Redes Neurais Artificiais

Redes Recorrentes - Variações

INFORMAÇÃO,

TECNOLOGIA

& INOVAÇÃO

Gate Recurrent Units (GRU)

A principal diferença entre RNNs convencionais (vanilla) e GRUs é que podemos aplicar portões no estado escondido. Assim temos mecanismos que nos possibilitam atualizar um estado escondido e resetar um estado escondido. Isso é útil para várias situações, como por exemplo:

- Uma observação prévia pode conter informação altamente significativa para prever todas as observações futuras. Nesse caso gostaríamos de armazenar a informação em algum tipo de célula de memória

- Pode haver situações em que tokens não carregam nenhuma informação pertinente. Nesse caso, gostaríamos de um mecanismo para “pular” esses tokens

- Pode haver situações em que há uma interrupção lógica entre partes de uma sequência. Nesse caso gostaríamos de resetar a representação do estado interno

- Uma GRU contém portas (portões ou gates) que permitem a realização dessas operações com os estados da rede

Gate Recurrent Units (GRU)

Reset Gate e Update Gate – Portões de Reinicialização e Atualização

- Um reset gate nos permite controlar quanto do estado anterior ainda queremos lembrar
- Um update gate nos permite controlar o quanto de um novo estado é apenas uma cópia do estado anterior
- Nas entradas para os reset e update gates de uma GRU, considera-se a entrada da iteração atual e o estado escondido da iteração anterior
- As saídas dos dois portões são dadas por duas camadas totalmente conectadas com função de ativação sigmoid

Gate Recurrent Units (GRU)

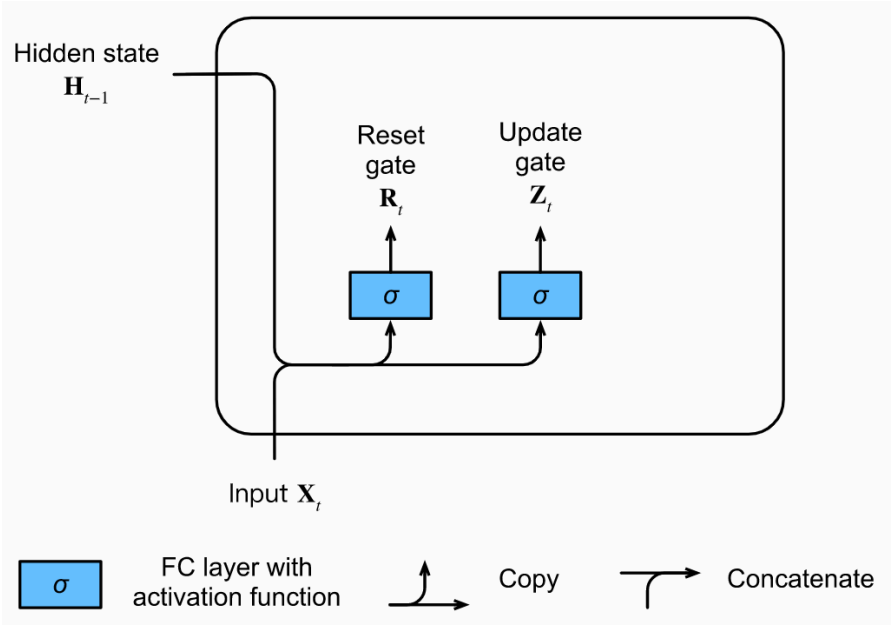
Reset e update gates de uma GRU

- Para uma iteração t, entrada $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ (n: número de exemplo, d: número de entradas), e estado escondido na iteração anterior $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$ (h: número de unidades escondidas), os gates reset $\mathbf{R}_t \in \mathbb{R}^{n \times h}$ e update $\mathbf{Z}_t \in \mathbb{R}^{n \times h}$ são calculados como:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r)$$

$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z)$$

- $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d \times h}$ e $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h \times h}$ são os pesos e $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1 \times h}$ são os bias



Gate Recurrent Units (GRU)

Vamos agora integrar o reset state $\mathbf{R}_t$ no mecanismo regular de atualização de estado da rede. Dessa forma, obtemos um estado escondido candidato $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$. Ele é considerado candidato porque ainda não incorporamos a ação do update gate

$$\tilde{\mathbf{H}}_t = tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$$

- $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ e $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ são os pesos e $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ é o bias. O símbolo $\odot$ representa o operador de multiplicação ponto a ponto (elementwise product)

