask(src_lengths)

scores.masked_fill_(src_seq_mask.unsqueeze(1), -float('Inf'))
```

Podemos então prosseguir para o cálculo da probabilidade de atenção, realizado o *softmax* do valor obtido acima:

```
prob = torch.softmax(scores,dim=2)
```

É então realizado o cálculo do *context vector* e, finalmente, o *output* da attention layer. Estas operações são realizadas da seguinte forma:

```
context = torch.bmm(prob, encoder_outputs)

attn_out = torch.tanh(self.linear_out(torch.cat((context, query), dim=2)))
```

Obtem-se então o seguinte gráfico para o *error rate*:

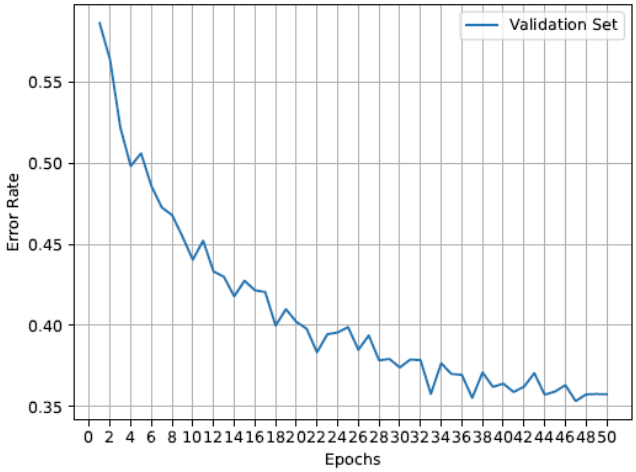

Figura 8: Error rate utilizando *Attention*

É possível então ver que, utilizando um mecanismo de atenção, se obtém um error rate muito mais baixo do que numa situação em que este não está presente. Obtiveram-se os seguintes valores:

Attention	Test Error Rate	Final Validation Error Rate
True	38.73%	35.75%

Questão 3.1 c)

Para melhorar os resultados obtidos, podemos: Aumentar o tamanho do *dataset* de treino, dado que deste modo é possível ao modelo aprender mais sobre as entradas e respetivas saídas, levando assim a uma tradução mais precisa, ou alterar os valores de *batch size* e número de layers, tentando deste modo encontrar um conjunto de valores que leve a um resultado melhor.

5 Conclusão

Ao longo do desenvolvimento deste *homework* foi possível aprofundar conhecimentos teóricos, bem como conhecimentos sobre a utilização de *Pytorch* e várias funções de *deep learning*. Foi necessária a implementação de um mecânismo de atenção, que permitiu compreender mais profundamente o funcionamento deste. Surgiram algumas dúvidas, nomeadamente em questões teóricas e de demonstração, mas foram esclarecidas através das aulas e consulta dos *slides* teóricos.

6 Contribuição dos elementos do grupo

Cada um dos elementos do grupo apresentou uma prestação equalitária, estando ambos os membros presentes tanto no desnvolvimento do código como na composição do relatório. As perguntas referentes à componente teórica foram divididas pelos membros do grupo de modo a realizar estas de forma mais rápida. Dado que surgiram algumas dúvidas, parte destas perguntas foram resolvidas em grupo.