Redes Recorrentes - RNN

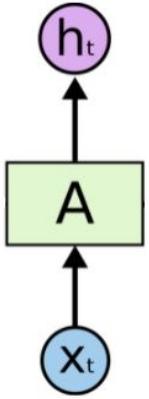
Deep Learning

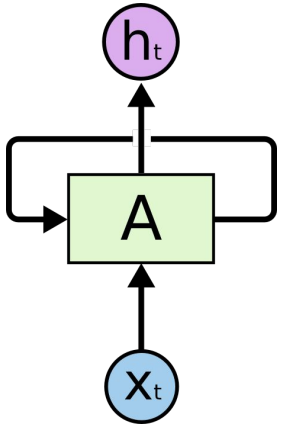
Referências complementares

- Understanding LSTM Networks
- Understanding LSTM and its diagrams
- Redes Neurais Recorrentes
- Livro **Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability** de **Danilo Mandic**
- Livro **Deep Learning: Recurrent Neurtal Networks in Python: LSTM, GRU, and more RNN machine learning architectures in Python and Theano** de **LazyProgrammer**Deep Learning: Recurrent Neural Networks in Python: LSTM, GRU, and more RNN machine learning architectures in Python and Theano (Machine Learning in Python) (English Edition)

Introdução

As arquiteturas de redes neurais que você já viu até agora foram treinadas usando apenas entradas atuais. Não consideramos entradas anteriores ao gerar a saída atual. Em outras palavras, nossos sistemas não continham nenhum elemento de **memória**.





As RNNs lidam com essa questão básica e importante usando **memória** (ou seja, entradas passadas da rede) ao produzir a entrada atual.

Um pouco da história neste link

Entendendo a evolução das RNNs

Os marcos importantes da evolução das RNN:

- TDNN - Rede neural de retardo de tempo.
- Elman Network - Publicação original de 1990.
- Redes Elman e Redes Jordan - Informações adicionais.
- LSTM *Long Short-Term Memory* - Artigo original escrito por Sepp Hochreiter e Jürgen Schmidhuber.

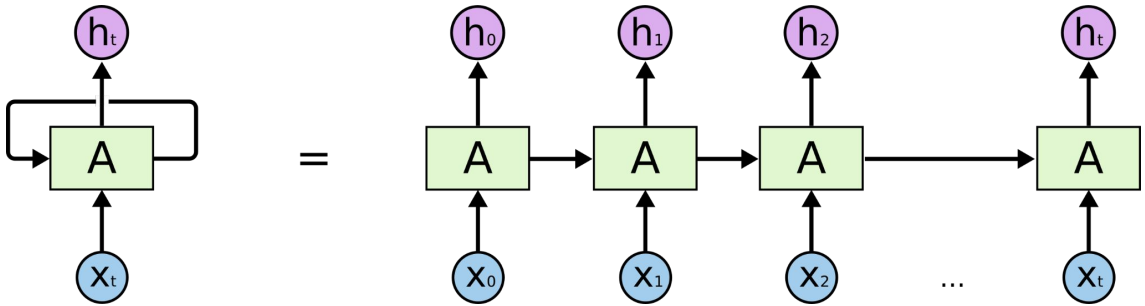
Aplicações de RNNs

As empresas líderes mundiais em tecnologia estão todas usando RNNs, especialmente LSTMs, em suas aplicações. Algumas aplicações interessantes:

- Você se interesse por jogos e bots? Veja este bot de DotA 2 da Open AI.
- Que tal adicionar sons a filmes mudos automaticamente?
- Aqui está uma ferramenta legal para geração automática de caligrafia.
- Transcrição de voz para texto da Amazon usando reconhecimento de fala de alta qualidade, Amazon Lex.
- O Facebook usa tecnologias de RNN e LSTM para construir modelos de linguagem.
- A Netflix também usa modelos RNN - aqui está uma leitura interessante.

Redes Neurais Recorrentes - RNNs

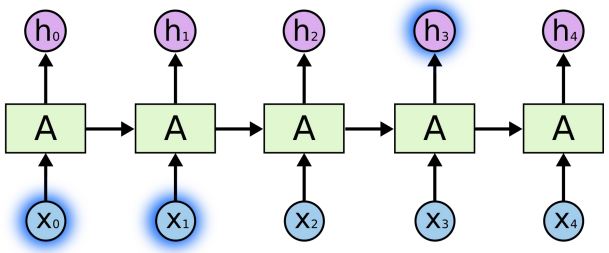
As RNNs são redes com loops permitindo que as informações persistam. Um loop permite que as informações sejam passadas de uma etapa da rede para a próxima. Uma RNN pode ser considerada como várias cópias da mesma rede, cada uma passando uma mensagem a um sucessor.

