Universidade Estadual de Campinas
MS960 - Projeto 2

Tradução automática de datas usando atenção

Caique Oliveira Alves da Silva - RA: 168023
Gabriel Cavalcanti de Arruda - RA: 247101
Nicolas Toledo de Camargo - RA: 242524

Dezembro
2023

1 Introdução

Desde tempos antigos, a barreira da comunicação tem sido um desafio persistente na sociedade. Devido às diferenças nos estilos de fala, escrita e cultura, diversos povos enfrentaram dificuldades para estabelecer acordos ou se comunicar entre si. Nesse contexto, a tradução se destaca como um elemento crucial para a comunicação global, desempenhando um papel fundamental na eliminação de barreiras linguísticas e na promoção da interação entre diferentes culturas.

A importância da tradução tornou-se ainda mais evidente durante a pandemia, quando as pessoas passaram mais tempo em casa e, como resultado, buscaram consumir uma variedade maior de conteúdos. A barreira linguística tornou-se um desafio significativo, destacando a necessidade de soluções eficazes nesse cenário.

Essa relevância da tradução abrange desde a disseminação de informações até a facilitação da colaboração internacional em setores diversos, como comércio, pesquisa e comunicação. Em um mundo globalizado e interconectado, a capacidade de superar as barreiras linguísticas é crucial para o progresso e a compreensão mútua.

Para enfrentar esse desafio, os métodos de tradução automática evoluíram ao longo do tempo. Inicialmente baseados em regras e dicionários, esses métodos eram limitados pela extensão das regras gramaticais e pelo vocabulário disponível nos dicionários da época. A introdução da tradução estatística representou um avanço, aproveitando grandes conjuntos de dados bilíngues para aprender padrões de tradução de maneira mais dinâmica.

Com o advento do *deep learning*, surgiram abordagens mais avançadas, como as Redes Neurais Recorrentes (RNN) e modelos Seq2Seq, que incorporam mecanismos de atenção para capturar relações complexas entre palavras de diferentes idiomas. A revolucionária arquitetura dos Transformers impulsionou ainda mais essa evolução, permitindo o processamento eficiente de sequências longas. Modelos como o "Transformer"do Google, BERT e GPT representam marcos significativos nessa trajetória.

Esses avanços tecnológicos resultaram em modelos pré-treinados capazes de compreender contextos complexos e nuances linguísticas, elevando consideravelmente a precisão e fluidez da tradução automática nos dias atuais. Essa evolução contínua reflete o compromisso em superar desafios linguísticos e promover uma comunicação mais eficaz em um mundo diversificado e interconectado.

Neste projeto, buscamos explorar uma arquitetura mais simples com o propósito de aprofundar nosso entendimento nas abordagens empregadas na tradução automática utilizando mecanismos de atenção. O nosso objetivo é realizar a conversão de datas, como por exemplo, "17th of October, 2023"ou "Tuesday, October 17, 2023", para um formato padronizado e mais acessível aos computadores, seguindo o padrão YYYY-MM-DD, o qual no primeiro exemplo citado seria "2023-10-17".

2 Modelo de Atenção

2.1 Aspectos teóricos e práticos

A tradução automática de datas usando atenção se baseia em Redes Neurais Recorrentes (RNNs) e modelos baseados em transformadores. Esses sistemas são fundamentais para compreender e gerar sequências, algo crítico ao lidar com datas em diferentes formatos. No âmbito teórico, destaca-se o mecanismo de atenção, permitindo aos modelos focarem em partes especificas das datas, melhorando significativamente a precisão das traduções.

Efetivamente ocorre que o mecanismo de atenção não traduz a data de uma só vez, mas opera em etapas, focando em partes específicas a cada passo. Por exemplo, ao traduzir "04/12/2023", ele pode inicialmente focar no dia "04", ajustando as ponderações para essa parte e, em seguida, concentrar-se no mês "12"em passos subsequentes. Essa abordagem dinâmica permite uma tradução mais precisa e contextual, já que o modelo atribui atenção diferenciada a diferentes partes da data durante o processo de tradução.

