PSI3471 – Fundamentos de Sistemas Eletrônicos Inteligentes

Evitando mínimos locais e *overfitting*

Magno T. M. Silva e Renato Candido

Escola Politécnica da USP

1 Função de ativação – Sigmoidal

- Também conhecida como função logística

$$\boxed{\varphi(v_k^{(j)}) = \mathrm{sgm}(a\, v_k^{(j)}) = \frac{1}{1 + e^{-a\, v_k^{(j)}}}, \quad a > 0,}$$

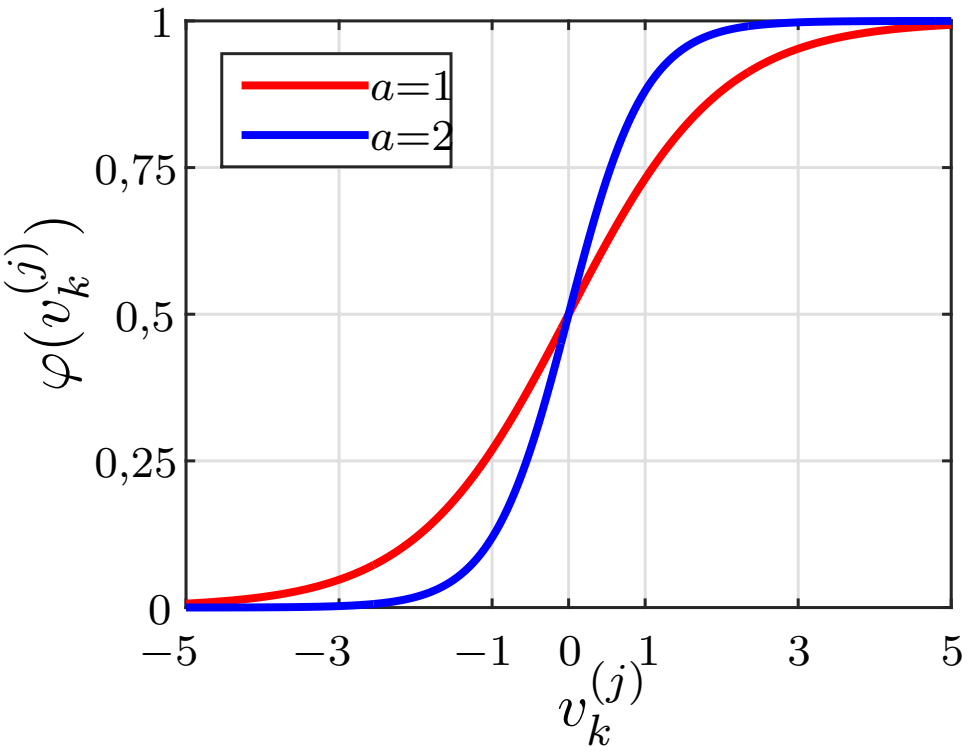
em que

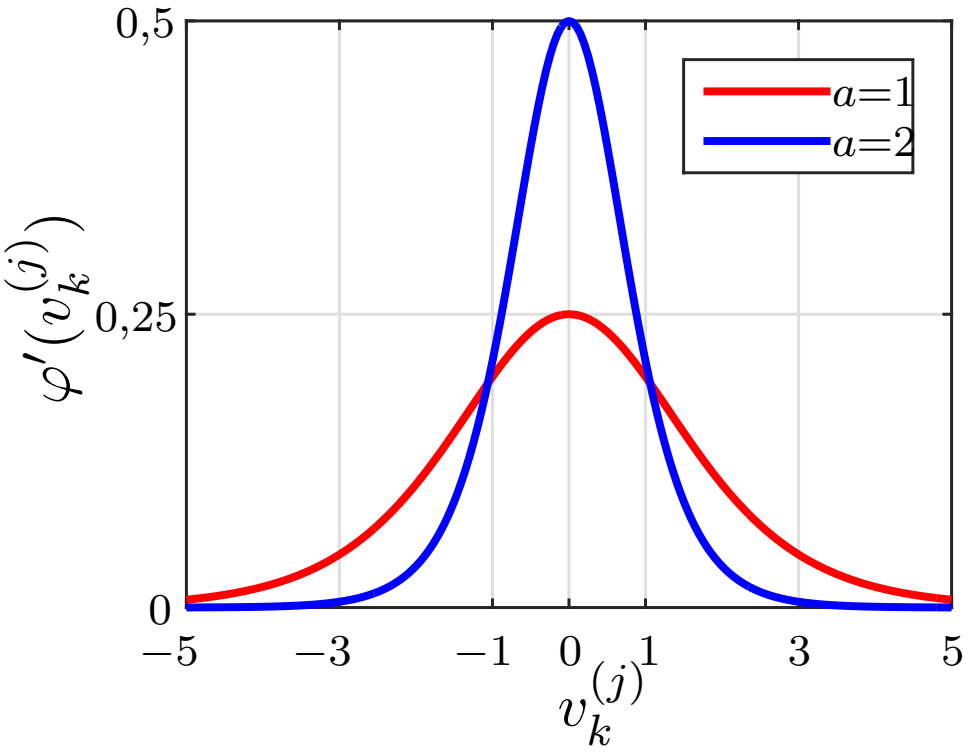
- $v_k^{(j)}$ é o resultado da combinação linear entre as entradas e os pesos do Neurônio k da Camada j
- a é um parâmetro positivo ajustável

- Derivada

$$\boxed{\varphi'(v_k^{(j)}) = \frac{d\,\mathrm{sgm}(a\, v_k^{(j)})}{dv_k^{(j)}} = \frac{a\, e^{-a\, v_k^{(j)}}}{\left[1 + e^{-a\, v_k^{(j)}}\right]^2} = a\varphi(v_k^{(j)})[1 - \varphi(v_k^{(j)})]}$$

2 Função de ativação – Sigmoidal





Função sigmoidal e sua derivada para dois valores do parâmetro a

3 Função de ativação – Tangente hiperbólica

- Definição

$$\varphi(v_k^{(j)}) = \tanh(a\, v_k^{(j)}) = \frac{e^{a\, v_k^{(j)}} - e^{-a\, v_k^{(j)}}}{e^{a\, v_k^{(j)}} + e^{-a\, v_k^{(j)}}}, \quad a > 0,$$

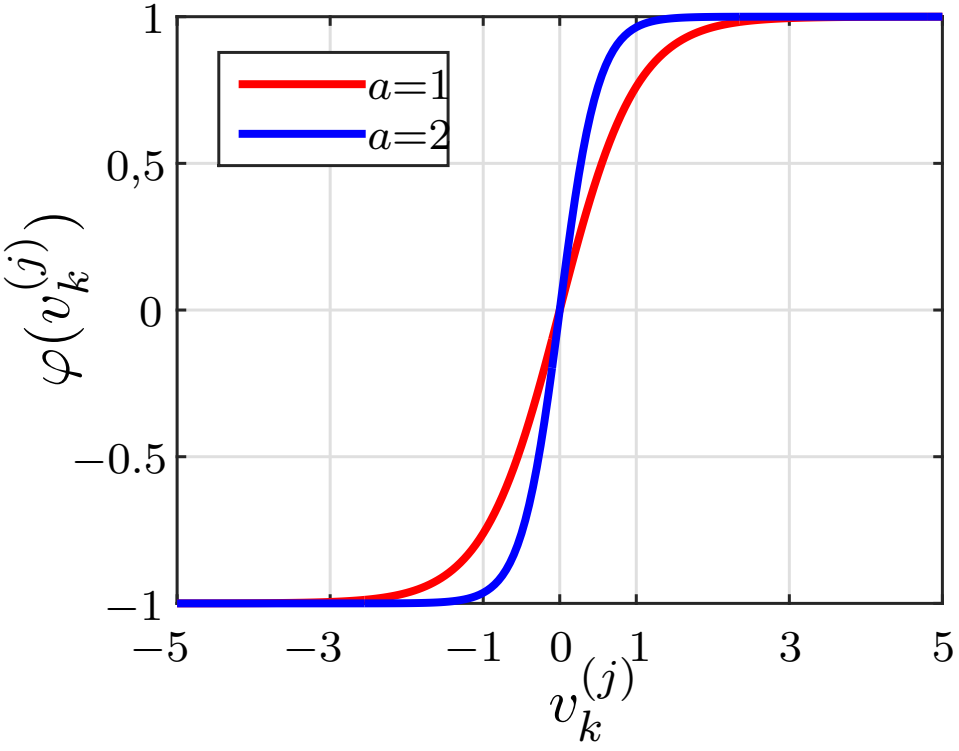
em que

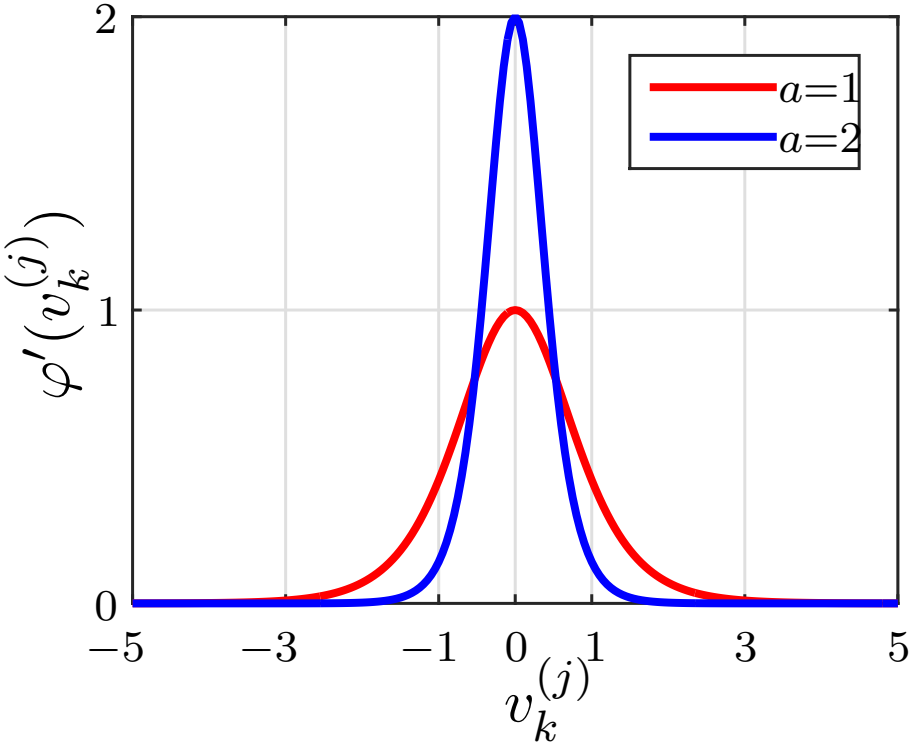
- $v_k^{(j)}$ é o resultado da combinação linear entre as entradas e os pesos do Neurônio k da Camada j
- a é uma constante positiva

