# Deep Learning – Parte 1

# Aspectos Adicionais de Otimização em Treinamento Supervisionado

## Índice Geral

# 1. Otimização não-linear em deep learning

- Existem muitos desafios associados ao treinamento supervisionado em *deep learning*, dentre os quais se destacam a inicialização dos pesos, a não-convexidade da superfície de erro e a necessidade de convergência no menor número de iterações possível (dada a elevada dimensão do problema de otimização). Esses três desafios estão associados ao fato de não se conhecer a superfície de erro a ser minimizada, sendo possível apenas obter informações locais acerca de seu comportamento. Isso sem falar na questão do sobreajuste (*overfitting*), ou seja, nem sempre é conveniente baixar demais o erro de treinamento (BENGIO, 2012).
- Mesmo antes do advento das redes neurais profundas, já havia diversas propostas alternativas de algoritmos de otimização não-linear irrestrita, que é o problema que caracteriza o processo de treinamento supervisionado. Elas diferem na forma como passo e direção são estimados a cada iteração $k$:

$$\theta_{k+1} = \theta_k + \text{passo}_k * \text{direção}_k$$

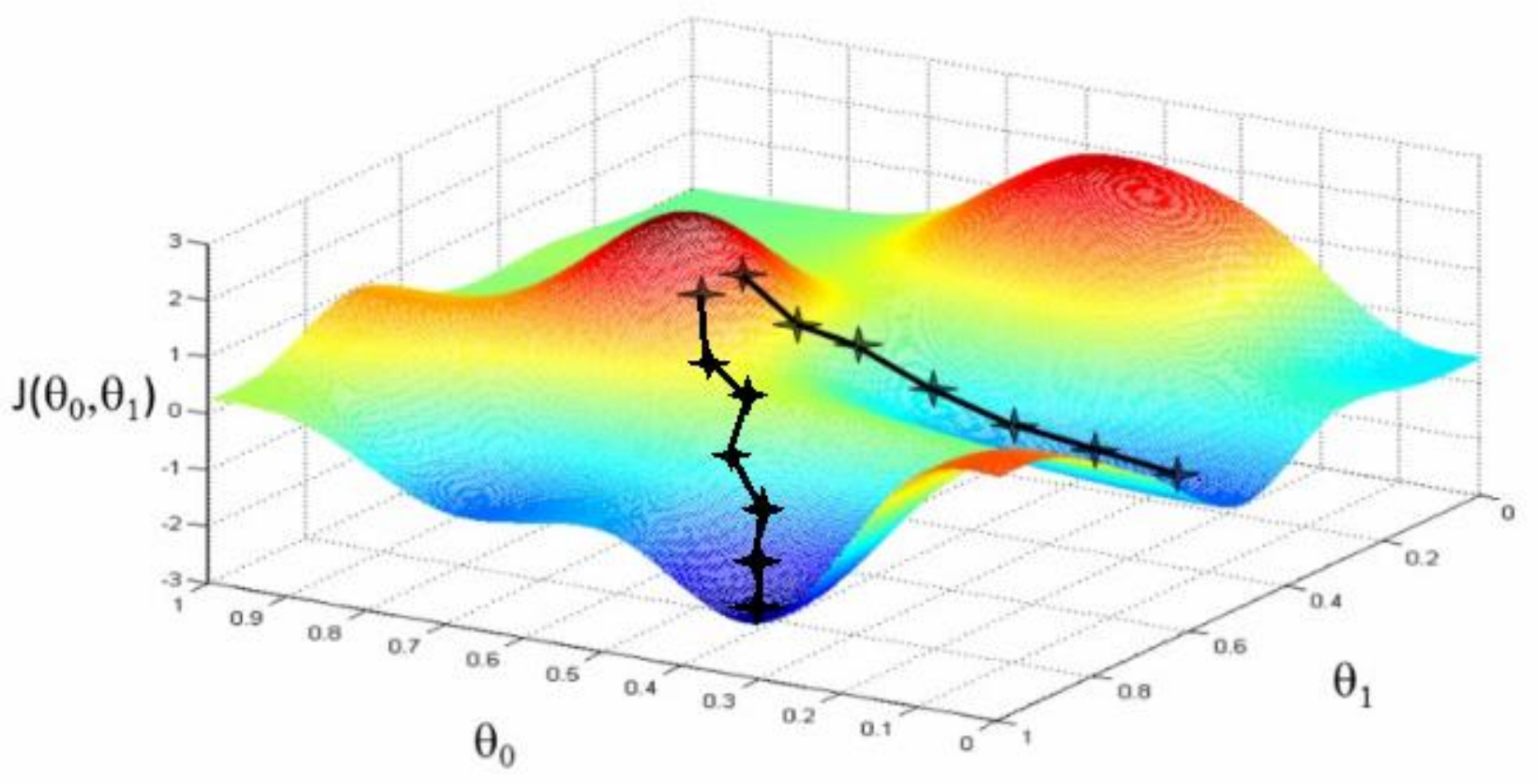



Figura 1 – Trajetória pela superfície de erro a partir de duas condições iniciais distintas