A implementação prática desse conceito abrange funcionalidades desde a conversão de caracteres para *one-hot vectors* até a implementação de rede de atenção para geração de contextos. O processo lida com os dados caractere por caractere, garantindo um processamento adequado mesmo em casos de entrada com tamanhos variados. Detalhes cruciais incluem a definição de funções como *softmax*, utilizada para identificar os pesos de atenção que compõem o mapa de atenção e a escolha de otimizadores como o Adam, combinado com a *crossetropy* categórica como função de perda.

A estrutura do modelo é definida para lidar com entradas e saídas de tamanhos fixos, com a aplicação de *padding* quando necessário. Essa abordagem permite experimentação com diferentes parâmetros e ajutes para aprimorar o desempenho do modelo, capacitando uma implementação prática da tradução automática de datas usando atenção.

As LSTMs (Long Short-Term Memory) são uma variação das RNNs, projetadas para lidar com sequências de dados e capturar dependências temporais complexas. São especialmente úteis em tarefas de tradução automática de datas, onde a compreensão do contexto temporal é crucial. A LSTM unidirecional é útil quando apenas o contexto passado é relevante, onde as informações anteriores podem ser suficientes para a conversão correta. Por outro lado, a LSTMS bidirecional é apropriada em situações em que o contexto passado e futuro são necessários, especialmente quando a ordem das informações da data podem variar.

Neste projeto vamos focar em utilizar um encoder LSTM bidirecional e um decoder LSTM unidirecional, com a comunicação entre eles intermediada por um mecanismo de atenção.

2.2 Arquitetura

A primeira rede do modelo é uma rede LSTM bidirecional que recebe t' *inputs* $x^{<t'>}$ de *one-hot vectors* de tamanho T_x, e a ultima é uma rede LSTM unidirecional com t *outputs* $y^{<t>}$ de *one-hot vectors* de tamanho T_y.

A primeira rede LSTM possui n_a unidades de ativação por bloco e produz as ativações $\overrightarrow{a}$ (ativação *forward*) e $\overleftarrow{a}$ (ativação *backward*), que são concatenadas para termos apenas a em cada passo temporal. Estas ativações serão utilizadas para o cálculo dos pesos de atenção $\alpha^{<t,t'>}$ de cada $a^{<t'>}$ para cada $y^{<t>}$.

A rede de saída gera as ativações chamadas de 'estados' $s^{<t>}$ e os $y^{<t>}$, escolhidos por uma

softmax. Cada bloco recebe o estado anterior e contexto, $s^{<t-1>}$ e $c^{<t>}$. O contexto $c^{<t>}$ é um produto interno dos pesos de atenção $\alpha^{<t,t'>}$ e as ativações $a^{<t'>}$.

Agora para o mecanismo de atenção em si, esta parte que é a responsável pelos pesos $\alpha^{<t,t'>}$. O mecanismo recebe o estado anterior $s^{<t-1>}$ e as ativações $a^{<t'>}$ (ativação de todos os t'), a seguir, $s^{<t-1>}$ e cada $a^{<t'>}$ passa por uma pequena rede totalmente conectada, com pesos também treinados, gerando t' 'energias' $e^{<t,t'>}$. Finalmente, cada $\alpha^{<t,t'>}$ é tido como uma *softmax* levando em conta $e^{<t,t'>}$ e a soma de todas as energias.

Confira diagramas da arquitetura no Apêndice A.

2.3 Implementação

Para essa implementação, empregou-se Python com as bibliotecas Numpy, Matplotlib, Pickle, e Keras. Foram elaborados dois arquivos: «implementacao.py» e «main.py». Os arquivos do projeto estão em [3].

Nos foi concedido alguns arquivos para auxiliar a implementação, são eles: um arquivo *dataset* com uma lista de tuplas de pares de datas em formato humano e datas em formato de máquina, um arquivo de dicionário mapeando caracteres das datas em formato humano para índices inteiros, um arquivo de dicionário mapeando caracteres das datas em formato de máquina para índices inteiros, um arquivo de dicionário mapeando os índices do mapeamento de máquina de volta para os caracteres, um arquivo de função que converte uma *string* para uma lista de inteiros usando o dicionário de mapeamento, um arquivo de função que desenha um mapa de atenção, e um arquivo com pesos pré-treinados para serem usados em testes.

