AULA 11
ESTRATÉGIAS DE OTIMIZAÇÃO

1. Objetivos

- Apresentar diferentes formas de utilizar os dados durante o treinamento.
- Apresentar alguns algoritmos avançados de otimização.
- Apresentar o método do corte do gradiente.
- Apresentar o método de normalização de batelada para acelerar o treinamento de RNAs com muitas camadas.

2. Formas de utilização do conjunto de dados de treinamento

- Com vimos, 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.
- Uma pergunta que surge: **Como dividir os dados de treinamento nas mini-bateladas?**
  - Na prática pode-se realizar essa divisão de qualquer forma, utilizando mini-bateladas de 1 a *m*, onde *m* é o número total de exemplos de treinamento ⇒ mas existem algumas regras para fazer essa divisão.

- Se o conjunto de dados de treinamento for pequeno (menor do que 1000), então, não se divide os exemplos em conjuntos menores, ou seja, usa-se o método do Gradiente Descendente de Batelada.

- Se os dados são obtidos de uma forma tal que a cada momento surge um dado novo, então, talvez seja interessante usar poucos exemplos de cada vez, na medida em que forem surgindo. Esse tipo de situação é comum em sistemas que monitoram a internet e atualizam o seu aprendizado na medida em que surgem novos dados. Isso tem a vantagem de não necessitar armazenar os dados de treinamento, que em alguns casos podem facilmente atingir milhões de exemplos em poucos dias.

- O número típico de exemplos em uma mini-batelada é um número que seja igual a 2^n, onde n é um número inteiro, ou seja, conjuntos de 32 (2^5), 64 (2^6), 128 (2^7), ..., 1024 (2^{10}). Em geral não se usam mini-bateladas com mais do que 1024 exemplos.

- A razão de se usar um número que seja 2^n é que a memória dos computadores são otimizadas para armazenar quantidades de números desse tipo.

➢ **Ao dividirmos os exemplos do conjunto de dados de treinamento em mini-bateladas o comportamento do processo de otimização se altera** ⇒ na Figura 1 é apresentada a variação do valor da função de custo ao longo de um processo de otimização (treinamento) para obtenção dos parâmetros de uma RNA.

➢ Quando não se divide os dados em mini-bateladas o valor da função de custo somente diminui na medida em que o processo de otimização avança.

➢ No método de mini-batelada o valor da função de custo apresenta pequenas oscilações em função das características dos exemplos contidos em cada mini-batelada ⇒ para mini-bateladas com exemplos mais "fáceis" o valor do custo pode ter uma diminuição maior, enquanto que para mini-bateladas com exemplos mais "difíceis", o valor do custo pode aumentar um pouco.

➢ No entanto a tendência do valor da função de custo é sempre diminuir com o avanço do treinamento.

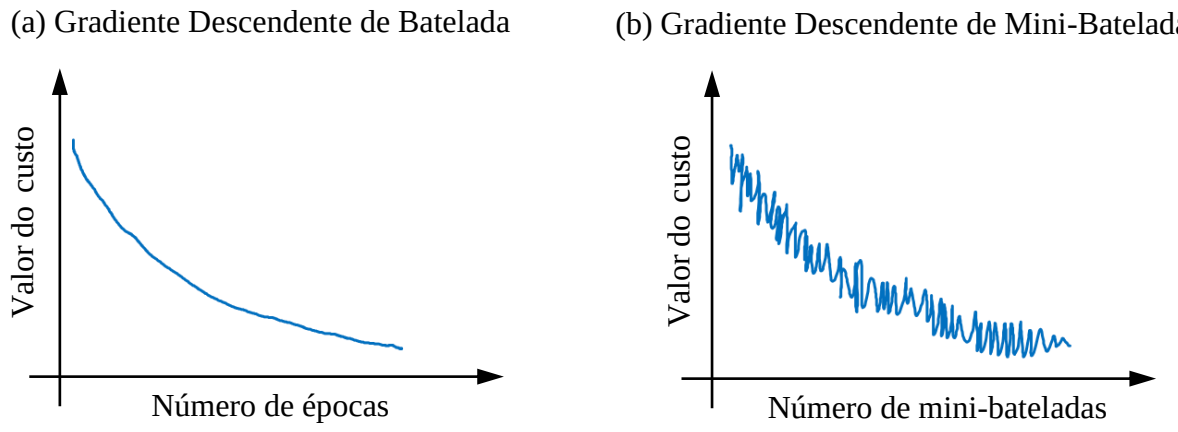

Figura 1. Comportamento do valor da função de custo durante o processo de otimização. (a) Método do Gradiente Descendente de Batelada; (b) método do Gradiente Descendente de Mini-Batelada.

- **Ressalta-se que é muito importante monitorar o valor do custo ao longo do processo de treinamento, pois um aumento do custo significa que existe algum problema, que pode ser:**
  - Incapacidade da RNA configurada de aprender os exemplos;
  - Taxa de aprendizagem inadequada;
  - Dados com problemas.

- No Keras basta definir no treinamento o número de exemplos que deseja utilizar em cada batelada usando o parâmetro `batch_size` no método `fit`, como no exemplo a seguir, que usa 64 exemplos por batelada:

```
rna.fit(x_train, y_train, epochs=20, batch_size=64)
```

  - O tamanho da batelada padrão no Keras é 32.

