AULA 9
PRÁTICA DE DEEP-LEARNING
PROCESSO DE DESENVOLVIMENTO

1. Objetivos

- Apresentar o processo de desenvolvimento de um sistema de inteligência artificial baseado em RNAs profundas (deep-learning).
- Apresentar o conceito de parâmetros e hiperparâmetros de uma RNA.
- Apresentar as etapas do processo de desenvolvimento de RNAs profundas.
- Apresentar alguns aspectos da etapa de preparação de dados.
- Apresentar alguns aspectos da configuração de RNAs profundas.
- Apresentar alguns detalhes de como treinar a RNA (número de épocas, taxa de aprendizado e formas de usar os dados de treinamento).

2. Parâmetros e Hiperparâmetros

- Em uma RNA podemos diferenciar entre parâmetros e hiperparâmetros.
- **Parâmetros de uma RNA** $\Rightarrow$ são os valores calculados durante o treinamento.
  - Pesos das ligações $\Rightarrow \mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \mathbf{W}^{[3]},...., \mathbf{W}^{[L]}$
  - Vieses dos neurônios $\Rightarrow \mathbf{b}^{[1]}, \mathbf{b}^{[2]}, \mathbf{b}^{[3]},...., \mathbf{b}^{[L]}$
- **Hiperparâmetros de uma RNA** $\Rightarrow$ consistem em grandezas, características ou métodos que escolhemos ao configurar uma RNA:
  - Tipos de funções de ativação;
  - Número de camadas $\Rightarrow L$;
  - Número de neurônios em cada camada $\Rightarrow n^{[l]}$, para $l = 1,...,L$;
  - Função de custo;
  - Taxa de aprendizagem $\Rightarrow \alpha$;
  - Método de otimização para calcular os parâmetros da RNA;
  - Número de épocas de treinamento;
  - Outros.

- Uma vez escolhida a configuração da RNA o cálculo dos seus parâmetros é feito durante o treinamento e os resultados podem ser bons ou ruins.

- A eficiência da RNA depende das escolhas dos hiperparâmetros e dos dados utilizados para o treinamento.

- Admitindo que os dados usados para treinamento são adequados, então, surge o problema de como obter os hiperparâmetros da melhor forma possível.

- A eficiência da RNA depende muito dos hiperparâmetros que escolhemos e o processo de escolha desses hiperparâmetros é totalmente empírico e iterativo.

- Veremos posteriormente formas de ajustar os hiperparâmetros para obter uma RNA com o desempenho desejado.

3. Processo de desenvolvimento de uma RNA

- **O processo de desenvolvimento de uma RNA deep-learning consiste das seguintes etapas:**

  1. Definição do problema e obtenção dos dados de treinamento;
  2. Escolha de uma métrica para medir a eficiência da RNA;
  3. Preparação dos dados;
  4. Configuração da RNA;
  5. Codificação da RNA;
  6. Treinamento da RNA;
  7. Ajuste dos hiperparâmetros;
  8. Análise dos resultados.

- Esse processo é altamente iterativo e consiste de um ciclo como o mostrado na Figura 1 $\Rightarrow$ nesse processo repetimos principalmente as etapas 4 a 8 repetidamente até atingirmos a eficiência desejada ou concluirmos que o problema não tem solução satisfatória.

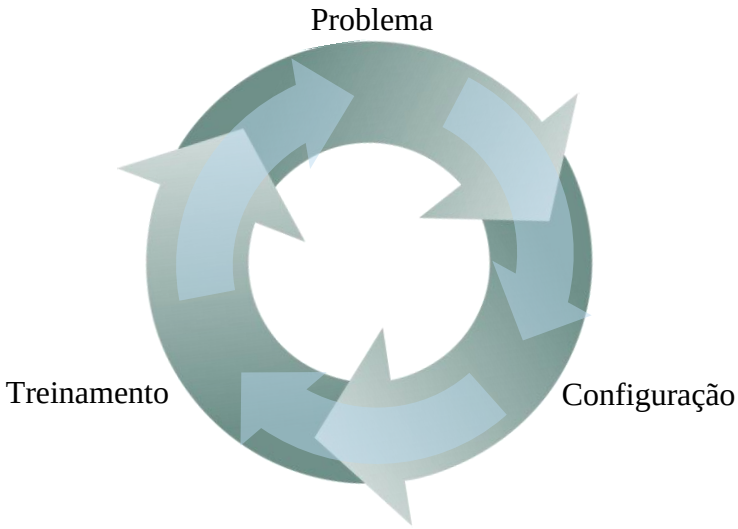


Figura 1. Processo iterativo de solução de um problema usando uma RNA deep-learning (Adaptado de Andrew Ng, deeplearning.ai).