O arquivo «implementacao.py» contém a implementação das seguintes funcionalidades:

- Converter os caracteres para *one-hot vectors*;
- Função *softmax*;
- Mecanismos compartilhados globalmente, como a rede totalmente conectada que gera as energias e ferramentas operadoras de tensores como concatenação, produto etc;
- A rede de atenção que gera os contextos;
- O modelo completo;
- Treinamento do modelo;
- Teste de acurácia;
- Além disso, transferimos as função fornecidas para cá também para termos todas as funções em um só arquivo.

Já o arquivo «main.py» contém o carregamento dos dados e utilização das funções pré-definidas para treinar modelos e fazer testes. Os parâmetros possíveis para alterar neste arquivo são: tamanho de entrada/saída, número de unidades de ativação pré/pós mecanismo de atenção, número de épocas de treino, número de exemplos de treino e teste, pesos carregados para o modelo e exemplos para o mapa de atenção.

Alguns detalhes adicionais da implementação:

- É processado caractere por caractere, ao invés de palavras inteiras;

- Os tamanhos da entrada e saída são fixos e, quando a entrada não tiver o mesmo tamanho da pré-definição, é utilizado *padding*;
- Foi definida a função *softmax* para poder dar o nome à camada dos pesos de atenção, que são usados no mapa de atenção;
- O otimizador do modelo é Adam e a *loss* é a *crossentropy* categórica;
- A inicialização da ativação da rede LSTM de saída (s) e o contexto (c) é com zeros;
- Uma parte dos dados é utilizada para treino e outra para teste;

3 Resultados e Discussão

3.1 Seleção e Avaliação de Modelos

Inicialmente, dividiu-se o *dataset*, contendo 10.000 exemplos rotulados, de modo que 99% foram utilizados para o treino do modelo e 1% para o teste de acurácia. A abordagem adotada visou maximizar a quantidade de exemplos para treinamento, mantendo uma quantidade mínima para testes.

Não houve necessidade de embaralhar os exemplos para evitar um viés de seleção, pois eles já estavam dispostos aleatoriamente no *dataset*. Portanto tomou-se os 9.900 primeiros dados para o conjunto de treino, e os 100 restantes para o conjunto de teste.

Com relação à escolha dos hiper-parâmetros, o modelo foi treinado por 50 épocas com o conjunto de treino, para 6 configurações diferentes de unidades de pré-ativação e pós-ativação, que foram comparadas com base em sua acurácia, medida com o conjunto de teste. Os resultados obtidos estão na Tabela 1.

Tabela 1: Modelos treinados por 50 épocas

Unidades de pré-ativação:	**Unidades de pós-ativação:**	**Acurácia:**
16	16	80%
16	32	81%
32	16	81%
32	32	93%
32	64	99%
64	32	99%

Os resultados mostraram que, conforme esperado teoricamente, dentro do intervalo testado, à medida em que o número de unidades de pré-ativação e de pós-ativação foi sendo dobrado, as acurácias aumentaram, ao custo de um maior tempo gasto com o treinamento.

Além disso, a influência destes dois hiper-parâmetros no resultado final é muito similar, visto que as acurácias obtidas foram as mesmas ao se inverter o número de unidades de pré-ativação e de pós-ativação.

Como as duas últimas configurações tiveram a mesma performance (99%), definiu-se arbitrariamente a configuração com 32 unidades de pré-ativação e 64 unidades de pós-ativação como sendo o modelo principal.

Posteriormente, para avaliar a influência do número de épocas na acurácia, treinou-se do zero outros modelos com a mesma configuração do modelo principal, porém por diferentes épocas: 1, 10, 20, 30, 40 e 50. Em seguida mediu-se as suas respectivas acurácias no conjunto de teste, e os resultados obtidos estão no gráfico da Figura 1.