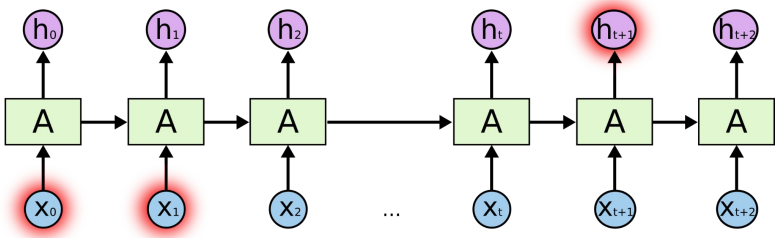


blog de Andrej Karpathy

O Problema das Dependências de Longo Prazo

Um dos apelos das RNNs é a ideia de que elas podem ser capazes de conectar informações anteriores à tarefa presente, não precisando de contexto adicional. Isto ocorre quando não temos previsão de longo prazo em relação às entradas.



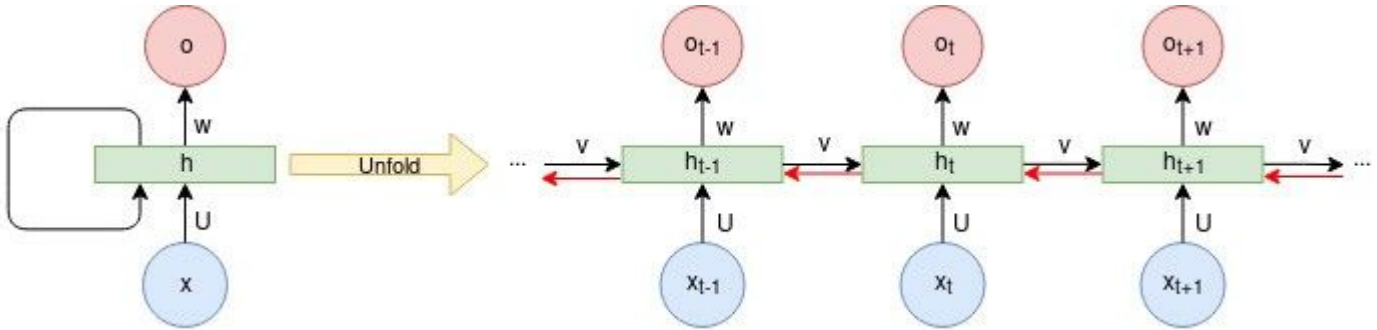


Infelizmente, à medida que essa lacuna aumenta, os RNNs tornam-se incapazes de aprender a conectar as informações.

O problema foi explorado em profundidade por Hochreiter (1991) [alemão] e Bengio, et al. (1994) , que encontrou algumas razões fundamentais pelas quais isso pode ser difícil.

Backpropagation Through Time (BPTT)

As RNNs utilizam um algoritmo de treinamento chamado *Backpropagation Through Time* (BPTT), o qual é uma extensão do *backpropagation*. Em uma RNN, a saída de uma célula é também a entrada dessa célula, então, o erro é propagado de uma célula para a própria célula. Então, a interpretação aqui é a de que o erro se propaga no tempo (para o passado, no caso).

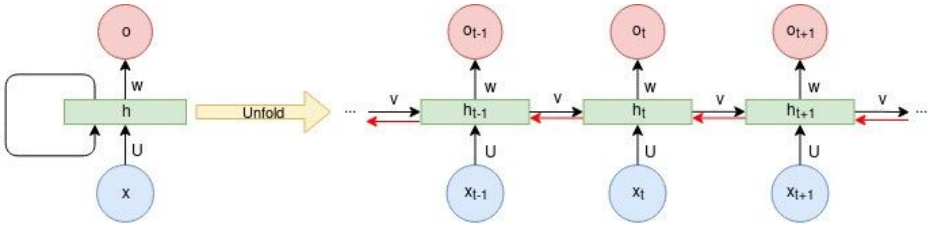


Backpropagation Through Time (BPTT)

Considere que estamos criando uma rede neural recorrente que seja capaz de prever a próxima palavra em um texto. Aqui temos as equações básicas da nossa RNN:

$$h_t = tanh(U x_{t-1} + V h_{t-1})$$

$$\hat{o}_t = W h_t$$



OBS: Note a diferença entre o(t) e ô(t). O primeiro se refere ao valor de algo real que queremos prever, enquanto o segundo é o valor da nossa predição.

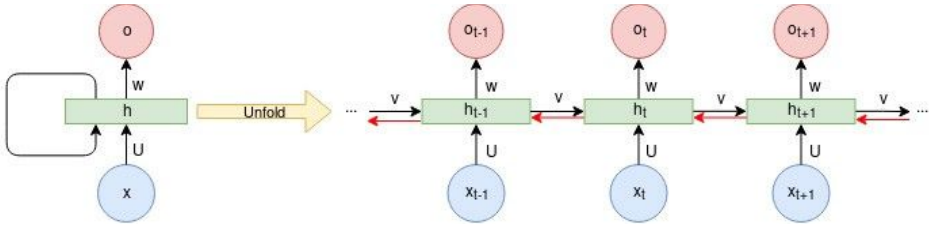
Vamos considerar aqui um problema de classificação. Então, também definimos nossa perda, ou erro, como a perda de entropia cruzada, dada por: $E_t(o_t, \hat{o}_t) = -o_t * log(\hat{o}_t)$