4. Definição do problema

- Vamos considerar que temos um problema que queremos resolver e decidimos usar uma RNA para solucioná-lo.

- Para resolver um determinado problema com uma RNA devemos estabelecer os dados de treinamento e o objetivo da RNA.

- **Objetivo da RNA:**
  - A primeira coisa que devemos fazer é definir o que a RNA deve calcular (prever).
  - Definir o tipo de problema, como por exemplo:
    - Classificação binária;
    - Classificação multiclasse;
    - Regressão escalar ou vetorial (ajuste de função);
    - Outros.

- **Dados de treinamento:**
  - A disponibilidade de dados de treinamento é usualmente a parte mais limitante nas aplicações de RNAs;
  - Obtenção dos dados de treinamento é a parte mais difícil, demorada e de maior custo de todo o processo de desenvolvimento de uma RNA;
  - Devemos garantir que as saídas podem ser previstas pelos dados de entradas $\Rightarrow$ ou seja, garantir que os dados de treinamento contém informação suficiente para que a RNA aprenda a relação entre as entradas e as saídas.

Importante ⇒ somente é possível é possível aprender algo que está presente no conjunto de dados utilizado no treinamento da RNA.

5. Escolha de uma métrica para medir a eficiência da RNA

- Para controlar o treinamento e verificar a eficiência e o desempenho de uma RNA temos que saber o que queremos alcançar.

- Para medir a eficiência da RNA temos que definir uma métrica ⇒ existem diversos tipos de métricas possíveis. Algumas são as seguintes:
  - Exatidão;
  - Erro absoluto médio;
  - Precisão e revocação (“precison/recal”);
  - Pontuação F1 (“F1 score”);
  - Outras.

- Exemplos de utilização de métricas:
  - Problemas de classificação binária balanceado ⇒ exatidão;
  - Problemas de classificação não balanceado ⇒ precisão/revocação ou pontuação F1;
  - Problemas de classificação de múltiplas classes ⇒ exatidão;
  - Problema de ajuste de função ⇒ erro absoluto médio.

- A métrica não precisa ser igual à função de custo utilizada para treinar a RNA ⇒ mas a escolha da métrica está relacionada com a função de custo, ou seja, com o que se deseja otimizar durante o treinamento da RNA.

- Veremos métricas com mais detalhes posteriormente.

6. Preparação dos dados

Divisão do conjunto de dados

- **Os dados devem ser divididos em 3 conjuntos diferentes:**
  - **Conjunto de dados de treinamento** ⇒ é o conjunto de dados utilizado no treinamento da RNA;

- **Conjunto de dados de validação** ⇒ é o conjunto de dados utilizado para monitorar o treinamento da RNA;
- **Conjunto de dados de teste** ⇒ é o conjunto de dados utilizado para testar a RNA após o treinamento.

➢ **Para que dividir os dados? Para que servem esses 3 conjuntos de dados?**

- Uma RNA é treinada usando o conjunto de treinamento.
- Durante o treinamento a RNA é avaliada usando o conjunto de validação.
- Uma vez que o modelo está treinado, ele é testado no conjunto de testes.
- O conjunto de validação é usado durante o treinamento para evitar avaliar o desempenho da RNA usando os mesmos dados utilizados para o treinamento.
- Para garantir a eficiência da RNA, a avaliação do seu desempenho é realizada com dados que ela nunca tenha utilizado ⇒ é muito importante comparar o desempenho da RNA nos dados utilizados no treinamento e nos dados de teste.
- O objetivo final é obter uma RNA capaz de generalizar o modelo, ou seja, uma RNA que apresenta um bom desempenho com dados nunca vistos.