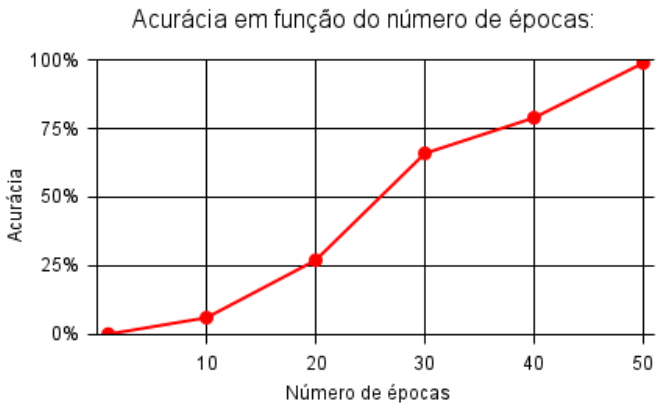


Figura 1: Gráfico da acurácia em função do número de épocas

Por meio deste gráfico, é possível observar que, novamente, conforme esperado teoricamente, dentro do intervalo testado, à medida em que o número de épocas aumentou, o modelo melhorou sua performance, ao custo de um maior tempo gasto com o treinamento.

Todos os modelos testados estão salvos na pasta *models* dos arquivos do projeto [3]. Não houve necessidade de treinar e testar modelos com um número ainda maior de épocas ou de unidades de ativação, pois com os valores utilizados já foi possível obter uma acurácia de 99%.

3.2 Análise das Traduções dos Modelos

Conforme visto na seção 3.1, o modelo principal (50 épocas, 32 unidades de pré-ativação, 64 unidades de pós-ativação) obteve uma acurácia de 99% no conjunto de teste. Como este conjunto possui 100 exemplos, o modelo principal errou apenas um deles, que pode ser visualizado na Figura 2, retirada do *output* do programa responsável pela testagem:

```
1/1 [==============================] - 0s 28ms/step
Original: 31 12 86

Predito: 1986-12-11

Correto: 1985-12-31
```

Figura 2: Exemplo com tradução incorreta

Com base na imagem, a data deste exemplo é expressa como “31 12 86”, portanto deveria ser convertida para a forma “1986-12-31”. Entretanto, curiosamente, para este exemplo, tanto o rótulo no *datset* quanto a predição do modelo principal estão incorretos, e por motivos diferentes: o valor do ano e do dia do mês, respectivamente.

Para se certificar de que não houve nenhum caso em que tanto o rótulo quanto a predição cometeram o mesmo erro, sendo contabilizado como um acerto, realizou-se uma inspeção manual no conjunto de teste, na qual constatou-se que aquele era de fato o único caso de rótulo incorreto no conjunto de testes. Portanto, a acurácia do modelo neste conjunto é realmente de 99%.

Já no conjunto de treinamento, é possível que exemplos com rótulo incorreto tenham sido considerados, mas mesmo assumindo esta hipótese, a influência de tais exemplos teria sido desprezível, visto que não impediram o modelo de generalizar muito bem, a ponto de obter uma acurácia de

99% no conjunto de teste.

O modelo principal performou melhor do que o modelo pré-treinado fornecido nos arquivos do projeto (*modelo.h5*), que também possui 32 unidades de pré-ativação e 64 unidades de pós-ativação, mas que obteve uma acurácia de 92%[1] no conjunto de teste.

Não se sabe exatamente como havia sido treinado este modelo, portanto não é possível ter certeza dos motivos por trás dessa diferença, supondo que ela se manteria em conjuntos de teste maiores, com milhares de exemplos. É provável que o modelo pré-treinado tenha tido um treinamento com menos épocas ou com um conjunto de treino que não generaliza tão bem quanto o que foi utilizado no modelo principal, devido à qualidade ou quantidade dos dados.

Uma visão geral dos resultados está sintetizada na Tabela 2, que compara o modelo principal, o modelo pré-treinado e os demais modelos do gráfico da Figura 1, para 2 exemplos do conjunto de teste que o modelo principal acertou e o modelo pré-treinado errou.