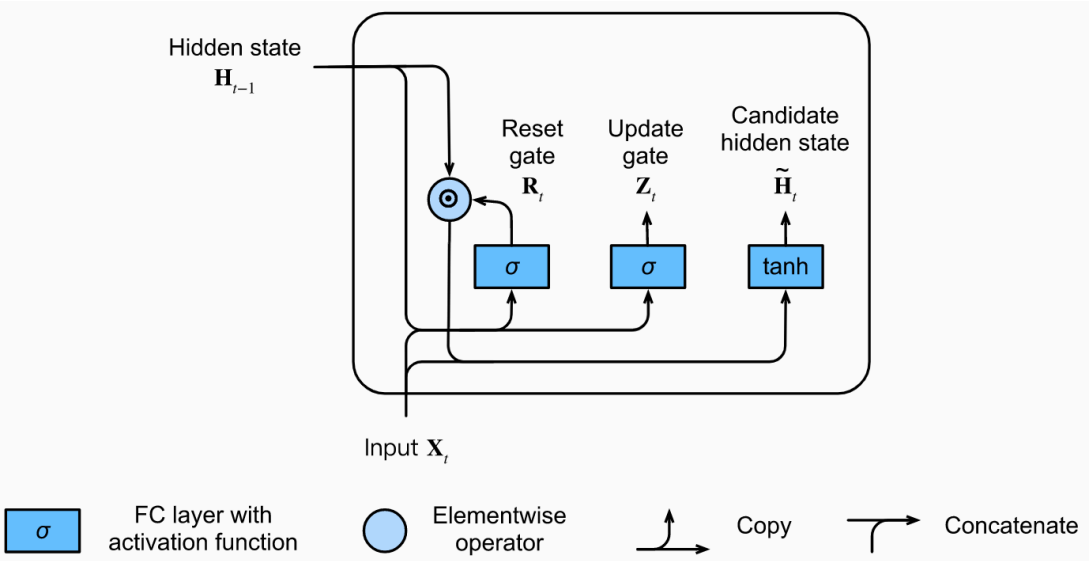
- A função de ativação tangente hiperbólica garante saídas no intervalo (-1,1)

Gate Recurrent Units (GRU)

A multiplicação ponto a ponto entre $\mathbf{R}_t$ e $\mathbf{H}_{t-1}$ serve para reduzir a influência do estado anterior no estado atual da rede

- Sempre que as entradas vindas do reset gate $\mathbf{R}_t$ são próximas de 1, o estado escondido candidato é resultado de uma RNN convencional (Vanilla)

- Sempre que todas as entradas vindas do reset gate $\mathbf{R}_t$ são próximas de 0, o estado escondido candidato é o resultado de uma MLP tendo $\mathbf{X}_t$ como entrada



Gate Recurrent Units (GRU)

O próximo passo agora é incorporar o efeito do update gate $\mathbf{Z}_t$.

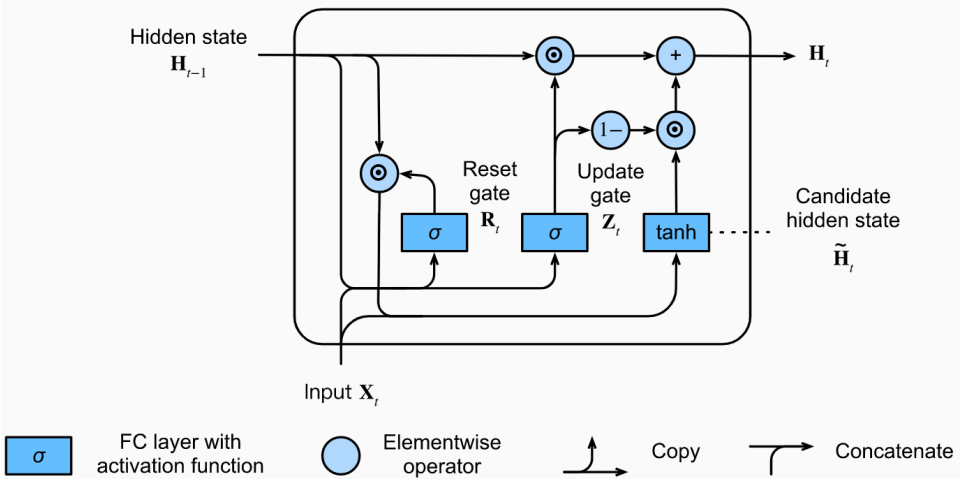
- Esse portão define o quanto o novo estado escondido $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ é simplesmente o estado anterior $\mathbf{H}_{t-1}$ e o quanto o novo estado candidato $\tilde{\mathbf{H}}_t$ é utilizado

- O update gate $\mathbf{Z}_t$ faz uma combinação elemento a elemento entre $\mathbf{H}_{t-1}$ e $\tilde{\mathbf{H}}_t$, de acordo com a seguinte equação

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$

Gate Recurrent Units (GRU)