➢ **Para que ter conjuntos de validação e teste diferentes?**

- Tanto o conjunto de validação e teste não são usados diretamente para treinar a RNA.
- O conjunto de validação é usado para monitorar o treinamento da RNA e muitas decisões que tomamos para melhorar o desempenho da RNA são baseadas no desempenho da RNA no conjunto de validação ⇒ ou seja, o conjunto de validação é usado para ajustar os hiperparâmetros da RNA.
- O ajuste dos hiperparâmetros da RNA é de fato uma forma de treinamento da RNA, realizada pela pessoa que está desenvolvendo o modelo.
- Toda vez que alteramos os hiperparâmetros de uma RNA durante o seu treinamento, baseado no seu desempenho no conjunto de validação, alguma informação desse conjunto é incorporada na RNA.
- Durante o treinamento ajustamos e otimizamos os hiperparâmetros para obter uma RNA que apresenta um bom desempenho tanto no conjunto de treinamento como no de validação, assim:
  - muita informação do conjunto de validação é incorporada aos parâmetros da RNA;
  - se queremos realmente ter uma boa avaliação da RNA temos que usar um conjunto de dados nunca visto pela RNA durante o treinamento.
- Concluindo ⇒ o conjunto de validação não pode ser considerado um conjunto nunca visto pela RNA, como é de fato o conjunto de teste.

Em muitos casos é comum usar somente o conjunto de validação, ou somente o conjunto de teste para avaliar a RNA.

- **Alguns aspectos importantes:**
  - Antes de dividir os dados de treinamento entre os 3 conjuntos, deve-se misturá-los aleatoriamente para eliminar qualquer tendência que pode existir na forma com que os dados foram originalmente ordenados.
  - Se estamos desenvolvendo uma RNA para tentar prever o futuro baseado em dados passados, como por exemplo, previsão de tempo, comportamento da bolsa de valores etc, então, nesse caso não se deve rearranjar aleatoriamente os dados porque a sequência temporal é importante. Nesse caso, os dados de teste devem ser de datas posteriores às datas dos dados de treinamento.
  - Dados redundantes ⇒ não se deve ter o mesmo exemplo aparecendo duas vezes em nenhum dos conjuntos de dados, pois isso pode gerar avaliações de desempenho melhores do que de fato são na realidade.
  - Mais detalhes de como dividir os dados será visto posteriormente.

Pré-processamento dos dados

- Os dados devem ser pré-processados ⇒ é necessário definir como serão formatados e colocá-los em tensores.
- A forma como os dados são formatados depende muito do tipo de dado, por exemplo, imagens e textos têm suas características próprias.

- **Vetorização dos dados**
  - Todos os dados de entrada e de saída utilizados por uma RNA devem estar na forma de tensores ⇒ a colocação dos dados em tensores consiste de um processo de vetorização dos dados.
  - A colocação dos dados em tensores permite que a RNA processe-os de forma vetorizada, que é extremamente eficiente computacionalmente.
  - A colocação dos dados em tensores permite dividir os dados de treinamento em grupos menores de exemplos de forma a não exigir muita memória do computador.

- **Normalização dos dados**
  - Em geral não é adequado ter dados de entrada para uma RNA com valores absolutos grandes, ou dados que são muito heterogêneos, como por exemplo, algumas características das entradas terem valores entre 0 e 1, e outras entre 100 e 1000.
  - Para tornar o treinamento mais eficiente e mais rápido, os dados de entrada (e em geral os de saída também) devem ter as seguintes características:
    - Possuírem valores pequenos ⇒ tipicamente entre 0 e 1, ou entre –1 e 1, dependendo do tipo de problema;

  - Serem homogêneos ⇒ todos os elementos dos vetores de entrada devem possuir o mesmo intervalo de variação.

- Mais detalhes sobre normalização de dados será visto posteriormente.

➢ **Valores ausentes**

- É comum os vetores de dados de entrada ou de saída terem dados ausentes ⇒ pois nem sempre é possível obter todos os dados para todos os exemplos de treinamento.

- No caso de valores ausentes pode-se fazer o seguinte:
  - colocar zero no lugar, na condição de que zero já não esteja presente nos dados;
  - no caso de se usar a linguagem Python pode-se colocar `None`, que significa ausente;
  - cuidado ao usar `None`, pois a rede deve estar preparada para lidar com isso.

  Nesses casos, a RNA irá aprender que zero ou `None` significa que está faltando um dado e irá ignorar esse valor.

- Importante ⇒ se for esperado valores ausentes nos dados de entrada, quando a RNA estiver sendo utilizada após o treinamento e ela não tiver sido treinada para lidar com essa situação isso irá causar problemas.