Tabela 2: Tradução de datas em diferentes modelos

Modelo:	**Acurácia:**	**19 dec 2020:**	**2 apr 2008:**
1 época	0%	1900——0	1000—-00
10 épocas	6%	2002-12-12	2009-04-29
20 épocas	27%	2022-12-19	2008-04-20
30 épocas	66%	2022-12-19	2008-04-02
40 épocas	79%	2000-12-19	2008-04-02
Pré-Treinado	92%	2022-12-19	2008-04-22
Principal	99%	2020-12-19	2008-04-02

Como pode-se observar, a partir de 10 épocas os modelos pararam de cometer erros grosseiros e chegaram muito perto da data esperada, na maioria das vezes errando apenas o dia, o mês ou o ano. Isto demonstra a eficiência deste algoritmo para este tipo de problema.

Para uma maior compreensão do aprendizado destes modelos, a Figura 3 apresenta o mapa de atenção do modelo principal e do modelo pré-treinado para um exemplo escolhido arbitrariamente.

Em resumo, cada coordenada de um mapa de atenção está associada a um caractere do *input* (eixo horizontal), um caractere do *output* (eixo vertical) e uma cor, de modo que quanto mais escuro é um ponto do mapa, pode-se interpretar que maior é a influência do caractere de *input* no caractere de *output* daquela coordenada.

[1] Este já é o valor de acurácia corrigido, levando em consideração que o modelo acertou o exemplo rotulado incorretamente. O mesmo se aplica para os valores de acurácia da Tabela 2.

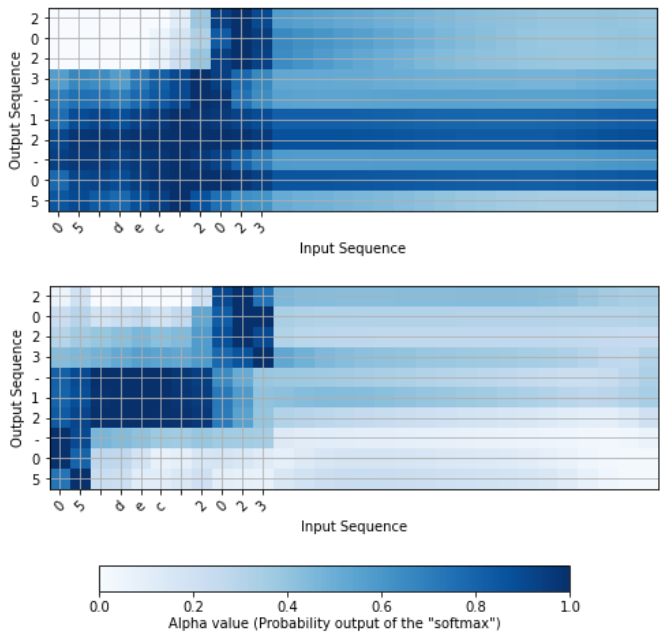


Figura 3: Exemplo 1 - Mapas de atenção do modelo principal (em cima) e do modelo pré-treinado (embaixo)

Pode-se observar que, embora ambos os modelos tenham traduzido corretamente a data "05 dec 2023", seus mapas de atenção são bem diferentes. Uma vantagem do modelo principal (mapa superior) é que o *output* do ano recebeu influência praticamente nula dos *inputs* do dia e do mês, ainda menor do que em comparação ao modelo pré-treinado.

Por outro lado, o modelo pré-treinado se saiu melhor no fato do *output* do dia ter recebido uma influência praticamente nula dos *inputs* do mês e do ano, e em geral seu mapa de atenção teve zonas de influências bem mais bem definidas, sendo mais próximo do que seria esperado por um modelo 100% preciso. Dessa forma, é curioso que ele tenha tido uma performance pior no conjunto de teste.