Fonte: https://medium.com/ai-society/hello-gradient-descent-ef74434bdfa5

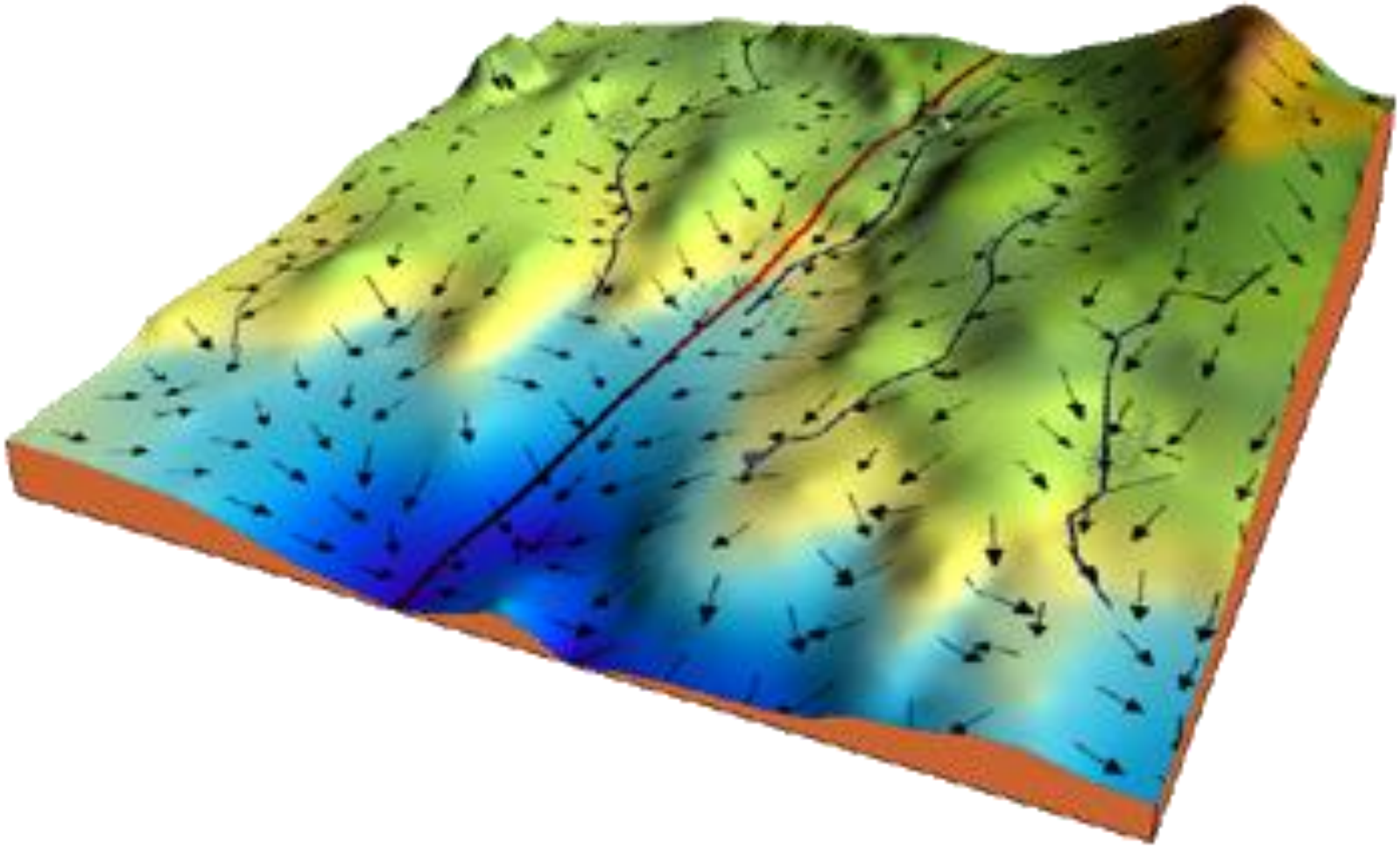


Figura 2 – Direções do gradiente descendente para uma superfície multimodal

Fonte: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

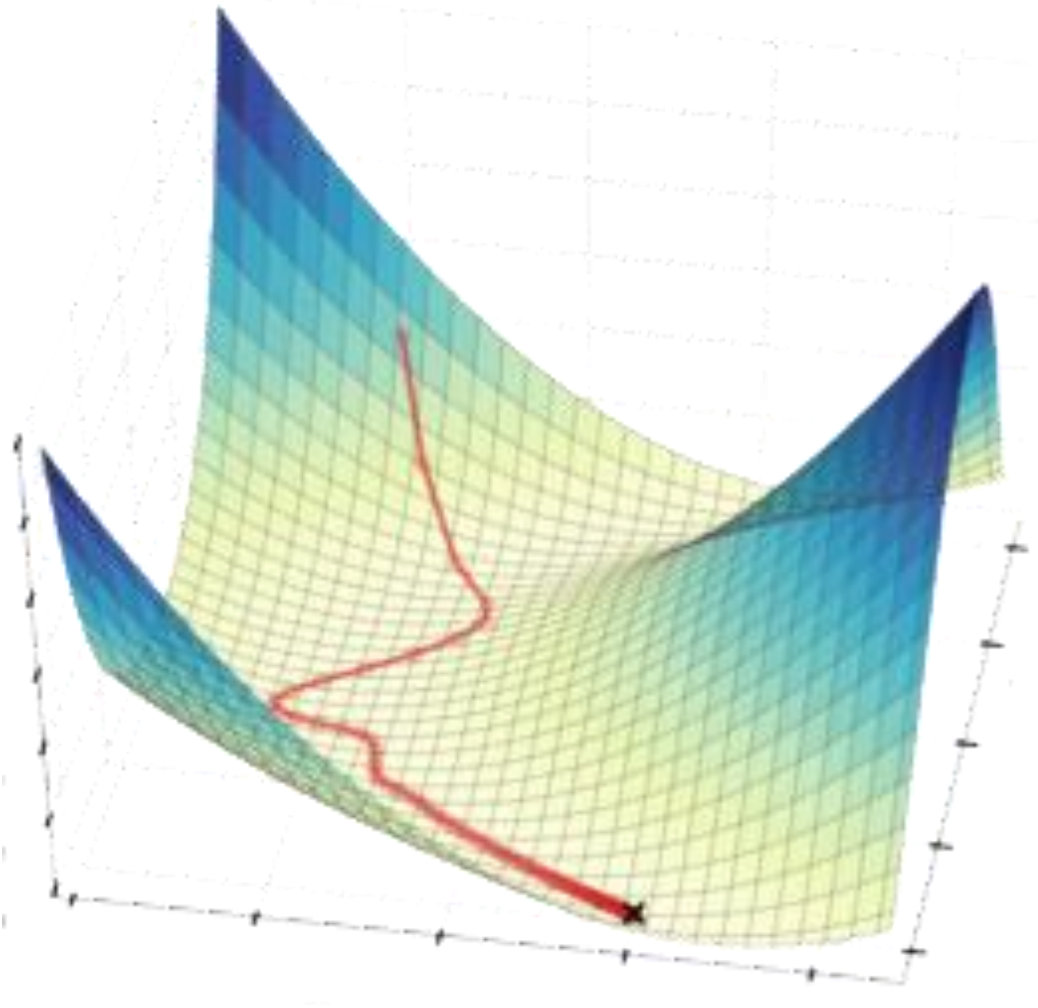


Figura 3 – Comportamento do gradiente descendente com passo adequado

Fonte: http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html

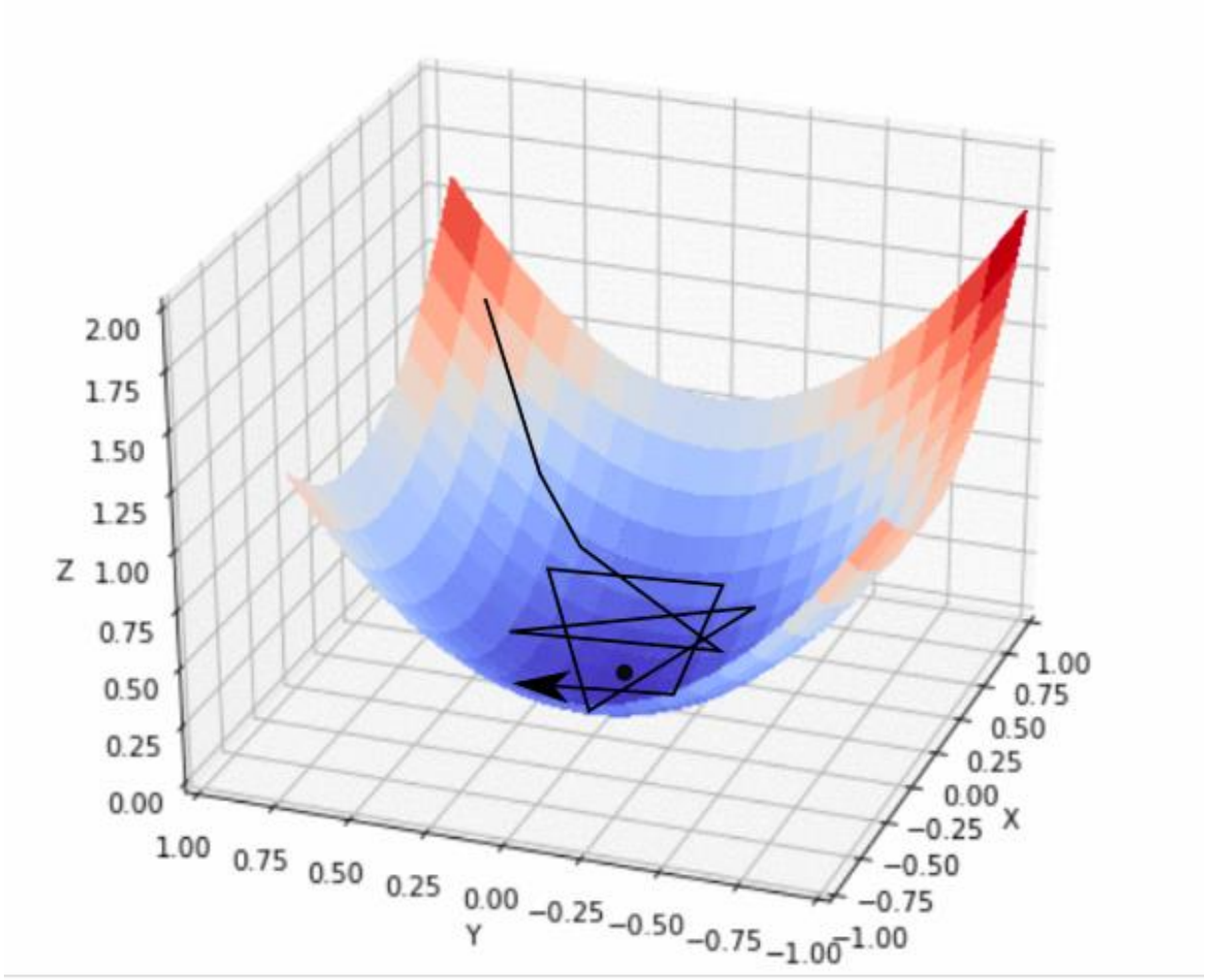



Figura 4 – Comportamento do gradiente descendente com passo inadequado

Fonte: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

- A questão do sobreajuste já foi abordada quando discutimos *holdout* e *k-fold crossvalidation* e voltará a ser tratada quando discutirmos *dropout*.
- Os desafios associados ao treinamento de redes neurais profundas (*deep learning*) levaram à proposição de algoritmos bastante competentes para a inicialização e o ajuste de pesos, no sentido de promoverem, em média, progressos expressivos na exploração da superfície de erro a cada iteração (RUDER, 2017).
- Esses avanços envolvem truques computacionais de natureza empírica, suportados por aspectos conceituais, como no caso de algoritmos adaptativos que não recorrem diretamente a informações de 2a. ordem (RUDER, 2017; MARTENS & SUTSKEVER, 2012), *batch normalization* (IOFFE & SZEGEDY, 2015) e *fixup initialization* (ZHANG et al., 2019). São apresentados, a seguir, noções de parte desses progressos recentes em otimização não-linear para *deep learning*.
- Evidentemente, existem iniciativas voltadas para outras estratégias de aprendizado de máquina, além do treinamento de RNAs (BOTTOU *et al*., 2018).