Backpropagation Through Time (BPTT)

Normalmente, tratamos a sequência completa (sentença) como um exemplo de treinamento, portanto, o erro total é apenas a soma dos erros em cada etapa de tempo (palavra).

$$E(o, \hat{o}) = \sum_t E_t(o_t, \hat{o}_t) = -\sum_t o_t * log(\hat{o}_t)$$

Lembre-se de que nosso objetivo é calcular os gradientes do erro em relação aos nossos parâmetros U, V e W e, em seguida, aprender bons parâmetros usando o *Stochastic Gradient Descent*. Assim como resumimos os erros, também somamos os gradientes em cada etapa de tempo para um exemplo de treinamento:

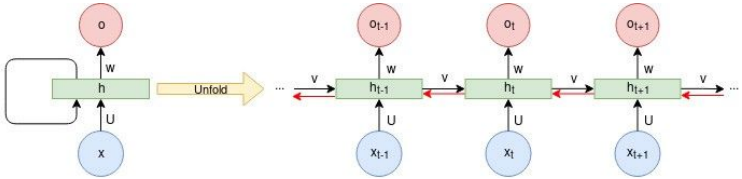


$$\frac{\partial E}{\partial V} = \sum_t \frac{\partial E_t}{\partial V}$$

Backpropagation Through Time (BPTT)

Para calcular esses gradientes, usamos a regra de diferenciação da cadeia. Esse é o algoritmo de retropropagação quando aplicado para trás a partir do erro. Usaremos o E_3 como exemplo:

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{o_3}} \frac{\partial \hat{o_3}}{\partial W} = \frac{\partial E_3}{\partial \hat{o_3}} \frac{\partial \hat{o_3}}{\partial z_3} \frac{\partial z_3}{\partial W} = (\hat{o_3} - o_3) \otimes h_3$$



Acima, $z_3 = Wh_3$ e a última expressão é o produto vetorial entre $\hat{o}_3$-o_3 e h_3.

O foco aqui é entender que $\frac{\partial E_3}{\partial W}$

depende apenas dos valores no momento atual, $\hat{o}_3$, o_3, h_3. Se você tem estes valores, calcular o gradiente para W é uma multiplicação simples de matriz.

Backpropagation Through Time (BPTT)

Mas a história é diferente para: $\frac{\partial E_3}{\partial V}$

Para entender porque, escrevemos a regra da cadeia, como acima:

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{o_3}} \frac{\partial \hat{o_3}}{\partial h_3} \frac{\partial h_3}{\partial V}$$

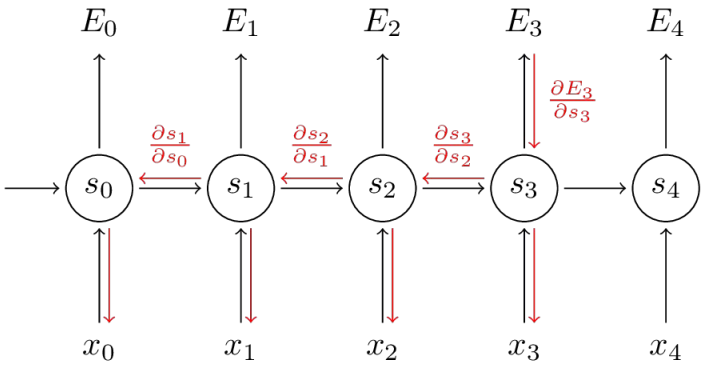
Agora, note que h_3 = tanh (Ux_3+Vh_2) depende de h_2, que depende de V e h_1, e assim por diante. Então, se pegarmos a derivada em relação a V, não podemos simplesmente tratar h_2 como uma constante. Precisamos aplicar a regra da cadeia novamente e o que realmente temos é o seguinte:

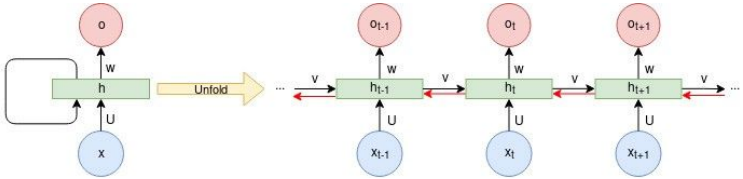
$$\frac{\partial E_3}{\partial V} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{o_3}} \frac{\partial \hat{o_3}}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial V}$$

Backpropagation Through Time (BPTT)

Somamos as contribuições de cada passo de tempo para o gradiente. Em outras palavras, como *V* é usado em todas as etapas até a saída que nos interessa, precisamos retroceder gradientes na rede de *t* = 3 até *t* = 0:

A imagem acima ilustra todo o processo comentado. A única diferença está no modo como cada estado foi nomeado. Aqui, usamos a notação h(t), e na imagem, cada estado é representado por s(t).