3. 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.

- Ressalta-se que existem outros métodos de otimização dos parâmetros de uma RNA, mas esses são os mais utilizados.

- Mais detalhes sobre esses e outros métodos de otimização e outros métodos podem ser encontrados em: (1) https://arxiv.org/pdf/1906.06821.pdf; (2) https://www.deeplearning.ai/ai-notes/optimization/ e

- Evitar mínimos locais é um aspecto muito importante no cálculo dos parâmetros de uma RNA em razão de que as funções de custo são multidimensionais e possuem inúmeros mínimos locais $\Rightarrow$ se o processo de otimização ficar preso em um mínimo local a eficiência da RNA fica limitada em relação ao que ela poderia alcançar.

- Na Figura 2 é apresentada uma função de uma única variável que possui inúmeros mínimos locais. Se o método do gradiente descendente for usado para obter o mínimo dessa função, dependendo do valor inicial do parâmetro, não é possível obter o ponto de mínimo global.

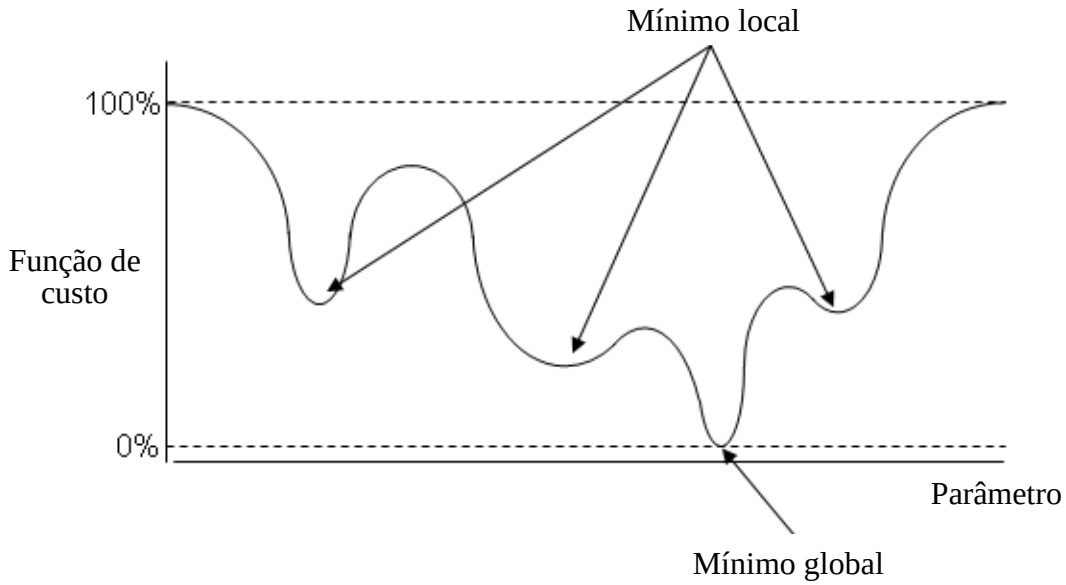

Figura 2. Função de uma única variável e seus mínimos locais (Adaptado de Lazy Programmer https://lazyprogrammer.me/deep-learning-tutorial-part-33-deep-belief/).

- Acelerar o processo de otimização é outro aspecto muito importante no cálculo dos parâmetros de uma RNA, pois dado o tamanho de algumas RNAs o seu treinamento pode levar dias, mesmo em um supercomputador.

Gradiente Descendente com Momento

- O método do Gradiente Descendente com Momento no lugar de utilizar somente o último valor do gradiente para atualizar os pesos usa também valores passados.

- O gradiente descendente com momento implementa um filtro passa baixa de 1ª ordem nos valores do gradiente para não permitir que a otimização dos parâmetros fique presa em um mínimo local.

- A principal observação que levou ao desenvolvimento dessa técnica é que a atualização dos parâmetros em cada iteração pode ser extremamente errática em razão dos valores dos gradientes mudarem frequentemente, especialmente quando a função é complexa $\Rightarrow$ a adição de um filtro suaviza o processo ao usar as tendências anteriores para ajudar a escolher o próximo passo.

- Na Figura 3 é apresentada uma função com muitas oscilações (analogia com muito mínimos locais), como é o caso dos gradientes dos parâmetros de uma RNA, e a sua versão filtrada com um filtro passa baixa.