## *1.1 Mini-batch gradient descent*

- Trata-se de uma técnica intermediária entre os métodos padrão-a-padrão (*stochastic gradient descent* – SGD) e batelada (*batch*), já vista no Tópico 4 do curso. Mais detalhes podem ser obtidos em: [https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/].
- Definem-se partições ou lotes (subconjuntos) de dados de treinamento, sendo que o treinamento para cada lote é em batelada. A evolução do erro não oscila tanto quanto no caso padrão-a-padrão, mas possivelmente oscila o suficiente para escapar de mínimos locais ruins (LI *et al.*, 2014).
- Além disso, há um favorecimento das implementações em computação paralela e uma redução no uso de memória, já que não é necessário manter simultaneamente todos os dados de treinamento em memória.
- Supondo um lote de tamanho $N_L$: $\mathbf{w}(k+1) = \mathbf{w}(k) - \frac{\alpha}{N_L} \sum_{l=1}^{N_L} \nabla J_l\left(\mathbf{w}(k)\right)$.

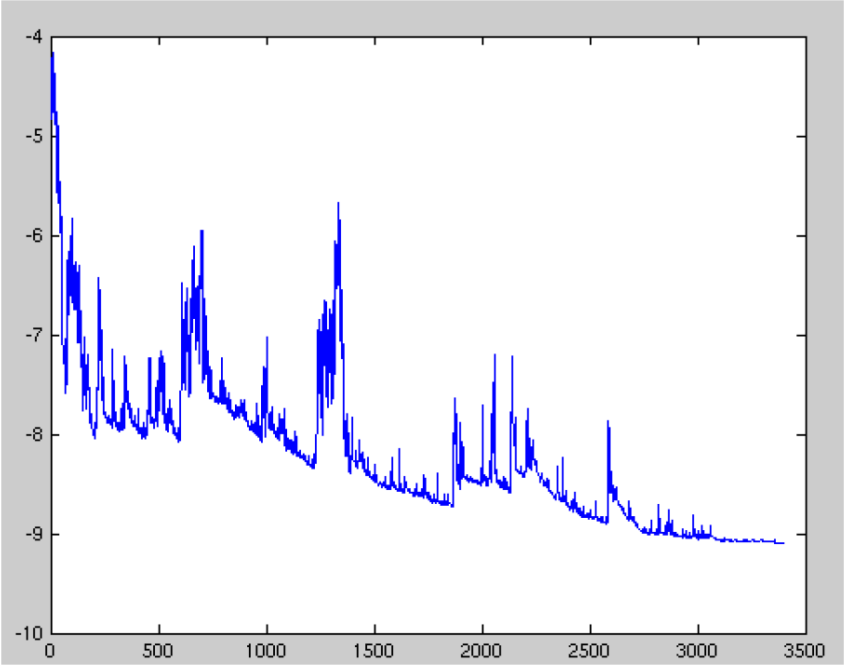



Figura 5 – Comportamento típico da função-objetivo numa busca por gradiente descendente.

Fonte: Wikipedia. Após RUDER (2017).

- Observação: Daqui em diante, vamos usar a notação $\nabla J_l\big(\mathbf{w}(k)\big)$ para se referir tanto ao gradiente associado à *l*-ésima amostra (abordagem padrão-a-padrão) como ao gradiente resultante do processamento do *l*-ésimo lote de amostras (abordagem *mini-batch*).

## 2. Algoritmos adaptativos

- Os algoritmos adaptativos foram motivados basicamente pela limitação associada ao uso de uma taxa fixa de aprendizado para todos os pesos sinápticos e para todas as iterações ou então pela definição prévia de uma política de adaptação dessa taxa de aprendizado. Em ambos os casos, é desafiador definir adequadamente e a priori essa única taxa de aprendizado ou então essa política de ajuste da taxa.
- Taxa de aprendizado aqui diz respeito ao passo do ajuste iterativo.
- Deve-se levar em conta que uma taxa de aprendizado muito pequena produz uma convergência muito lenta do treinamento, enquanto uma taxa de aprendizado muito elevada pode impedir a convergência do processo de ajuste, seja por apresentar oscilações em torno de um ótimo local, seja por divergir.
- Além disso, usar uma mesma taxa de aprendizado para todos os pesos ajustáveis, seja ela fixa ou adaptativa, pode não ser a melhor solução em aplicações práticas caracterizadas, por exemplo, pela existência de dados esparsos, em que o ajuste

dos pesos se dá em frequências distintas. Neste caso, pode não ser uma boa estratégia ajustar os pesos com a mesma intensidade a cada iteração. Tende a ser mais produtivo, assim, promover ajustes mais intensos para pesos que sofrem ajustes menos frequentes num histórico de ajustes recentes.

- Outro aspecto-chave em *deep learning* é a existência de uma quantidade expressiva de pontos de sela e de *plateaus* (DAUPHIN et al., 2014), além de mínimos locais de baixa qualidade. Escapar dessas "armadilhas" geralmente requer passos de treinamento adaptativos.

## 2.1 Algoritmo adaptativo 1: SGD + momentum

- Sua principal missão é reduzir oscilações durante a exploração da superfície de erro, mas também opera como acelerador sempre que gradientes subsequentes preservarem direções de busca (QIAN, 1999):

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha \nabla J_l\left(\mathbf{w}(k)\right) + \gamma\left[\mathbf{w}(k) - \mathbf{w}(k-1)\right]$$

- Um valor típico geralmente adotado para $\gamma$ é 0,9. De qualquer modo, deve-se considerar o fato de que o processo de ajuste ganhou uma ordem na dinâmica e um hiperparâmetro a mais, a ser devidamente definido.

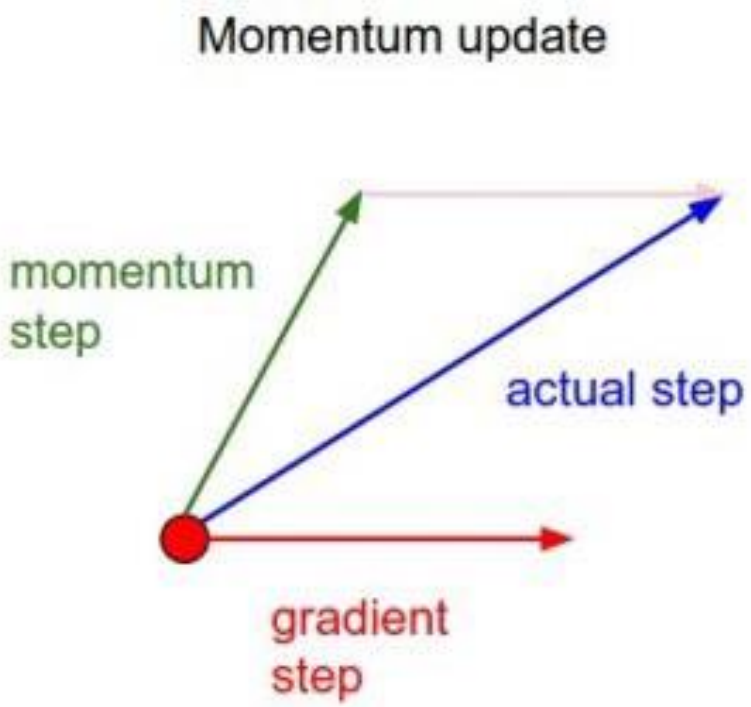



Figura 6 – Operação associada ao método SGD + momentum. Fonte: [https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d]