- Derivada

$$\varphi'(v_k^{(j)}) = \frac{d\tanh(a\, v_k^{(j)})}{dv_k^{(j)}} = \frac{a\, e^{-a\, v_k^{(j)}}}{\left[1+e^{-a\, v_k^{(j)}}\right]^2} = a\, \left[1-\tanh^2(a v_k^{(j)})\right]$$

4 Função de ativação – Tangente hiperbólica





Função tangente hiperbólica e sua derivada para dois valores do parâmetro a

5 Função de ativação – ReLU

- A unidade linear retificada (*Rectified Linear Unit* – ReLU) é uma função de ativação dada por

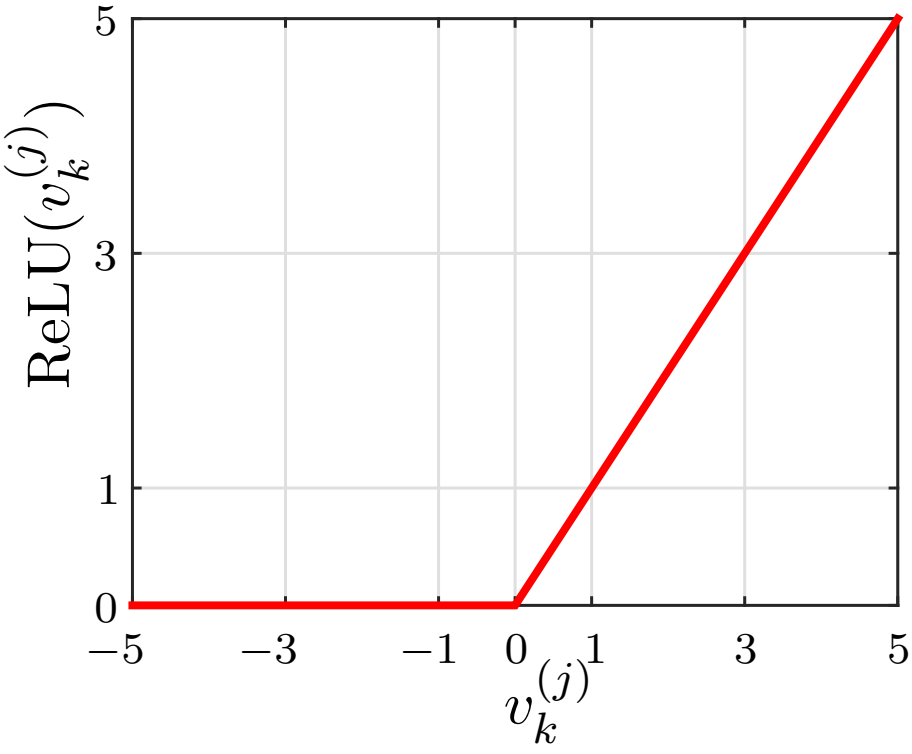
$$\varphi(v_k^{(j)}) = \text{ReLU}(v_k^{(j)}) = \max(0, v_k^{(j)}) = \begin{cases} 0, & v_k^{(j)} \leq 0 \\ v_k^{(j)}, & v_k^{(j)} > 0 \end{cases}$$

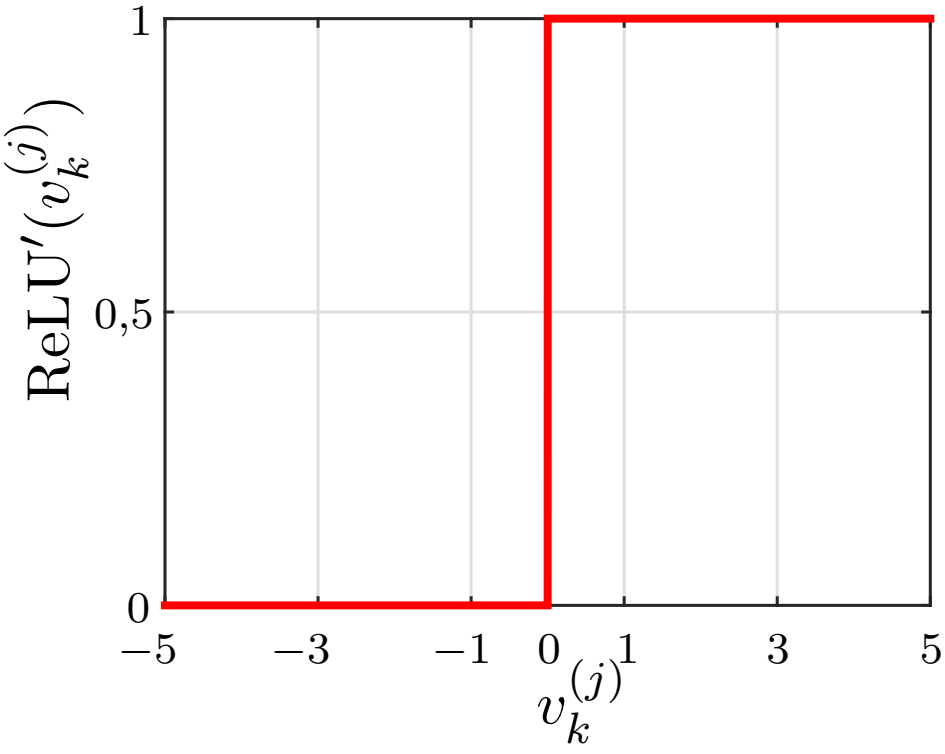
- Derivada

$$\varphi'(v_k^{(j)}) = \text{ReLU}'(v_k^{(j)}) = \begin{cases} 0, & v_k^{(j)} < 0 \\ 1, & v_k^{(j)} > 0 \\ \nexists, & v_k^{(j)} = 0. \end{cases}$$

- Não é diferenciável em $v_k^{(j)} = 0$. O valor de sua derivada em 0 pode ser arbitrariamente escolhido como 0 ou 1
- Baseada no princípio que os modelos são mais facilmente otimizados quando o seu comportamento é próximo do linear
- O treinamento de redes MLP profundas com a ReLU é mais rápido

6 Função de ativação – ReLU





Função ReLU e sua derivada (a derivada em $v_k^{(j)} = 0$ foi arbitrariamente escolhida como 0)

7 Função de ativação – ReLU

Na literatura, há diferentes variantes da ReLU, como:

1. *Softplus*;
2. *Gaussian Error Linear Unit* (GELU);
3. *Leaky rectified linear unit* (*Leaky* ReLU);
4. *Parametric rectified linear unit* (PReLU);
5. *Exponential linear unit* (ELU);
6. *Sigmoid linear unit* (SiLU).

Algumas dessas funções são diferenciáveis em todos os pontos, o que evita ter que escolher arbitrariamente o valor da derivada em $v_k^{(j)} = 0$.

8 Função de ativação – ReLU

- A ReLU apresenta vantagens como:
  1. ativação esparsa: em uma rede inicializada aleatoriamente, apenas 50% dos neurônios ocultos são ativados (saída não nula)
  2. melhor propagação do gradiente: consegue escapar de mínimos locais em comparação com funções do tipo sigmoidal
  3. computação eficiente
  4. invariante à escala: $\max(0, ax) = a \max(0, x), \ a > 0$
- Apesar disso, a ReLU é ilimitada, o que pode levar o algoritmo de treinamento à divergência.
- Neurônios com ReLU podem se tornar inativos para todas as entradas: o gradiente não é retropropagado e o neurônio "morre". Esse problema geralmente surge quando a taxa de aprendizado é muito alta e pode ser evitado usando ReLUs com vazamento

9 Função de ativação – *Softmax*

- Na classificação multiclasse, é comum considerar uma rede com N_L neurônios de saída, sendo N_L o número de classes.
- A saída esperada da rede é a ativação de apenas um dos N_L neurônios e a inativação dos $N_L - 1$ restantes
- A função *softmax* limita a saída do neurônio entre 0 e 1
- Considera-se uma normalização que faz com que a soma de todas as saídas dos neurônios seja unitária, o que faz com que o vetor saída da rede seja um vetor de probabilidades
- A função *softmax* para o k-ésimo neurônio da camada de saída é dada por

$$\boxed{\varphi(v_k^{(L)}) = \text{Softmax}(v_k^{(L)}) = \frac{e^{v_k^{(L)}}}{\sum_{\ell=1}^{N_L} e^{v_\ell^{(L)}}},}$$

em que $0 \leq \varphi(v_k^{(L)}) \leq 1$ e $\sum_{\ell=1}^{N_L} \varphi(v_\ell^{(L)}) = 1$

10 Função custo – MSE

Em problemas de regressão, é comum usar o erro quadrático médio (*Mean Squared Error* – MSE) definido por

$$J_{\text{MSE}} = \frac{1}{N_L} \sum_{\ell=1}^{N_L} e_\ell^2(n),$$

em que

$$e_\ell(n) = d_\ell(n) - y_\ell^{(L)}(n)$$

são os erros dos neurônios da camada de saída da rede.

11 Função custo – EC

Em problemas de classificação, é comum usar a entropia cruzada. Para classificação binária com $d = 0$ ou $d = 1$, a entropia cruzada é dada por

$$J_{\mathrm{EC}} = -\left[d_1(n) \ln\left(y_1^{(L)}(n)\right) + [1 - d_1(n)] \ln\left(1 - y_1^{(L)}(n)\right)\right]$$

- Vamos retomar o problema das meias-luas, com $d = 1$ para a Região A e $d = 0$ para a Região B
- Quando $y_1^{(L)}(n) \geq 0{,}5$ a rede classifica o dado como pertencente à Região A e para $y_1^{(L)}(n) < 0{,}5$ como pertencente à Região B
- Quando $d_1(n) = y_1^{(L)}(n)$, $J_{\mathrm{EC}} = 0$, que é o valor mínimo dessa função custo
- Para $d_1(n) = 1$ e $y_1^{(L)}(n) = 0{,}1$, a rede erra e $J_{\mathrm{EC}} = 1 \times \ln(0{,}1) = 2{,}3026$. A função custo tem o mesmo valor para $d_1(n) = 0$ e $y_1^{(L)}(n) = 0{,}9$ e novamente há erro de classificação

12 Função custo – EC

No caso de classificação entre N_L classes, essa função é chamada de entropia cruzada categórica e é dada por

$$J_{\mathrm{ECC}} = -\frac{1}{N_L} \sum_{\ell=1}^{N_L} d_\ell(n) \ln \left(y_\ell^{(L)}(n) \right).$$

13 Função custo – Entendendo a EC

- Vem da Teoria da Informação (Shannon, 1948) para explicar o "conteúdo" das mensagens e como elas são "corrompidas" pelos canais de comunicação
- O conceito chave é a Informação, que é a medida de aleatoriedade de uma mensagem
- Se uma mensagem é totalmente predizível, ela não contém informação
- Em contrapartida, algo muito inesperado contém uma grande quantidade de informação
- A informação é associada inversamente com a probabilidade de um evento