Backpropagation Through Time (BPTT)

Considerações Finais: Observe que isso é exatamente o mesmo que o algoritmo de retropropagação padrão que usamos nas Redes Neurais Profundas da Feedforward. A principal diferença é que resumimos os gradientes para V em cada etapa de tempo. Em um rede neural tradicional, não compartilhamos parâmetros entre camadas, portanto, não precisamos somar nada. Mas o BPTT é apenas um nome sofisticado para retropropagação padrão em uma RNN “desenrolada”. Assim como com Backpropagation, você pode definir um vetor delta que você repassa, por exemplo:

$$\delta_2^{(3)} = \frac{\partial E_3}{\partial z_2} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial z_2}$$

Isso também deve lhe dar uma ideia do motivo pelo qual as RNNs são difíceis de treinar: as sequências podem ser bastante longas e você precisa retroceder através de várias camadas. Na prática, é comum truncar a retropropagação em poucos passos.

Problema do Gradiente Desaparecendo

O **problema do gradiente de desaparecimento** é encontrado ao treinar redes neurais artificiais com métodos de aprendizado baseados em gradiente e retropropagação . Em tais métodos, cada um dos pesos da rede neural recebe uma atualização proporcional à derivada parcial da função de erro em relação ao peso atual em cada iteração de treinamento. O problema é que, em alguns casos, o gradiente será extremamente pequeno, evitando efetivamente que o peso mude de valor. Na pior das hipóteses, isso pode impedir completamente o treinamento adicional da rede neural.

Algumas soluções:

- Utilização do BPTT;
- Inicialização dos pesos: Xavier Initialization
- LSTM - *Long Short Term Memory*

*Quando funções de ativação são usadas cujas derivadas podem assumir valores maiores, corre-se o risco de encontrar o **problema de gradiente explosivo**.

Problema do Gradiente Explosivo

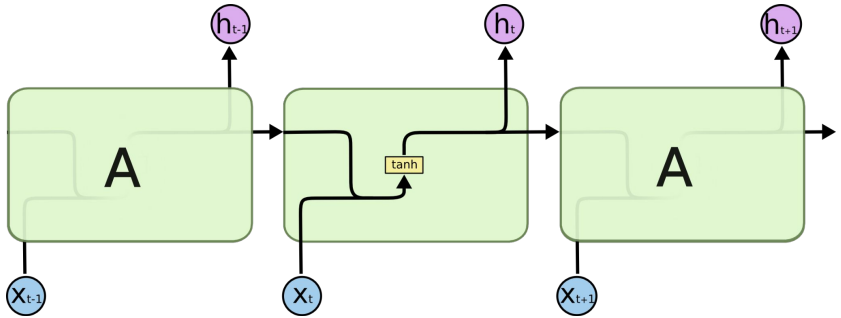
Algumas soluções:

- Não visitar todas as camadas ocultas**
- RMSProp - tenta resolver o problema de que gradientes podem variar amplamente em magnitudes. Alguns gradientes podem ser minúsculos e outros podem ser enormes, o que resulta em um problema muito difícil - tentar encontrar uma única taxa de aprendizado global para o algoritmo.
- *Cliping gradient* - O recorte de gradiente envolve forçar os valores de gradiente (elemento a elemento) para um valor mínimo ou máximo específico se o gradiente exceder um intervalo esperado.

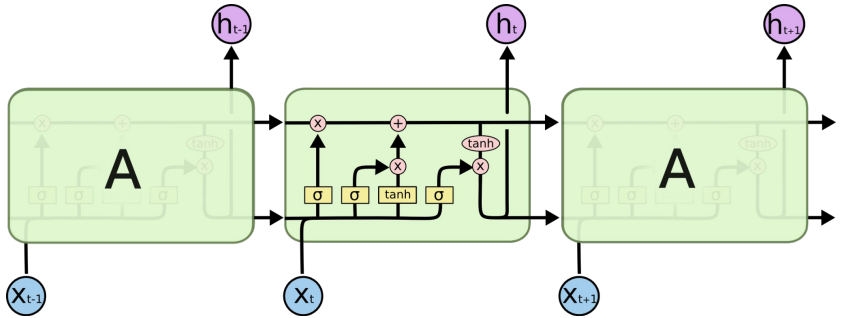
*Quando funções de ativação são usadas cujas derivadas podem assumir valores maiores, corre-se o risco de encontrar o **problema de gradiente explosivo**.

Long Short Term Memory (LSTM)

Todas as redes neurais recorrentes têm a forma de uma cadeia de módulos repetidos de rede neural. Em RNNs padrão, esse módulo de repetição terá uma estrutura muito simples, como uma única camada de *tanh*.