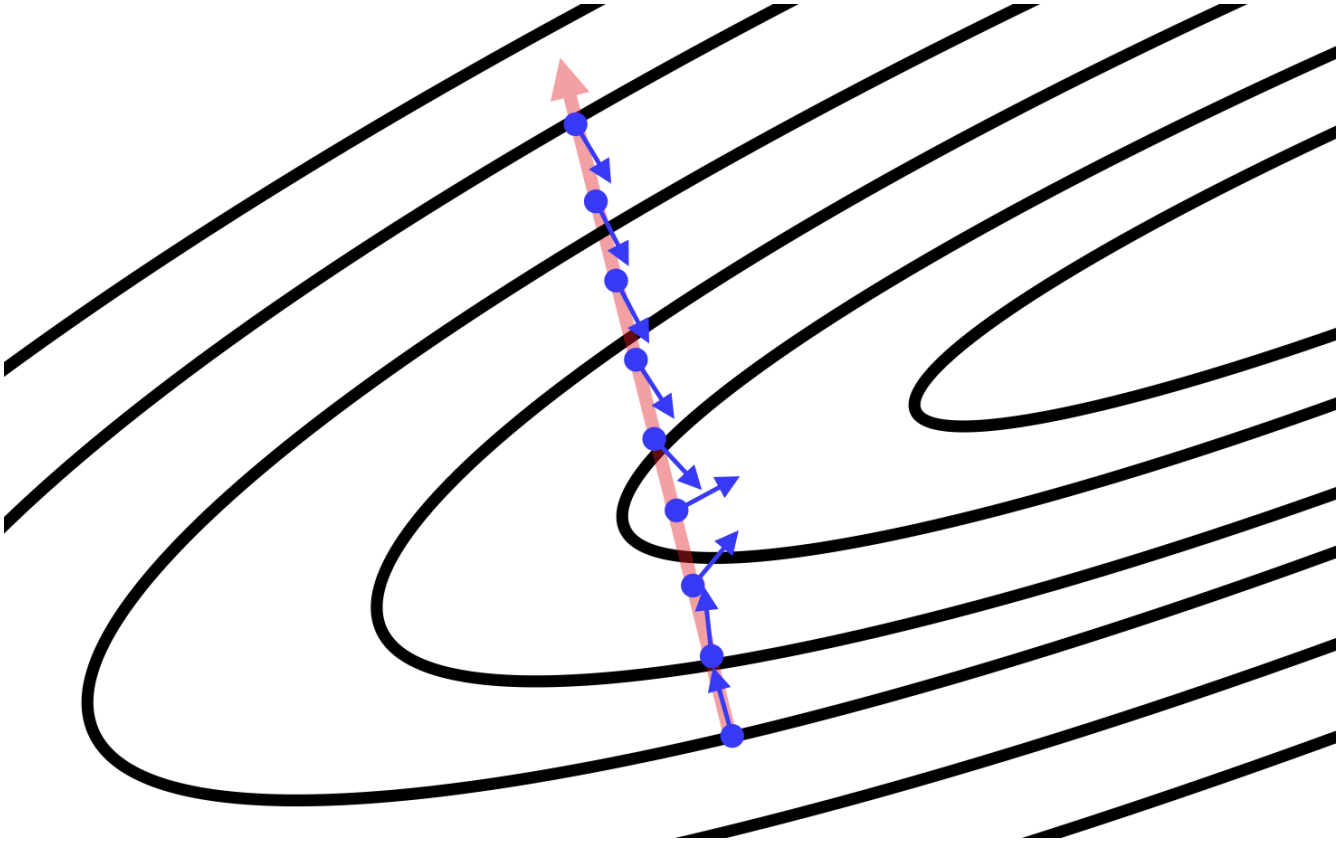


Figura 7 – Relação entre direção atual do gradiente descendente e direção seguinte (azul)

Fonte: https://learning.oreilly.com/library/view/fundamentals-of-deep/9781491925607/ch04.html

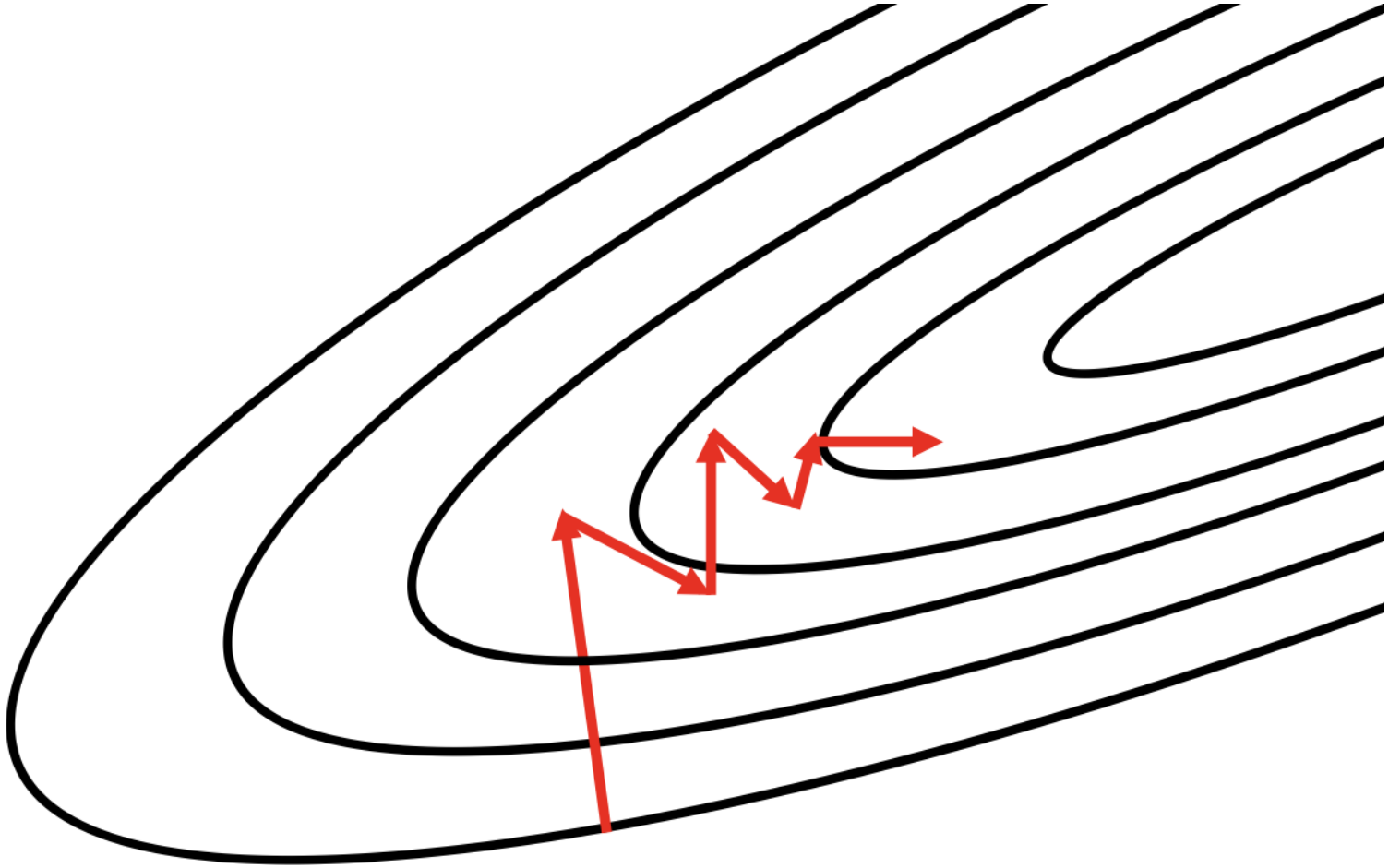


Figura 8 – Efeito de zig-zag na busca em virtude do fenômeno ilustrado na Figura 7

Fonte: https://learning.oreilly.com/library/view/fundamentals-of-deep/9781491925607/ch04.html