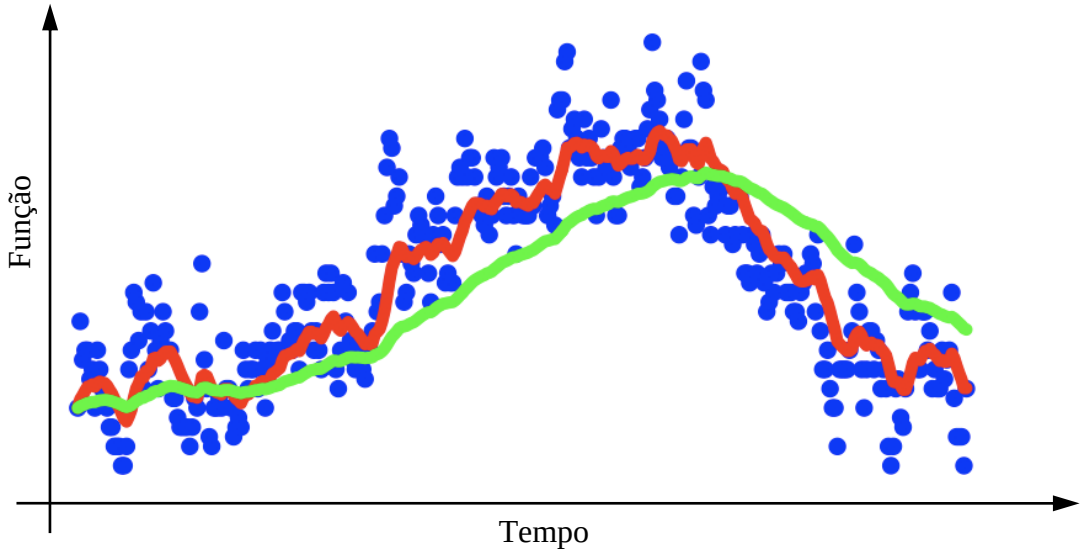

Figura 3. Função com oscilações e sua versão filtrada com filtro passa baixa. Pontos azuis: curva original; pontos vermelhos: $\beta = 0{,}5$; ponto verdes: $\beta = 0{,}9$
(Adaptado de Andrew Ng, deeplearning.ai).

- O parâmetro β na Figura 3 representa o quanto de peso é dado aos valores passados da função. Quanto maior o valor de beta mais importância é dada aos valores passados e mais suavizada fica a curva.

- Fazendo uma analogia de que cada depressão presente da curva original é um mínimo local, então, pode-se perceber que as curvas filtradas suavizam os mínimos locais e, assim, torna mais fácil passar por mínimos locais sem ficar “preso”.

- A ideia do método do gradiente descendente com momento é que o processo iterativo de otimização tenha “velocidade suficiente” para passar pelos mínimos locais sem ficar preso.

- Na Figura 4 é apresentada uma função de custo de um único parâmetro e um esquema do processo de minimização dessa função usando a técnica de momento.

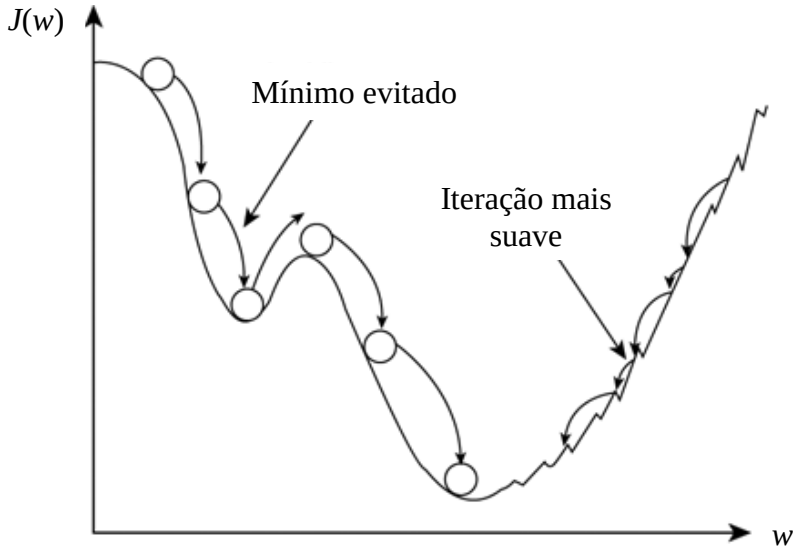

Figura 4. Uso do momento para ultrapassar mínimos locais e suavizar os passos para prevenir oscilações em regiões com grandes variações da função de custo (Adaptado de AI Game Development, http://www.yaldex.com/game-development/1592730043_ch18lev1sec4.html).

- As equações do método do gradiente descendente com momento são as seguintes:

$$\mathbf{V}_w^{[l]} = \beta_1 \mathbf{V}_w^{[l]} + (1-\beta_1) d\mathbf{W}^{[l]} \tag{1}$$

$$\mathbf{v}_b^{[l]} = \beta_1 \mathbf{v}_b^{[l]} + (1-\beta_1) d\mathbf{b}^{[l]} \tag{2}$$

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \alpha \mathbf{V}_w^{[l]} \tag{3}$$

$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \alpha \mathbf{v}_b^{[l]} \tag{4}$$

onde:

$\mathbf{V}_w^{[l]}$ = matriz dos gradientes suavizados referentes aos pesos das ligações da *l*-ésima camada;

$\mathbf{v}_b^{[l]}$ = vetor dos gradientes suavizados referentes aos vieses da *l*-ésima camada;

$\mathbf{W}^{[l]}$ = matriz de pesos das ligações da *l*-ésima camada;

$d\mathbf{W}^{[l]}$ = matriz de gradientes referentes aos pesos das ligações da *l*-ésima camada;

$\mathbf{b}^{[l]}$ = vetor de vieses da *l*-ésima camada;

$d\mathbf{b}^{[l]}$ = vetor de gradientes referentes aos vieses da *l*-ésima camada;

α = taxa de aprendizagem;

β_1 = constante (hiperparâmetro da RNA).