- Se forem esperados dados ausentes nas entradas quando a RNA for ser utilizada, então ela deve ser treinada com esse tipo de dado ⇒ nesse caso deve-se gerar artificialmente exemplos de treinamento com valores entradas/saídas ausentes.

- Uma forma de gerar dados ausentes é copiar alguns exemplos e trocar os elementos que se espera estejam ausentes pelo "código" de dado ausente.

Engenharia de dados

➢ Em algumas situações, no lugar de fornecer à RNA dados brutos, tal como, imagens e ondas sonoras de áudio, pode ser interessante trabalhar esses dados para extrair características mais elaboradas que constituirão os dados de treinamento.

➢ Esse tipo de operação era muito comum até alguns anos atrás, quando os computadores ainda não possuíam capacidade computacional e de memória suficientes para a utilização de RNAs profundas em processamento de imagens. Nessa época os dados de entrada para uma RNA eram características extraídas das imagens, que o desenvolvedor considerava ideais para resolver o problema em questão.

➢ Atualmente, em alguns tipos de problemas ainda pode ser interessante processar os dados para extrair e selecionar características que serão utilizadas como dados de treinamento ⇒ contudo, isso está caindo em desuso porque as RNAs profundas são capazes de extrair informações dos dados brutos melhor do que qualquer ser humano.

7. Configuração da RNA

- A definição da melhor arquitetura de uma RNA para resolver um problema específico é um processo iterativo.

- O ideal é iniciar esse processo com uma RNA simples, de tamanho pequeno, cujo desempenho seja simplesmente melhor do que um resultado aleatório, que chamamos de resultado base.

  Um resultado melhor do que um resultado aleatório seria, por exemplo, para um problema de classificação binária obter uma taxa de acerto maior do que 50%, pois se aleatoriamente escolhermos uma das classes temos 50% de chance de acertar.

- Observa-se que nem sempre é possível obter uma RNA que seja melhor do que o resultado base, isso pode significar que as saídas desejadas não estão contidas nas entradas dos exemplos. Observa-se que ao tentar resolver um problema usando uma RNA estamos fazendo duas hipóteses:

  1. As saídas desejadas podem ser previstas pelos dados de entradas;
  2. Os dados disponíveis contêm informação suficiente para a RNA aprender a relação entre as entradas e as saídas.

  Pode acontecer de que essas hipóteses sejam falsas e se for esse o caso tem-se que pensar em novas soluções.

- Se esse primeiro modelo simples for capaz de ser melhor do que o resultado base então aprimoramos esse modelo segundo um procedimento que veremos posteriormente.

- Para configurar essa primeira RNA temos muitas escolhas para fazer, tais como:

  1. Número de camadas da RNA;
  2. Número de neurônios em cada camada;
  3. Funções de ativação de todas as camadas;
  4. Função de erro/custo;
  5. Método de otimização.

  Uma das formas de realizar essas escolhas é pesquisar na literatura se algum problema semelhante já foi resolvido e se for o caso utilizar o mesmo modelo de RNA, ou algum modelo semelhante.

- Posteriormente veremos formas para obter uma RNA com o desempenho desejado.

Número de camadas e número de neurônios

- Não existem muitas regras para escolher o número de camadas intermediárias e o número de neurônios em cada uma das camadas intermediárias no primeiro modelo $\Rightarrow$ a experiência e a intuição são os fatores principais que podem ajudar nessa definição.

- Em geral inicia-se o processo de desenvolvimento usando uma RNA simples com uma única camada intermediária.

- Uma indicação para definir o número de camadas e de neurônios em uma RNA está associada ao número de dados do conjunto de treinamento $\Rightarrow$ **o número de parâmetros de uma RNA deve ser inferior ao número de dados total presente nos dados de treinamento**.

- Outra regra importante para definir o número neurônios das camadas intermediárias é não criar **gargalhos de informação**, ou seja, uma camada intermediária nunca pode ter menos neurônios do que a camada de saída. Em um conjunto de camadas, cada camada tem acesso somente à informação na saída da camada anterior. Se uma camada tem poucos neurônios, ela vai perder alguma informação, que não será possível de ser recuperar nas camadas posteriores.

- Os números de neurônios das camadas intermediárias dependem muito do número de entradas e de saídas dos exemplos de treinamento e em geral podemos fazer alguns testes iniciais rápidos para determinar o número mínimo adequado.