14 Função custo – Entendendo a EC

- A quantidade de informação de um evento aleatório x_k com probabilidade $p(x_k)$ é dada por

$$I(x_k) = \log\left(\frac{1}{p(x_k)}\right)$$

- A entropia é a medida da quantidade de informação média de um evento. Considerando o conjunto das $2K+1$ mensagens discretas com probabilidades $p_k = p(x_k)$ temos

$$H(x) = \sum_{k=-K}^{K} p_k I(x_k)$$

- Assuma agora que temos duas funções de probabilidade $\{p_k\}$ e $\{q_k\}$. A entropia relativa da distribuição de probabilidade P com relação à distribuição Q é dada por

$$D(p||q) = \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right),$$

que é conhecida como divergência de Kullback-Leibler (KL)

15 Função custo – Entendendo a EC

- Como a saída da rede neural é aproximadamente a resposta desejada no sentido estatístico, é razoável usar a divergência KL como custo
- Neste caso, $p(x)$ é a função de probabilidade alvo construída com a resposta desejada e $q(x)$ é a função de probabilidade da saída da rede
- Isso é adequado para problemas de classificação. Para N_L classes, o critério KL se torna

$$J = \sum_n \sum_k d_{n,k} \log \left(\frac{d_{n,k}}{y_{n,k}} \right),$$

em que n é o índice relacionado aos padrões de entrada e k às classes

16 Reduzindo o *overfitting* com a regularização

- Uma das maneiras de se reduzir *overfitting* é usar regularização na função custo
- Isso controla o ajuste dos pesos, possibilitando que a rede tenha uma boa capacidade de generalização
- A regularização ℓ_2 é a mais comum e consiste em somar à função custo o termo

$$\frac{\lambda}{2N_L}\sum_{\ell=1}^{N_L}\|\mathbf{w}_\ell^{(L)}(n-1)\|^2,$$

  em que λ é um hiperparâmetro
- Assim, ao minimizar a função custo somada a esse termo, o algoritmo também procura minimizar a norma dos vetores de peso da camada de saída, evitando dessa forma que ocorra divergência.

17 Inicialização

- Nos experimentos das meias-luas, os pesos e *biases* foram inicializados considerando $\mathrm{U} \sim [-10^{-2}, 10^{-2}]$
- Qual o intervalo "ideal" da distribuição uniforme para inicializar os pesos e *biases*?
- A inicialização considerando uma distribuição uniforme é a mais adequada?
- No *backpropagation*, $\delta_k^{(j)}$ carrega consigo a multiplicação dos gradientes locais das camadas mais profundas da rede.
- Para redes profundas, se os gradientes locais forem menores do que um, as atualizações dos pesos e *biases* das camadas mais rasas assumem valores muito pequenos, tornando o processo de aprendizado lento e ineficiente
- Para gradientes locais maiores que um, as atualizações dos pesos das camadas menos profundas assumem valores muito elevados, levando o algoritmo de treinamento à divergência
- Esse problema é conhecido como desvanecimento ou explosão dos gradientes

18 Inicialização de Xavier

- Considere uma MLP com função de ativação do tipo sigmoidal e pesos grandes. Como a sigmoidal é plana para valores grandes da entrada, as ativações ficarão saturadas e os gradientes se aproximam de zero
- A inicialização de Xavier busca garantir que a variância de $y_k^{(j)}$ seja mantida igual ao longo das camadas
- Considerando função de ativação linear, temos

$$y_k^{(j)} = b_k^{(j)} + w_{k1}^{(j)} y_1^{(j-1)} + w_{k2}^{(j)} y_2^{(j-1)} + \cdots + w_{kN_{j-1}}^{(j)} y_{N_{j-1}}^{(j-1)}.$$

- A variância de $y_k^{(j)}$ é

$$\mathrm{var}(y_k^{(j)}) = \mathrm{var}\left(b_k^{(j)} + w_{k1}^{(j)} y_1^{(j-1)} + w_{k2}^{(j)} y_2^{(j-1)} + \cdots + w_{kN_{j-1}}^{(j)} y_{N_{j-1}}^{(j-1)}\right)$$

19 Inicialização de Xavier

- Assumindo que os *biases* foram inicializados com zero, sua variância também é nula
- Assumindo independência entre os pesos e as entradas da camada j, temos

$$\mathrm{var}(w_{k\ell}^{(j)} y_\ell^{(j-1)}) = [\mathrm{E}\{y_\ell^{(j-1)}\}]^2 \mathrm{var}(w_{k\ell}^{(j)}) + [\mathrm{E}\{w_{k\ell}^{(j)}\}]^2 \mathrm{var}(y_\ell^{(j-1)}) + \mathrm{var}(w_{k\ell}^{(j)}) \mathrm{var}(y_\ell^{(j-1)}),$$

  $\ell = 1,2,\cdots,N_{j-1}$
- Para entradas e pesos com médias nulas

$$\mathrm{var}(w_{k\ell}^{(j)} y_\ell^{(j-1)}) = \mathrm{var}(w_{k\ell}^{(j)}) \mathrm{var}(y_\ell^{(j-1)})$$

- Usando esse resultado no cálculo da variância de $y_k^{(j)}$:

$$\mathrm{var}(y_k^{(j)}) = N_{j-1} \mathrm{var}(w_{k\ell}^{(j)}) \mathrm{var}(y_\ell^{(j-1)}).$$

- Como se deseja que $\mathrm{var}(y_k^{(j)}) = \mathrm{var}(y_\ell^{(j-1)})$, obtemos

$$\mathrm{var}(w_{k\ell}^{(j)}) = \frac{1}{N_{j-1}}$$

20 Inicialização de Xavier

- A inicialização de Xavier propõe inicializar os pesos com a distribuição

$$\boxed{w_{k\ell}^{(j)} \sim \mathrm{N}\left(0, \frac{1}{N_{j-1}}\right)}$$

- Uma variante, a inicialização de Glorot, leva em conta também o número de número de neurônios da camada j:

$$\boxed{w_{k\ell}^{(j)} \sim \mathrm{N}\left(0, \frac{2}{N_{j-1} + N_j}\right)}$$

- Variantes com a distribuição uniforme:

$$\boxed{w_{k\ell}^{(j)} \sim \mathrm{U}\left[-\sqrt{\frac{3}{N_{j-1}}}, +\sqrt{\frac{3}{N_{j-1}}}\right]}$$

$$\boxed{w_{k\ell}^{(j)} \sim \mathrm{U}\left[-\sqrt{\frac{6}{N_{j-1} + N_j}}, +\sqrt{\frac{6}{N_{j-1} + N_j}}\right].}$$

21 Inicialização de He

- O problema de desvanecimento ou explosão dos gradientes com funções de ativação do tipo sigmoidal geralmente não ocorre quando se usa ReLU
- A inicialização de He foi proposta como alternativa à de Xavier para neurônios que consideram ReLU
- Basicamente, a inicialização de He propõe que os pesos tenham o dobro da variância calculada anteriormente, o que leva à seguinte inicialização considerando a distribuição normal

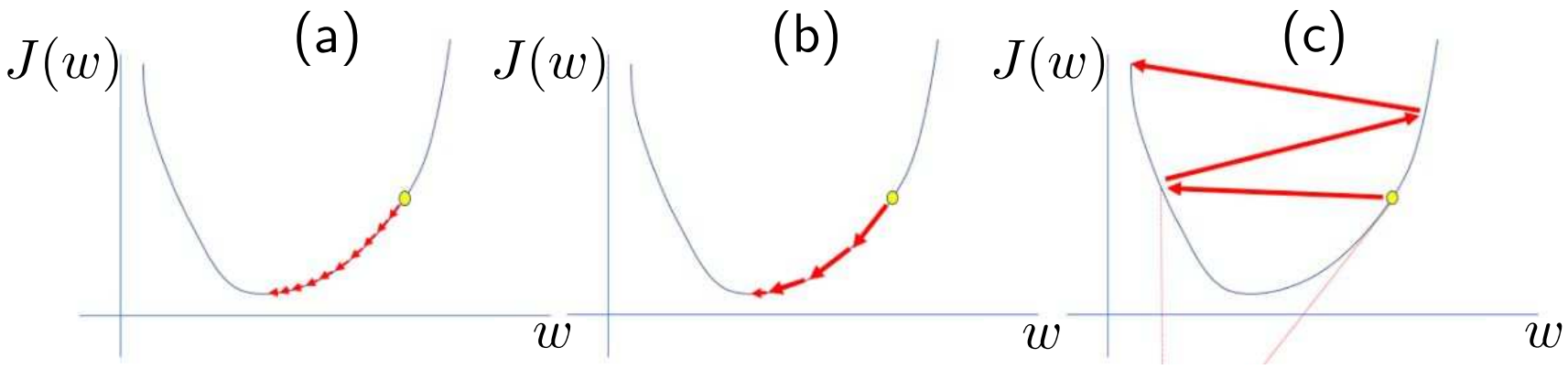
$$\boxed{w_{k\ell}^{(j)} \sim \mathrm{N}\left(0, \frac{2}{N_{j-1}}\right)}$$

e à seguinte variante para distribuição uniforme

$$\boxed{w_{k\ell}^{(j)} \sim \mathrm{U}\left[-\sqrt{\frac{6}{N_{j-1}}}, +\sqrt{\frac{6}{N_{j-1}}}\right].}$$