- Observa-se que os valores iniciais para $\mathbf{V}_w = \mathbf{v}_b = 0$.

- As constantes α e β_1 são hiperparâmetros usadas no treinamento da rede.

- Obrigatoriamente, $0 < \beta_1 < 1$, sendo que o valor padrão utilizado mais comumente é $\beta_1 = 0{,}9$.

RMSprop

- O método RMSprop é uma alteração do método do Gradiente Descendente normal para incluir uma taxa de aprendizagem variável ou adaptativa.

- A taxa de aprendizagem efetiva varia em função do valor do gradiente.

- As equações do método RMSprop são as seguintes:

$$\mathbf{S}_w^{[l]} = \beta_2 \mathbf{S}_w^{[l]} + (1-\beta_2)(d\mathbf{W}^{[l]})^2 \quad (5)$$

$$\mathbf{s}_b^{[l]} = \beta_2 \mathbf{s}_b^{[l]} + (1-\beta_2)(d\mathbf{b}^{[l]})^2 \quad (6)$$

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \alpha \frac{d\mathbf{W}^{[l]}}{\sqrt{\mathbf{S}_w^{[l]} + \varepsilon}} \quad (7)$$

$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \alpha \frac{d\mathbf{b}^{[l]}}{\sqrt{\mathbf{s}_b^{[l]} + \varepsilon}} \quad (8)$$

Operações realizadas elemento por elemento

onde:

$\mathbf{S}_w^{[l]}$ = matriz de fatores de variação das taxas de aprendizado referentes aos pesos das ligações da l-ésima camada;

$\mathbf{s}_b^{[l]}$ = vetor de fatores de variação das taxas de aprendizado referentes aos vieses da l-ésima camada;

$\mathbf{W}^{[l]}$ = matriz de pesos das ligações da l-ésima camada;

$d\mathbf{W}^{[l]}$ = matriz de gradientes referentes aos pesos das ligações da l-ésima camada;

$\mathbf{b}^{[l]}$ = vetor de vieses da l-ésima camada;

$d\mathbf{b}^{[l]}$ = vetor de gradientes referentes aos vieses da l-ésima camada;

α = taxa de aprendizagem;

β_2 = constante (hiperparâmetro da RNA).

ε = constante pequena = 10^{-8}.

- Observa-se que os valores iniciais para: $\boldsymbol{S}_w = \boldsymbol{s}_b = 0$.

- Nota-se que as potências, divisões e raízes quadradas nas equações (5), (6), (7) e (8) são realizadas elemento por elemento.

- A constante ε é incluída nas equações de atualização dos parâmetros somente para evitar uma divisão por zero, caso algum elemento de $\mathbf{S}_w^{[l]}$ ou $\mathbf{s}_b^{[l]}$ seja em algum instante igual a zero;

- As constantes α e β_2 são hiperparâmetros usados no treinamento da rede.

- Obrigatoriamente, $0 < \beta_2 < 1$, sendo que o valor padrão utilizado mais comumente é $\beta_2 = 0{,}999$.

ADAM

- O método ADAM une a técnica de momento com a taxa de aprendizado variável.

- O método ADAM é a união dos métodos do Gradiente Descendente com Momento com o método RMSprop.

- O método ADAM é mais complexo do que os métodos do Gradiente Descendente com Momento e RMSprop, realizando mais cálculo em cada iteração e, assim, necessita de mais memória de computador.

- Apesar do método ADAM exigir a realização de mais cálculos que nos outros métodos, em geral o treinamento é mais rápido porque é mais efetivo.

- As equações do método ADAM são as seguintes:

$$\mathbf{V}_w^{[l]corrigido} = \frac{\mathbf{V}_w^{[l]}}{(1-\beta_1)} \tag{9}$$

$$\mathbf{v}_b^{[l]corrigido} = \frac{\mathbf{v}_b^{[l]}}{(1-\beta_1)} \tag{10}$$

$$\mathbf{S}_w^{[l]corrigido} = \frac{\mathbf{S}_w^{[l]}}{(1-\beta_2)} \tag{11}$$

$$\mathbf{s}_b^{[l]corrigido} = \frac{\mathbf{s}_b^{[l]}}{(1-\beta_2)} \tag{12}$$

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \alpha \frac{d\mathbf{V}_w^{[l]corrigido}}{\sqrt{\mathbf{S}_w^{[l]corrigido} + \varepsilon}} \tag{13}$$

$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \alpha \frac{d\mathbf{v}_b^{[l]corrigido}}{\sqrt{\mathbf{s}_b^{[l]corrigido} + \varepsilon}} \tag{14}$$

Operações realizadas elemento por elemento

onde $\mathbf{V}_w^{[l]}$, $\mathbf{v}_b^{[l]}$, $\mathbf{S}_w^{[l]}$ e $\mathbf{s}_b^{[l]}$ são calculados da mesma forma que nos métodos do gradiente com momento e RMSprop. Os outros símbolos tem a mesma definição usada nos outros métodos.