Uma falha de ambos os modelos é que o primeiro dígito do *input* do ano não está sendo devidamente considerado no *output* do ano. Provavelmente, isso ocorre pois nos dados de treinamento, com base nos últimos três dígitos do ano, já era possível determinar o primeiro dígito: por exemplo, toda vez, ou na maioria das vezes, que o ano terminava em 990, o exemplo de treinamento se referia ao ano 1990, de modo que foi desnecessário para os modelos aprenderem a considerar o primeiro dígito. Com base nesta hipótese, foi criado um exemplo artificialmente com o intuito de "enganar"ambos os modelos, conforme a Figura 4.

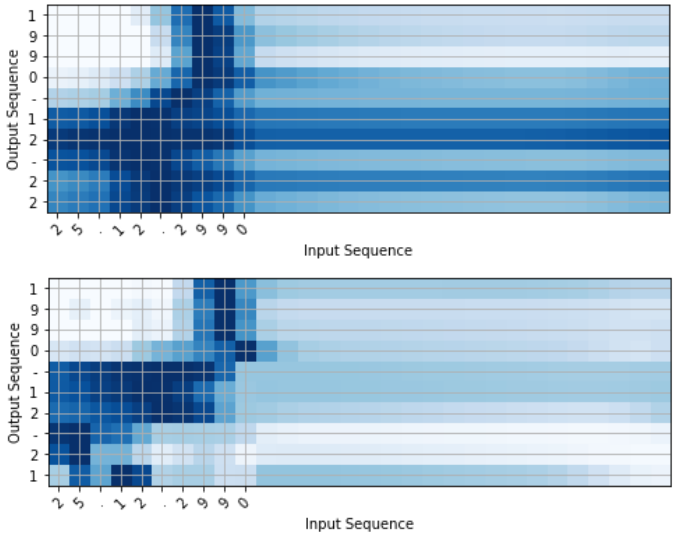


Figura 4: Exemplo 2 - Mapas de atenção do modelo principal (em cima) e do modelo pré-treinado (embaixo)

Neste exemplo ("25.12.2990"), não apenas ambos os modelos erraram o primeiro dígito do ano, confirmando a hipótese anterior, como também foi possível observar como os modelos se comportam quando o *input* do mês é dado na forma numérica: embora este *input* tenha influenciado corretamente no *output* do mês, também gerou uma forte influência indevida no *output* do dia, levando ambos os modelos a cometerem um segundo erro.

4 Conclusão

Com base na discussão realizada na seção 3, pode-se concluir que o objetivo do projeto foi atingido com sucesso, visto que nosso modelo apresentou uma excelente acurácia (99%), superando o modelo pré-treinado.

Embora 99% seja geralmente um valor muito raro de acurácia, não é tão incomum neste caso onde o problema modelado não é muito complexo (uma vez que envolve apenas a conversão de datas) e o conjunto de teste utilizado é relativamente pequeno, de modo que cada exemplo correto contribui com 1% de acurácia.

Isto posto, como possíveis melhorias para futuras implementações e análises posteriores, sugerimos:

1) Realizar o treinamento de outro modelo com outro *dataset* que contenha datas em um intervalo mais amplo de datas, de modo que o algoritmo generalize bem mesmo em casos extremos (exemplo: ano 3999).

2) Realizar o treinamento do modelo dividindo o *dataset* de outras maneiras, como 90% para treino e 10% para teste, ou 80% para treino e 20% para teste, e verificar se ainda assim a sua acurácia no conjunto de teste é superior à do modelo pré-treinado.

3) Realizar o treinamento do modelo com um número ainda maior de unidades (ex: 128 de pré-ativação e de pós-ativação), por um número ainda maior de épocas (ex: 100) e com um *dataset* ainda maior, e então verificar se isso torna as zonas de influência dos mapas de atenção mais bem definidas, como seria esperado em um modelo ideal.

Referências

[1] João Florindo - *Semana 16 - Modelos de Atenção*. Disponível em: <https://drive.google.com/drive/folders/1T1Y8RJ50OQ2hm9CRZlX4zuZfkcTs_4Ej?usp=drive_link>. Acesso em: Dezembro de 2023.