- Sempre que o update gate $\mathbf{Z}_t$ for próximo de 1, o estando anterior é mantido. Nesse caso, a informação de $\mathbf{X}_t$ é essencialmente ignorada, como se pulássemos a iteração t na sequência
- Sempre que $\mathbf{Z}_t$ for próximo de 0, o novo estado $\mathbf{H}_t$ se aproxima do estado candidato $\widetilde{\mathbf{H}}_t$
- Os portões ajudam a melhor capturar dependências em sequências grandes. Por exemplo, se o update gate se mantiver próximo de 1 em todas as iterações de uma subsequência inteira, um estado escondido antigo na iteração do início da subsequência será conservado e passado para o fim da subsequência, independentemente do comprimento da mesma



Reset gates ajudam a capturar dependências de curto prazo em sequências

Update gates ajudam a capturar dependências de longo prazo em sequências

Gate Recurrent Units (GRU)

- GRUs melhor capturam dependências em sequências grandes
- Reset gates ajudam a capturar dependências de curto prazo em sequências
- Update gates ajudam a capturar dependências de longo prazo em sequências
- GRUs contém RNNs básicas como caso extremo, sempre que o reset gate é ativado
- Subsequências são ignoradas quando o update gate é ativado

Long Short-Term Memory (LSTMs)

O desafio de preservar informação de longo prazo e ignorar entradas existe há muito tempo. A LSTM foi um dos primeiros modelos propostos para isso. Ela compartilha muitas das propriedades da GRU, tem uma estrutura um pouco mais complexa e é anterior à GRU em quase duas décadas.

- A LSTM introduz o conceito de célula de memória, que é controlada por um número de portões: portão de saída (output gate), portão de entrada (input gate) e portão de esquecimento (forget gate)

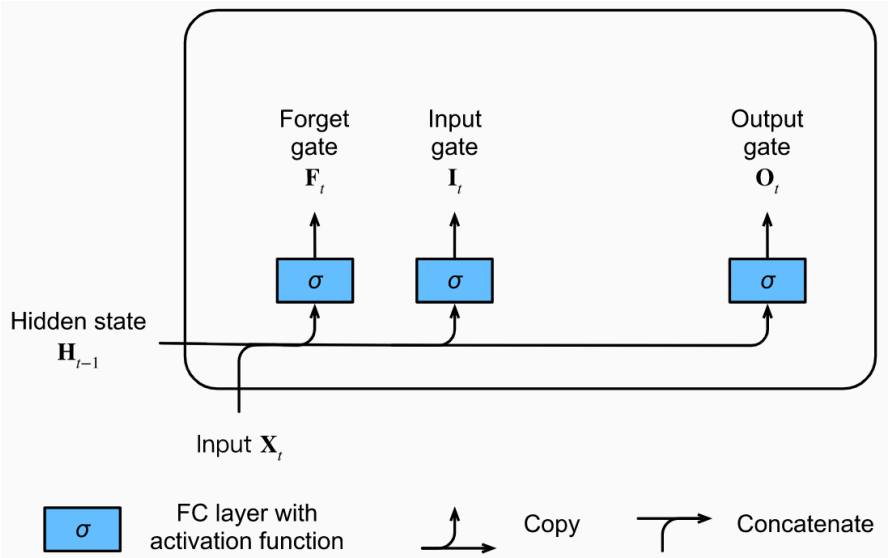
- A motivação é a mesma da GRU: tornar possível a decisão de quando lembrar e quando ignorar entradas no estado escondido

Long Short-Term Memory (LSTMs)

Input gate, forget gate e output gate

- Assim como nas GRUs, a entrada nos portões da LSTM é dada pela entrada na iteração atual, e pelo estado escondido na iteração anterior.

- O processamento é feito por três camadas totalmente conectadas com função de ativação sigmoide



Long Short-Term Memory (LSTMs)

Considerando h unidades escondidas, batch de tamanho n, e número de entradas igual a d, a entrada da rede é dada por $\mathbf{X}_t \in \mathbb{R}^{n \times d}$, e o estado escondido na iteração anterior é dado por $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$.

- Os portões na iteração *t* são definidos como $\mathbf{I}_t \in \mathbb{R}^{n \times h}$ (input gate), $\mathbf{F}_t \in \mathbb{R}^{n \times h}$ (forget gate), e $\mathbf{O}_t \in \mathbb{R}^{n \times h}$ (output gate):

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i)$$

$$\mathbf{F}_t = \sigma\left(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f\right)$$