- Obviamente que o número de neurônios da camada de saída é definido pelo número de saídas dos exemplos de treinamento.

Funções de ativação

- Em relação à escolha das funções de ativação, a função de ativação da última camada impõe restrições nas saídas da RNA, portanto, a sua escolha depende do tipo de problema que queremos resolver.

  Por exemplo, a função sigmoide somente gera valores positivos entre 0 e 1, enquanto a tangente hiperbólica fornece valores entre –1 e 1, e a função ReLu gera somente valores positivos entre 0 e infinito. Na Tabela 1 são apresentas as funções de ativação indicadas para diversos tipos de problemas.

Tabela 1. Indicações para escolha da função de ativação da camada de saída e da função de custo em função do tipo de problema.

Tipo de problema	Função de ativação da última camada	Função de custo
Classificação binária	Sigmoide	Função logística (entropia cruzada)
Classificação multi-classe	Softmax	Entropia cruzada para múltiplas classes
Regressão (ajuste de função) com valores arbitrários	Linear	Erro quadrático médio
Regressão com valores entre 0 e 1	Sigmoide ou ReLu	Erro quadrático médio ou entropia cruzada
Regressão com valores entre –1 e 1	Tangente hiperbólica	Erro quadrático médio

- Em relação às funções de ativação das camadas intermediárias a escolha não é tão direta. Mas existem algumas dicas:
  - A função sigmoide pode ser usada se tanto os dados de entrada como os de saída estiverem no intervalo entre 0 a 1;
  - É regra geral usarmos a tangente hiperbólica no lugar da sigmoide se os dados de entrada e de saída estiverem no intervalo de −1 a 1;
  - Observa-se, contudo, que o uso de funções sigmoide e tangente hiperbólica nas camadas intermediárias aumenta a dificuldade de treinar a RNA, pois em geral causam problemas de saturação e de "vanishing gradients";
  - **Uma escolha melhor para a função de ativação das camadas intermediárias é a ReLu ou a leReLu, pois elas não apresentam problemas de saturação e nem de pequenos gradientes. Essas são os tipos de funções mais usadas nas camadas intermediárias.**

Função de custo

- A escolha correta da função de custo é extremamente importante, pois o algoritmo de otimização, durante o treinamento, irá minimizar essa função de custo, portanto, se ela não estivar muito bem relacionada como problema o desempenho final da RNA não será bom.
- A função de custo deve ser adequada ao tipo de problema que estamos resolvendo.
  - Para problemas comuns, tais como, de classificação, regressão e previsão de sequências, existem regras simples para definir a função de custo correta ⇒ a Tabela 1 apresenta as funções de custo indicadas para diversos tipos de problemas.
  - Somente quando estamos trabalhando em problemas realmente novos podemos ser obrigados a desenvolver novas funções de custo.
- Observa-se novamente, que a função de custo não precisa ser igual à métrica que definimos para avaliar o desempenho da RNA.
- Nem sempre é possível transformar uma métrica em uma função de custo ⇒ a função de custo deve ser diferenciável e possível de ser calculada para um único exemplo, para ser possível usar o método do gradiente descendente para treinar a RNA.
  - Métricas cujas derivadas apresentam descontinuidades, como por exemplo, uma métrica definida como sendo a soma dos valores absolutos dos erros entre as previsões da RNA e as saídas desejadas, possui derivada não contínua ⇒ portanto, não é adequada para ser usada como função de custo.
  - Métricas que dependem de valores médios entre as previsões e as saídas desejadas, tais como, precisão/revocação e pontuação F1 não são possíveis de serem calculadas para um único exemplo ⇒ portanto, não tem como se tornar uma função de custo.

Inicialização dos parâmetros

- Como o método do Gradiente descendente é um método iterativo é necessário inicializar os parâmetros da RNA de alguma forma.
- Podemos inicialmente pensar em duas formas de inicializar os parâmetros de uma RNA:
  - Inicialização com zero ou uma constante qualquer;
  - Inicialização com números aleatórios.
- Existem diversos problemas que podem aparecer durante o treinamento da RNA se os parâmetros forem inicializados de forma inadequada $\Rightarrow$ cuidado deve ser tomado para inicializar os parâmetros de uma RNA.
- Existem métodos eficientes para inicialização dos parâmetros, veremos isso posteriormente.