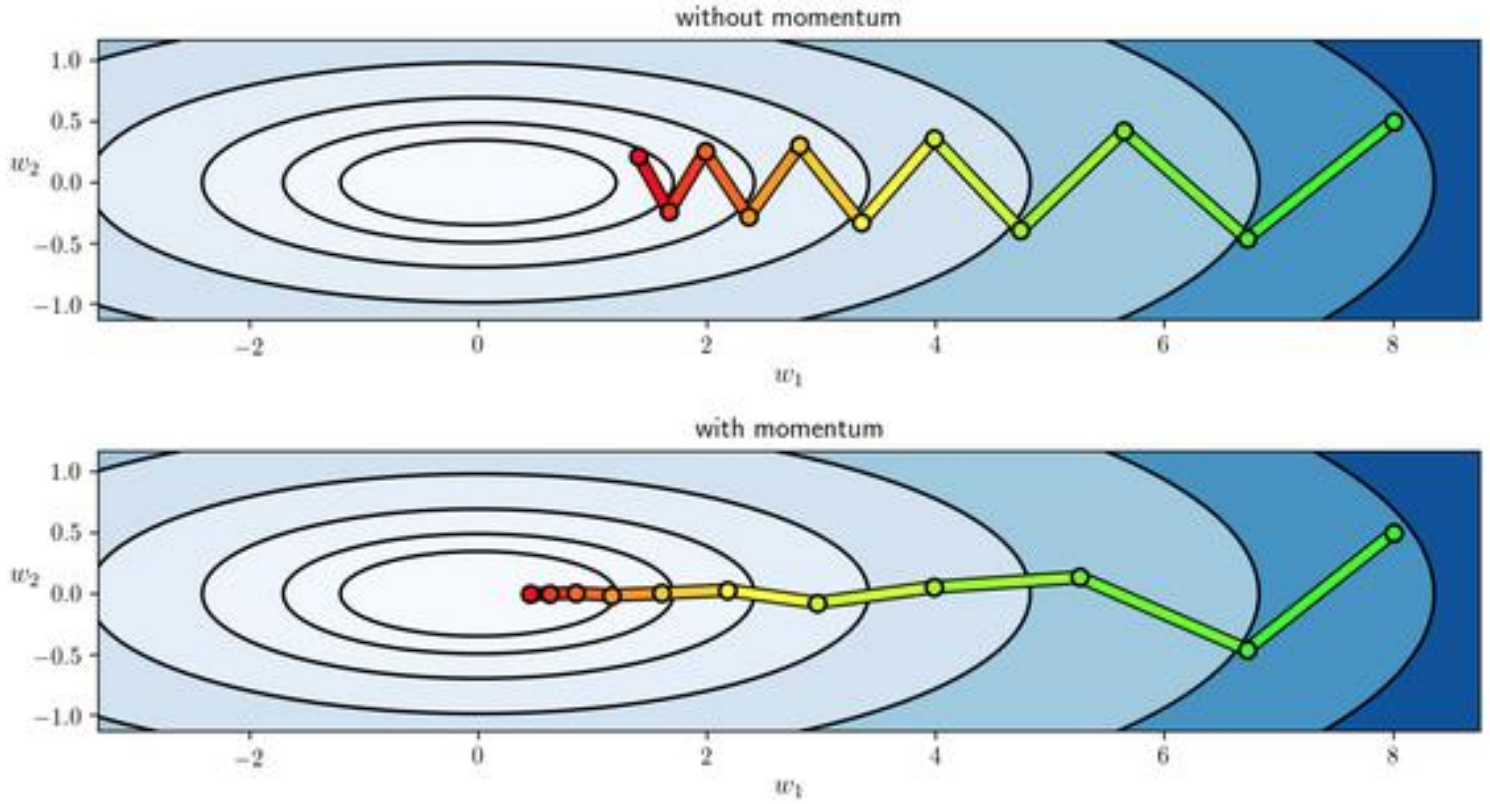



Figura 9 – Contribuição típica do termo de momentum na redução de oscilações.

Fonte: [https://www.quora.com/What-exactly-is-momentum-in-machine-learning]

## 2.2 Algoritmo adaptativo 2: *Nesterov accelerated gradient* (NAG)

- Numa tentativa de acelerar a busca, é possível calcular o vetor gradiente não em $\mathbf{w}(k)$, mas em $\mathbf{w}(k)+\gamma\left[\mathbf{w}(k)-\mathbf{w}(k-1)\right]$, que é uma aproximação do novo valor do vetor de pesos da rede neural. Com isso, a regra de ajuste assume a forma:

$$\mathbf{w}(k+1)=\mathbf{w}(k)-\alpha\nabla J_l\left(\mathbf{w}(k)+\gamma\left[\mathbf{w}(k)-\mathbf{w}(k-1)\right]\right)+\gamma\left[\mathbf{w}(k)-\mathbf{w}(k-1)\right]$$

- Trata-se de uma tentativa de antecipar o comportamento da superfície de erro.

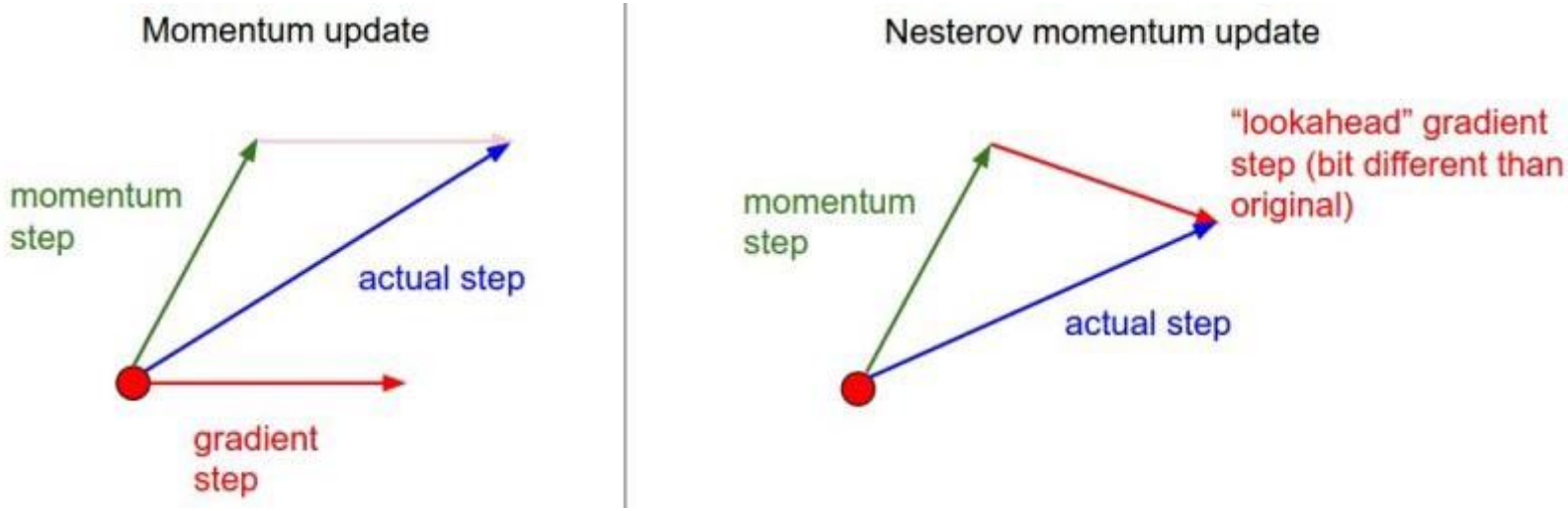



Figura 10 – Motivação geométrica da diferença entre SGD+momentum e NAG. Fonte: [https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d]

## 2.3 Estado-da-arte em algoritmos adaptativos

- Antes de abordar as propostas mais avançadas em algoritmos adaptativos, é importante salientar que há dois aspectos a serem considerados ao medir a competência de um algoritmo de otimização não-linear no treinamento de redes neurais artificiais: (1) eficiência computacional; e (2) capacidade de produzir redes neurais que generalizam bem.
- De fato, os algoritmos adaptativos de uso mais frequente na literatura tendem a produzir ganhos de eficiência computacional, mas podem levar a redes neurais que generalizam pior que aquelas produzidas por *mini-batch* SGD (*stochastic gradient descent*), conforme investigado de forma empírica e também com evidências teóricas no trabalho de WILSON *et al*. (2018). A principal conclusão de WILSON *et al*. (2018) é que métodos adaptativos e não-adaptativos tendem a obter soluções de otimização diferentes e apresentar capacidades de generalização bem distintas.

- Como as propostas de algoritmos adaptativos são bem recentes, assim como essas críticas ao seu comportamento em aplicações práticas, associadas ao treinamento de redes neurais artificiais profundas ou não, cabem estudos mais detalhados visando chegar a regras de uso mais objetivas e efetivas. Havendo recursos computacionais disponíveis, por hora, a recomendação é treinar a rede neural com *mini-batch SCG* e também com ADAM (veja descrição mais adiante), comparando o desempenho das redes neurais produzidas em cada caso, em termos de capacidade de generalização.
- O trabalho de RUDER (2017) apresenta uma formalização didática acerca das principais propostas em algoritmos adaptativos, sendo que daremos destaque aqui às seguintes propostas:
  - ✓ Adagrad (DUCHI *et al.*, 2011)
  - ✓ Adadelta (ZEILER, 2012)
  - ✓ ADAM (*Adaptive Moment Estimation*) (KINGMA & BA, 2015)

## 2.4 Algoritmo adaptativo 3: Adagrad

- O Adagrad realiza um ajuste do passo de aprendizado de acordo com a intensidade acumulada com que cada peso já foi ajustado ao longo da busca, até a iteração corrente.
- Para a *l*-ésima amostra (na abordagem padrão-a-padrão) ou então para o *l*-ésimo lote de amostras (na abordagem *mini-batch*), considere que $\nabla J_l\left(\mathbf{w}_i(k)\right)$ representa o *i*-ésimo elemento do vetor gradiente de dimensão *P*. No método convencional de *stochastic gradient descent* (SGD), adota-se um mesmo passo $\alpha$ para todos os pesos, produzindo:

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) - \alpha \nabla J_l\left(\mathbf{w}_i(k)\right)$$

- Logo, SGD não é um algoritmo adaptativo.
- Por outro lado, o Adagrad vai adotar um passo específico para cada peso:

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) - \frac{\alpha}{\sqrt{G_{ii}(k)+\varepsilon}} \nabla J_l\left(\mathbf{w}_i(k)\right)$$

onde $G_{ii}(k)$ representa o *i*-ésimo elemento da diagonal da matriz $G(k)$ de dimensão $P \times P$. A matriz $G(k)$ é uma matriz diagonal em que seu *i*-ésimo elemento da diagonal armazena a soma do quadrado dos gradientes associados ao *i*-ésimo peso da rede neural, até a iteração *k*. O coeficiente $\varepsilon$ geralmente assume o valor $10^{-8}$ e evita divisão por zero. Já o passo $\alpha$ passa a ter uma influência menor no desempenho da otimização, tendo atribuído a si o valor fixo 0,01.