[2] Bahdanau et al. - *Neural machine translation by jointly learning to align and translate. 2014*

[3] *Arquivos do projeto*. Disponível em: <https://drive.google.com/drive/folders/1T1Y8RJ50OQ2hm9CRZlX4zuZfkcTs_4Ej?usp=drive_link>. Acesso em: Dezembro de 2023.

A Diagramas da arquitetura

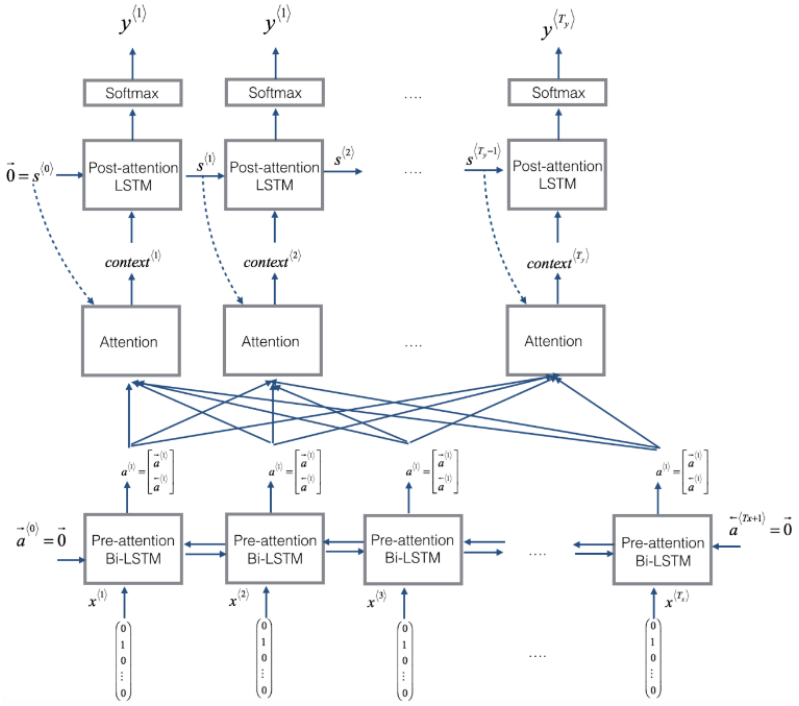
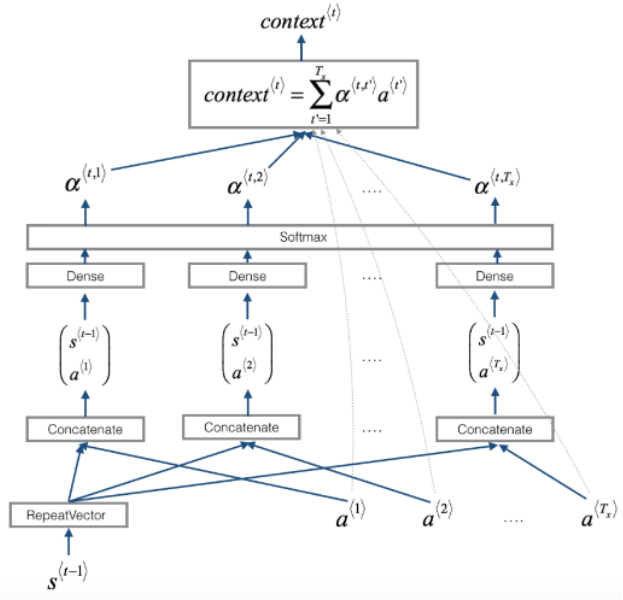
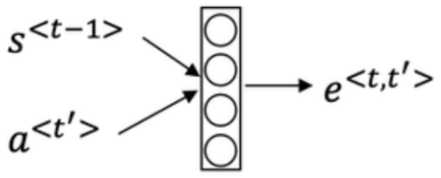


Figura 5: Diagrama geral da rede. [1]



(a) Mecanismo geral de atenção.

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$



(b) Obtenção dos pesos de atenção.

Figura 6: Diagrama do mecanismo de atenção. [1]