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o)$$

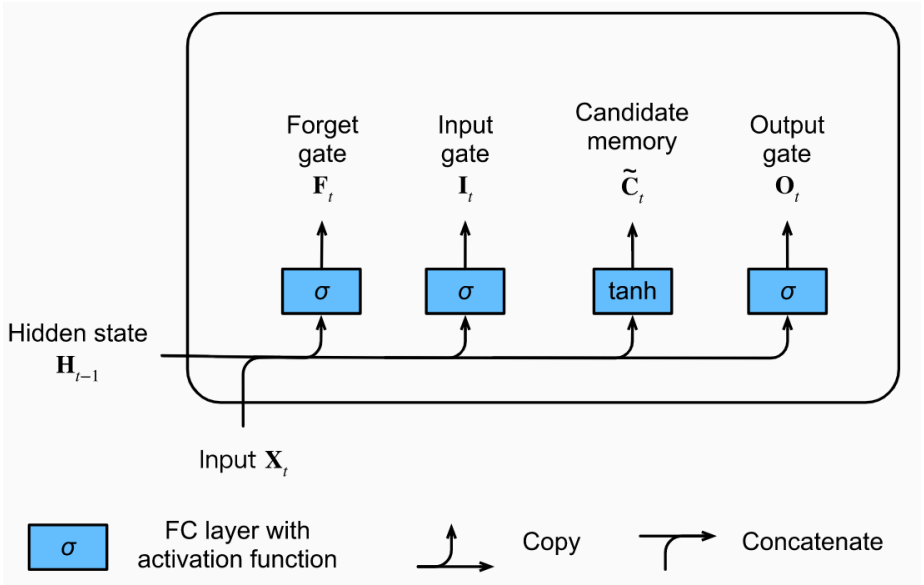
- Nas equações, $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo} \in \mathbb{R}^{n \times h}$ e $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho} \in \mathbb{R}^{h \times h}$ são os pesos, e $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o \in \mathbb{R}^{1 \times h}$ são os bias

Long Short-Term Memory (LSTMs)

Não definimos ainda como os portões vão atuar. Vamos inicialmente introduzir nossa célula de memória $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n \times h}$ como candidata, calculada como segue

$$\tilde{\mathbf{C}}_t = tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c)$$

- $\mathbf{W}_{xc} \in \mathbb{R}^{d \times h}$ e $\mathbf{W}_{hc} \in \mathbb{R}^{h \times h}$ são os pesos, e $\mathbf{b}_c \in \mathbb{R}^{1 \times h}$ é o bias



Long Short-Term Memory (LSTMs)

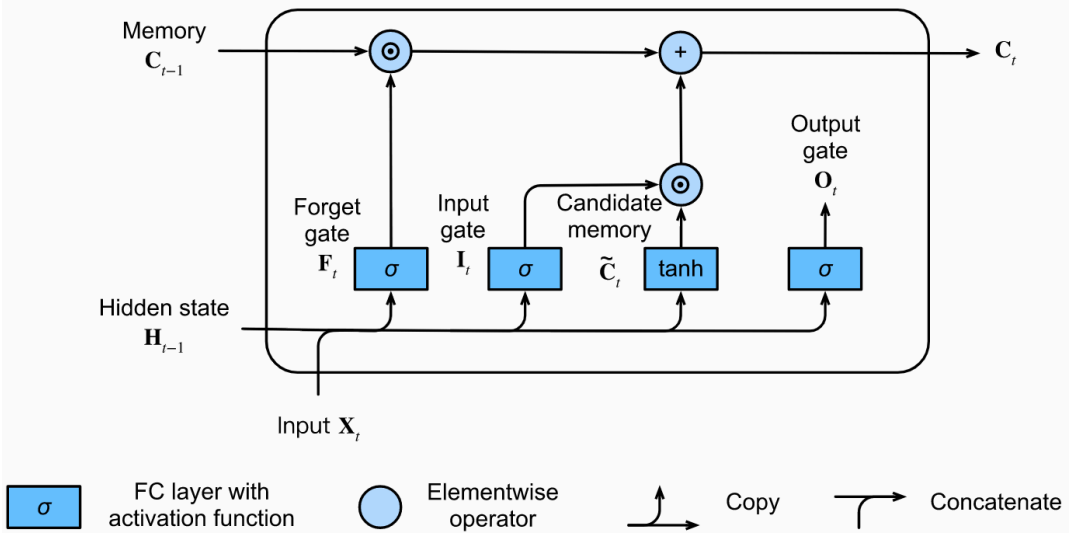
Nas GRUs temos mecanismos para controlar a entrada e o esquecimento (ignorar entradas). Na LSTM há dois portões para isso:

- O input gate $\mathbf{I}_t$ diz o quanto levamos em conta os dados novos usando $\tilde{\mathbf{C}}_t$
- O forget gate $\mathbf{F}_t$ nos diz o quanto da célula de memória anterior $\mathbf{C}_{t-1}$ devemos manter
- Usando o operador de multiplicação ponto a ponto, temos a seguinte equação:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

Long Short-Term Memory (LSTMs)

