AULA 10
MÉTRICAS – INICIALIZAÇÃO - DADOS

1. Objetivos

- Apresentar alguns tipos de métricas.
- Apresentar como dividir dos dados em conjuntos de treinamento, validação e teste.
- Apresentar algumas formas de inicializar os parâmetros de uma RNA.
- Apresentar normalização de dados.

2. Métricas para medir a eficiência da RNA

- Para controlar o treinamento e verificar a eficiência de uma RNA temos que saber o que queremos alcançar.
- Métricas são funções usadas para medir o desempenho das RNAs.
- Algumas métricas mais usuais são:
  - Exatidão;
  - Erro absoluto médio;
  - Precisão e revocação (“precison/recal”);
  - Pontuação F1 (“F1 score”);
  - Outras.
- A métrica não precisa ser igual à função de erro ou função de custo utilizada para treinar a RNA $\Rightarrow$ mas a escolha da métrica está relacionada com a escolha da função de custo, ou seja, o que se deseja otimizar durante o treinamento da RNA.

Exatidão

- A exatidão de uma solução representa o quão próxima ela está da solução correta.
- A forma de calcular a exatidão depende do tipo de problema.
- **Para um problema de classificação binária a exatidão (*acc*) pode ser calculada da seguinte forma:**

$$acc = 1 - \frac{1}{m}\sum_{i=1}^{m}\left|c^{(i)} - y^{(i)}\right| \qquad (1)$$

onde:

m = número de exemplos;

c = classe prevista pela RNA (igual a 0 ou 1);

y = saída ou classe real (igual a 0 ou 1).

- A saída de uma RNA em um problema de classificação binária é um valor real entre 0 e 1 para cada caso analisado. Assim, dada a saída, $0 < \hat{y} < 1$, devemos decidir em qual classe esse caso pertence e para isso fazemos o seguinte:
  - Casos são previstos como sendo da classe $c = 1$, se $\hat{y} \geq$ limiar;
  - Casos são previstos como sendo da classe $c = 0$, se $\hat{y} <$ limiar;
  - O valor do limiar é um número real entre 0 e 1, o mais usual é adotar o limar igual a 0,5.
- Observe que a exatidão definida pela equação (1) representa a fração de exemplos classificada corretamente.

➢ Para escolhermos a métrica exatidão no Keras basta incluir essa informação na etapa de compilação da RNA, como já visto na Aula 7 para um problema de classificação binária, ou seja:

```
from tensorflow.keras import optimizers
rna.compile(optimizer='SGD', loss='binary_crossentropy',
            metrics=['accuracy'])
```

Erro absoluto médio

➢ O erro absoluto médio representa o quão longe a solução obtida está da solução correta.

➢ A forma de calcular o erro absoluto médio depende do tipo de problema.

➢ Em problemas de ajuste de **ajuste de funções** ⇒ um elemento do conjunto de treinamento consiste de um vetor $\mathbf{x}^{(i)}$, de dimensão $(n_x, 1)$, e a sua saída correspondente $\mathbf{y}^{(i)}$ é um vetor de números reais de dimensão $(n_y, 1)$.

➢ Nesse caso o erro absoluto médio (*mae*) é definido por:

$$mae = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{n_y}\left|y_j^{(i)} - \hat{y}_j^{(i)}\right|\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})\right\|_1 \qquad (2)$$

onde $\|\mathbf{v}\|_1$ representa a norma 1 do vetor $\mathbf{v}$, ou seja:

$$\|\mathbf{v}\|_2 = |v_1| + |v_2| + \ldots |v_{n_y}| \quad (3)$$

- Para escolhermos a métrica erro absoluto médio no Keras basta incluir essa informação na etapa de compilação da RNA, como já visto na Aula 7, ou seja:

```
rna.compile(optimizer='SGD', loss='mse', metrics=['mae'])
```

  - No comando acima a função de custo `mse` representa o erro quadrático médio visto na Aula 3, que é normalmente utilizada para problemas de ajuste de funções.

- Podemos também utilizar mais de uma métrica para avaliar a nossa RNA ⇒ para calcular tanto a exatidão como e erro absoluto médio a compilação deve ser realizada da seguinte forma:

```
rna.compile(optimizer='SGD', loss='mse', metrics=['accuracy','mae'])
```

Precisão e revocação ("precision/recal")

- **Para problemas de classificação onde o número de exemplos de uma classe é desbalanceado não se deve usar a exatidão como métrica.**

- Por exemplo, um problema de identificação de pacientes com câncer.

  - Nesse exemplo queremos identificar se um paciente tem ou não câncer, a partir de dados de exames médicos.

  - Isso consiste em um problema de classificação binária, onde:

    - $y = 1 \Rightarrow$ paciente com câncer;
    - $y = 0 \Rightarrow$ paciente não tem câncer.

  - Para resolver esse problema é usada uma RNA e o resultado final após o treinamento da RNA é, por exemplo, uma exatidão de 99%, ou seja, a RNA acertou 99% dos diagnósticos e errou 1%.

  - Sabendo que somente 0,5% dos pacientes que fazem os exames têm de fato câncer ⇒ isso pode ser considerado um bom resultado?

  - Obviamente que esse não é um bom resultado e obviamente a exatidão não é uma boa métrica para esse tipo de problema.

- Problemas onde uma das classes aparece raramente em relação a outra a exatidão não é uma boa métrica.

- Então, como resolver esse problema? Que métrica usar?

  A solução é usar as métricas precisão e revocação.

- Na Tabela 1 é apresentado um quadro dos possíveis acertos e erros das previsões da RNA. Observa-se que as previsões podem ser erradas tanto na **classificação de positivo (*y* = 1)**, quanto na **classificação de negativo (*y* = 0)**.