22 Passo de adaptação

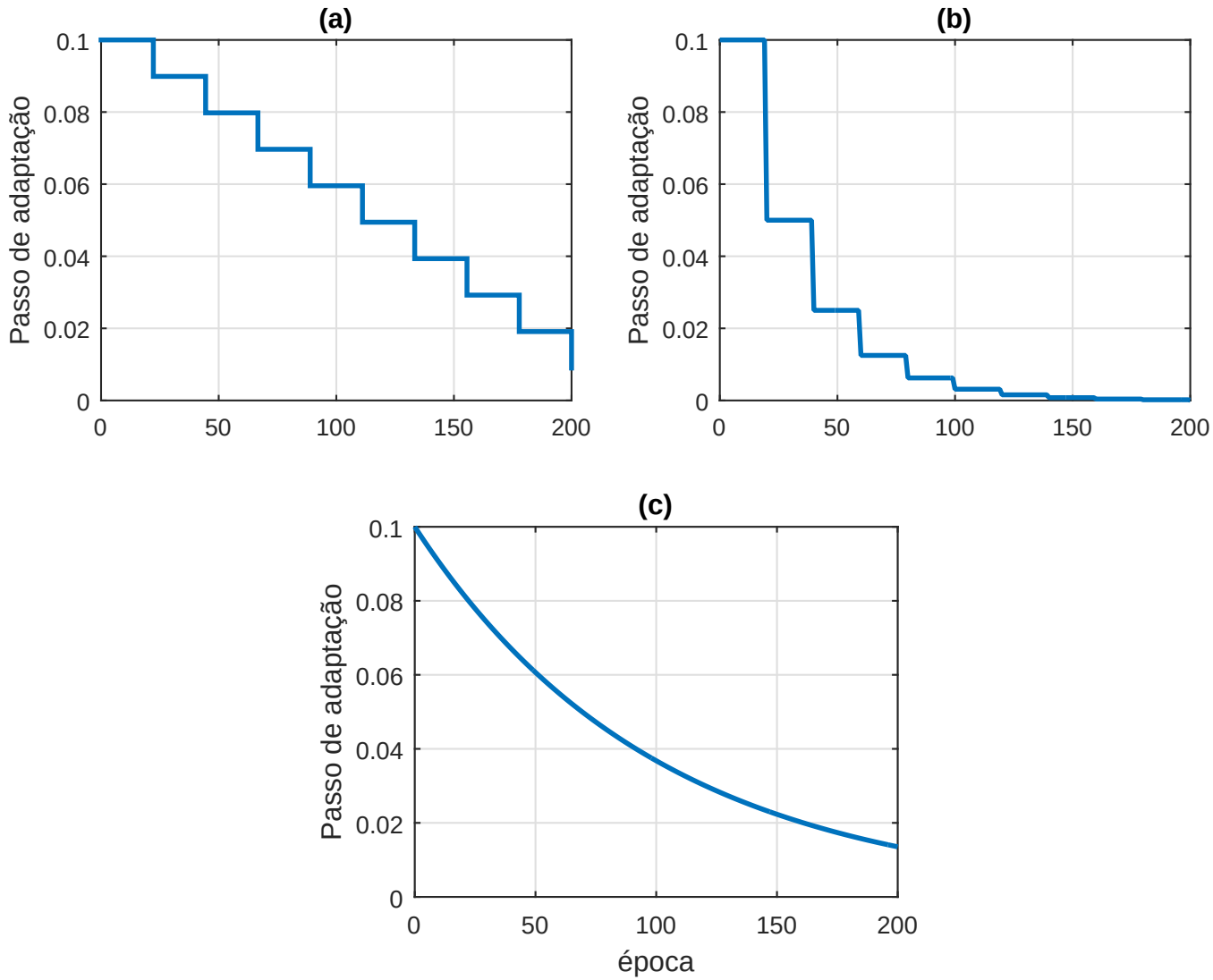
- Se o passo for muito baixo, a convergência do algoritmo de treinamento será muito lenta [Fig (a)]
- Um passo muito elevado pode causar divergência [Fig. (c)]
- Um passo ideal proporciona uma rápida convergência [Fig. (b)]
- O passo de adaptação ideal depende da superfície de desempenho, da arquitetura da rede e do conjunto de dados



23 Passo de adaptação

- Na técnica *learning rate annealing*, o passo deve ser alto no início e diminuir gradualmente ao longo do treinamento
- Uma taxa alta no início pode fazer com que o algoritmo "pule" mínimos locais
- Em seguida, uma taxa pequena faz um ajuste fino, possibilitando o algoritmo explorar as partes "mais profundas" da função custo
- A seguir vamos considerar três tipos usuais de decaimento: escada uniforme, escala não uniforme e exponencial

24 Passo de adaptação – *learning rate annealing*



(a) decaimento em escada uniforme ($\eta_0 = 0{,}1$, $\Delta\eta = 0{,}0101$ e $\Delta k = 20$), (b) decaimento em escada não uniforme ($\eta_0 = 0{,}1$, $\Delta\eta = 0{,}5$ e $\Delta k = 20$) e (c) decaimento exponencial ($\eta_0 = 0{,}1$ e $a = 0{,}01$)

25 Passo de adaptação – *learning rate annealing*

- No decaimento em escada com degraus uniformes, o passo da k-ésima época é calculado como

$$\eta(k) = \eta_0 - \Delta\eta\lfloor k/\Delta k \rfloor,$$

em que η_0 é o valor inicial do passo, $\Delta\eta$ o valor do decaimento e Δk o número de épocas em que o passo é mantido fixo

- No decaimento em escada com degraus não uniformes, o passo da k-ésima época é calculado como

$$\eta(k) = \eta_0 \Delta\eta^{\lfloor k/\Delta k \rfloor}.$$

- No decaimento exponencial, o passo da k-ésima época é calculado como

$$\eta(k) = \eta_0 e^{-ak}, \ a > 0.$$

26 Passo de adaptação – *learning rate annealing*

- O desafio de usar esquemas de ajuste dos passos de adaptação é que seus hiperparâmetros precisam ser definidos com antecedência e dependem da arquitetura da rede e do problema
- Pode ser conveniente adaptar pesos de neurônios de camadas diferentes com passos distintos.
- Algoritmos de otimização como Adam e RMSprop resolvem esses problemas, pois ajustam os passos de adaptação de forma automática com o uso de normalização, como veremos posteriormente.

27 *Mini-batch*

- Como o algoritmo LMS foi proposto para aplicações de tempo real, o modo estocástico é o mais utilizado
- No caso das redes neurais, o modo *mini-batch* é o mais utilizado:
  - A inferência não é realizada durante o treinamento
  - A saída e o erro são utilizados no treinamento apenas para atualizar os pesos do algoritmo
  - Depois do treinamento, fixam-se os pesos para então se fazer a inferência e testar o classificador ou regressor

28 *Mini-batch*

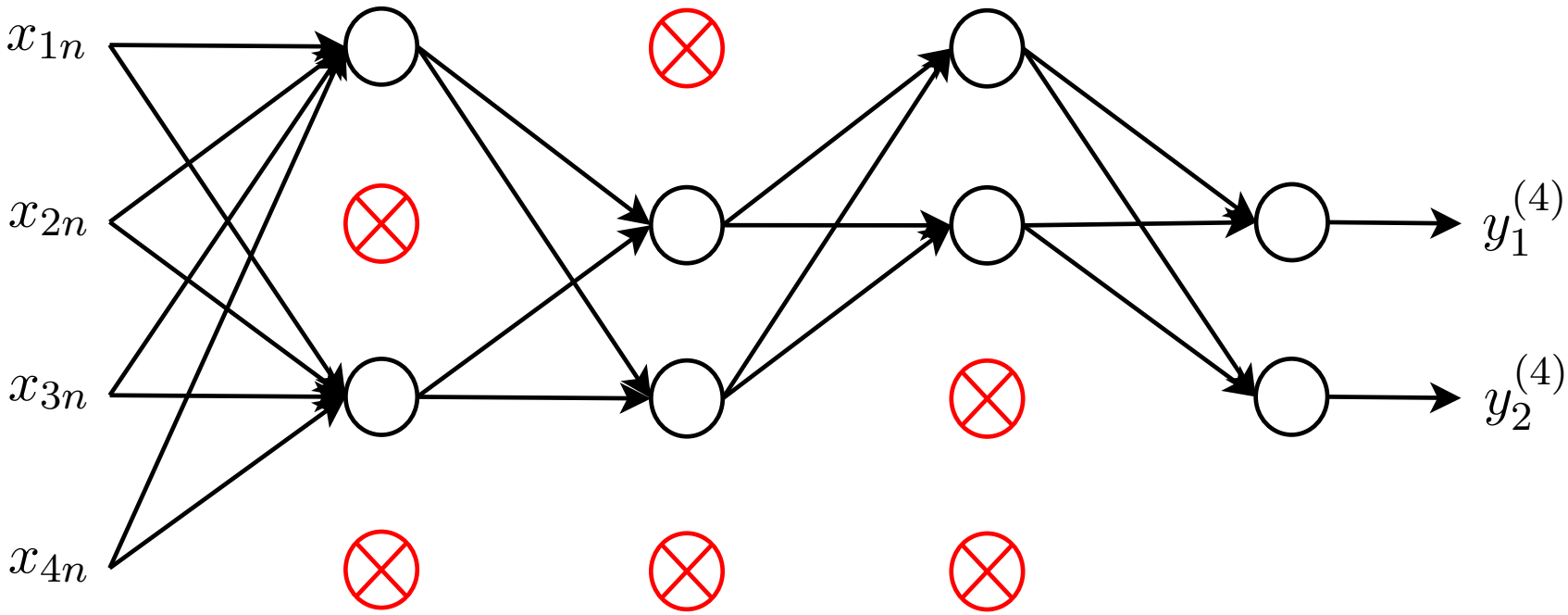
- O uso do *mini-batch* consiste em dividir aleatoriamente o conjunto de treinamento da rede em blocos de tamanho previamente definido, embaralhando-se as amostras do conjunto
- A atualização dos parâmetros ocorre apenas depois que são calculados os gradientes de todos os elementos de um *mini-batch* e está associada à média dos gradientes de um *mini-batch*
- Considera-se passada uma época quando todos os *mini-batches* são percorridos
- Após cada época, a divisão do conjunto de treinamento entre *mini-batches* é refeita de maneira aleatória, embaralhando-se novamente o conjunto de treinamento
- O tamanho de cada *mini-batch* é um hiperparâmetro e não muda no decorrer das épocas

29 *Mini-batch*

- Quando cada *mini-batch* é formado apenas por uma amostra do conjunto de treinamento, trata-se do modo estocástico
- O modo estocástico é pouco eficiente, pois a atualização ocorre em direções distintas do mínimo da função custo, o que faz com que o algoritmo leve mais épocas para convergir. O modo estocástico também anula as vantagens computacionais de uma implementação matricial
- Quando um *mini-batch* possui todos os elementos do conjunto de treinamento, trata-se do modo *batch*. Neste modo, os parâmetros são sempre atualizados na direção do mínimo da função custo. O *batch* seria o modo de treinamento ideal se não houvesse limitações computacionais
- Como é necessário esperar que todo o conjunto de treinamento seja percorrido para se realizar a atualização, o modo *batch* é muito demorado e computacionalmente ineficiente

30 *Dropout* para evitar *overfitting*

- *Dropout* é uma das técnicas mais utilizadas para evitar o *overfitting*
- Ela inativa aleatoriamente, em cada iteração do algoritmo *backpropagation*, diferentes neurônios de cada camada oculta
- Cada neurônio é inativado com probabilidade p



dropout em uma rede MLP com $p = 0{,}5$

31 *Dropout* para evitar *overfitting*

- No exemplo, metade dos neurônios de cada camada oculta foram inativados em uma determinada iteração
- Quando um neurônio é inativado, seu gradiente é nulo e seus pesos não são atualizados
- A eliminação temporária de diferentes conjuntos de neurônios leva ao treinamento de redes neurais distintas
- O procedimento de eliminação é equivalente ao cálculo da média dos efeitos de um grande número de redes distintas
- Como elas vão se adaptar de diferentes maneiras, isso possibilita a redução do *overfitting*, pois será mais difícil para a rede se especializar nos dados de treinamento