Se o forget gate é sempre aproximadamente 1 e o input gate é sempre aproximadamente 0, as células de memória passadas $\mathbf{C}_{t-1}$ serão salvas ao longo do tempo e passadas para a iteração atual. Assim como na GRU, isso ajuda a capturas dependências em sequências grandes

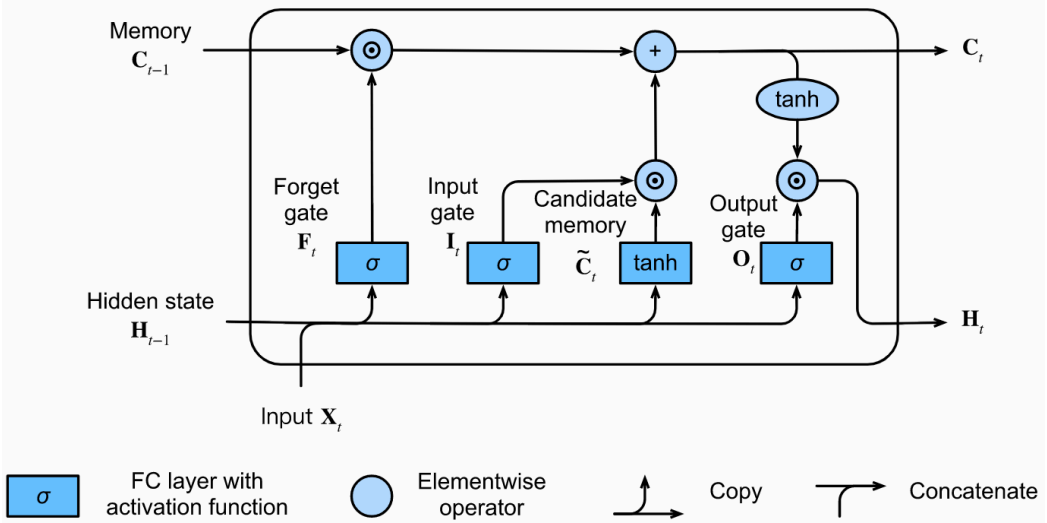


Long Short-Term Memory (LSTMs)

Finalmente, precisamos definir o cálculo do estado escondido $\mathbf{H}_t \in \mathbb{R}^{n \times h}$. Vamos utilizar agora o output gate $\mathbf{O}_t$ e a memória $\mathbf{C}_t$:

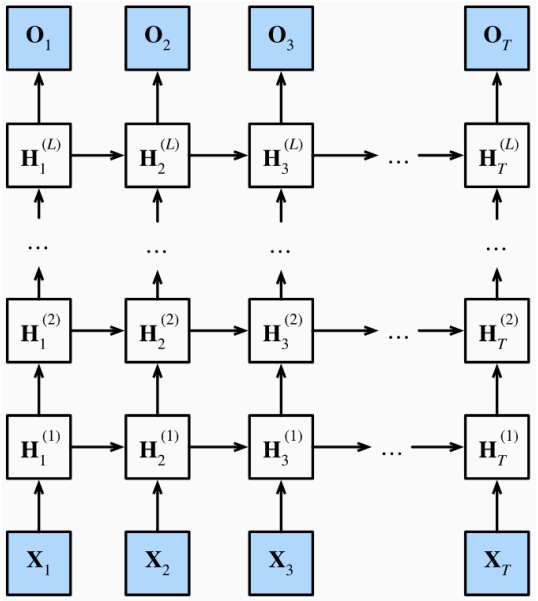
$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t)$$

- Sempre que o output gate se aproxima de 1, toda a informação da memória é passada pelo preditor. Para o output gate próximo de 0, toda a informação é retida somente na célula de memória



Deep Recurrent Neural Networks

Em uma Deep RNN com L camadas, cada estado escondido é continuamente passado tanto para a próxima iteração da camada atual, quanto para a iteração corrente da próxima camada.



Deep Recurrent Neural Networks

A dependência funcional da arquitetura profunda com L camadas pode ser formalizada. Vamos fazer a formalização com a RNN simples (vanilla), mas a formalização também serve para outros modelos

- Suponha um minibatch $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ (d: número de exemplos, d: número de entradas para cada exemplo) na iteração t. Na mesma iteração t, o estado escondido da l^{th} camada escondida ($l = 1, \dots, L$) é dado por $\mathbf{H}_t^{(l)} \in \mathbb{R}^{n \times h}$ (h: número de unidades escondidas), e a camada de saída é dada por $\mathbf{O}_t \in \mathbb{R}^{n \times q}$ (q: número de saídas)