Tabela 1. Acertos e erros das previsões de uma RNA em um problema de classificação binária.

		Classe real	
		1	0
Classe prevista	1	Positivo Verdadeiro (*PV*)	Positivo Falso (*PF*)
	0	Negativo Falso (*NF*)	Negativo Verdadeiro (*NV*)

- As definições de *PV*, *NV*, *NF* e *NP* são as seguintes:
  - Previsto 1, real 1 $\Rightarrow$ Positivo Verdadeiro (*PV*);
  - Previsto 0, real 0 $\Rightarrow$ Negativo Verdadeiro (*NV*);
  - Previsto 0, real 1 $\Rightarrow$ Negativo Falso (*NF*);
  - Previsto 1, real 0 $\Rightarrow$ Positivo Falso (*PF*).
- A partir dos erros e acertos mostrados na Tabela 1 é fácil definir o que é precisão e revocação.
- Note que as previsões corretas são somente as *PV* e *NV*.

Precisão

- No caso do exemplo de pacientes com câncer, precisão é a fração de pacientes que foram detectados como tendo câncer e que de fato tem câncer.
- A precisão é calculada de acordo com a seguinte equação:

$$\text{Precisão} = \frac{PV}{\text{número previstos como positivos}} = \frac{PV}{PV + PF} \leq 1 \qquad (4)$$

Revocação

- No caso do exemplo de pacientes com câncer, revocação é a fração dos pacientes previstos como tendo câncer em relação a todos os pacientes que de fato tem câncer.

- A revocação é calculada a partir da seguinte equação:

$$\text{Revocação} = \frac{PV}{\text{número real de pacientes que de fato tem cancer}} = \frac{PV}{PV + NF} \leq 1 \qquad (5)$$

- Podemos generalizar os conceitos de precisão e revocação para qualquer problema de classificação binária.

  Considerando que a classe rara que queremos detectar tem saída real $y = 1$, então:

> **Precisão** é a relação entre a fração de previstos como sendo $y = 1$, que de fato pertencem à classe $y = 1$, em relação ao número total de previstos com pertencendo à classe $y = 1$.
>
> **Revocação** é a relação entre o número total de previstos como pertencendo à classe $y = 1$, em relação ao número total de elementos que realmente pertencem a essa classe.

- O desejado em um problema de classificação binária é ter ambos precisão e revocação altas e iguais a 1 $\Rightarrow$ mas isso nem sempre é possível. Porque?

  Porque existe um compromisso entre precisão e revocação.

- Como visto, a saída de uma RNA em um problema de classificação binária é um valor real entre 0 e 1 para cada caso analisado, que representa a probabilidade do caso pertencer à uma das classes.

- Também como já vimos, dada a saída da RNA, $0 < \hat{y} < 1$, devemos decidir em qual classe esse caso pertence, ou seja:
  - Casos são previstos como sendo da classe $y = 1$, se $\hat{y} \geq$ limiar;
  - Casos são previstos como sendo da classe $y = 0$, se $\hat{y} <$ limiar.

- **Dependendo do valor do limar utilizado teremos resultados diferentes para a precisão e a revocação $\Rightarrow$ existe um compromisso entre precisão e revocação que depende do que queremos e em função disso podemos escolher o valor do limiar.**

- No exemplo de diagnóstico de câncer, se quisermos uma previsão com precisão alta, então, uma forma de obter isso é aumentar o limiar, por exemplo, para 0,7, ou seja:
  - Casos são previstos como sendo da classe $y = 1$, se $\hat{y} \geq 0{,}7$;
  - Casos são previstos como sendo da classe $y = 0$, se $\hat{y} < 0{,}7$;
  - Dessa forma, somente prevemos que um paciente tem câncer se a probabilidade for maior do que 70%;
  - Fazendo isso teremos uma precisão maior, mas também teremos uma revocação menor.

- Por outro lado, ainda no exemplo de diagnóstico de câncer, se quisermos previsões mais seguras então podemos diminuir o limiar, ou seja:
  - Casos são previstos como sendo da classe $y = 1$, se $\hat{y} \geq 0{,}3$;
  - Casos são previstos como sendo da classe $y = 0$, se $\hat{y} < 0{,}3$;
  - Dessa forma, prevemos que um paciente tem câncer para todos os casos onde a probabilidade for maior do que 30%, ou seja, vamos errar poucos diagnósticos de câncer reais;
  - Com isso teremos maior segurança na previsão, ou seja, teremos uma revocação alta, mas uma precisão baixa.

- **Concluindo:**
  - Quanto maior o limiar, maior a precisão e menor a revocação;
  - Quanto menor o limiar, maior a revocação e menor a precisão.

- Para escolhermos as métrica PV, PF, NV e NF no Keras do TensorFlow usamos na compilação o seguinte comando:

```
rna.compile('sgd', loss='mse',
            metrics=[tf.keras.metrics.TruePositives(),
            tf.keras.metrics.TrueNegatives(),
            tf.keras.metrics.FalsePositives(),
            tf.keras.metrics.FalseNegatives()])
```

- Para escolhermos as métricas precisão e revocação no Keras do TensorFlow usamos na compilação o seguinte comando:

```
rna.compile('sgd', loss='mse', metrics=[tf.keras.metrics.Recall(),
           tf.keras.metrics.Precision()])
```

Pontuação F1 (“F1 score”)

- Uma métrica melhor, que combina a precisão com a revocação é a pontuação F1.

- A pontuação $F1$ transforma a precisão e a revocação em um único valor $\Rightarrow$ o que é muito melhor e mais conveniente.

- A pontuação $F1$ é definida por:

$$F1 = \frac{2PR}{P + R} \tag{6}$$

onde: P = precisão; e R = revocação.