- Nota-se que as raízes quadradas e as divisões nas equações (13) e (14) são realizadas elemento por elemento.

- As constantes α, β_1 e β_2 são hiperparâmetros usados no treinamento da rede e os valores usados para as mesmas são os mesmos que os utilizados no gradiente descendente com momento e RMSprop.

Escolha do método de otimização no Keras

- No Keras o método de otimização é escolhido na etapa de compilação da RNA usando a classe `optimizers`.

- Para usar o método do **gradiente descendente normal**, temos por exemplo:

```
from tensorflow.keras import optimizers

sgd = optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0,
                     nesterov=False)
rna.compile(loss='binary_crossentropy', optimizer=sgd)
```

  - Nesse caso a taxa de aprendizado é constante e igual a 0,01.

  - Essa configuração é o padrão para o método do gradiente descendente no Keras.

- Para usar o método do **gradiente descendente com momento** e taxa de aprendizado decrescente segundo o método de Xavier, temos por exemplo:

```
from tensorflow.keras import optimizers

sgd = optimizers.SGD(lr=0.01, decay=1e-02, momentum=0.9,
                     nesterov=True)
rna.compile(loss='binary_crossentropy', optimizer=sgd)
```

  - Observa-se que para usar o método do momento, temos que definir o parâmetro `momentum` do otimizador padrão do gradiente descendente e definir o parâmetro `nesterov` como sendo `True`.

  - O parâmetro `momentum` deve ser algum valor maior do que 0 e menor do que 1, recomenda-se usar 0,9. Esse parâmetro controla a "suavização" da função de custo, quanto maior esse valor mais suavizada é a curva.

- Para usar o método **RMSprop**, temos por exemplo:

```
from tensorflow.keras import optimizers

rms = optimizers.RMSprop(lr=0.01, rho=0.9, decay=0.0))
rna.compile(loss='binary_crossentropy', metrics=['accuracy'],
            optimizer=rms)
```

- O parâmetro `rho` deve ser algum valor maior do que 0 e menor do que 1.

- Observa-se que no manual do Keras recomenda-se usar os valores padrão do otimizador RMSprop, a menos da taxa de aprendizagem.

➢ Para usar o método **ADAM**, temos por exemplo:

```
from tensorflow.keras import optimizers

adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
                       decay=0.0)
rna.compile(loss='binary_crossentropy', metrics=['accuracy'],
            optimizer=adam)
```

- Os parâmetros `beta_1` e `beta_2` correspondem respectivamente aos parâmetros `momentum` do método do Gradiente Descendente com Momento e `rho` do método RMSprop.

- Em geral para o parâmetro `beta_2` usam-se valores próximos de 1.

4. Limitação do gradiente

➢ Como visto na Aula 10, podem ocorrer problemas de "exploding gradients" durante o treinamento de uma RNA.

➢ Uma forma de lidar com problemas de "exploding gradients" é realizar uma operação de limitação dos gradientes durante o treinamento.

➢ No Keras existem duas formas de realizar a limitação dos gradientes dos parâmetros da RNA durante o treinamento:

- Limitação da norma de todos os parâmetros da RNA em determinado valor;

- Limitação dos parâmetros da RNA entre determinados valores mínimos e máximos.

➢ Na operação de limitação da norma dos gradientes parâmetros, define-se um valor limite para a norma dos gradientes dos parâmetros e se a norma dos gradientes dos parâmetros for maior do que esse limite estabelecido, então os valores dos gradientes são alterados. A forma de alteração dos valores dos gradientes é dada por:

$$d\mathbf{W} = \frac{limiar * d\mathbf{W}}{\|d\mathbf{W}\|_2}, \text{ se } \|d\mathbf{W}\|_2 > limiar \qquad (22)$$

onde $d\mathbf{W}$ representa um vetor com todos os parâmetros da rede, *limiar* é o valor máximo pré-estabelecido para a norma dos gradientes dos parâmetros e $\|.\|_2$ é a norma 2 de um vetor.

- Na operação de limitação dos valores dos gradientes parâmetros, define-se um valor limite para os gradientes dos parâmetros e se um gradiente estiver fora desses limites é fixado igual ao valor limite de acordo com a equação (23) a seguir.

$$\begin{aligned} &\text{Se } d\mathbf{W}_k < -\,d\mathbf{W}_{limite}, \text{ então, } d\mathbf{W}_k = -\,d\mathbf{W}_{limite} \\ &\text{Se } d\mathbf{W}_k > d\mathbf{W}_{limite}, \text{ então, } d\mathbf{W}_k = d\mathbf{W}_{limite} \end{aligned} \tag{23}$$

onde $d\mathbf{W}_k$ representa o gradiente do k-ésimo parâmetro da rede e $d\mathbf{W}_{limite}$ é o valor limite definido para os gradientes dos parâmetros.