32 *Momentum*

- O algoritmo LMS é uma aproximação estocástica do algoritmo *steepest descent*
- Existe um compromisso entre a **velocidade de convergência** e a **precisão da solução**
- Quanto menor o passo, mais lento é o algoritmo e os pesos variam menos em torno da solução de Wiener
- Quanto maior o passo, maior a sua velocidade de convergência e maior a variação dos pesos. O algoritmo também pode divergir se o passo for muito alto
- O mesmo acontece com o algoritmo *backpropagation*: quanto menor for o passo de adaptação, menores serão as mudanças nos pesos de uma iteração para outra, mais suave será a trajetória no espaço dos pesos e mais lenta a taxa de aprendizagem
- Se aumentarmos muito o passo, as mudanças dos pesos de uma iteração para outra também aumentam e o algoritmo pode divergir

33 *Momentum*

- *Momentum* é um método simples de aumentar a velocidade de convergência do *backpropagation* sem causar divergência
- Atualização da matriz de pesos da Camada j da MLP:

$$\mathbf{W}^{(j)}(n) = \mathbf{W}^{(j)}(n-1) + \eta \boldsymbol{\Delta}_{\delta}^{(j)}(n),$$

em que

$$\boldsymbol{\Delta}_{\delta}^{(j)}(n) = \boldsymbol{\delta}^{(j)}(n)[\mathbf{x}^{(j)}(n)]^T$$

- Define-se a matriz

$$\boldsymbol{\Delta}\mathbf{W}^{(j)}(n-1) \triangleq \mathbf{W}^{(j)}(n-1) - \mathbf{W}^{(j)}(n-2)$$

- A atualização do *backpropagation* com *momentum* fica

$$\boxed{\mathbf{W}^{(j)}(n) = \mathbf{W}^{(j)}(n-1) + \alpha \boldsymbol{\Delta}\mathbf{W}^{(j)}(n-1) + \eta \boldsymbol{\Delta}_{\delta}^{(j)}(n)}$$

em que $0 \leq \alpha < 1$ é a constante de *momentum*.

- Observe que $\alpha = 0$ leva essa atualização à forma padrão do *backpropagation* sem *momentum*

34 *Momentum*

- Usando a definição $\boldsymbol{\Delta}\mathbf{W}^{(j)}(n)$, podemos reescrever

$$\boxed{\boldsymbol{\Delta}\mathbf{W}^{(j)}(n) = \alpha\boldsymbol{\Delta}\mathbf{W}^{(j)}(n-1) + \eta\boldsymbol{\Delta}_{\delta}^{(j)}(n)}$$

- Para entender o efeito do termo de *momentum*, note que

$$\begin{aligned}
\boldsymbol{\Delta}\mathbf{W}^{(j)}(1) &= \alpha\boldsymbol{\Delta}\mathbf{W}^{(j)}(0) + \eta\boldsymbol{\Delta}_{\delta}^{(j)}(1) \\
\boldsymbol{\Delta}\mathbf{W}^{(j)}(2) &= \alpha\boldsymbol{\Delta}\mathbf{W}^{(j)}(1) + \eta\boldsymbol{\Delta}_{\delta}^{(j)}(2) \\
&= \alpha\left[\alpha\boldsymbol{\Delta}\mathbf{W}^{(j)}(0) + \eta\boldsymbol{\Delta}_{\delta}^{(j)}(1)\right] + \eta\boldsymbol{\Delta}_{\delta}^{(j)}(2) \\
&= \alpha^2\boldsymbol{\Delta}\mathbf{W}^{(j)}(0) + \eta\left[\alpha\boldsymbol{\Delta}_{\delta}^{(j)}(1) + \boldsymbol{\Delta}_{\delta}^{(j)}(2)\right] \\
\boldsymbol{\Delta}\mathbf{W}^{(j)}(3) &= \alpha\boldsymbol{\Delta}\mathbf{W}^{(j)}(2) + \eta\boldsymbol{\Delta}_{\delta}^{(j)}(3) \\
&= \alpha^3\boldsymbol{\Delta}\mathbf{W}^{(j)}(0) + \eta\left[\alpha^2\boldsymbol{\Delta}_{\delta}^{(j)}(1) + \alpha\boldsymbol{\Delta}_{\delta}^{(j)}(2) + \boldsymbol{\Delta}_{\delta}^{(j)}(3)\right] \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
\boldsymbol{\Delta}\mathbf{W}^{(j)}(n) &= \alpha^n\boldsymbol{\Delta}\mathbf{W}^{(j)}(0) + \eta\sum_{k=1}^{n}\alpha^{n-k}\boldsymbol{\Delta}_{\delta}^{(j)}(k).
\end{aligned}$$

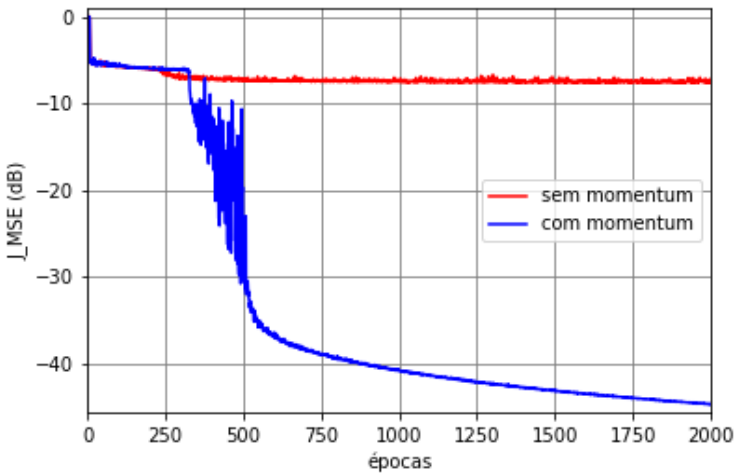
35 *Momentum*

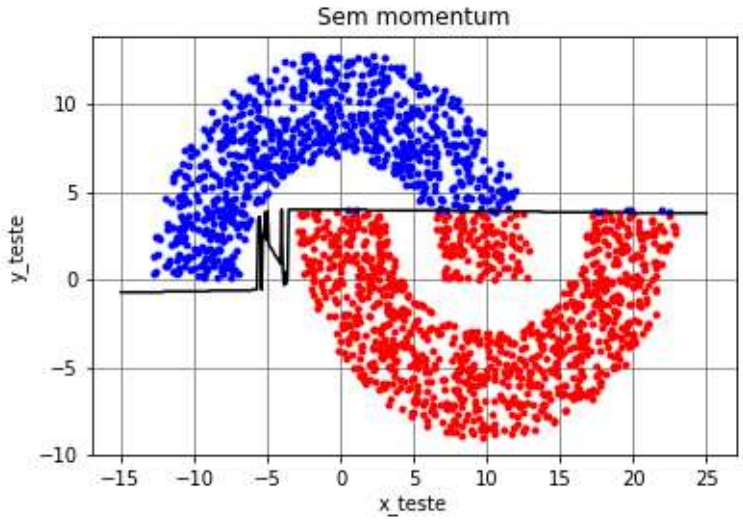
- $\alpha^n \mathbf{\Delta W}^{(j)}(0)$ tende a zero a medida que n aumenta. Assim:

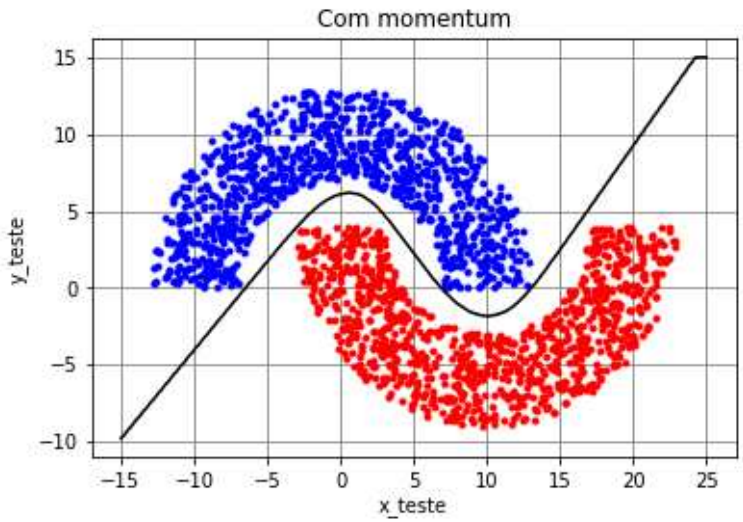
$$\mathbf{\Delta W}^{(j)}(n) = \eta \sum_{k=1}^{n} \alpha^{n-k} \mathbf{\Delta}_{\delta}^{(j)}(k).$$

- Efeitos benéficos do *momentum*:
  1. Ponderação exponencial: como $0 \leq \alpha < 1$, consideram-se pesos maiores para ajustes recentes e pesos menores para os mais antigos
  2. quando $\mathbf{\Delta}_{\delta}^{(j)}(n)$ tem o mesmo sinal algébrico para n's sucessivos, $\mathbf{\Delta W}^{(j)}(n)$ cresce em magnitude e $\mathbf{W}^{(j)}(n)$ é ajustada com uma grande quantidade. O *momentum* tende a acelerar a convergência em direções de descida mais íngreme
  3. quando o sinal algébrico de $\mathbf{\Delta}_{\delta}^{(j)}(n)$ muda para n's sucessivos, $\mathbf{\Delta W}^{(j)}(n)$ diminui em magnitude e $\mathbf{W}^{(j)}(n)$ é ajustada com uma pequena quantidade. O *momentum* tem o efeito de estabilizador em direções que oscilam em sinal

36 *Momentum* – Meia-luas com MLP (3–10–1)







Meias-luas ($r_1 = 10$, $r_2 = -4$ e $r_3 = 6$). Função custo, classificação dos dados de teste ($N_{\text{teste}} = 2 \times 10^3$) e curva de separação das regiões. Rede MLP (3–10–1) treinada em *mini-batch* ($N_0 = 2$, $N_t = 10^3$, $N_b = 50$) com o algoritmo *backpropagation* sem *momentum* ($\eta = 0{,}1$) e com *momentum* ($\eta = 0{,}1$, $\alpha = 0{,}9$); função de ativação tangente hiperbólica e pesos e *biases* inicializados com $\mathrm{U} \sim [-10^{-2}, 10^{-2}]$

37 *Momentum*

- O comportamento observado no slide anterior nem sempre se repete, pois depende da inicialização
- Em muitos casos, os algoritmos com e sem *momentum* apresentam comportamentos semelhantes
- Ainda podem ocorrer situações em que o algoritmo com *momentum* não consegue evitar mínimos locais, enquanto o algoritmo sem *momentum* consegue
- Apesar disso, o uso de *momentum* é considerado benéfico na maior parte das vezes
- Isso de deve ao fato de que quando implementado junto com outras técnicas pode fazer com que a rede atinja valores de MSE mais baixos no treinamento, o que é indício de que mínimos locais foram evitados

38 Otimizador Adam

O algoritmo de otimização Adam (*adaptive moment estimation*) é uma extensão do algoritmo do gradiente estocástico e apresenta os seguintes benefícios:

1. simples de implementar, computacionalmente eficiente e requer poucos requisitos de memória
2. adequado quando se usa muitos dados e/ou parâmetros
3. apropriado para problemas não estacionários e problemas com gradientes muito ruidosos e/ou esparsos e
4. os hiperparâmetros têm interpretação intuitiva e são simples de ajustar

39 Otimizador Adam

- O Adam atualiza os parâmetros a partir dos gradientes calculados na iteração atual e em iterações passadas
- Ele combina o gradiente estocástico com *momentum* com o otimizador RMSprop (*root mean squared propagation*)
- À medida que os dados se propagam na rede, os gradientes podem ficar muito pequenos ou muito grandes
- Gradientes muito pequenos podem levar à estagnação enquanto os muito grandes podem levar à divergência
- O RMSprop usa uma média móvel dos gradientes ao quadrado, o que gera uma normalização no algoritmo, que passa a ser encarado como um algoritmo de passo variável
- Quando os gradientes são grandes, o método diminui o passo para evitar a divergência e quando os gradientes são pequenos, ele aumenta o passo para evitar a estagnação.

