Redes Neurais Artificiais

Introdução às Redes Recorrentes

INFORMAÇÃO,

TECNOLOGIA

& INOVAÇÃO

Motivação

Enquanto CNNs são boas para processar informação espacial, Redes Neurais Recorrentes (RNNs) são adequadas para lidar com informações sequenciais

- Textos escritos em sequência
- Frames em um vídeo
- Sinais de áudio em uma conversa
- Padrões de navegação em sites

Além de receber uma sequência como entrada, muitas vezes precisamos completar essa sequência (continuar a série, por exemplo 2, 4, 6, 8, 10, ...)

- Isso é bastante comum na análise de séries temporais, para prever o mercado de ações, a curva de febre de um paciente ou a aceleração necessária para um carro de corrida.

Motivação

RNNs são melhores para lidar com dados sequencias. Elas introduzem variáveis de estado para armazenar informações anteriores, junto com as entradas atuais, para determinar as saídas atuais.

- Música, fala, texto e vídeos são todos sequenciais por natureza. Se fôssemos permutá-los, eles fariam pouco sentido. A frase *cachorro morde homem* é muito menos surpreendente do que *homem morde cachorro*, embora as palavras sejam idênticas

- Os humanos interagem uns com os outros de maneira sequencial naturalmente, como pode ser visto nas discussões no Twitter, padrões de dança e debates

- Um texto pode ser visto simplesmente como uma sequência de palavras ou mesmo uma sequência de caracteres

Estados Escondidos

Redes Neurais Recorrentes (RNNs) são redes neurais com estados escondidos.

- Estados escondidos e camadas escondidas são coisas diferentes
- Camadas escondidas são camadas que ficam ocultas da visualização no caminho da camada de entrada até a camada de saída.
- Os estados escondidos são entradas para tudo o que fazemos em uma determinada etapa, e só podem ser calculados observando os dados em etapas anteriores.

Estados Escondidos

Vamos recapitular a rede neural sem estados escondidos. Vamos considerar uma MLP com uma camada escondida.

- Dado um conjunto de exemplos $\mathbf{X}$ e uma camada escondida $\mathbf{H}$ com função de ativação ϕ, a saída da camada escondida é dada por $\mathbf{H} = \phi(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h)$, com $\mathbf{W}_{xh}$ os pesos e $\mathbf{b}_h$ os bias

- $\mathbf{H}$ é então utilizada como entrada para a camada de saída, dada por $\mathbf{O} = \mathbf{H}\mathbf{W}_{hq} + \mathbf{b}_q$, com $\mathbf{W}_{hq}$ os pesos e $\mathbf{b}_q$ os bias

- Em um problema de classificação, por exemplo, podemos aplicar $softmax(\mathbf{O})$ para calcular a distribuição de probabilidade das categorias de saída

Estados Escondidos

Vamos agora utilizar a mesma notação para incluir estados escondidos na nossa rede neural. Teremos agora um conjunto de exemplos $\mathbf{X}_t$. Assim, cada exemplo (linha) em $\mathbf{X}_t$ corresponde a um exemplo no tempo t (time step t) em uma sequência de exemplos.

- Definimos agora um $\mathbf{H}_t$ como a variável escondida no tempo t. Agora, diferente da MLP, salvamos a variável $\mathbf{H}_{t-1}$ do passo anterior, e introduzimos novos pesos $\mathbf{W}_{hh}$ que nos dizem como utilizar a variável escondida do passo anterior $t-1$ no passo atual t.

- Especificamente, o cálculo da variável escondida no time step atual é determinado pela entrada no time step atual junto com a variável oculta do time step anterior:

$$\mathbf{H}_t = \phi(\mathbf{X}_t\mathbf{W}_{xh} + \boldsymbol{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h)$$

Estados Escondidos

As variáveis $\mathbf{H}_t$ e $\mathbf{H}_{t-1}$ de time steps adjacentes capturaram e retêm as informações históricas da sequência até o time step atual, assim como o estado ou memória no time step atual da rede neural

- Dessa forma, a variável escondida é chamada de estado escondido. Como esse estado usa a mesma definição do passo anterior no passo atual, seu cálculo é recursivo

- É por isso que as redes são chamadas de redes recorrentes

Uma Rede Neural Recorrente

Vamos analisar a lógica por trás de uma RNN com 3 time steps adjacentes. Em qualquer time step t, a obtenção do estado escondido é dada por:

- (i) concatenar a entrada corrente $\mathbf{X}_t$ com o estado escondido anterior $\mathbf{H}_{t-1}$; (ii) a concatenação é dada como entrada para uma camada totalmente conectada com função de ativação ϕ

- A saída da camada totalmente conectada é o estado escondido $\mathbf{H}_t$ do time step atual

- O estado escondido do time step atual t, $\mathbf{H}_t$, participará do cálculo do estado escondido $\mathbf{H}_{t+1}$ do próximo time step $t + 1$

- Além disso, $\mathbf{H}_t$ também será fornecido à camada de saída totalmente conectada para o cálculo da saída $\mathbf{O}_t$ do time step corrente t: $\mathbf{O}_t = \mathbf{H}_t\mathbf{W}_{hq} + \mathbf{b}_q$

Uma Rede Neural Recorrente

O cálculo de $\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh}$ para o estado escondido é equivalente à multiplicação de matrizes entre a concatenação de $\mathbf{X}_t$ e $\mathbf{H}_{t-1}$ e a concatenação de $\mathbf{W}_{xh}$ e $\mathbf{W}_{hh}$

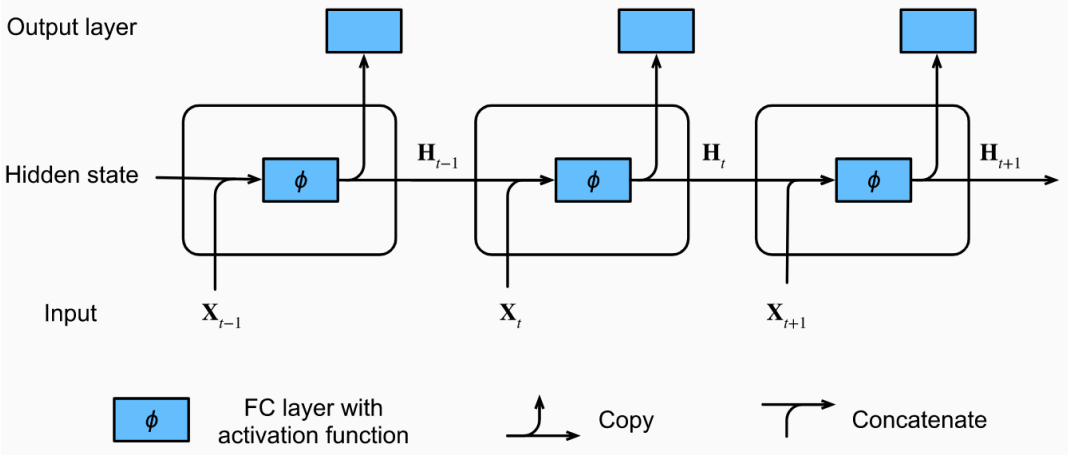


Ilustração Simples de um Modelo de Linguagem

De maneira simples, para modelos de linguagem, queremos prever o próximo token com base no token atual e no token passado

- Aqui nesse exemplo, para ilustração, vamos trabalhar no nível de predição de caracteres do texto, e um token é o próprio caractere do texto

- Vamos ilustrar uma rede para prever o próximo caractere utilizando o caractere atual e o caractere anterior

Ilustração Simples de um Modelo de Linguagem

No exemplo, vamos utilizar o texto “machine”. A entrada da rede é a sequência “machin” e o rótulo é a sequência “achine”.