- No Keras é muito simples implementar o corte do gradiente ⇒ isso é feito durante a configuração do otimizador usando os parâmetros `clipnorm` e `clipvalue`, da seguinte forma:

```
from tensorflow.keras import optimizers

# Para limitar o valor de todos os gradientes de forma que a norma
máxima seja igual a 1
sgd = optimizers.SGD(lr= 0.01, clipnorm=1.0)

# Para limitar o valor de todos os gradientes entre -0.5 e 0.5
sgd = optimizers.SGD(lr= 0.01, clipvalue=0.5)
```

- **Observações:**

  - O método da limitação da norma dos gradientes é mais efetivo do que o método de limitar os valores dos gradientes;

  - O método de limitação da norma do gradiente é bastante eficiente para facilitar o treinamento de uma RNA, principalmente quando os dados de saída apresentam grandes variações de amplitude.

5. Normalização de Batelada ("Batch Normalization")

- A normalização de batelada não é de fato um método de otimização e sim uma alteração **na estrutura clássica das RNAs**.

- Esse método foi criado por Ioffe, S. e Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (https://arxiv.org/pdf/1502.03167.pdf).

- **Uma das grandes vantagens de usar esse método é poder usar taxas de aprendizado maiores e com isso o aprendizado é muito mais rápido.**

- A ideia básica do método é **normalizar os estados dos neurônios de uma camada** de forma a terem sempre média igual a zero e desvio padrão igual a 1, da mesma forma que normalizamos os dados de entrada de uma RNA para minimizar problemas de saturação (ver Aula 10).

- A normalização de batelada é extremamente eficiente para acelerar o processo de treinamento, pois evita problemas de saturação das funções de ativação e de "vanishing gradientes" ⇒ RNAs de muitas camadas só conseguem ser treinadas usando esse método.

- Observa-se que esse método não tem nenhuma relação com a divisão dos exemplos de treinamento em mini-bateladas.

- Na Aula 10 vimos com fazer a normalização dos dados de entrada. Somente para relembrar, as equações usadas para normalizar os dados de entrada, de forma a terem média igual a zero e variância igual a um, são as seguintes:

$$\overline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)} \tag{24}$$

$$\boldsymbol{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}^{(i)} - \overline{\mathbf{x}})^2 \tag{25}$$

$$\mathbf{x}^{(i)} = \frac{\mathbf{x}^{(i)} - \overline{\mathbf{x}}}{\boldsymbol{\sigma}} \tag{26}$$

Operações realizadas elemento por elemento

onde:

$\mathbf{x}^{(i)} \Rightarrow$ vetor do *i*-ésimo exemplo de treinamento;

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

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

- Na normalização de batelada esse mesmo tipo de normalização é realizada nos estados das camadas de neurônios ⇒ para a *l*-ésima camada, essa normalização é feita de acordo com as seguintes equações:

$$\boxed{\overline{\mathbf{z}}^{[l]} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{z}^{[l](i)}} \tag{27}$$

$$\boldsymbol{\sigma}^{[l]^2} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{z}^{[l](i)} - \bar{\mathbf{z}})^2 \tag{28}$$

$$\mathbf{z}^{[l](i)}_{normalizado} = \frac{\mathbf{z}^{[l](i)} - \bar{\mathbf{z}}^{[l]}}{\sqrt{\boldsymbol{\sigma}^{[l]^2} + \varepsilon}} \tag{29}$$

$$\tilde{\mathbf{z}}^{[l](i)} = \boldsymbol{\gamma}^{[l]}\mathbf{z}^{[l](i)}_{normalizado} + \boldsymbol{\beta}^{[l]} \tag{30}$$

Operações realizadas elemento por elemento

onde:

$\mathbf{z}^{[l](i)} \Rightarrow$ vetor de estados da camada *l* referente ao exemplo *i*;

$$\bar{\mathbf{z}}^{[l]} = \begin{bmatrix} \bar{z}_1^{[l]} \\ \vdots \\ \bar{z}_{n^{[l]}}^{[l]} \end{bmatrix}_{(n^{[l]},1)} \Rightarrow \text{vetor com as médias de cada elemento dos estados da camada } l;$$

$$\boldsymbol{\sigma}^{[l]^2} = \begin{bmatrix} \sigma_1^{[l]^2} \\ \vdots \\ \sigma_{n^{[l]}}^{[l]^2} \end{bmatrix}_{(n^{[l]},1)} \Rightarrow \text{vetor com as variâncias de cada elemento dos estados da camada } l;$$

$$\mathbf{z}^{[l](i)}_{normalizado} = \begin{bmatrix} z^{[l](i)}_{1,normalizdo} \\ \vdots \\ z^{[l](i)}_{n^{[l]},normalizado} \end{bmatrix}_{(n^{[l]},1)} \Rightarrow \text{vetor de estados normalizados dos neurônios da camada } l;$$

$$\tilde{\mathbf{z}}^{[l](i)} = \begin{bmatrix} \tilde{z}_1^{[l](i)} \\ \vdots \\ \tilde{z}_{n^{[l]}}^{[l](i)} \end{bmatrix}_{(n^{[l]},1)} \Rightarrow \text{vetor de novos estados dos neurônios da camada } l;$$

$$\boldsymbol{\gamma}^{[l]} = \begin{bmatrix} \gamma_1^{[l]} \\ \vdots \\ \gamma_{n^{[l]}}^{[l]} \end{bmatrix}_{(n^{[l]},1)} \Rightarrow \text{vetor de parâmetros de normalização dos neurônios da camada } l;$$

$$\boldsymbol{\beta}^{[l]} = \begin{bmatrix} \beta_1^{[l]} \\ \vdots \\ \beta_{n^{[l]}}^{[l]} \end{bmatrix}_{(n^{[l]},1)} \Rightarrow \text{vetor de parâmetros de normalização dos neurônios da camada } l;$$

ε = número pequeno, da ordem de 10^{-8}, incluído somente para evitar divisão por zero.