8. Codificação da RNA

- Após termos definido a configuração da RNA devemos codificá-la para iniciar o seu treinamento.
- Praticamente ninguém desenvolve o seu próprio código para implementar RNAs $\Rightarrow$ na prática utiliza-se ferramentas de desenvolvimento já existentes e disponíveis.
- O uso de ferramentas de desenvolvimento facilita muito o desenvolvimento de RNAs, pois fornecem toda a estrutura necessária para configurar, codificar, treinar e testar uma RNA.
- Uma das grandes vantagens de usar uma ferramenta de desenvolvimento é de não precisarmos implementar o algoritmo de retro-propagação para treinar a RNA.

9. Treinamento da RNA

- Para iniciar o treinamento de uma RNA temos que definir os seguintes hiperparâmetros:
  1. Número de épocas;
  2. Taxa de aprendizagem;
  3. Forma de utilização do conjunto de dados de treinamento;
  4. Método de otimização.

Número de épocas

- Como visto uma época consiste no processamento de todos os dados de treinamento e a consequente atualização de todos os parâmetros da RNA.

- No treinamento da RNA temos que definir o número de épocas que desejamos utilizar para treinar a RNA.

- Não existe nenhuma fórmula para definir o número de épocas necessário para o treinamento eficiente de uma RNA profunda.

- O número de épocas necessário é um valor muito empírico e temos que realizar alguns testes para conseguir defini-lo.

- A única forma que temos para verificar se o treinamento está sendo eficiente é monitorar a função de custo ⇒ se a função de custo estiver diminuindo, então, a RNA está aprendendo, caso contrário, a RNA não está sendo capaz de aprender.

- Existem algumas formas automáticas para finalizar o treinamento sem precisar definir o número de épocas, que consiste em monitorar a função de custo e se o custo não diminuir após um certo número de épocas pré-estabelecido, então o treinamento é interrompido.

- Temos que tomar cuidado ao impor um critério de parada do treinamento automático, pois o número de épocas necessário para o treinamento está muito relacionado à taxa de aprendizado utilizada ⇒ ou seja, a função de custo pode ter parado de diminuir em razão de uma escolha errada da taxa de aprendizagem e não porque a RNA já está treinada.

- **Para definir um critério de parada do treinamento no Keras é necessário usar uma classe de `callback`.**

- Um `callback` é uma função aplicada durante o estágio de treinamento da RNA, que pode ser usada para monitorar um estado interno ou estatísticas do modelo durante o treinamento. Uma lista de `callbacks` é passada para o método `fit`.

- O código a seguir define um critério de parada usando a classe `EarlyStopping`. Nesse exemplo se o valor da função de custo não diminuir após cinco épocas o treinamento é parado.

```
# Importa classe EarlyStopping do Keras
from tensorflow.keras.callbacks import EarlyStopping

# Define critério de parada e treina o modelo
parada = EarlyStopping(patience=5)
history = rna.fit(x_train, y_train, epochs=100, callbacks=[parada])
```

onde `patience` é o número de épocas para parar o treinamento se o valor da função de custo não diminuir.

- Existem muitas outras funções de `callback` e muitos outros parâmetros na função `EarlyStopping` ⇒ para mais detalhes ver documentação do Keras.

Taxa de aprendizado

- O treinamento de uma RNA termina, em princípio, quando a função de custo atinge o seu valor mínimo e os seus parâmetros convergem para um determinado valor e não se alteram mais, mesmo se o treinamento continuar.

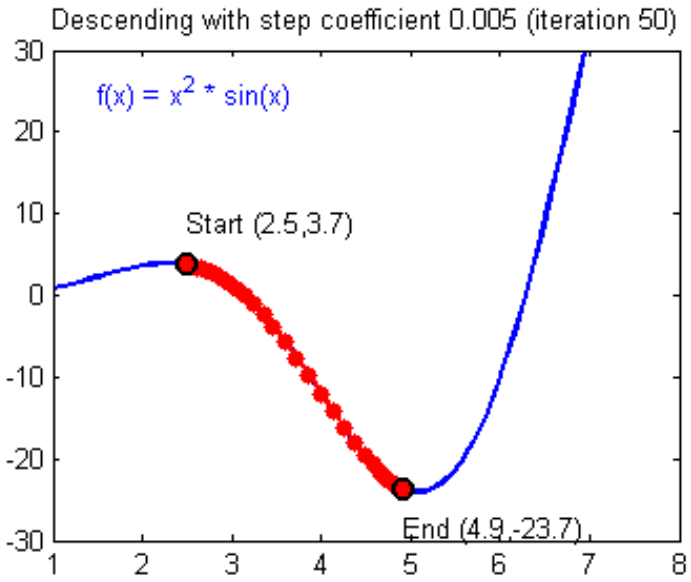
- A taxa de aprendizado controla o quão rápido ou quão lento é o treinamento da RNA.