- Considerando $\mathbf{H}_t^{(0)} = \mathbf{X}_t$, o estado escondido na l^{th} camada escondida com função de ativação ϕ_l é dado por:

$$\mathbf{H}_t^{(l)} = \phi_l \left(\mathbf{H}_t^{(l-1)} \mathbf{W}_{xh}^{(l)} + \mathbf{H}_{t-1}^{(l)} \mathbf{W}_{hh}^{(l)} + \mathbf{b}_h^{(l)} \right)$$

- Os pesos $\mathbf{W}_{xh}^{(l)} \in \mathbb{R}^{h \times h}$, $\mathbf{W}_{hh}^{(l)} \in \mathbb{R}^{h \times h}$ e o bias $\mathbf{b}_h^{(l)} \in \mathbb{R}^{1 \times h}$ são os parâmetros do modelo na l^{th} camada escondida

Deep Recurrent Neural Networks

O cálculo da camada de saída é baseado apenas no estado escondido da L^{th} camada escondida:

$$\mathbf{O}_t = \mathbf{H}_t^{(L)}\mathbf{W}_{hq} + \mathbf{b}_q$$

- Os pesos $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ e o bias $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ são os parâmetros do modelo na camada de saída

- Assim como nas MLPs, o número de camadas escondidas L e o número de unidades escondidas h são hiperparâmetros

- Uma Deep RNN com portões pode ser facilmente obtida substituindo o cálculo do estado escondido apresentado pelos cálculos da GRU ou da LSTM

Bidirectional Recurrent Neural Networks

Até agora, para aprender sequências, consideramos que o objetivo é prever a próxima saída dado o que foi visto até agora. No entanto há cenários mais complicados.

- Eu estou _____.
- Eu estou _____ fome.
- Eu estou _____ fome, e poderia comer metade de um boi.

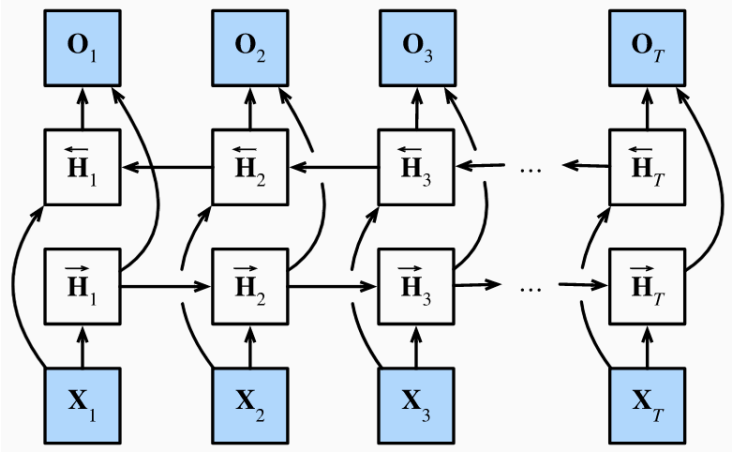
Dependendo da quantidade de informação disponível, podemos preencher as lagunas com diferentes palavras, tais como “faminto”, “sem” ou “com”. Claramente, o fim da frase, se disponível, provê informação importante sobre qual palavra escolher.

- Um modelo que é incapaz de utilizar essa informação terá desempenho ruim em tais tarefas, como por exemplo: laranja (fruta ou testa de ferro?)

Bidirectional Recurrent Neural Networks

Se queremos um mecanismo nas RNNs que torne-as capaz de “olhar mais pra frente”, precisamos modificar as RNNs que temos visto

- Modificação: ao invés de executar uma RNN somente em uma direção, começando do primeiro token, executamos também outra RNN iniciando do último token até o primeiro



Bidirectional Recurrent Neural Networks

Para um time step t, considere um minibatch $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ (n: número de exemplos, d: número de entradas para cada exemplo), e função de ativação ϕ na camada escondida

- Na arquitetura bidirecional, assumimos que os estados ocultos para frente e para trás no time step t são $\overrightarrow{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ e $\overleftarrow{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$, com h o número de unidades escondidas

$$\overrightarrow{\mathbf{H}}_t = \phi\left(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}\right)$$

$$\overleftarrow{\mathbf{H}}_t = \phi\left(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}\right)$$

- Em seguida, concatenamos os estados escondidos $\overrightarrow{\mathbf{H}}_t$ e $\overleftarrow{\mathbf{H}}_t$ para obter um estado escondido $\mathbf{H}_t \in \mathbb{R}^{n \times 2h}$, que é então fornecido para a camada de saída. Em Deep Bidirectional RNNs com mais camadas ocultas, essa informação é dada como entrada para a próxima camada bidirecional. A camada de saída $\mathbf{O}_t \in \mathbb{R}^{n \times q}$ (q: número de saídas) é obtida por meio de