- Note que a média e a variância são calculadas para os exemplos de treinamento, para cada estado da *l*-ésima camada de forma independente.

- Na normalização de batelada o vetor de estados da camada *l* ($\mathbf{z}^{[l](i)}$) é trocado pelo vetor de estado normalizado ($\tilde{\mathbf{z}}^{[l](i)}$) para calcular a ativação dos neurônios da camada.

- **Note que se $\boldsymbol{\gamma}^{[l]} = \sqrt{\boldsymbol{\sigma}^{[l]^2} + \varepsilon}$ e $\boldsymbol{\beta}^{[l]} = \bar{\mathbf{z}}^{[l]}$, então $\tilde{\mathbf{z}}^{[l](i)} = \mathbf{z}^{[l](i)}$.**

- Observa-se que os vetores de parâmetros de normalização da *l*-ésima camada da RNA, $\boldsymbol{\gamma}^{[l]}$ e $\boldsymbol{\beta}^{[l]}$, são **"aprendidos" durante o processo de treinamento** $\Rightarrow$ dessa forma, obviamente deve-se calcular os seus gradientes e atualizá-los, da mesma forma como é feito para os pesos das ligações e os vieses.

- Como o vetor de estados normalizado possui, em princípio, média zero e desvio padrão igual a um, então os seus valores ficam dentro de um intervalo próximo de –1 a 1 $\Rightarrow$ com isso, os níveis de ativação dos neurônios não apresentam problema de saturação.

- Nota-se que a normalização de batelada pode ser realizada em todas as camadas de uma RNA, ou somente em algumas camadas.

- Na equação (30) utilizada para calcular $\tilde{\mathbf{z}}^{[l](i)}$, já existe um viés para cada estado normalizado dos neurônios da camada, que é o vetor de parâmetros $\boldsymbol{\beta}^{[l]}$, assim, os vieses $\mathbf{b}^{[l]}$ não são necessários e, portanto, são retirados (note que tanto $\boldsymbol{\beta}^{[l]}$ como $\mathbf{b}^{[l]}$ tem a mesma função e somar duas constante é a mesma coisa que somar somente uma).

- **As equações da propagação para frente em uma RNA com normalização de batelada em todas as camadas são as seguintes:**

Para a *l*-ésima camada ($l = 1,..., L$):

$$\mathbf{z}^{[l](i)} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1](i)} \tag{31}$$

$$\bar{\mathbf{z}}^{[l]} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{z}^{[l](i)} \tag{32}$$

$$\boldsymbol{\sigma}^{[l]^2} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{z}^{[l](i)} - \bar{\mathbf{z}})^2 \tag{33}$$

$$\mathbf{z}^{[l](i)}_{normalizado} = \frac{\mathbf{z}^{[l](i)} - \bar{\mathbf{z}}^{[l]}}{\sqrt{\boldsymbol{\sigma}^{[l]^2} + \varepsilon}} \tag{34}$$

Operações realizadas elemento por elemento

$$\tilde{\mathbf{z}}^{[l](i)} = \boldsymbol{\gamma}^{[l]}\mathbf{z}^{[l](i)}_{normalizado} + \boldsymbol{\beta}^{[l]} \tag{35}$$

$$\mathbf{a}^{[l]} = g^{[l]}(\tilde{\mathbf{z}}^{[l]}) \tag{36}$$

Para a primeira camada ($l = 1$), então, $\mathbf{a}^{[0](i)} = \mathbf{x}^{(i)}$.

Saídas da RNA:

$$\boxed{\hat{\mathbf{y}} = \mathbf{a}^{[L]}} \tag{37}$$

Obviamente que esses cálculos são implementados vetorizadamente nos exemplos, como visto na Aula 6.

- Na Figura 5 é mostrado o fluxo de dados em uma RNA deep-learning de L camadas com normalização de batelada.