- Observa-se que para a pontuação *F*1 ser alta, tanto a precisão quanto a revocação devem ser altas.
  - Note que *F*1 =1 somente se *P* e *R* forem ambos iguais a 1.
  - Note que se *P* ou *R* for igual a zero, então, *F*1 é igual a 0.
  - Pontuação *F*1 é uma forma de comparar precisão e revocação.
- Pontuação *F*1 é a melhor métrica para problemas de classificação onde o número de exemplos de uma classe **é desbalanceado.**
- O Keras do TensorFlow não possui a métrica pontuação F1, mas ela pode ser facilmente calculada de acordo com a equação (6) tendo a precisão e a revocação.
- **Observações:**
  - Existem inúmeras métricas disponíveis para serem usadas no Keras do TensorFlow. A lista das métricas existentes pode ser consultada no manual do TensorFlow no link https://www.tensorflow.org/api_docs/python/tf/keras/metrics/.
  - Existem também inúmeras funções de custo disponíveis para serem usadas no Keras do TensorFlow. A lista das funções de custo existentes pode ser consultada no link https://www.tensorflow.org/api_docs/python/tf/keras/losses.
  - Outras métricas e funções de custo, inclusive a métrica pontuação F1, estão também disponíveis no módulo Addons do TensorFlow. Esse módulo precisa ser instalado antes de ser usado. Mais informação sobre esse módulo pode ser obtida no link https://medium.com/tensorflow/introducing-tensorflow-addons-6131a50a3dcf.

3. Como dividir os dados nos conjuntos de treinamento, validação e teste

- Como visto na aula anterior, para poder monitorar a função de custo e a métrica durante o treinamento com dados que não são usados no treinamento, deve-se usar um conjunto de validação.
- Existem alguns princípios básicos que devem ser seguidos para dividir os dados de treinamento entre os conjuntos de treinamento, validação e teste.

1. **O tipo de dados do conjunto de treinamento deve ser o mesmo dos conjuntos de validação e teste.**

   Por exemplo, em um problema de classificação binária de determinar se existe ou não um gato em uma imagem, as imagens devem ser obtidas de uma mesma fonte, ou seja:

   - Se as imagens do conjunto de treinamento foram obtidas da internet, então, as imagens dos conjuntos de validação e teste também devem ser obtidas da internet;

- Se as imagens do conjunto de treinamento foram obtidas de usuários usando um sistema especialmente desenvolvido para obter as imagens, então, as imagens dos conjuntos de validação e teste também devem ser obtidas dessa mesma forma.

Na área de inteligência artificial esse aspecto é chamado tecnicamente de dados com mesma distribuição de probabilidade. No caso de imagens, isso quer dizer, por exemplo, que as imagens são obtidas de forma similar, com câmeras similares, com mesma luminosidade, todas coloridas ou todas preto e branco, obtidas em mesma região do mundo, obtidas todas na internet etc.

2. **Escolher os conjuntos de validação e de teste de forma a refletir os dados que se espera utilizar no futuro quando o sistema estiver em operação.**

3. **Escolher os conjuntos de validação e de teste de forma a conter dados que são considerados importantes, no sentido de que com esses dados o sistema desenvolvido apresenta um bom desempenho.**

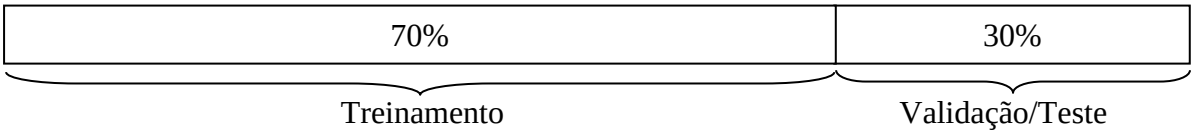
4. **Definir o conjunto de validação de forma que seja suficientemente grande para ser capaz de detectar diferenças entre os diversos modelos que eventualmente estamos testando.**

5. **Definir o conjunto de teste de forma que seja suficientemente grande para fornecer confiança nos resultados obtidos.**

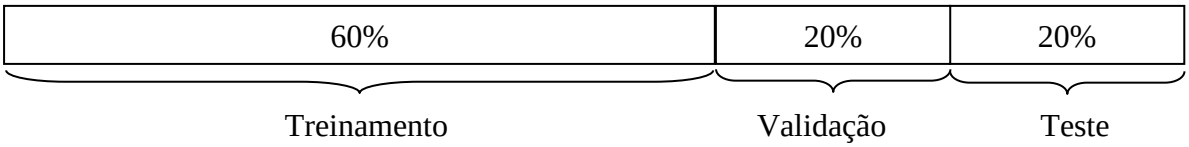
6. **Numericamente a divisão dos dados deve seguir algumas regras.**

- **Regras para dividir os dados nos três conjuntos:**

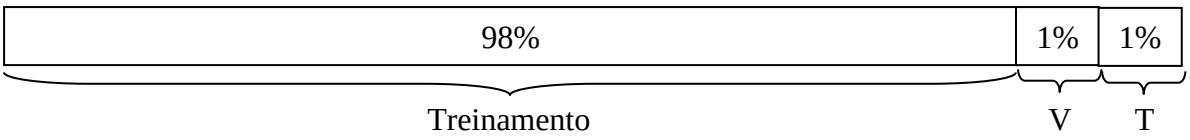
  - Para um **conjunto de dados pequeno**, contendo de 100 a 10.000 exemplos, se não tivermos o conjunto de testes, ou o de validação, então, a divisão deve ser feita da seguinte forma:



  - Para um **conjunto de dados pequeno**, contendo de 100 a 10.000 exemplos, se tivermos tanto o conjunto de validação com o de teste a divisão deve ser feita da seguinte forma:



  - Para um **conjunto de dados grande**, contendo cerca de 1 milhão ou mais de exemplos, a divisão segue a seguinte regra:



- Para um **conjunto de dados de tamanho intermediário**, contendo entre 10 mil e 1 milhão de exemplos, para definir o tamanho dos conjuntos de validação e teste pode-se utilizar uma regra de três entre as indicações para conjuntos pequeno e grande.