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q$$

Bidirectional Recurrent Neural Networks

Importante: em RNNs bidirecionais, utilizamos informação do passado e do futuro para prever o presente. Sendo assim, elas não são adequadas para tarefas de previsão do futuro. Elas são mais adequadas para tarefas como:

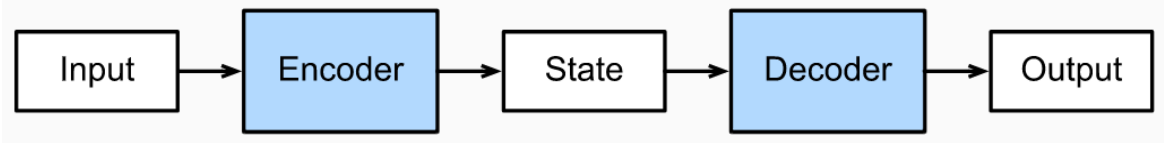
- Preenchimento de palavras faltantes
- Anotação, por exemplo reconhecimento de entidades nomeadas
- Tradução automática

Arquitetura Encoder-Decoder

Em muitos problemas, quando trabalhamos com modelos de sequência, precisamos transformar uma sequência de entrada em uma sequência de saída (sequence transduction). Em muitos casos, a entrada e a saída do modelo são diferentes e com tamanhos diferentes. Para tratar esses tipos de problema, pode ser utilizada uma arquitetura com dois tipos de componentes:

- Encoder: recebe uma sequência de tamanho variável como entrada e transforma em um estado com tamanho fixo

- Decoder: mapeia o estado codificado de tamanho fixo para uma sequência de tamanho variável



Arquitetura Encoder-Decoder

Um problema de sequence transduction é a tradução de textos. A tarefa pode ser traduzir um texto do Inglês para o Português, ou do Inglês para o Francês.

- Dada por exemplo uma sequência de entrada “they”,“are”,“very”,“hungry”,“.”, a arquitetura encoder-decoder primeiro codifica a sequência para um estado escondido de tamanho fixo, e então decodifica esse estado para gerar uma sequência traduzida token por token na saída: “eles”,“estão”,“com”,“muita”,“fome”,“.”

- Outro exemplo do Inglês para o Francês: “they”,“are”,“watching”,“.” traduzido para “ils”,“regardent”,“.”

- O modelo encoder-decoder é a base para diferentes modelos de sequence transduction

Sequence to Sequence

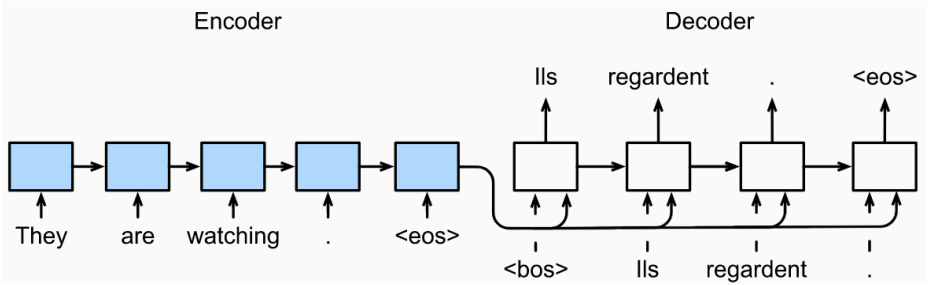
Tomando como base a arquitetura encoder-decoder, podemos utilizar duas RNNs para implementar o encoder e o decoder da arquitetura. Vamos usar o exemplo da tradução de textos

- A RNN encoder recebe como entrada uma sequência de tamanho variável e a transforma em um estado escondido de tamanho fixo. Assim, informação de entrada é codificada em um estado escondido

- A sequência de saída é gerada token por token, com outra RNN decoder predizendo o próximo token baseado em quais tokens já foram vistos, juntamente com a informação codificada (encoded) da sequência de entrada

Sequence to Sequence

A Figura ilustra como usar duas RNNs para a tarefa de tradução



- O caractere especial <eos> marca o fim da sequência, indicando que o modelo pode parar de fazer predições quando esse token é gerado
- No decoder, um caractere especial de início de sequência <bos> é utilizado como entrada
- O estado escondido final do encoder é utilizado para iniciar o decoder
- Em algumas implementações (como na figura), o estado escondido final do encoder também é fornecido ao decoder como parte da entrada em cada iteração
- Os rótulos podem ser a sequência de saída original, deslocada de um token
- “<bos>”, “Ils”, “regardent”, “.” → “Ils”, “regardent”, “.”, “<eos>”

Sequence to Sequence

O **Encoder** transforma uma sequência de entrada de tamanho variável em uma variável de contexto $\mathbf{c}$ de tamanho fixo. A informação da sequência de entrada é codificada nessa variável