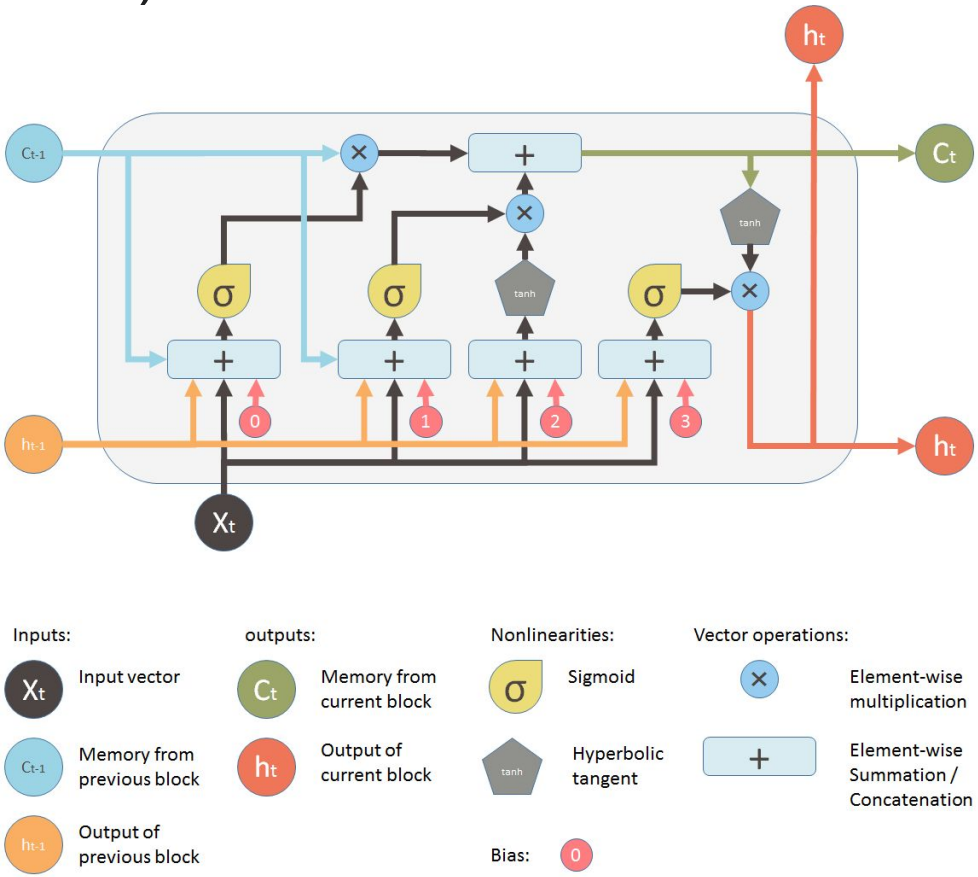
Redes de memória de longo prazo - geralmente chamadas apenas de “LSTMs” - são um tipo especial de RNN, capaz de aprender dependências de longo prazo. Eles foram introduzidos por Hochreiter & Schmidhuber (1997) para resolver esse problema, introduzindo explicitamente uma unidade de memória na rede chamada de célula.



Long Short Term Memory (LSTM)

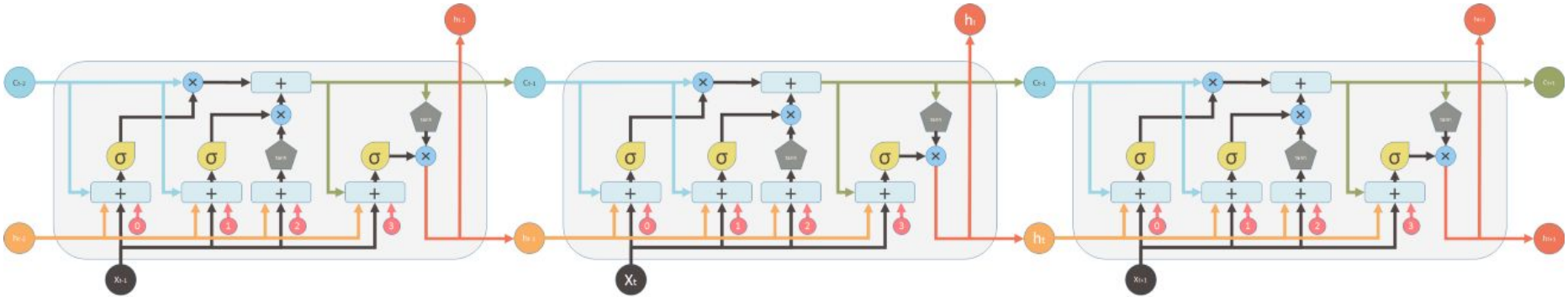
A rede tem três entradas. X_t é a entrada da etapa de tempo atual. h_t-1 é a saída da unidade LSTM anterior e C_t-1 é a "memória" da unidade anterior, que eu acho que é a entrada mais importante. Quanto às saídas, h_t é a saída da rede atual. C_t é a memória da unidade atual.

Portanto, esta única unidade toma decisões considerando a entrada atual, a saída anterior e a memória anterior. E gera uma nova saída e altera sua memória.