40 Otimizador Adam

- Na dedução do *backpropagation*, definimos

$$\boldsymbol{\Delta}_{\delta}^{(j)}(n) = \boldsymbol{\delta}^{(j)}(n)[\mathbf{x}^{(j)}(n)]^T,$$

que contém o negativo dos vetores gradiente de todos os neurônios da Camada j.

- Vamos agora definir a matriz:

$$\mathbf{S}^{(j)}(n) = \beta_2 \mathbf{S}^{(j)}(n-1) + (1-\beta_2)\left[\boldsymbol{\Delta}_{\delta}^{(j)}(n)\right]^{\odot 2}$$

em que

  - $\mathbf{S}^{(j)}(0) = \mathbf{0}$, $0 \ll \beta_2 < 1$ (fator de esquecimento)
  - a operação $[\boldsymbol{\Delta}_{\delta}^{(j)}(n)]^{\odot 2}$ indica que cada elemento da matriz $\boldsymbol{\Delta}_{\delta}^{(j)}(n)$ é elevado ao quadrado

- Levando em conta a inicialização com valores nulos, temos

$$\mathbf{S}^{(j)}(n) = (1-\beta_2)\sum_{k=1}^{n} \beta_2^{n-k}\left[\boldsymbol{\Delta}_{\delta}^{(j)}(k)\right]^{\odot 2}$$

  - Essa estimativa considera pesos maiores para os gradientes ao quadrado mais recentes e pesos menores para os mais antigos

41 Otimizador Adam

- Utilizando a matriz $\mathbf{S}^{(j)}(n)$, a atualização dos pesos e *bias* da Camada j da rede segundo o otimizador RMSprop é dada por

$$\mathbf{W}^{(j)}(n) = \mathbf{W}^{(j)}(n-1) + \eta\, \boldsymbol{\Delta}_{\delta}^{(j)}(n) \oslash \left[\left[\mathbf{S}^{(j)}(n) \right]^{\odot \frac{1}{2}} + \varepsilon \mathbf{1} \right],$$

em que

- $\oslash$ se refere a divisão de Hadamard, que resulta em uma matriz em que cada elemento é igual à divisão do respectivo elemento da matriz à esquerda pelo respectivo elemento da matriz à direita,
- ε é uma constante positiva pequena (e.g., $\varepsilon = 10^{-8}$) usada para evitar divisões por zero
- $\mathbf{1}$ é uma matriz com todos os elementos iguais a 1 e dimensões adequadas para que a soma seja possível de ser calculada.

42 Otimizador Adam

- Para entender melhor essas operações, suponha que na iteração n dispomos das matrizes

$$\boldsymbol{\Delta}_{\delta}^{(j)}(n) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \text{e} \quad \mathbf{S}^{(j)}(n) = \begin{bmatrix} e & f \\ g & h \end{bmatrix}.$$

Assim,

$$\boldsymbol{\Delta}_{\delta}^{(j)}(n) \oslash \left(\left[\mathbf{S}^{(j)}(n) \right]^{\odot \frac{1}{2}} + \varepsilon \mathbf{1} \right) = \begin{bmatrix} \dfrac{a}{\sqrt{e}+\varepsilon} & \dfrac{b}{\sqrt{f}+\varepsilon} \\ \dfrac{c}{\sqrt{g}+\varepsilon} & \dfrac{d}{\sqrt{h}+\varepsilon} \end{bmatrix}.$$

43 Otimizador Adam

- O otimizador Adam considera uma janela exponencial para estimar os gradientes:

$$\mathbf{V}^{(j)}(n) = \beta_1 \mathbf{V}^{(j)}(n-1) + (1-\beta_1)\boldsymbol{\Delta}_\delta^{(j)}(n)$$

em que $\mathbf{V}^{(j)}(0) = \mathbf{0}$ e $0 \ll \beta_1 < 1$ (fator de esquecimento)

- Levando em conta a inicialização com valores nulos:

$$\mathbf{V}^{(j)}(n) = (1-\beta_1)\sum_{k=1}^{n} \beta_1^{n-k}\boldsymbol{\Delta}_\delta^{(j)}(k).$$

- As inicializações das matrizes $\mathbf{S}^{(j)}$ e $\mathbf{V}^{(j)}$ com elementos nulos podem gerar distorções no início do treinamento. Na atualização do RMSprop,

$$\mathbf{S}^{(j)}(1) = (1-\beta_2)[\boldsymbol{\Delta}_\delta^{(j)}(1)]^{\odot 2}$$

o que pode ser muito pequeno, pois $0 \ll \beta_2 < 1$.

44 Otimizador Adam

- Para amenizar isso, são definidas as as matrizes de correção

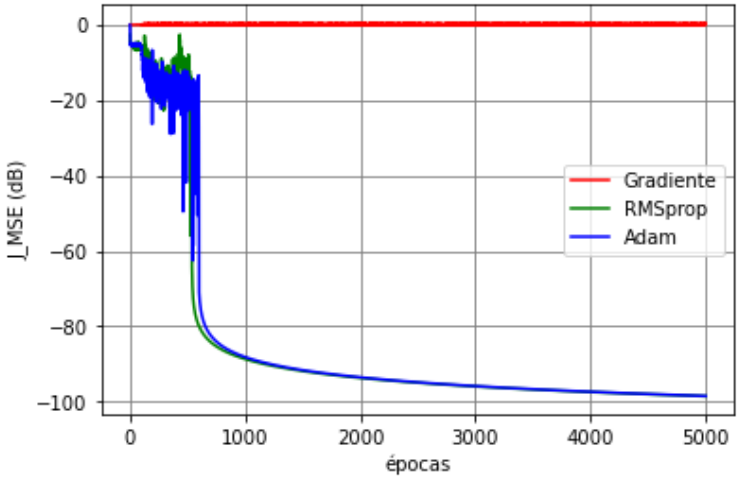
$$\overline{\mathbf{V}}^{(j)}(n) = \frac{1}{1-\beta_1^n}\,\mathbf{V}^{(j)}(n)$$

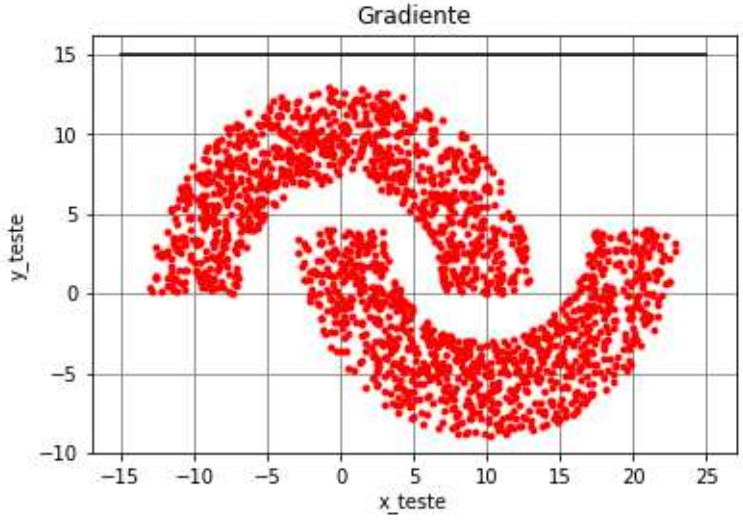
$$\overline{\mathbf{S}}^{(j)}(n) = \frac{1}{1-\beta_2^n}\,\mathbf{S}^{(j)}(n)$$

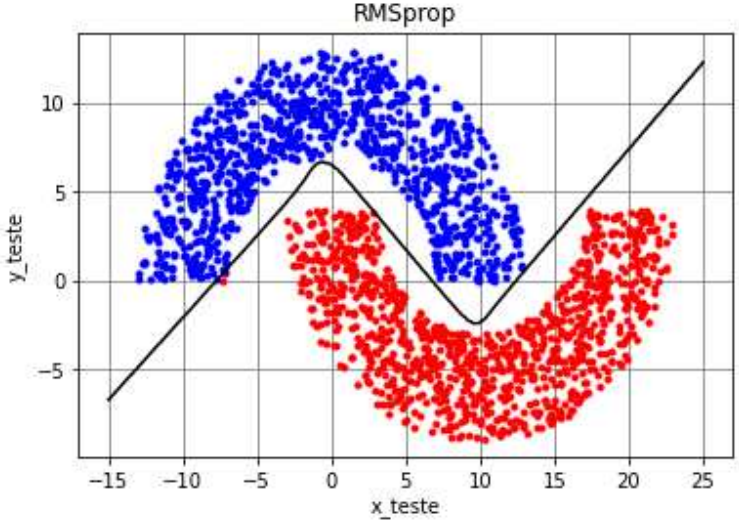
- Como $0 \ll \beta_1, \beta_2 < 1$, $\overline{\mathbf{V}}^{(j)}(n)$ e $\overline{\mathbf{S}}^{(j)}(n)$ tendem a $\mathbf{V}^{(j)}(n)$ e $\mathbf{S}^{(j)}(n)$, respectivamente, a medida que n aumenta (efeito apenas no início do treinamento)
- Utilizando as matrizes corrigidas, a atualização segundo o otimizador Adam é dada por

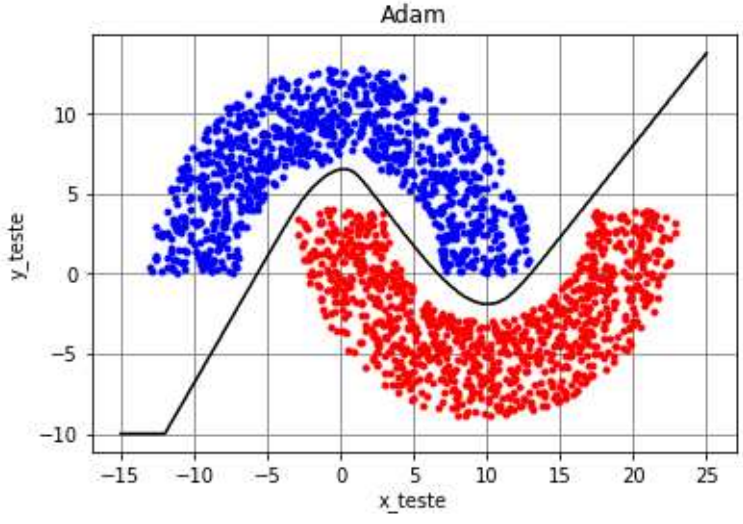
$$\mathbf{W}^{(j)}(n) = \mathbf{W}^{(j)}(n-1) + \eta\,\overline{\mathbf{V}}^{(j)}(n) \oslash \left[\left[\overline{\mathbf{S}}^{(j)}(n)\right]^{\odot\frac{1}{2}} + \varepsilon\mathbf{1}\right].$$