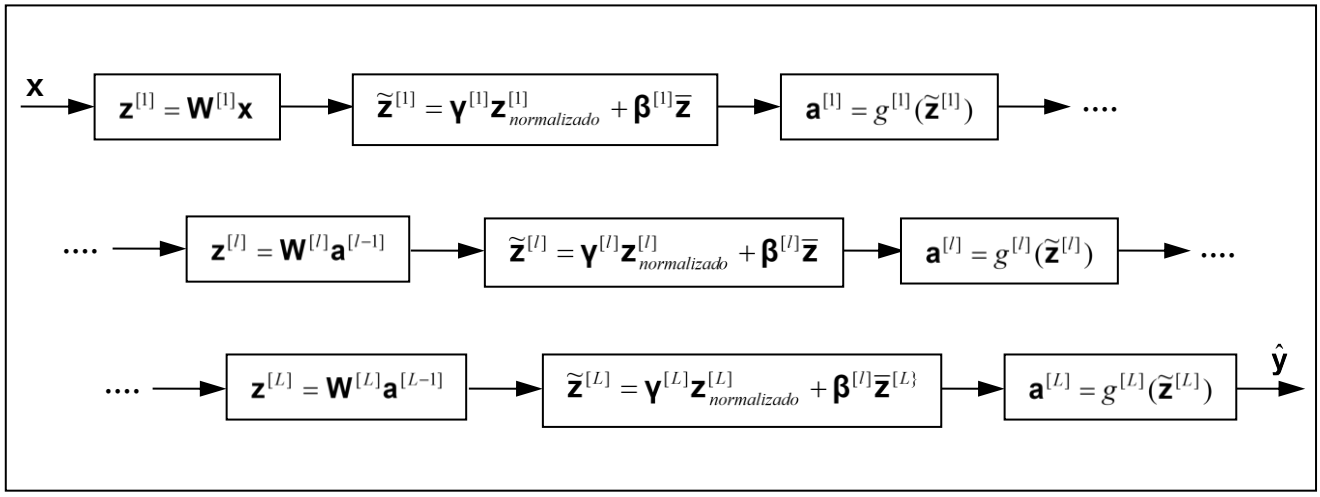

Figura 5. Fluxo de cálculo de uma RNA deep-learning de L camadas com normalização de batelada em todas as camadas. As equações (31) a (34) foram omitidas por motivo de simplicidade.

- Os parâmetros de uma RNA com normalização de batelada são os seguintes: $\mathbf{W}^{[l]}$, $\mathbf{b}^{[l]}$, $\boldsymbol{\gamma}^{[l]}$ e $\boldsymbol{\beta}^{[l]}$ para $l = 1,...,L$.

- A dimensão dos vetores $\boldsymbol{\gamma}^{[l]}$ e $\boldsymbol{\beta}^{[l]}$ é $(n^{[l]}, 1) \Rightarrow$ assim, uma RNA com normalização de batelada possui $n^{[l]}$ parâmetros a mais em cada camada do que uma RNA normal.

- No Keras a normalização de batelada funciona como se fosse outra camada da RNA que incluímos após a camada que queremos realizar a normalização de batelada.

- No trabalho original da normalização de batelada, o autor sugere incluir uma limitação do gradiente de forma que a norma dos parâmetros da camada não exceda 3,0.

- No Keras para incluir a normalização de batelada em uma camada da RNA deve-se na configuração da rede, após adicionar a camada, adicionar o processo de normalização de batelada para essa camada. Porém algumas modificações na camada original devem ser realizadas:
  - Retirar o viés da camada (parâmetro **b**);
  - Calcular a ativação dos neurônios após a camada de normalização de batelada;

- Incluir, como sugerido, a limitação do gradiente de forma que a norma dos parâmetros da camada não exceda 3,0. Observa-se que incluir essa limitação é opcional.

➢ O exemplo de código a seguir configura e compila uma rede de duas camadas intermediárias com normalização de batelada nessas camadas.

```
# Importa classes necessárias
from tensorflow.keras import models
from tensorflow.keras.layers import Dense, BatchNormalization
from tensorflow.keras.layers import Activation
from tensorflow.keras import optimizers

# Cria a instância de uma RNA
rna = models.Sequential()

# Configuração das camadas do modelo
rna.add(Dense(64, input_dim=12288, use_bias=False))
rna.add(BatchNormalization())
rna.add(Activation('relu'))
rna.add(Dense(64, use_bias=False))
rna.add(BatchNormalization())
rna.add(Activation('relu'))
rna.add(Dense(1, activation='sigmoid'))

# Configuração do otimizador e compilação da RNA
sgd = optimizers.SGD(lr=0.01, decay=1e-02, momentum=0.9,
                     nesterov=True, clipnorm=3.0)
rna.compile(loss='binary_crossentropy', metrics=['accuracy'],
            optimizer=sgd)
```

Nesse código é criada uma RNA de duas camadas intermediárias e uma de saída com as seguintes características:

- Primeira camada: 64 neurônios, função de ativação ReLu, dimensão dos dados de entrada 12.288 elementos, camada com normalização de batelada;
- Segunda camada: 64 neurônios, função de ativação ReLu, camada com normalização de batelada;
- Camada de saída: 1 neurônio, função de ativação sigmoide, camada sem normalização de batelada.
- Método de cálculo dos parâmetros: gradiente descendente com momento e com taxa de aprendizado variável segundo o método de Xavier.
- Nota-se que o padrão da camada de normalização de batelada assume que o primeiro eixo dos tensores é o eixo dos exemplos. Se esse não for o caso deve-se usar, por exemplo no caso do eixo dos exemplos ser o primeiro, a opção `axis=-1`.

➢ A classe BatchNormalization possui diversos parâmetros que podem ser configurados como desejado. Para detalhes sobre esses parâmetros consultar o manual do Keras do TensorFlow.