➢ **Existe um método mais elaborado e mais adequado para dividir os dados entre os três conjuntos, que é aplicável quando se tem poucos exemplos ⇒ "K-fold cross-validation"** (François Chollet, Deep Learning with Python, pág. 87).

➢ Como exemplo, seja um conjunto de dados com as seguintes características:

- Dado de entrada de cada exemplo consiste de um vetor linha de dimensão (1024);
- Dado de saída de cada exemplo é um escalar;
- Número de exemplos: 10.000;
- Tensor com dados de entrada de todos os exemplos ⇒ dimensão (10000, 1024);
- Tensor com dados de saída de todos os exemplos ⇒ dimensão (10000, 1);
- Conjunto de treinamento: 6000 exemplos;
- Conjunto de validação: 2000 exemplos;
- Conjunto de teste: 2000 exemplos;

Para separar os dados nos conjuntos de treinamento, validação e teste, podemos usar o seguinte código em Python:

```
# Dados de entrada dos três conjuntos
x_val = x[:2000,:]
x_test = x[2000:4000,:]
x_train = x[4000:,:]

# Dados de saída dos três conjuntos
y_val = y[:2000,:]
y_test = y[2000:4000,:]
y_train = y[4000:,:]
```

➢ Para treinar uma RNA e obter os valores da função de custo e da métrica também para o conjunto de validação com o Keras, usamos o método `fit` da seguinte forma:

```
history = rna.fit(x_train, y_train, epochs=20,
                  validation_data=(x_val, y_val))
```

- Note que somente é acrescida no método `fit` a informação do conjunto de validação.

- Nesse caso o dicionário `history` contém quatro entradas que são: os valores do custo e a da métrica para os conjuntos de treinamento e validação.
- Para acessar e salvar os valores da função de custo e da métrica para os conjuntos de treinamento e validação usa-se o mesmo procedimento da Aula 7, ou seja:

```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

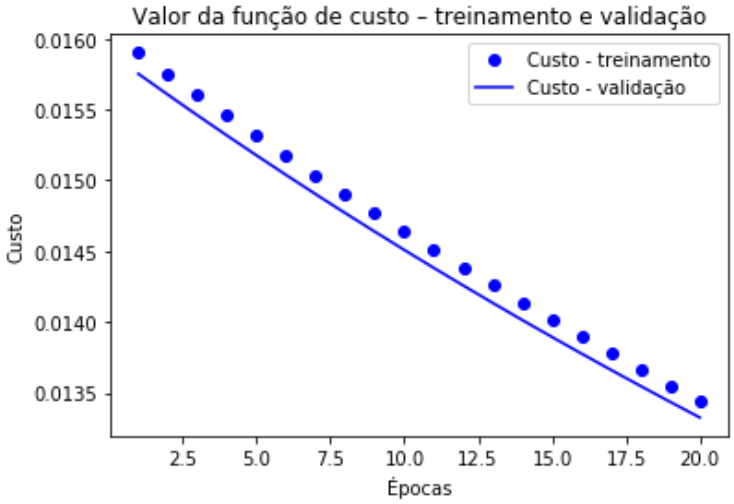
```
# Salva custos, métricas e épocas em vetores
custo = history_dict['loss']
exatidao = history_dict['accuracy']
val_custo = history_dict['val_loss']
val_acc = history_dict['val_accuracy']

# Cria vetor de épocas
epocas = range(1, len(custo) + 1)
```

➢ Para fazer o gráfico dos valores do custo para os dados dos conjuntos de treinamento e de validação em função das épocas usam-se as funções da biblioteca Python Matplotlib da seguinte forma:

```
# Importa biblioteca de funções gráficas
import matplotlib.pyplot as plt

# Gráfico dos valores de custo
plt.plot(epocas, custo, 'bo', label='Custo - treinamento')
plt.plot(epocas, val_custo, 'b', label='Custo - validação')
plt.title('Valor da função de custo - treinamento e validação')
plt.xlabel('Épocas')
plt.ylabel('Custo')
plt.legend()
plt.show()
```



- Os gráficos das métricas em função do número de épocas são realizados de forma similar, usando as variáveis `exatidao` e `val_acc`, no lugar das variáveis `custo` e `val_custo`.

4. Inicialização dos parâmetros

- Como o método do Gradiente descendente é um método iterativo é necessário inicializar os parâmetros da RNA de alguma forma.

- Existem diversos problemas que podem aparecer durante o treinamento da RNA se os parâmetros forem inicializados de forma inadequada ⇒ cuidado deve ser tomado para inicializar os parâmetros de uma RNA.

- Os parâmetros devem ser sempre inicializados com números aleatórios pequenos.

- **Importante:**
  - Nunca se deve inicializar os parâmetros com zeros, ou com uma constante qualquer ⇒ a inicialização de todos os parâmetros com um único valor transforma a RNA em uma função linear ⇒ uma demonstração desse fato se encontra no item 6 dessa aula.
  - Se inicializarmos todos os parâmetros de uma RNA com zero ou alguma outra constante, no treinamento eles passam a ter todos o mesmo valor e a RNA se comporta como se as camadas tivessem um único neurônio.

- Ressalta-se que inicializar os vieses com zeros não gera nenhum problema, porque basta que os pesos das ligações sejam diferentes para quebrar a simetria entre os neurônios.

- O valor médio dos parâmetros de cada camada deve ser pequeno para evitar saturação das funções de ativação.

- Observa-se, contudo, que, mesmo números aleatórios pequenos podem gerar problemas de saturação em uma RNA profunda.

- O problema de saturação das funções de ativação pode ser visualizado na Figura 1.

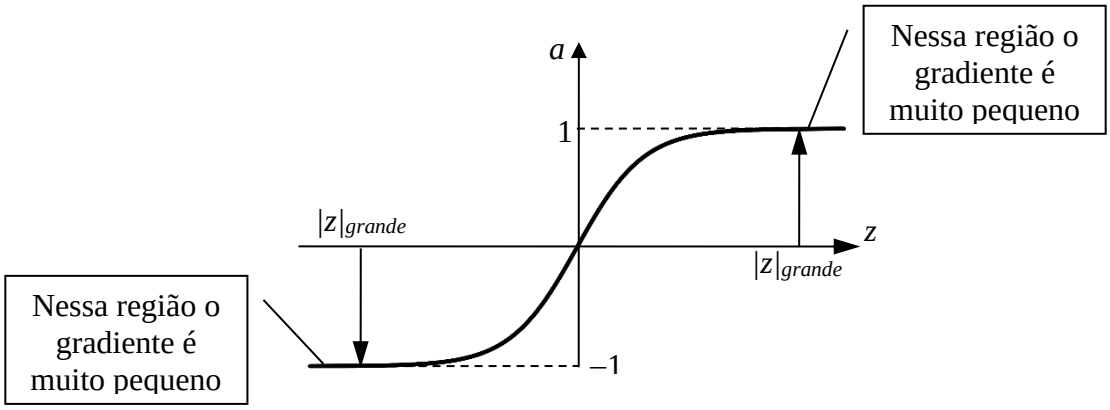


Figura 1. Problema de saturação da função de ativação tangente hiperbólica para valores grandes do estado do neurônio.

- A saturação das funções de ativação gera problemas de "vanishing gradients" e "exploding gradients" em RNA profundas.

- Problemas de "vanishing gradients" e "exploding gradients" dificultam o treinamento da RNA.

- No caso de ocorrer saturação não é possível sair dessa condição se forem usadas funções de ativação tipo sigmoide ou tangente hiperbólica ⇒ função de ativação tipo ReLu é mais robusta a problemas de saturação.

- **"Vanishing Gradients"**

  - Para RNAs profundas, em razão dos parâmetros terem valores pequenos, o valor absoluto dos gradientes se torna cada vez menor na medida em que avançamos para as camadas iniciais na retro-propagação ⇒ com isso, as camadas iniciais das RNAs são as mais lentas para serem treinadas.

  - A atualização dos pesos nas camadas iniciais é pequena e resulta em uma convergência lenta fazendo com que a otimização da função de custo seja demorada ⇒ no pior caso, os gradientes tendem a zero e o treinamento é impossível de ser realizado.

  - No caso das funções de ativação serem sigmoide, ou tangente hiperbólica, e se os pesos tiverem valor absoluto grande, os gradientes irão tender a zero impedindo o treinamento da RNA.

  - No caso das funções de ativação serem ReLu, a ocorrência de problemas de "vanishing gradients" é menor, pois somente ocorreria se os estados de todos os neurônios de uma mesma camada forem negativos, o que provocaria gradientes iguais a zero em todos os neurônios dessa camada impedindo, assim, o treinamento da RNA.

- **"Exploding Gradients"**

  - Esse caso é o oposto ao problema de "vanishing gradients".

- Se os pesos das ligações forem valores grandes, os estados dos neurônios serão também valores grandes e consequentemente as ativações serão próximas de um, ou zero, em todos os neurônios, se forem utilizadas funções de ativação sigmoide ou tangente hiperbólica, e valores muito grandes se forem utilizadas funções de ativação ReLu ⇒ quanto maior o número de neurônios nas camadas pior será o problema.

- Na medida em que a propagação para frente é realizada os estados podem aumentar cada vez mais, gerando valores de saída muito grandes e, assim, erros entre as saídas desejadas e as saídas calculadas muito grandes.

- Erros grandes geram gradientes também muito grandes, provocando grandes alterações dos parâmetros em cada iteração ⇒ isso resulta em oscilação da função de custo em torno do mínimo e a convergência do treinamento pode não ocorrer.

- Outro impacto de valores grandes de gradientes é poder provocar problemas numéricos de overflow.

➢ **Uma boa prática para evitar esses problemas de saturação, "vanishing gradients" em RNAs profundas é utilizar funções de ativação tipo ReLu ou leRelu nas camadas intermediárias. Porém, funções de ativação tipo ReLu são mais sensíveis a problemas de "exploding gradientes".**

Métodos práticos de inicialização dos parâmetros

➢ Existem métodos para inicializar os parâmetros de uma RNA de forma a evitar os problemas de saturação, "vanishing gradients" e "exploding gradients".

➢ Em todos esses métodos, os pesos das ligações de uma RNA são inicializados com **números aleatórios** com **distribuição uniforme** e com **variância inversamente proporcional ao número de neurônios da camada**.

➢ O método de inicialização mais utilizado é o **método de Xavier**, que é também chamado de **Glorot Normal**. Nesse método a inicialização dos pesos da *l*-ésima camada é feita da seguinte forma:

$$\mathbf{W}^{[l]} = \text{random}(n^{[l]}, n^{[l-1]})\sqrt{\frac{1}{n^{[l-1]}}} \tag{7}$$

➢ Outras formas de inicializar parâmetros podem ser usadas. Para funções de ativação ReLu a inicialização dos pesos da *l*-ésima camada pode ser feita por:

$$\mathbf{W}^{[l]} = \text{random}(n^{[l]}, n^{[l-1]})\sqrt{\frac{2}{n^{[l-1]}}} \tag{8}$$

onde random($n^{[l]}$, $n^{[l-1]}$) é uma função que gera uma matriz de números aleatórios com distribuição uniforme de dimensão ($n^{[l]}$, $n^{[l-1]}$).

➢ Ainda outro método comumente usado é inicializar os pesos da *l*-ésima camada por:

$$\mathbf{W}^{[l]} = \text{random}(n^{[l]}, n^{[l-1]})\sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}} \tag{9}$$

- No Keras a inicialização dos parâmetros é realizada na configuração da RNA ao adicionar as camadas.

- Abaixo seguem alguns exemplos de inicialização dos parâmetros usando o Keras.