- Computacionalmente, o ajuste do vetor de pesos é feito de forma matricial, como segue:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \frac{\alpha}{\sqrt{G(k) + \varepsilon}} \odot \nabla J_l\left(\mathbf{w}(k)\right)$$

- Uma desvantagem evidente deste algoritmo adaptativo é a tendência de passos cada vez menores ao longo das iterações, possivelmente impedindo o progresso do aprendizado após muitas iterações.

## 2.5 Algoritmo adaptativo 4: Adadelta

- O algoritmo Adadelta veio para combater essa redução agressiva e monotônica do passo de aprendizado presente no algoritmo Adagrad.
- Em lugar de acumular todos os gradientes prévios, desde o início da busca, uma solução seria definir uma janela de iterações passadas de tamanho *W*. O problema com esta estratégia, além de requerer uma definição adequada de *W*, é o custo computacional associado à atualização desta janela na soma cumulativa.
- Uma solução mais eficiente envolve a inclusão de um fator de esquecimento $\gamma$, geralmente definido em torno de 0,9, permitindo que ajustes mais recentes pesem mais no cálculo cumulativo, produzindo:

$$\mathbf{v}_k = \gamma \mathbf{v}_{k-1} + (1-\gamma)\nabla J_l^{[2]}(\mathbf{w}(k))$$

  onde $\nabla J_l^{[2]}(\mathbf{w}(k))$ é um vetor contendo o quadrado dos elementos de $\nabla J_l(\mathbf{w}(k))$.
- O mesmo é feito para a média do ajuste de pesos, resultando em:

$$\mathbf{q}_k = \gamma \mathbf{q}_{k-1} + (1-\gamma)\Delta \mathbf{w}^{[2]}(k)$$

onde $\Delta \mathbf{w}^{[2]}(k)$ é um vetor contendo o quadrado dos elementos de $\Delta \mathbf{w}(k) = \mathbf{w}(k) - \mathbf{w}(k-1)$.

- Obtém-se, finalmente, o passo iterativo do algoritmo Adadelta:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \frac{\sqrt{\mathbf{q}_{k-1} + \varepsilon}}{\sqrt{\mathbf{v}_k + \varepsilon}} \odot \nabla J_l\left(\mathbf{w}(k)\right)$$

- O coeficiente $\varepsilon$ geralmente assume o valor $10^{-8}$ e evita divisão por zero.

## 2.6 Algoritmo adaptativo 5: ADAM

- O ADAM (*adaptive moment estimation*) também recorre a um passo de aprendizado específico para cada peso da rede neural. Para tanto, ele recorre a estimativas de primeiro momento (média) e segundo momento (variância) dos gradientes consecutivos, na forma:

$$\mathbf{m}_k = \gamma_1 \mathbf{m}_{k-1} + (1-\gamma_1)\nabla J_l\left(\mathbf{w}(k)\right);$$

$$\mathbf{v}_k = \gamma_2 \mathbf{v}_{k-1} + (1-\gamma_2)\nabla J_l^{[2]}(\mathbf{w}(k)).$$

- Como $\mathbf{m}_0$ e $\mathbf{v}_0$ são tomados como sendo vetores nulos, esses momentos são enviesados e requerem uma correção, que é realizada na forma:

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1-\gamma_1^k};$$

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\gamma_2^k}.$$

- A regra de ajuste de pesos, então, assume a forma:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_k} + \varepsilon}\hat{\mathbf{m}}_k,$$

sendo que os valores recomendados para os hiperparâmetros são: $\gamma_1 = 0{,}9$; $\gamma_2 = 0{,}999$ e $\varepsilon = 10^{-8}$.

## 2.7 Métodos de segunda ordem

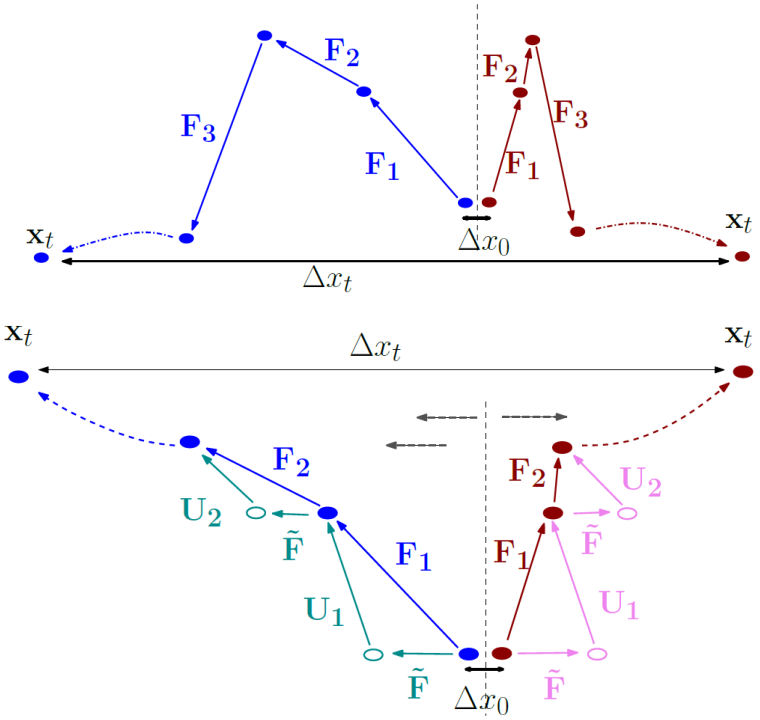
- Embora métodos de segunda ordem tradicionais, como o método de Newton, não sejam factíveis quando o número de parâmetros ajustáveis é elevado, pois requerem o armazenamento e a inversão da matriz hessiana, existem iniciativas que viabilizam o emprego da informação de segunda ordem sem recorrer diretamente à matriz hessiana, sendo técnicas bastante promissoras em *deep learning* (MARTENS, 2010; MARTENS & SUTSKEVER, 2012), mas ainda pouco exploradas.
- Tomando ainda o método de Newton como referência, evidencia-se que a informação de segunda ordem exerce dois papéis fundamentais: corrigir a direção associada à informação de primeira ordem e determinar o passo da busca, embora requeira aqui que a matriz hessiana seja definida positiva.
- Portanto, um método de segunda ordem tende a ser mais custoso por iteração, mas tende a levar bem menos iterações para convergir.

## 3. O problema dos gradientes explosivos e que se anulam

- A aplicação de métodos baseados em gradiente para o treinamento de redes neurais profundas pode se defrontar com um de dois fenômenos que inviabilizam o ajuste de pesos: gradientes explosivos e gradientes que se anulam (do inglês *exploding and vanishing gradients*).
- Os gradientes explosivos podem promover ajustes exagerados nos vetores de pesos, saturando neurônios e destruindo progressos anteriores obtidos pelo ajuste iterativo vinculado ao treinamento supervisionado.
- Os gradientes que se anulam impedem que pesos sinápticos continuem a sofrer ajustes significativos durante o treinamento supervisionado.
- Um estudo mais aprofundado desses dois fenômenos e de como lidar com eles foi apresentado em PASCANU *et al*. (2013), no contexto de redes neurais recorrentes.
- Entende-se que os gradientes explosivos ocorrem quando a operação do sistema dinâmico, representado pela rede neural recorrente sob treinamento, cruza a

fronteira entre duas bacias de atração. Logo, a solução proposta é evitar cruzar esta fronteira ou definir heurísticas capazes de lidar com o processo de ajuste durante este processo de cruzamento de fronteiras entre bacias de atração.

- As figuras a seguir, extraídas de PASCANU *et al.* (2013), mostram cenários de gradiente explosivo para sistemas dinâmicos com e sem entrada externa.



- Por outro lado, entende-se que os gradientes que se anulam são produzidos quando o sistema dinâmico se encontra próximo de um ponto fixo, ou seja, próximo da região de convergência de uma bacia de atração.
- Logo, a solução proposta é permanecer, durante o treinamento, próximo a fronteiras entre bacias de atração, sem cruzá-las.
- Para redes neurais não-recorrentes, profunda ou não, gradientes explosivos surgem com superfícies de erro localmente muito acidentadas, enquanto gradientes que se anulam surgem com superfícies de erro localmente muito planas.
- A estratégia de inicialização denominada FIXUP (ZHANG et al. 2019) tenta contornar esse problema, sendo que foram reportados casos de sucesso no treinamento de redes neurais com mais de 10.000 camadas. Também *batch normalization*, a ser descrita mais adiante neste tópico do curso, foi demonstrada ser capaz de reescalar adequadamente a superfície de erro (SANTURKAR et al., 2019).

# 4. A função softmax e sua derivada

- A função softmax toma um vetor *n*-dimensional, cujos elementos assumem valores reais arbitrários, e produz outro vetor *n*-dimensional com valores reais no intervalo (0,1) e que somam na unidade. Trata-se de um mapeamento:

$$S(\mathbf{x}):\Re^n \rightarrow \Re^n, \text{ ou seja, } S(\mathbf{x}):\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{bmatrix}$$

com $S_j = \dfrac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}, \forall j \in \{1,2,...,n\}$.