  - Em princípio devemos usar uma taxa de aprendizado pequena para garantir convergência dos parâmetros, mas não pode ser muito pequena, senão o tempo de treinamento se torna inviável.

  - Uma taxa de aprendizado grande pode causar oscilação no valor da função de custo e a consequente demora para os parâmetros da RNA convergirem.

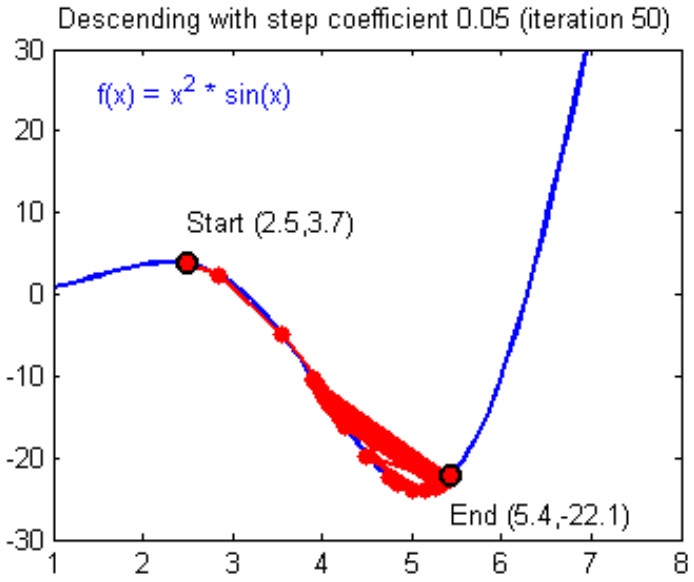
  - Uma taxa de aprendizado muito grande pode causar falha do treinamento, ou seja, a RNA não aprende os dados de treinamento e a função de custo no lugar de diminuir aumenta.

- Na Figura 2 é apresentado um exemplo de aplicação do método do gradiente descendente para determinar o mínimo de uma função, que é um processo semelhante ao de treinamento de uma RNA.



⇐ Nesse caso é utilizada uma taxa de aprendizado pequena (0,005) e o processo de convergência ocorre de forma satisfatória.

sgd.gif



⇐

Nesse caso é utilizada uma taxa de aprendizado grande (0,05) e o processo fica oscilando sem convergir e sem obter o valor mínimo da função.

sgd_bad.gif

Figura 2. Exemplo de aplicação do método do gradiente descendente com diferentes taxas de aprendizado: (a) taxa de aprendizado pequena; (b) taxa de aprendizado grande (Adam Harley, deeplearning.ai).

- Em princípio é uma boa prática diminuir a taxa de aprendizado na medida em que avançamos no processo de otimização para eliminar possíveis problemas de divergência.

- Para diminuir a taxa de aprendizado em função do andamento do treinamento existem algumas fórmulas:

$$\alpha = \alpha_0 (0{,}95)^t \tag{1}$$

$$\alpha = \frac{\alpha_0}{1 + \beta t} \tag{2}$$

$$\alpha = \frac{\alpha_0}{\sqrt{t}} \tag{3}$$

onde:

α_0 = taxa de aprendizado de referência, em geral usa-se 0,2;

β = constante de decaimento;

t = número da época do treinamento, que varia de um ao valor máximo pré-estabelecido.

- No Keras a taxa de aprendizado padrão é um valor constante igual a 0,01.

- Podemos definir qualquer valor para a taxa de aprendizado na etapa de configuração do otimizador, utilizando o seguinte código:

```
# Importa classe de otimizadores do keras
from tensorflow.keras import optimizers

# Define taxa de aprendizado do otimizador
sgd = optimizers.SGD(lr=0.1)
rna.compile(loss='binary_crossentropy', metrics=['accuracy'],
            optimizer=sgd)
```

onde `lr` é a taxa de aprendizagem desejada.