```
from tensorflow.keras.layers import Dense
from tensorflow.keras import initializers

# Camada inicializada com zeros nos pesos das sinapses e vieses
rna.add(Dense(units=64, activation='relu', kernel_initializer='zeros',
              bias_initializer='zeros', input_dim=12288))

# Camada inicializada com números aleatórios de distribuição normal,
com média zero e desvio padrão 0,05 nos pesos e bias
rna.add(Dense(units=64, activation='sigmoid',
    kernel_initializer=initializers.RandomNormal(mean=0.0,stddev=0.05),
    bias_initializer=initializers.RandomNormal(mean=0.0,stddev=0.05)))

# Camada inicializada com números aleatórios de distribuição uniforme
com valores entre -0.05 e 0.05 para os pesos das sinapses.
rna.add(Dense(units=64, activation='sigmoid',
              kernel_initializer=initializers.RandomUniform(minval=
              -0.05, maxval=0.05)))

# Camada inicializada com o método de Xavier
rna.add(Dense(units=32, activation='relu',
             kernel_initializer=initializers.glorot_normal(seed=None)))
```

  - Observa-se que para inicializar os parâmetros usando outra forma que não seja a padrão, é preciso antes importar do Keras a classe de inicializadores.

- O padrão de inicialização de parâmetros no Keras é o método de Xavier para os pesos das ligações e zeros para os vieses ⇒ como essa é a forma mais eficiente de inicialização de parâmetros sugere-se não alterar esse padrão.

5. 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.

- Ter dados com valores muito grandes, ou muito heterogêneos, pode gerar problema saturação das funções de ativação e, consequentemente, problemas de "vanishing gradients" e de "exploding gradients".

- 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 $\Rightarrow$ tipicamente entre 0 e 1, ou entre −1 e 1, dependendo do tipo de problema;

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

- É uma prática comum, mas nem sempre necessária, normalizar os exemplos de treinamento de forma que cada elemento de todos os exemplos de treinamento apresente:

  - média igual a zero;

  - variância igual a um.

- Essa normalização é obtida aplicando aos exemplos de treinamento as seguintes operações:

$$\boxed{\overline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)}} \tag{10}$$

$$\boxed{\boldsymbol{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}^{(i)} - \overline{\mathbf{x}})^2} \tag{11}$$

$$\boxed{\mathbf{x}^{(i)} = \frac{\mathbf{x}^{(i)} - \overline{\mathbf{x}}}{\boldsymbol{\sigma}}} \tag{12}$$

onde:

$$\overline{\mathbf{x}} = \begin{bmatrix} \overline{x}_1 \\ \vdots \\ \overline{x}_{n_x} \end{bmatrix}_{(n_x,1)} \Rightarrow \text{vetor com as médias de cada componente dos dados de entrada.}$$

$$\boldsymbol{\sigma}^2 = \begin{bmatrix} \sigma_1^2 \\ \vdots \\ \sigma_{n_x}^2 \end{bmatrix}_{(n_x,1)} \Rightarrow \text{vetor com as variâncias de cada componente dos dados de entrada.}$$

- **Importante: os vetores de médias e variâncias usados para normalização são calculados usando-se somente os dados de treinamento. Nunca deve usar os dados de validação e teste para calcular os parâmetros usados para fazer a normalização.**

- Note que:

  - A normalização é realizada independentemente para cada elemento dos dados de entrada;

- A equação (10) representa uma divisão elemento por elemento, onde cada elemento dos vetores ($\mathbf{x}^{(i)} - \overline{\mathbf{x}}$) é dividido pela sua respectiva variância.

➢ A Figura 2 apresenta um exemplo de dados não normalizados e após a sua normalização para um caso onde os vetores de entrada têm dimensão (2, 1), ou seja, $n_x = 2$.

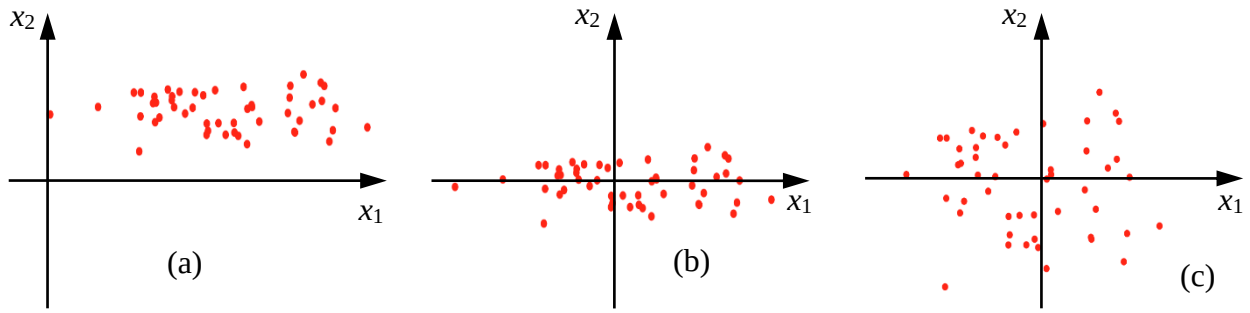


Figura 2. Processo de normalização dos dados de entrada. (a) dados originais; (b) dados com média zero; e (c) dados normalizados (adaptado de Andrew Ng, deeplearning.ai).

➢ O código abaixo mostra um exemplo de como fazer a normalização dos dados de entrada e de saída usando funções da biblioteca numpy.