45 Otimizador Adam – Meias-luas com MLP (3–4–4–2–1)

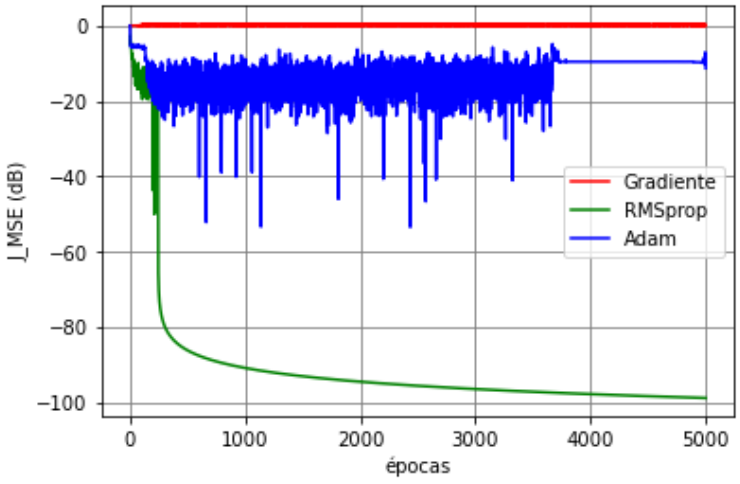


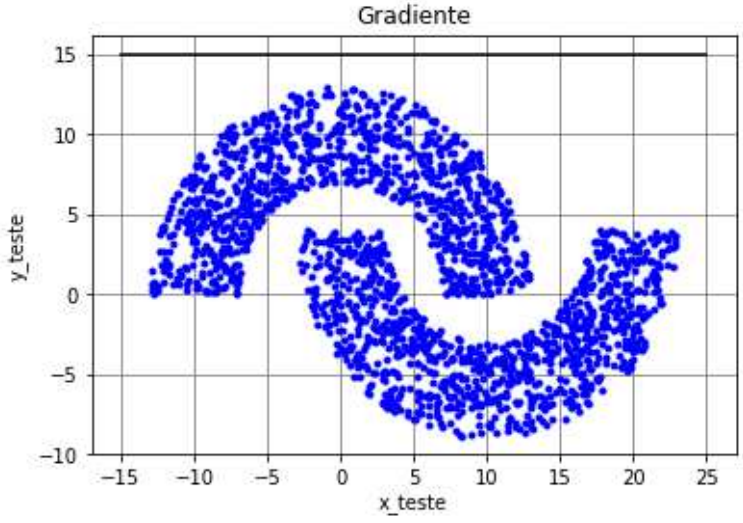


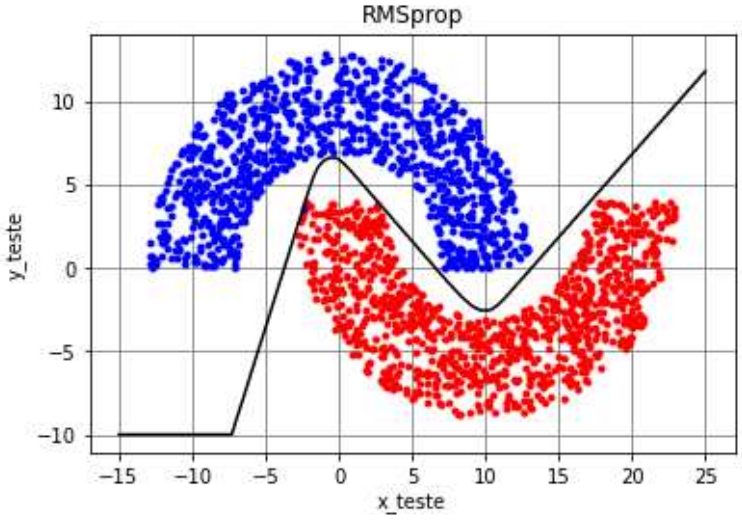


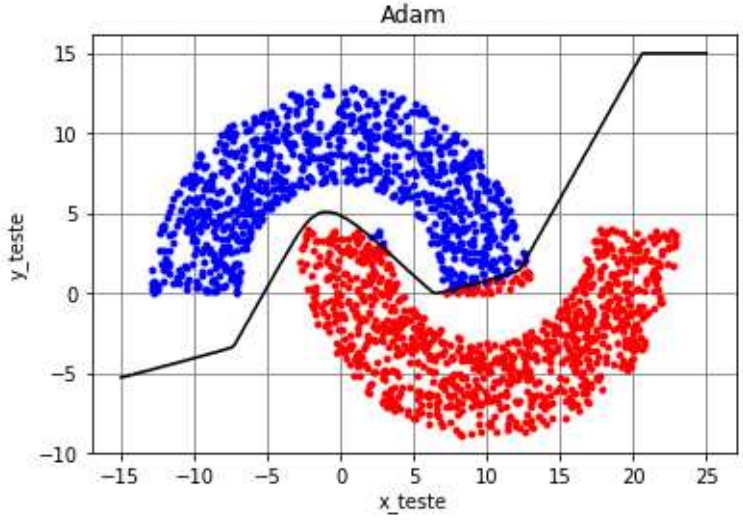


46 Otimizador Adam – Meias-luas com MLP (3–4–4–2–1)

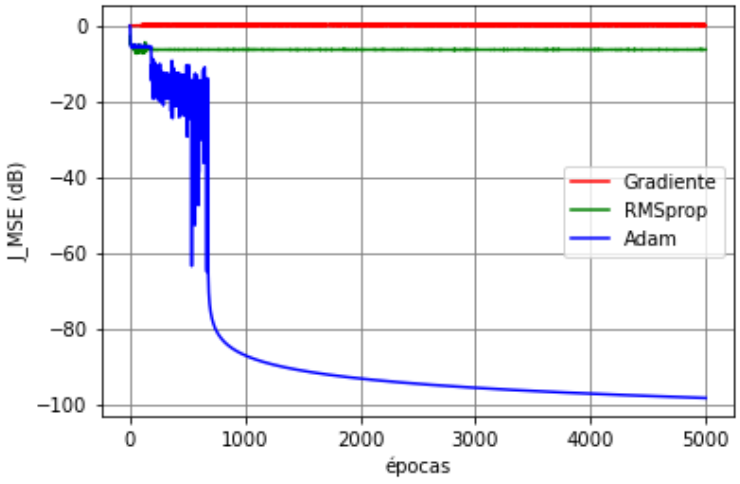


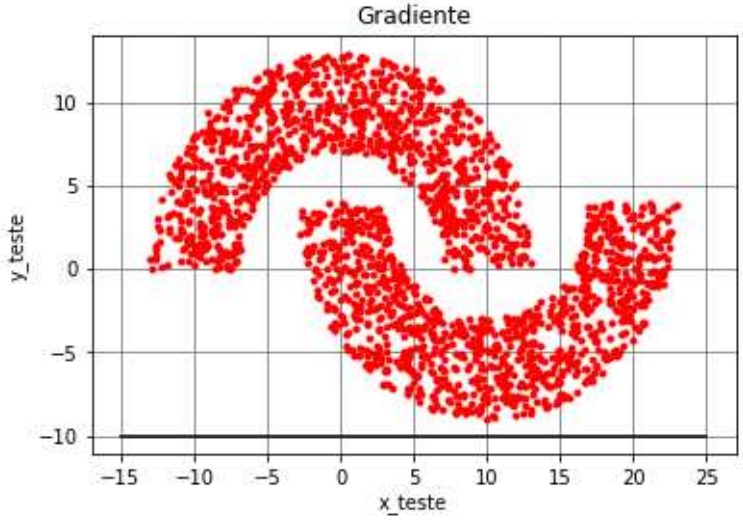


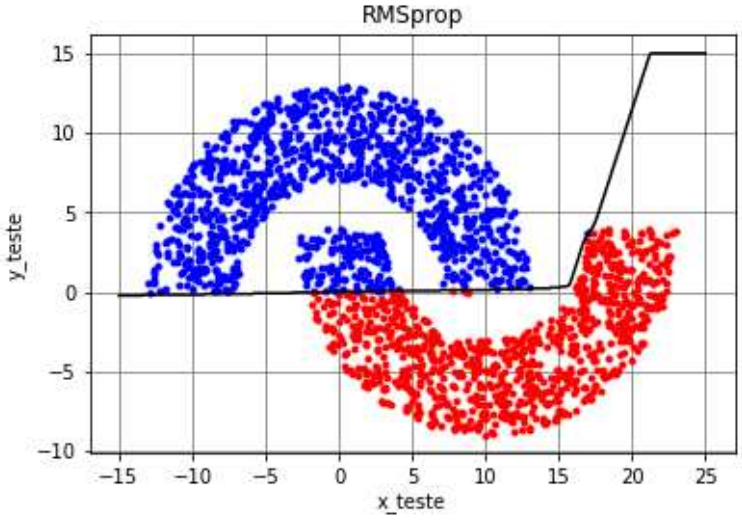


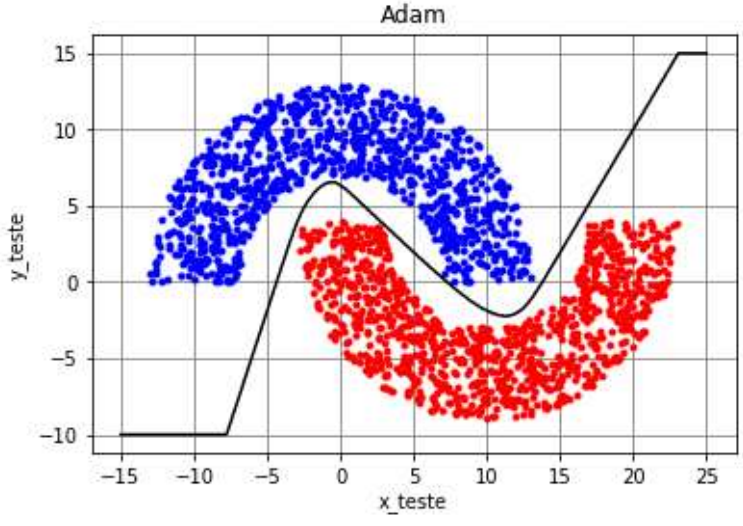


47 Otimizador Adam – Meias-luas com MLP (3–4–4–2–1)









48 Otimizador Adam

- Mudar o algoritmo de otimização é benéfico para evitar mínimos locais, principalmente em redes profundas.
- A adoção de um desses algoritmos apenas não é suficiente para evitar mínimos locais.
- O Adam tem algumas desvantagens:
  1. não converge em alguns exemplos simples
  2. o erro de generalização pode ser grande em muitos problemas de visão computacional
  3. requer mais memória que o método do gradiente e
  4. tem dois hiperparâmetros e portanto, alguns ajustes podem ser necessários