- O Keras está preparado para usar o modelo da equação (2) para diminuir a taxa de aprendizado em função do número de épocas. Isso pode ser feito da seguinte forma:

```
# Define taxa de aprendizado e seu decaimento
sgd = optimizers.SGD(lr=0.01, decay=1e-3)
rna.compile(loss='binary_crossentropy', metrics=['accuracy'],
            optimizer=sgd)
```

onde `lr` é α_0 e `decay` é a constante de decaimento β, ambos da equação (2).

- Existem outras formas de implementar o decaimento da taxa de aprendizado no Keras do TensorFlow. Essas formas podem ser vistas no manual do TensorFlow no link https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules.

Formas de utilização do conjunto de dados de treinamento

- Como vimos, o processo de otimização da função de custo, para obtenção dos parâmetros da RNA, é realizado repetindo-se sequencialmente quatro etapas:

1. Execução da RNA para todos os dados de treinamento, de forma a calcular as saídas previstas pela RNA;
2. Calculo da função de custo para todos os dados de treinamento;
3. Calculo do gradiente da função de custo em relação a todos os parâmetros da RNA;
4. Atualização dos parâmetros da RNA na direção oposta ao gradiente;
5. Repetição das etapas 1 a 4 até a obtenção de um valor desejado para a função de custo, ou os parâmetros da RNA convergirem.

- Nesse processo uma época de treinamento consiste das etapas 1 a 4.

- Nesse caso, a atualização dos parâmetros da RNA é feita somente após o cálculo do gradiente da função de custo para todos os exemplos de treinamento ⇒ isso pode ser um grande problema, ou até mesmo inviável, se tivermos uma grande quantidade de exemplos de treinamento, por exemplo, algo da ordem de 100 mil ou 1 milhão, o que não é incomum.

- Para evitar problemas de memória de computador, o processo de otimização pode ser alterado de forma a atualizar os parâmetros da RNA após o processamento de apenas alguns exemplos dos dados de treinamento ⇒ com essa alteração o processo de otimização passa a ser o seguinte:

  1. Escolha aleatória de um conjunto de exemplos dentro dos dados de treinamento;
  2. Execução da RNA para esse conjunto de exemplos;
  3. Cálculo da função de custo para esses exemplos;
  4. Cálculo do gradiente da função de custo para esses exemplos em relação a todos os parâmetros da RNA;
  5. Atualização dos parâmetros da RNA na direção oposta ao gradiente;
  6. Repetição das etapas 1 a 5 até varrer todos os exemplos do conjunto de dados de treinamento;
  7. Repetição das etapas 1 a 6 até a obtenção de um valor desejado para a função de custo, ou os parâmetros da RNA convergirem.

- Nesse processo uma época de treinamento consiste na execução das etapas 1 a 6.

- Esse processo é chamado na literatura de **Gradiente Descendente Estocástico**, em razão da divisão dos dados de treinamento em conjuntos menores ser feita de forma aleatória.

- Ressalta-se que existe uma pequena confusão entre os nomes dados para os métodos de otimização em função de como os dados de treinamento são divididos:

  - Uso de todos os dados de uma única vez ⇒ **Gradiente Descendente** ou **Gradiente Descendente de Batelada**
  - Uso de um único exemplo de cada vez ⇒ **Gradiente Descendente Estocástico**
  - Divisão do conjunto de dados em partes menores ⇒ **Gradiente Descendente Estocástico** ou **Gradiente Descendente de Mini-batelada**.

- Nota-se que essa confusão é irrelevante, pois o que importa é entender o que é de fato realizado.

- Comparando esse novo processo com o original, a única diferença é que dividimos os dados de treinamento em pequenos conjuntos (bateladas) e processamos cada um desses conjuntos separadamente.

- Uma pergunta que surge: **Como dividir os dados de treinamento nas mini-bateladas? ⇒** Veremos a resposta para essa pergunta posteriormente.

Métodos de otimização.

- Existem muitas variantes do método do Gradiente Descendente para a obtenção dos parâmetros de uma RNA.

- Algumas dessas variantes são as seguintes:
  - Gradiente Descendente com Momento;
  - Adam;
  - RMSprop;
  - Outros.
- Essas variantes têm como objetivo evitar mínimos locais da função de custo e acelerar o processo de otimização $\Rightarrow$ veremos posteriormente com detalhes alguns desses métodos.