```
# Cálculo da média e desvio padrão dos dados de entrada de
treinamento
meanx = x_train.mean(axis=0)
stdx = x_train.std(axis=0)

# Normalização das entradas dos dados de treinamento, validação e
teste usando as médias meanx e os desvios padrão stdx
x_train_norm = (x_train - meanx)/stdx
x_val_norm = (x_val - meanx)/stdx
x_test_norm = (x_test - meanx)/stdx

# Cálculo da média e desvio padrão dos dados de saída de
treinamento
meany = y_train.mean(axis=0)
stdy = y_train.std(axis=0)

# Normalização das saídas dos dados de treinamento, validação e
teste usando as médias meany e os desvios padrão stdy.
y_train_norm = (y_train - meany)/stdy
y_val_norm = y_val - meany)/stdy
y_test_norm = (y_test - meany)/stdy
```

- Nesse código, as entradas e as saídas de cada exemplo de treinamento são vetores linha.
- As funções `mean` e `std` calculam a média e o desvio padrão de arrays numpy.

- O tratamento dos dados de saídas em geral segue formatos diferentes, dependendo do tipo de problema.

- A grande vantagem de se normalizar os dados é obter uma função de custo que tende a ter um formato mais circular, o que facilita muito a convergência do gradiente descendente. Esse aspecto é mostrado esquematicamente na Figura 3 para um caso hipotético de uma RNA com somente um peso de ligação e um viés.

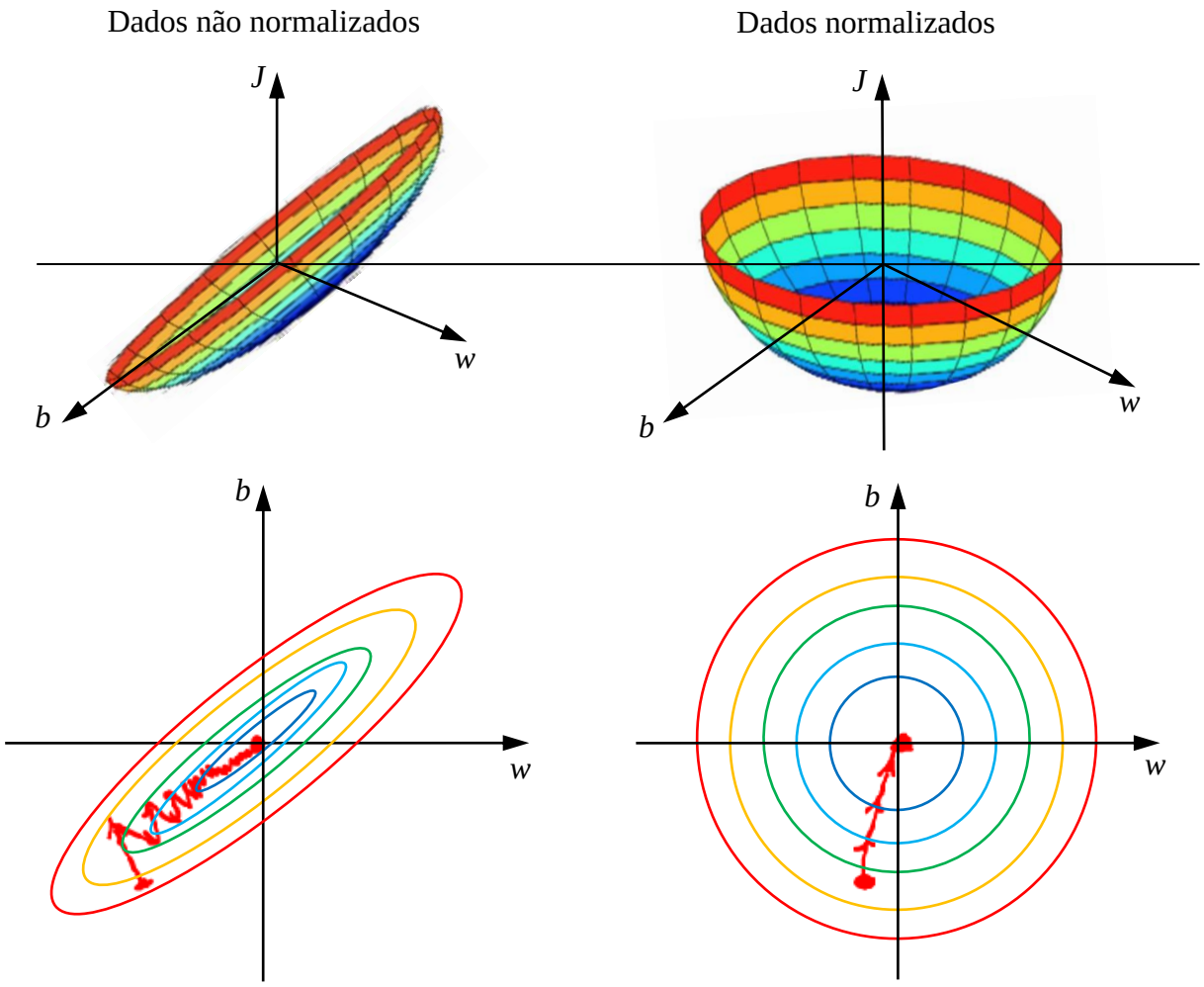


Figura 3. Função de custo *J* com dados não normalizados e com dados normalizados (adaptado de Andrew Ng, deeplearning.ai).

- Da Figura 3, observa-se que o processo de convergência do cálculo dos parâmetros da RNA, com o gradiente descendente é mais eficiente com os dados normalizados.

6. Inicialização de parâmetros com uma constante

- O que aconteceria se todos os parâmetros da RNA forem inicializados com zero?
- Considere uma RNA simples, como a mostrada na Figura 4.
  - Numero de entradas: $n_x = 2$;
  - Número de saídas: $n_y = 1$;
  - Numero de camadas: $L = 2$;
  - Numero de neurônios na camada intermediária: $n^{[1]} = 2$.

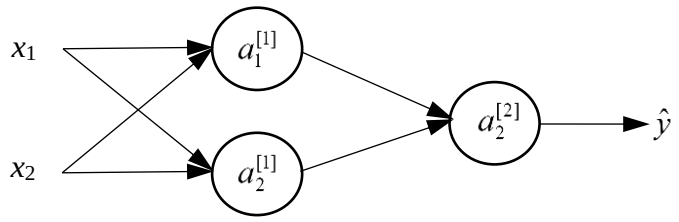


Figura 4. RNA rasa de duas entradas, uma saída e uma camada intermediária de dois neurônios.

- A propagação para frente nessa RNA é calculada por:

$$a_1^{[1]} = g^{[1]}(w_{1,1}^{[1]}x_1 + w_{1,2}^{[1]}x_2 + b_1^{[1]}) = g^{[1]}(z_1^{[1]}) \tag{13}$$

$$a_2^{[1]} = g^{[1]}(w_{2,1}^{[1]}x_1 + w_{2,2}^{[1]}x_2 + b_2^{[1]}) = g^{[1]}(z_2^{[1]}) \tag{14}$$

$$\hat{y} = a_2^{[2]} = g^{[2]}(w_1^{[2]}a_1^{[1]} + w_2^{[2]}a_2^{[1]} + b^{[2]}) = g^{[2]}(z^{[2]}) \tag{15}$$

Como todos os parâmetros são iguais a zeros, então:

$$z_1^{[1]} = z_2^{[1]} = 0 \quad \text{e} \quad a_1^{[1]} = a_2^{[1]} \tag{16}$$

- A retro-propagação é calculada usando as equações a seguir.