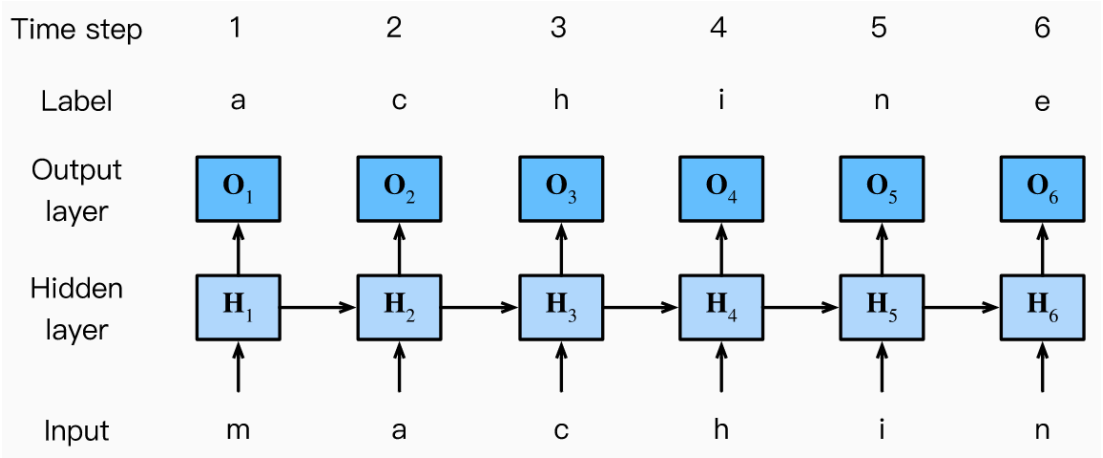


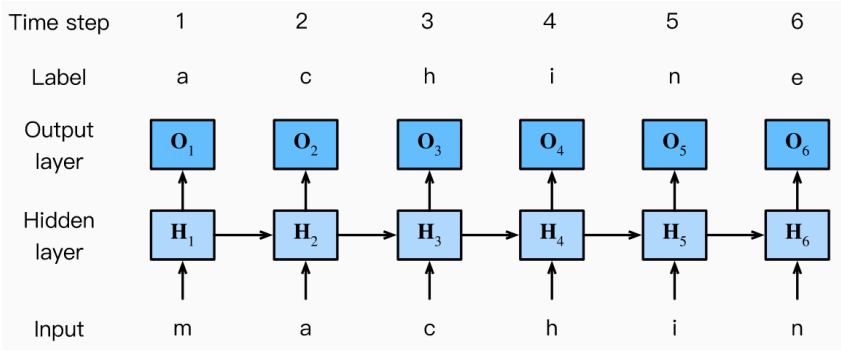
Ilustração Simples de um Modelo de Linguagem

Durante o processo de treinamento, é calculado o erro entre a saída da rede e o rótulo esperado.

- Vamos pegar como exemplo a saída $\mathbf{O}_3$. Devido à recorrência do estado escondido na camada escondida, a saída $\mathbf{O}_3$ é determinada pela sequência “m”, “a” e “c”

- Como o próximo caractere no conjunto de treino é “h”, o erro na iteração 3 depende da distribuição de probabilidade do próximo caractere gerado baseado na sequência m”, “a”, “c” e no rótulo “h” nessa iteração

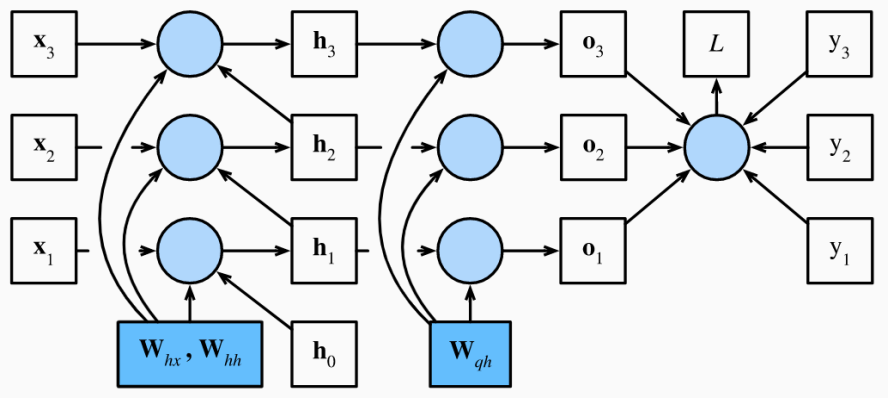
- Em situações práticas, cada token de entrada é representado por um vetor d-dimensional, e é utilizado um tamanho de batch $n > 1$. Nesse caso, a entrada $\mathbf{X}_t$ na iteração t será uma matriz $n \times d$



Back-propagation Through Time

O algoritmo Back-propagation Through Time (BPTT), utilizado para treinar uma RNN, é uma extensão do algoritmo Back-propagation tradicional. Pode ser derivado desdobrando a operação temporal da rede em uma rede feedforward em camadas, cuja topologia cresce em uma camada a cada iteração (time step)

- Exemplo: o cálculo do estado escondido na iteração 3 ($\mathbf{h}_3$) depende dos parâmetros $\mathbf{W}_{hx}$ e $\mathbf{W}_{hh}$, do estado escondido na iteração anterior ($\mathbf{h}_2$), e da entrada na iteração atual ($\mathbf{x}_3$)



Back-propagation Through Time

Considere que o conjunto de dados usado para treinar uma rede recorrente seja particionado em épocas independentes, com cada época representando um padrão temporal de interesse. Seja n_0 o início de uma época e n_1 o fim da época