Long Short Term Memory (LSTM)

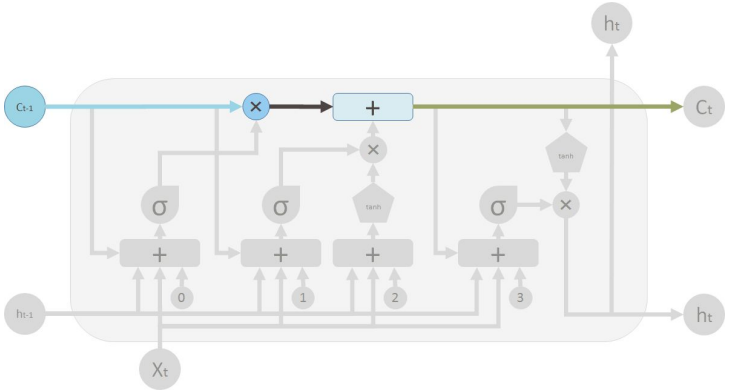
A forma como a memória interna C_t muda é muito semelhante a canalizar água por um cano. Supondo que a memória seja água, ela flui para um cano. Você deseja alterar esse fluxo de memória ao longo do caminho e essa alteração é controlada por duas válvulas.



Long Short Term Memory (LSTM)

A primeira válvula é chamada de válvula de esquecimento. Se você fechá-lo, nenhuma memória antiga será mantida. Se você abrir totalmente esta válvula, toda a memória antiga passará.

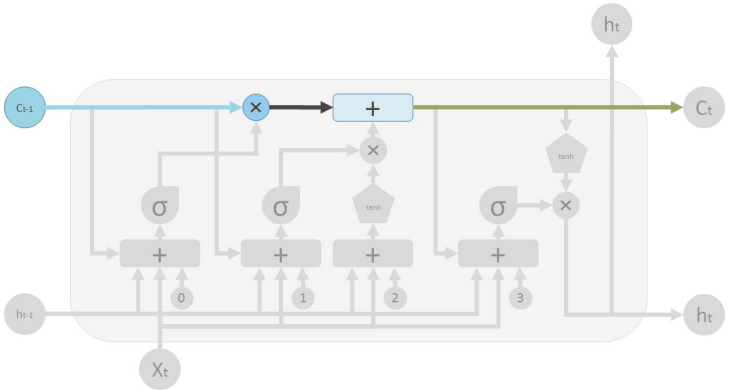
No diagrama LSTM, o “tubo” superior é o tubo de memória. A entrada é a memória antiga (um vetor). A primeira cruz ✖ pela qual ela passa é a válvula de esquecimento. Na verdade, é uma operação de multiplicação por elemento. Portanto, se você multiplicar a memória antiga C_t-1 por um vetor próximo a 0, isso significa que você deseja esquecer a maior parte da memória antiga. Você deixa a memória antiga passar, se sua válvula de esquecimento for igual a 1.



Long Short Term Memory (LSTM)

A segunda válvula é a nova válvula de memória. Uma nova memória virá através de uma junta em forma de T como acima e se fundirá com a memória antiga. Exatamente a quantidade de memória nova que deve entrar é controlada pela segunda válvula.

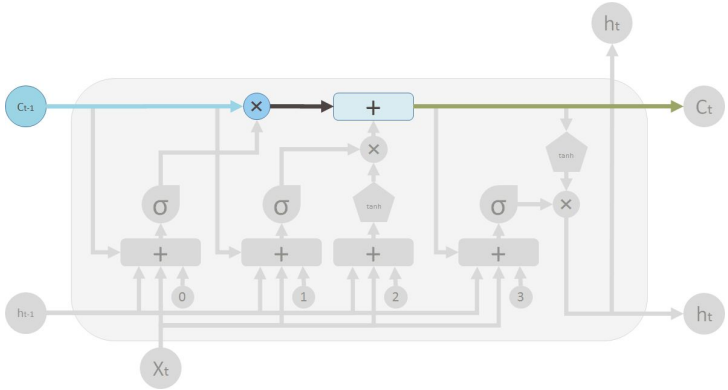
Então, a segunda operação pela qual o fluxo de memória passará é este operador +. Este operador significa soma por peça. Assemelha-se ao tubo de junção em forma de T. A memória nova e a memória antiga se fundirão por esta operação. A quantidade de memória nova que deve ser adicionada à memória antiga é controlada por outra válvula, o ✖ abaixo do sinal +.



Long Short Term Memory (LSTM)

A segunda válvula é a nova válvula de memória. Uma nova memória virá através de uma junta em forma de T como acima e se fundirá com a memória antiga. Exatamente a quantidade de memória nova que deve entrar é controlada pela segunda válvula.