- Suponha um sequência de entrada $x_1, \ldots, x_T$ tal que x_t é o t-ésimo token na sequência. Na iteração t, uma RNN transforma o vetor de atributos $\mathbf{x}_t$ de x_t e o estado escondido $\mathbf{h}_{t-1}$ da iteração anterior, no estado escondido atual $\mathbf{h}_t$. Essa transformação é dada pela função f

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

- O Encoder transforma os estados escondidos em todas as iterações na variável de contexto por meio de uma função q

$$\mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T)$$

- Por exemplo, quando $q(\mathbf{h}_1, \ldots, \mathbf{h}_T) = \mathbf{h}_T$, a variável de contexto é apenas o estado escondido $\mathbf{h}_T$ da sequência de entrada na última iteração (conforme figura do slide anterior)

Sequence to Sequence

Dada uma sequência de saída $y_1, y_2, \ldots, y_{T'}$ do conjunto de treino, para cada iteração t', a probabilidade da saída $y_{t'}$ do **Decoder** é condicionada à subsequência de saída anterior $y_1, \ldots, y_{t'-1}$ e à variável de contexto $\mathbf{c}$:

$$P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathbf{c})$$

- Para modelar essa probabilidade condicional, outra RNN é usada. Em toda iteração t' na sequência de saída, a RNN recebe como entrada a saída $y_{t'-1}$ da iteração anterior e a variável de contexto $\mathbf{c}$, e as transforma, juntamente com o estado escondido anterior $\mathbf{s}_{t'-1}$, no estado escondido $\mathbf{s}_{t'}$ da iteração atual:

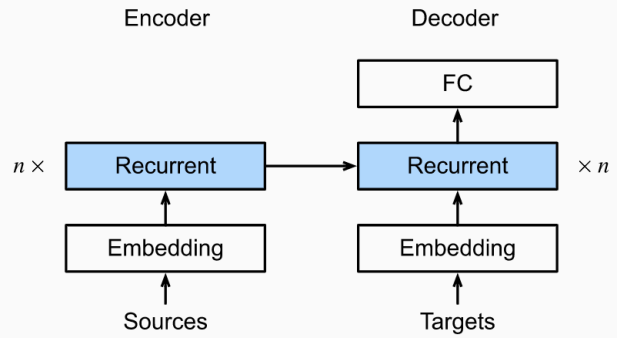
$$\mathbf{s}_{t'} = g(y_{t'-1}, \mathbf{c}, \mathbf{s}_{t'-1})$$

- Após obter o estado escondido do Decoder, uma camada de saída com softmax pode ser utilizada para calcular a distribuição de probabilidade condicional $P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathbf{c})$

Sequence to Sequence

Quando implementando o decoder, usamos diretamente o estado escondido obtido após a última iteração no encoder para inicializar o estado escondido do decoder. Isso requer que a RNN encoder e a RNN decoder tenham o mesmo número de unidades escondidas

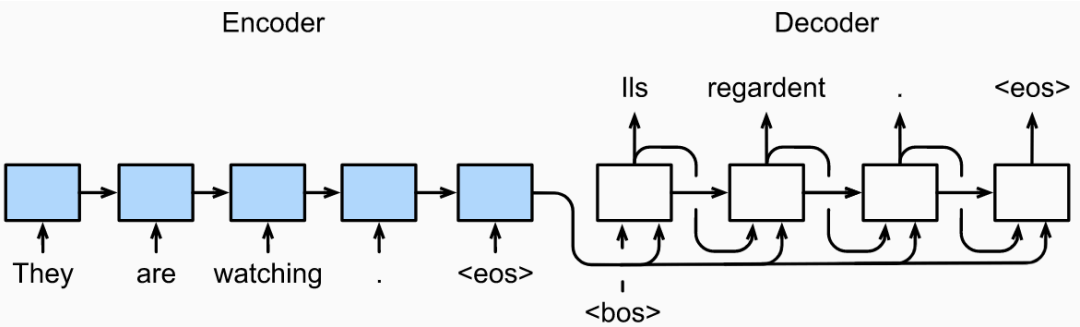
- Para incorporar a informação codificada da sequência de entrada, a variável de contexto é concatenada com a entrada do decoder em todas as iterações
- A predição da distribuição de probabilidade do token de saída é feita por uma camada totalmente conectada, transformando o estado escondido da camada final da RNN decoder



Sequence to Sequence

Para predizer uma sequência de saída token por token, a cada iteração do decoder, o token predito na iteração anterior é fornecido como entrada ao decoder na iteração atual

- Da mesma maneira que no treino, na iteração inicial o token “início de sequência” <bos> é fornecido ao decoder. Quando o token “fim de sequência” é predito, a predição da sequência de saída é finalizada



Perguntas?