Derivada parcial da função de erro em relação aos estados da segunda camada:

$$dEdz^{[2]} = \left[\frac{\partial E(a^{[2]}, y)}{\partial a^{[2]}}\right]\left[\frac{dg^{[2]}(z^{[2]})}{dz^{[2]}}\right] \tag{17}$$

As derivadas parciais dos pesos das ligações de 2ª camada são dadas por:

$$dEd\mathbf{W}^{[2]} = dEdz^{[2]}\mathbf{a}^{[1]T} \tag{18}$$

Como todas as ativações dos neurônios da primeira camada são iguais ($a_1^{[1]} = a_2^{[1]}$), então:

$$dEdw_1^{[2]} = dEdw_2^{[2]} \tag{19}$$

Ou seja, as derivadas parciais da função de custo em relação aos pesos da 2ª camada serão todas iguais. Assim, após a atualização dos pesos da 2ª camada eles serão diferentes de zero ⇒ mas serão todos iguais.

Como os pesos da segunda camada são todos iguais a zero ⇒ as derivadas parciais da função de erro em relação às ativações dos neurônios da 1ª camada são todos iguais a zero:

$$dEd\mathbf{a}^{[1]} = dEdz^{[2]}\mathbf{W}^{[2]T} = \begin{cases} 0 \\ 0 \end{cases} \tag{20}$$

Como as derivadas parciais $dEd\mathbf{a}^{[1]}$ são iguais para todos os neurônios da 1ª camada e os seus estados são todos iguais a zero, então, as derivadas parciais $dEd\mathbf{z}^{[1]}$ também serão iguais a zero, ou seja:

$$dEd\mathbf{z}^{[1]} = dEd\mathbf{a}^{[1]} * \frac{dg^{[1]}(\mathbf{z}^{[1]})}{d\mathbf{z}^{[1]}} = \begin{cases} 0 \\ 0 \end{cases} \tag{21}$$

Como as derivadas parciais $dEd\mathbf{z}^{[1]}$ são todas iguais a zero ⇒ temos que as derivadas parciais da função de erro em relação aos parâmetros da 1ª camada serão todas iguais:

$$dEd\mathbf{W}^{[1]} = dEd\mathbf{z}^{[1]}\mathbf{x}^{)T} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{22}$$

Assim, após a atualização dos parâmetros da 1ª camada eles serão diferentes de zero ⇒ mas serão todos iguais.

- Quando inicializamos todos os parâmetros com zero, ou com o mesmo valor, as derivadas da função de custo serão as mesmas para todos os parâmetros da RNA de uma mesma camada ⇒ isso faz com que todos os neurônios de uma mesma camada intermediária sejam iguais e continuam assim durante todo o treinamento.

- Ressalta-se que inicializar os vieses com zeros não gera nenhum problema, porque basta que os pesos das ligações sejam diferentes para quebrar a simetria entre os neurônios.

- **Conclusões:**

  - **Se inicializarmos todos os parâmetros de uma RNA com zero, no treinamento eles passam a ter todos o mesmo valor e a RNA se comporta como se as camadas tivessem um único neurônio;**

  - **O mesmo problema acontece se todos os parâmetros fossem inicializados com uma constante diferente de zero;**

  - **A inicialização de todos os parâmetros com um único valor transforma a RNA em uma função linear.**