49 Validação cruzada

- A MLP aproxima um mapeamento entrada-saída por meio dos pesos e *biases*, utilizado um conjunto de exemplos rotulados
- A rede deve aprender o suficiente com os dados do passado e generalizar para dados futuros
- É importante selecionar a "melhor" rede (número de camadas, número de neurônios, funções de ativação, passo de adaptação, etc.) dentro de um conjunto de redes candidatas, considerando um determinado critério.
- O MSE tende a diminuir ao longo das épocas. Em geral, quanto mais épocas, mais baixo é o MSE
- Um MSE baixo no treinamento não corresponde necessariamente a um desempenho satisfatório da rede com o conjunto de teste, ou seja, pode haver *overfitting*

50 Validação cruzada

- A pergunta que cabe fazer aqui é: quando devemos parar de treinar já que um treinamento longo pode gerar *overfitting*?
- Para responder e selecionar a melhor rede, é comum utilizar um conjunto de dados de validação
- O conjunto de dados disponível deve ser primeiramente particionado de maneira aleatória entre treinamento e teste.
- O conjunto de treinamento, por sua vez, deve ser particionado em dois subconjuntos disjuntos:
  1. subconjunto de estimação, usado para treinar o modelo
  2. subconjunto de validação, usado para testar o modelo durante o treinamento

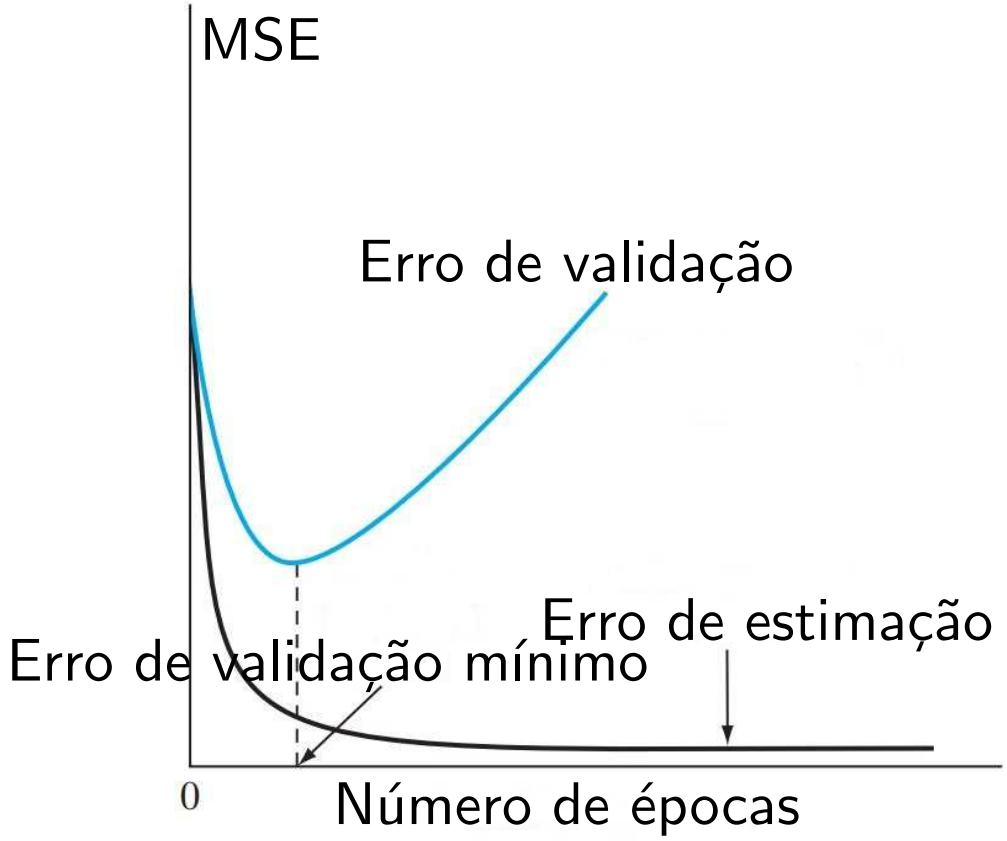
51 Validação cruzada

- A ideia é validar o modelo durante o treinamento com um conjunto de dados diferente do utilizado na estimação
- A avaliação final do modelo deve ser sempre feita com os dados do conjunto de teste, que não foram usados no treinamento, considerando fixos os pesos e *biases*
- Uma rede MLP aprende em etapas, passando da realização de funções de mapeamento simples para as mais complexas ao longo do treinamento
- Esse processo pode ser verificado pela diminuição do MSE ao longo das épocas: ele começa com um valor alto, diminui rapidamente e depois continua a diminuir lentamente quando a rede atinge um mínimo local
- Como o principal objetivo é obter uma rede com uma boa capacidade de generalização, é difícil descobrir quando parar de treinar
- Em particular, é possível que ocorra *overfitting* se o treinamento não for interrompido no ponto certo

52 Validação cruzada

- Podemos identificar o início do *overfitting* por meio da validação cruzada (*cross-validation*)
- O treinamento é interrompido periodicamente depois de um determinado número de épocas e e a rede é testada com o subconjunto de validação:
  1. após um intervalo de treinamento – a cada cinco épocas, por exemplo – os pesos e *biases* da MLP são mantidos fixos e apenas o cálculo progressivo é realizado. O erro de validação é então medido para cada exemplo do subconjunto de validação
  2. quando a fase de validação é concluída, o treinamento é retomado em um novo intervalo e o processo é repetido

53 Validação cruzada

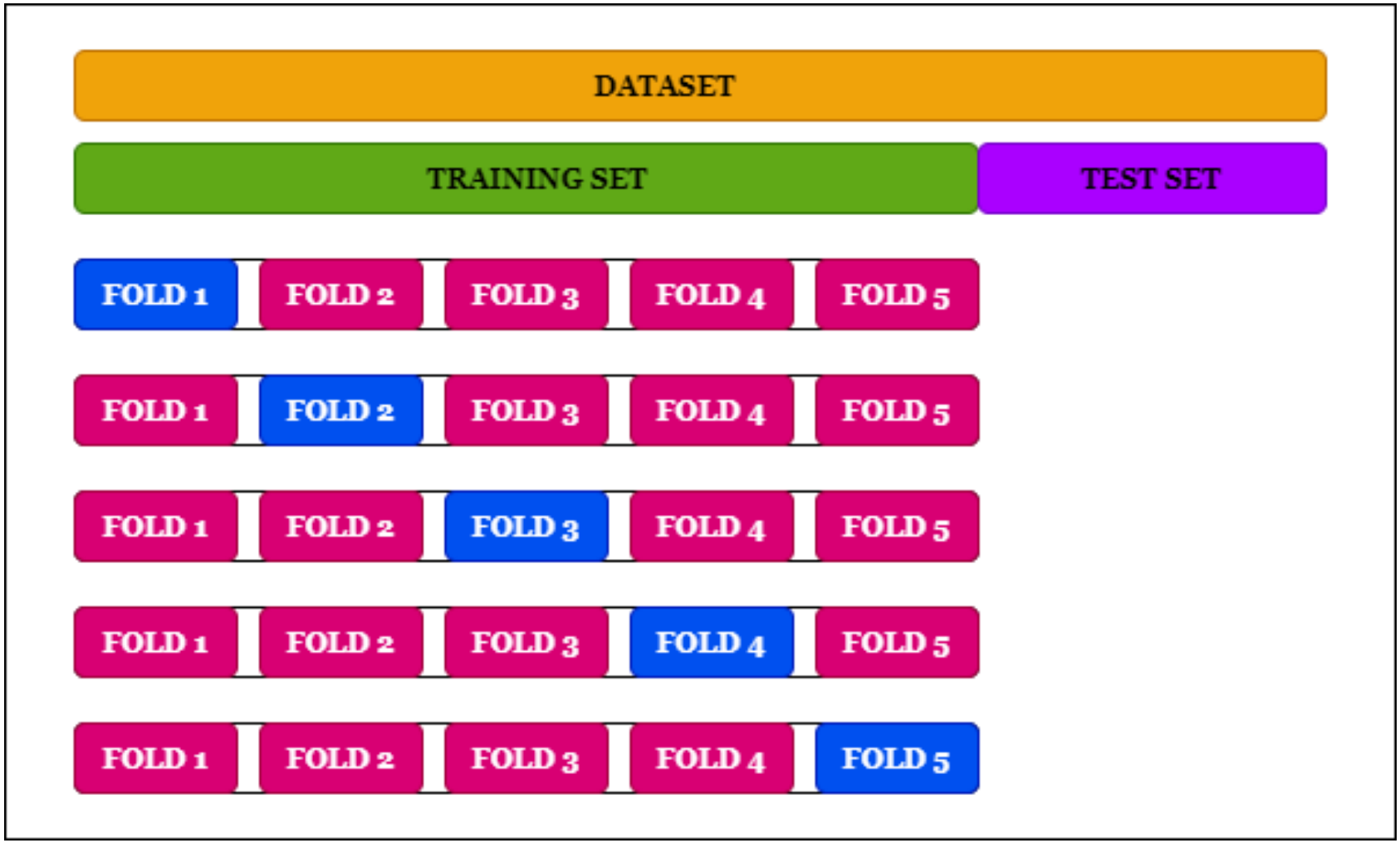


Curvas de erro de estimação e validação. O treinamento deve parar na época correspondente ao mínimo da curva de erro de validação. Fonte: Figura adaptada de [S. Haykin, 2009]

54 Validação cruzada

- A validação cruzada é conhecida como *holdout method*
- Uma das variantes mais utilizadas é a conhecida como *multifold cross-validation*
- O conjunto de treinamento de N_t exemplos é dividido em K subconjuntos com $K > 1$, sendo N_t divisível por K
- O modelo é treinado com todos os subconjuntos exceto um e o erro de validação é medido testando o modelo no subconjunto que é deixado de fora
- Este procedimento é repetido K vezes, cada vez usando um subconjunto diferente para validação
- O desempenho do modelo é avaliado pela média do erro quadrado de validação em todas as tentativas do experimento
- A desvantagem dessa variante é o custo computacional: o modelo tem que ser treinado K vezes, sendo $1 < K \leq N_t$

55 Validação cruzada



multifold cross-validation: para cada treinamento ($K = 5$), o subconjunto de dados destacado em azul é usado para validar o modelo treinado com os dados destacados em magenta. Fonte: `https://drigols.medium.com/`