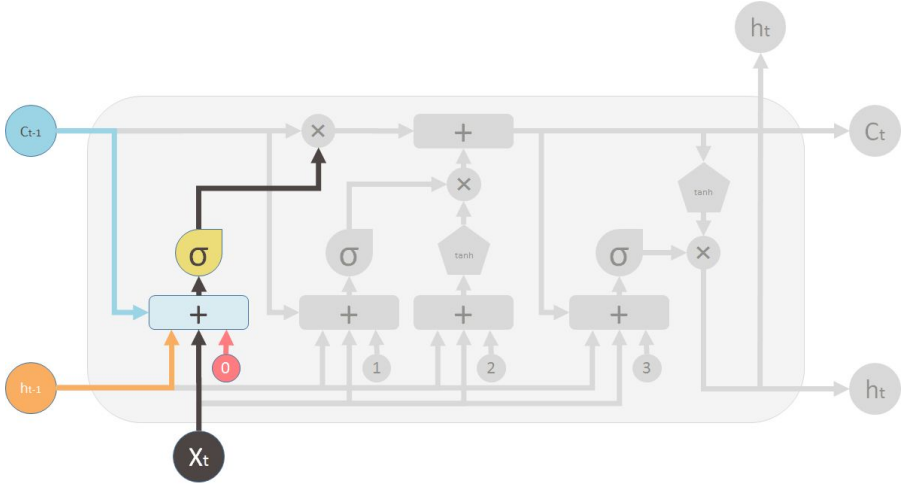
Então, a segunda operação pela qual o fluxo de memória passará é este operador +. Este operador significa soma por peça. Assemelha-se ao tubo de junção em forma de T. A memória nova e a memória antiga se fundirão por esta operação. A quantidade de memória nova que deve ser adicionada à memória antiga é controlada por outra válvula, o ✖ abaixo do sinal +.



Após essas duas operações, você tem a memória antiga C_t-1 alterada para a nova memória C_t.

Long Short Term Memory (LSTM)

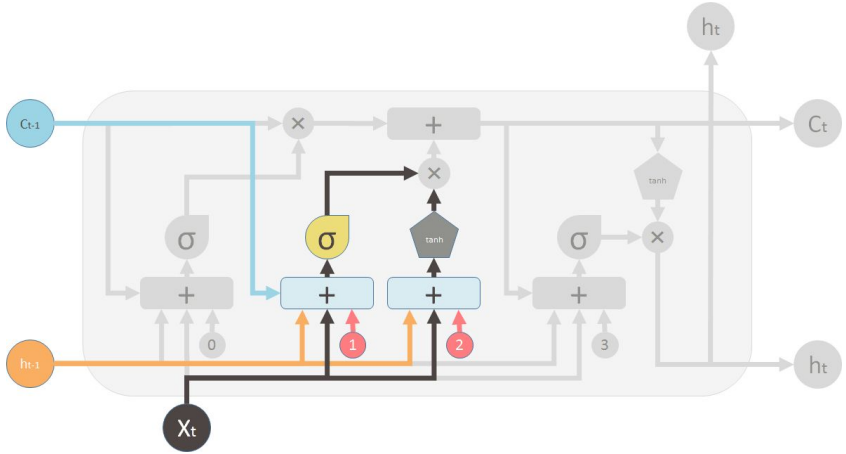
Agora vamos dar uma olhada nas válvulas. O primeiro é chamado de válvula de esquecimento. É controlado por uma rede neural simples de uma camada. As entradas da rede neural são h_t-1, a saída do bloco LSTM anterior, X_t, a entrada para o bloco LSTM atual, C_t-1, a memória do bloco anterior e, finalmente, um vetor de polarização b_0.



Esta rede neural tem uma função sigmóide como ativação, e seu vetor de saída é a válvula de esquecimento, que será aplicada à velha memória C_t-1 por multiplicação elemento a elemento.

Long Short Term Memory (LSTM)

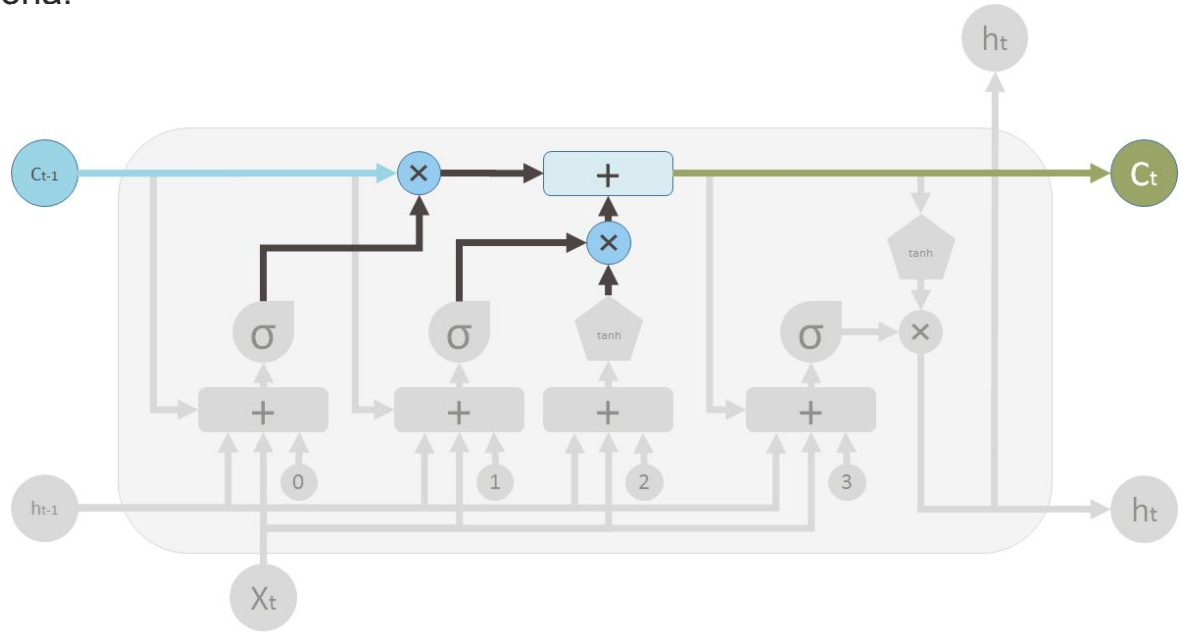
Agora, a segunda válvula é chamada de nova válvula de memória. Novamente, é uma rede neural simples de uma camada que recebe as mesmas entradas que a válvula de esquecimento. Esta válvula controla o quanto a nova memória deve influenciar a memória antiga.