- **Importante**: uma época de treinamento em uma MLP consiste em passar todo o conjunto de dados pela rede. Em uma RNN, uma época consiste em passar pela rede uma sequência de exemplos consecutivos em um intervalo de tempo

- Dada uma época em uma RNN, podemos definir uma função de custo nessa época:

$$\varepsilon_{total} = \frac{1}{2} \sum_{n=n_0}^{n_1} \sum_{j \in \mathcal{A}} e_{j,n}^2$$

- Na equação, $\mathcal{A}$ é o conjunto de índices j relacionados aos neurônios da rede para os quais saídas desejadas são especificadas, e $e_{j,n}$ é o erro calculado para o neurônio

Back-propagation Through Time

- Inicialmente, uma fase de propagação é executada, passando na rede os dados no intervalo (n_0, n_1). Todos os dados no intervalo são salvos: os estados da rede (pesos sinápticos) e as saídas desejadas

- Uma fase de retropropagação é então executada sobre esses dados do passado, de maneira a calcular os gradientes locais, para todos os j ∈ $\mathcal{A}$ e $n_0 < n \leq n_1$:

$$\delta_{j,n} = -\frac{\partial \varepsilon_{total}}{\partial v_{j,n}}$$

- O cálculo dos gradientes locais é executado da seguinte maneira:

$$\delta_{j,n} = \begin{cases} \varphi'(v_{j,n}) e_{j,n} & \text{para } n = n_1 \\ \varphi'(v_{j,n}) \left[e_{j,n} + \sum_{k \in \mathcal{A}} w_{jk} \delta_{k,n+1} \right] & \text{para } n_0 < n < n_1 \end{cases}$$

Back-propagation Through Time

Uma vez finalizada a fase de retropropagação, e tendo calculados todos os gradientes locais, um ajuste é aplicado no peso sináptico w_{ji} do neurônio j:

$$\Delta w_{ji} = -\eta \frac{\partial \varepsilon_{total}}{\partial w_{ji}} = \eta \sum_{n=n_0+1}^{n_1} \delta_{j,n} x_{i,n-1}$$

- η é a taxa de aprendizado e $x_{i,n-1}$ é a entrada aplicada à i-ésima sinapse do neurônio j na iteração $n-1$

- Comparando o BPTT com o aprendizado batch do back-propagation convencional, a diferença básica é que, no BPTT, é como se as saídas desejadas fossem utilizadas para os neurônios em muitas camadas, porque a camada de saída atual é replicada muitas vezes quando desdobramos o comportamento temporal da rede

Truncated Back-propagation Through Time

Em situação práticas, é minimizada a soma dos erros quadráticos em cada iteração

$$\varepsilon_n = \frac{1}{2}\sum_{j\in\mathcal{A}} e_{j,n}^2$$

- Como no back-propagation em modo padrão-padrão (online ou estocástico), os pesos sinápticos são ajustados a cada instante n

- Para isso, salvamos os dados de entrada e os estados da rede apenas para um número fixo de iterações h, chamado de profundidade de truncamento. Qualquer informação anterior a h iterações no passado é considerada irrelevante e pode, portanto, ser ignorada

Truncated Back-propagation Through Time

No algoritmo Trucated Back-propgation Through Time, BPTT(h), o gradiente local no neurônio j é definido como:

$$\delta_{j,l} = -\frac{\partial \varepsilon_l}{\partial v_{j,l}} \qquad \text{para todo } j \in \mathcal{A} \text{ e } n - h < l \leq n$$

- O cálculo dos gradientes locais é executado da seguinte maneira:

$$\delta_{j,l} = \begin{cases} \varphi'(v_{j,l}) e_{j,l} & \text{para } l = n \\ \varphi'(v_{j,l}) \sum_{k \in \mathcal{A}} w_{jk,l} \delta_{k,l+1} & \text{para } n - h < l < n \end{cases}$$

Truncated Back-propagation Through Time

Uma vez finalizada a fase de retropropagação, e tendo calculados todos os gradientes locais até a iteração $n - h + 1$, um ajuste é aplicado no peso sináptico w_{ji} do neurônio j:

$$\Delta w_{ji,n} = \eta \sum_{j=n-h+1}^{n} \delta_{j,l} x_{i,l-1}$$

- Comparando o BPTT(h) com o BPTT, vemos que o cálculo do erro é feito apenas na iteração corrente n. Por isso não precisamos manter o registro dos valores passados das saídas desejadas

Perguntas?