A nova memória em si, entretanto, é gerada por outra rede neural. É também uma rede de uma camada, mas usa ***tanh*** como função de ativação. A saída desta rede irá multiplicar os elementos da nova válvula de memória e adicionar à memória antiga para formar a nova memória.

Long Short Term Memory (LSTM)

Esses dois sinais ✖ são a válvula de esquecimento e a nova válvula de memória.

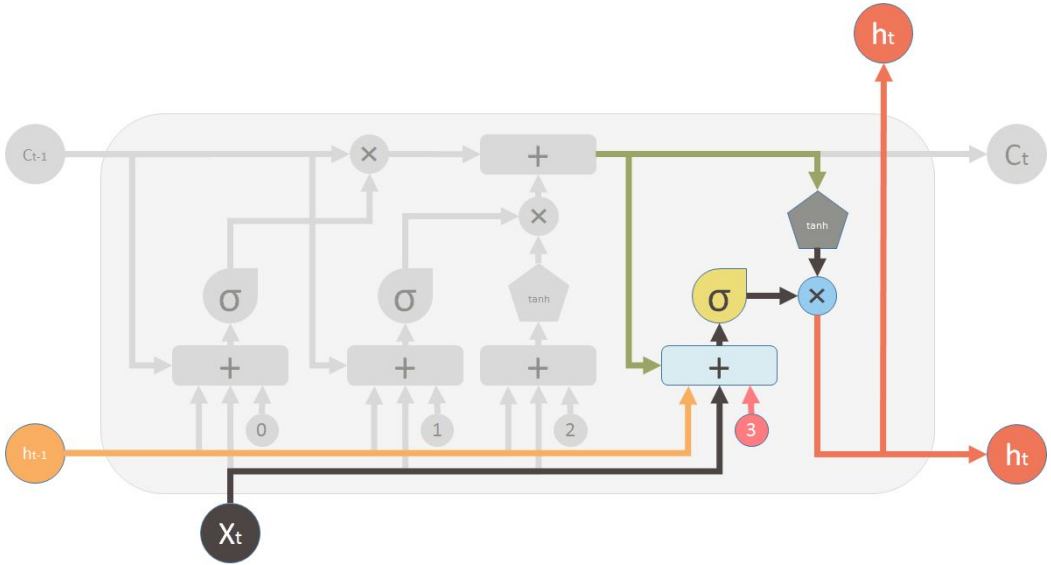


Long Short Term Memory (LSTM)

Por exemplo, Grid LSTMs de Kalchbrenner, *et al.* (2015) parecem extremamente promissores. Trabalho usando RNNs em modelos generativos - como Gregor, *et al.* (2015) , Chung, *et al.* (2015) , ou Bayer & Osendorfer (2015) - também parece muito interessante. Os últimos anos foram uma época empolgante para as redes neurais recorrentes, e os que estão por vir prometem ser ainda mais!

Long Short Term Memory (LSTM)

E, finalmente, precisamos gerar a saída para esta unidade LSTM. Esta etapa possui uma válvula de saída que é controlada pela nova memória, a saída anterior h_t-1, a entrada X_t e um vetor de polarização. Esta válvula controla quanta memória nova deve produzir para a próxima unidade LSTM.



Outros materiais interessantes

- O próximo vídeo irá se concentrar no modelo **desdobrado** conforme tentamos entendê-lo.
- O vídeo a seguir resume as RNN.
- (LSTM) dão uma solução para o problema do desaparecimento de gradiente, ao ajudar-nos a usar redes que têm dependências temporais. Elas foram propostas em 1997 por [Sepp Hochreiter] (https://en.wikipedia.org/wiki/Sepp_Hochreiter) e [Jürgen Schmidhuber] (http://people.idsia.ch/~juergen/).
- A postagem do Chris Olah sobre LSTM.
- Postagem do Edwin Chen's sobre LSTM.
- A aula do Andrej Karpathy sobre RNNs e LSTMs do curso CS231n.