- Exemplo 1:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \Rightarrow S(\mathbf{x}) = \begin{bmatrix} 0,09 \\ 0,24 \\ 0,67 \end{bmatrix}$$

- Exemplo 2:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} \Rightarrow S(\mathbf{x}) = \begin{bmatrix} 0,02 \\ 0,05 \\ 0,93 \end{bmatrix}$$

- Perceba que a função *softmax*, como o seu nome sugere, pode ser interpretada como uma versão suave (*soft*) da função máximo.
- O principal apelo da função *softmax* é permitir que a sua saída seja interpretada em termos de probabilidades. Por exemplo, em tarefas de aprendizado envolvendo múltiplas classes, é muito útil poder indicar a probabilidade de a entrada pertencer a uma dentre um conjunto de classes.
- Logo, se a saída *y* for uma variável aleatória, a interpretação é:

$$S_j = P(y = j|\mathbf{x})$$

- A forma mais simples de explorar esta propriedade é em regressão logística multiclasse, onde um vetor de entrada **x** é multiplicado por uma matriz *W* e o resultado é a entrada de uma função *softmax*.
- Como a função *softmax* recebe um vetor de entrada e produz na saída um vetor de mesma dimensão, então é possível obter a derivada da *i*-ésima saída em relação ao *j*-ésimo argumento da função *softmax*, produzindo a matriz jacobiana *D*:

$$D = \begin{bmatrix} \frac{\partial S_1}{\partial x_1} & \frac{\partial S_1}{\partial x_2} & \cdots & \frac{\partial S_1}{\partial x_n} \\ \frac{\partial S_2}{\partial x_1} & \frac{\partial S_2}{\partial x_2} & \cdots & \frac{\partial S_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial S_n}{\partial x_1} & \frac{\partial S_n}{\partial x_2} & \cdots & \frac{\partial S_n}{\partial x_n} \end{bmatrix}.$$

- Cálculo da derivada $\frac{\partial S_i}{\partial x_j} = \frac{\frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}}{\partial x_j}, \forall i, j \in \{1, 2, ..., n\}$.

- Lembre-se que, se $f(x) = \frac{g(x)}{h(x)}$, então:

$$f'(x) = \frac{h(x)g'(x) - g(x)h'(x)}{\left[h(x)\right]^2}$$

- Fazendo:

$$g_i = e^{x_i} \quad \text{e} \quad h = \sum_{k=1}^{n} e^{x_k}$$

constata-se que:

$$\frac{\partial h}{\partial x_j} = e^{x_j}, \forall j \in \{1, 2..., n\}$$

$$\frac{\partial g_i}{\partial x_j} = \begin{cases} e^{x_j} & \text{se } i = j \\ 0 & \text{caso contrário.} \end{cases}$$

- Começando pelo caso $i = j$:

$$\frac{\partial S_i}{\partial x_j} = \frac{\frac{g_i}{h}}{\partial x_j} = \frac{e^{x_i} h - e^{x_j} e^{x_i}}{h^2} = \frac{e^{x_i}}{h} \cdot \frac{h - e^{x_j}}{h} = S_i \left(1 - S_j\right)$$

- E agora deduzindo para o caso $i \neq j$:

$$\frac{\partial S_i}{\partial x_j} = \frac{\frac{g_i}{h}}{\partial x_j} = \frac{0 - e^{x_j} e^{x_i}}{h^2} = -\frac{e^{x_i}}{h} \cdot \frac{e^{x_j}}{h} = -S_i S_j$$

- Em resumo:

$$\frac{\partial S_i}{\partial x_j} = \begin{cases} S_i \left(1 - S_j\right) & \text{se } i = j \\ -S_i S_j & \text{caso contrário.} \end{cases}$$

- Uma forma condensada usa o delta de Kronecker $\delta_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases}$ produz:

$$\frac{\partial S_i}{\partial x_j} = S_i\left(\delta_{ij} - S_j\right)$$

- Particularmente em *deep learning*, em virtude da grande quantidade de parâmetros ajustáveis, existem formulações mais parcimoniosas que evitam o cálculo explícito da matriz jacobiana, a qual terá dimensões muito elevadas.
- Também cabe mencionar que, para valores elevados de $x_j$, $j \in \{1, 2..., n\}$, pode ocorrer *overflow*. Sendo assim, é possível aplicar um escalamento das saídas, pois é sabido que, para qualquer constante arbitrária *C*, vale:

$$S_j = \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}} = \frac{Ce^{x_j}}{\sum_{k=1}^{n} Ce^{x_k}} = \frac{e^{x_j + \ln(C)}}{\sum_{k=1}^{n} e^{x_k + \ln(C)}}.$$

- Um valor automático indicado é:

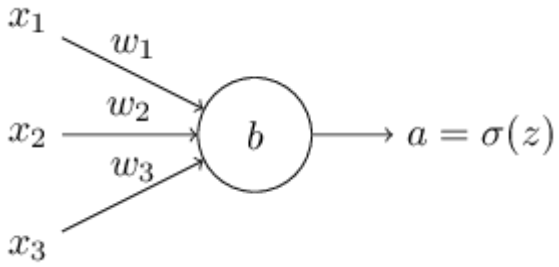
$$\ln(C) = -\max\{x_1, x_2, ..., x_n\}.$$

# 5. A entropia cruzada como função-custo

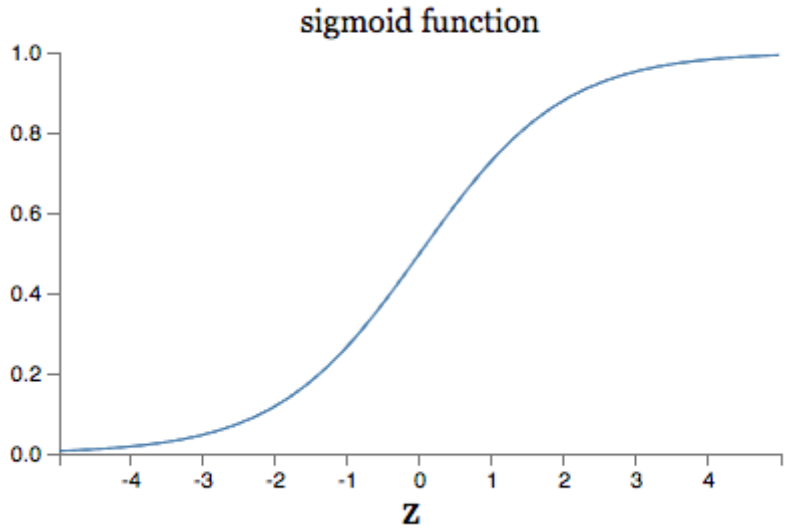
- Nesta seção, vamos motivar a escolha da entropia cruzada considerando o treinamento de um único neurônio do tipo perceptron para um problema de classificação binária. Tomaremos por base o material disponível em: http://deeplearningbook.com.br/cross-entropy-cost-function/
- Considerando *N* amostras de entrada:

$$\mathbf{x}_l = \begin{bmatrix} x_{l1} \\ x_{l2} \\ x_{l3} \end{bmatrix}, l \in \{1, 2, ..., N\},$$

o objetivo é ajustar os 3 pesos sinápticos do neurônio a seguir.



sendo $z_l = \sum_{i=1}^{3} w_i x_{li} + b, l \in \{1,2,...,N\}$ e a função de ativação σ dada na forma:

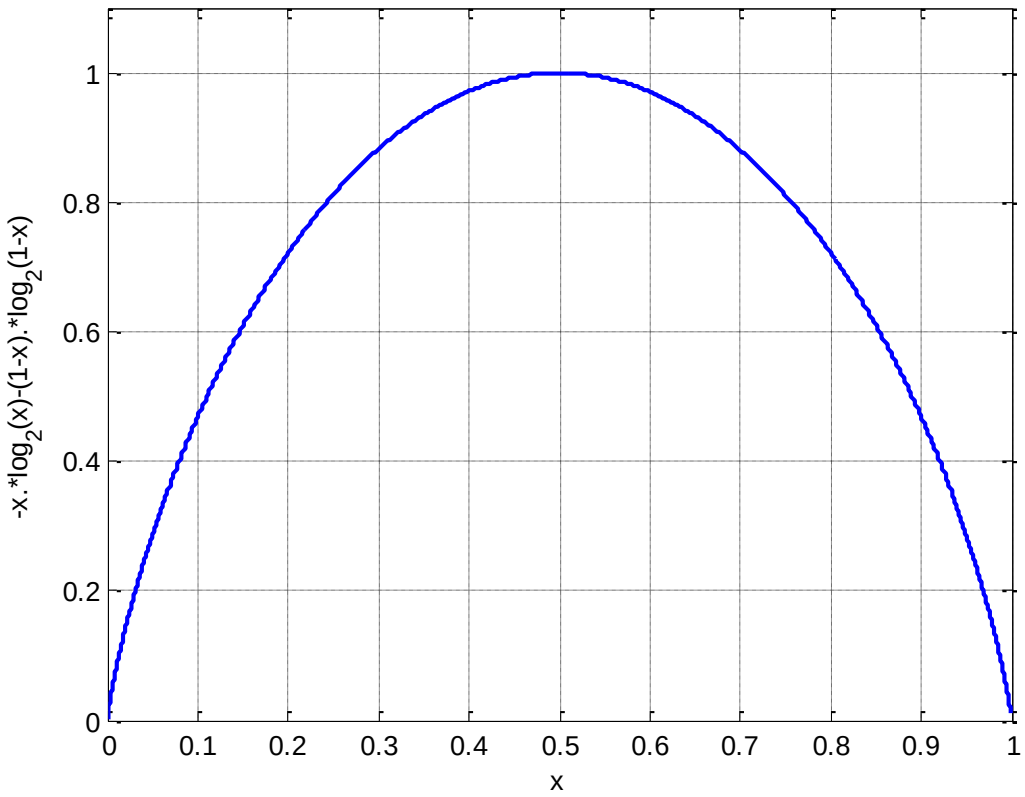



- A função-custo de entropia cruzada é como segue:

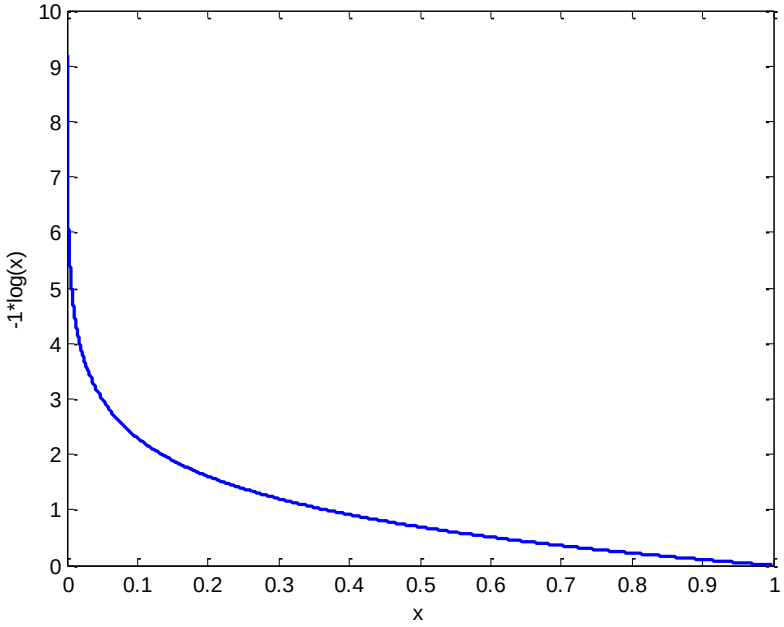
$$C = -\frac{1}{N}\sum_{l=1}^{N}\left[y_l \ln\left(a_l\right) + \left(1-y_l\right)\ln\left(1-a_l\right)\right].$$

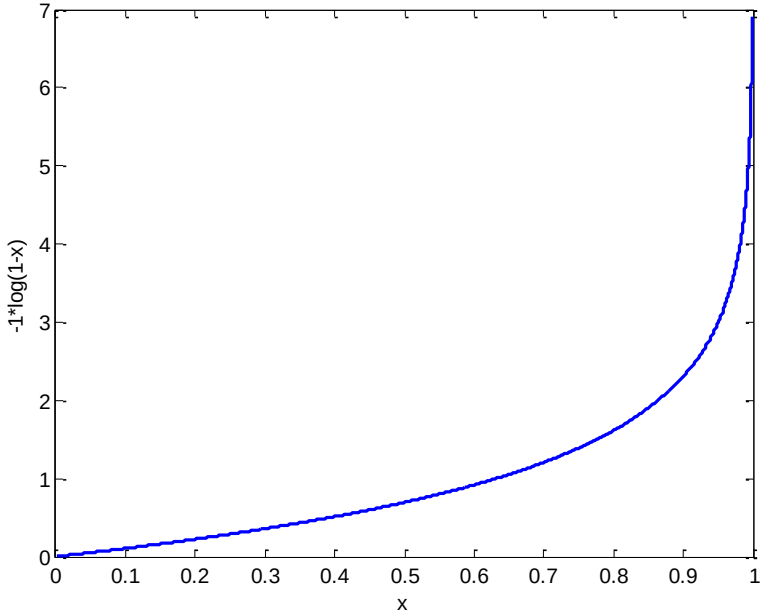
onde $y_l \in \{0,1\}$ é a saída desejada para a *l*-ésima entrada $\mathbf{x}_l$.

- Perceba que, como garantidamente $a_l \in (0,1)$, então é certo que $C > 0$.




Entropia de Shannon – grau médio de incerteza a respeito de uma fonte binária de informação.




- Portanto, a entropia cruzada é positiva e tende a zero à medida que a saída do neurônio se aproxima da saída desejada, para todo *N*.
- Uma outra propriedade decisiva para garantir a vantagem da entropia cruzada como função-custo, quando comparada, por exemplo, com o erro quadrático médio e supondo saídas restritas ao intervalo (0,1), é o fato de que o gradiente em

relação aos pesos não depende da derivada da função de ativação, de modo que o treinamento é acelerado mesmo para o caso de saturação da função de ativação:

$$\frac{\partial C}{\partial w_j} = \frac{1}{N}\sum_{l=1}^{N}\left[ x_{lj}\left(\sigma\left(z_l\right) - y_l\right)\right] = \frac{1}{N}\sum_{l=1}^{N}\left[ x_{lj}\left(a_l - y_l\right)\right].$$

- No caso do erro quadrático médio, seria: $\frac{\partial C}{\partial w_j} = \frac{1}{N}\sum_{l=1}^{N}\left[ x_{lj}\left(a_l - y_l\right)\sigma'\left(z_l\right)\right]$

- Para uma base $b > 0$ qualquer, sabe-se que:

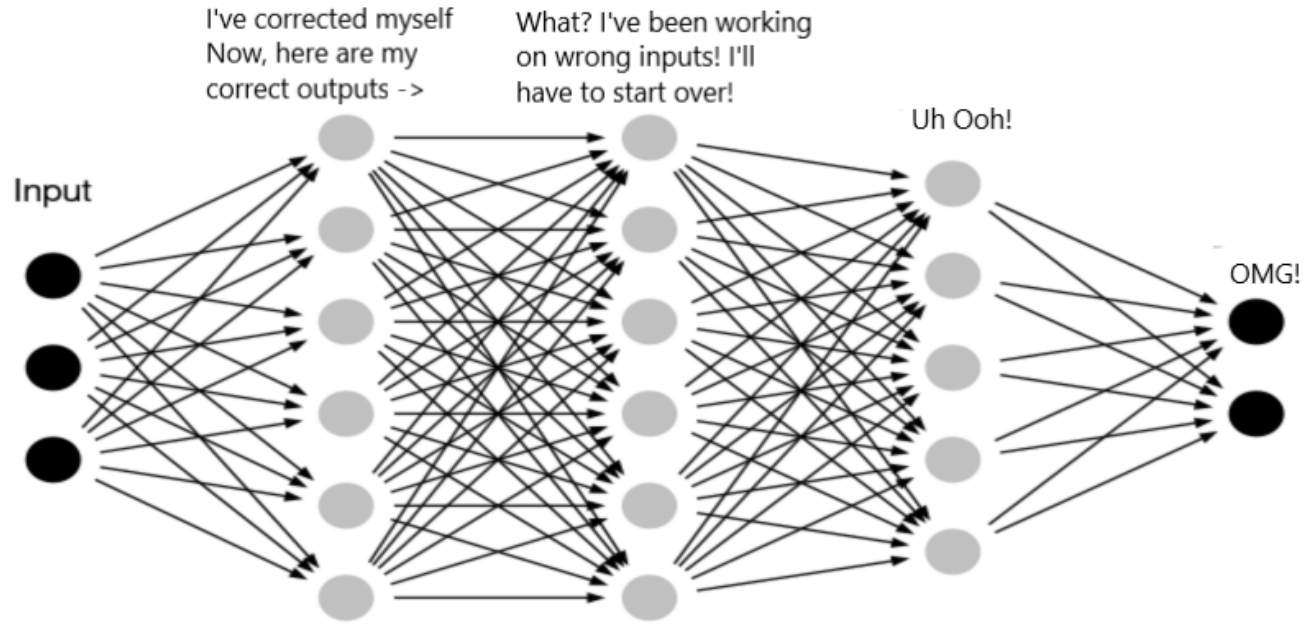
$$\frac{d}{dx}\log_b\left(x\right) = \frac{1}{x\ln\left(b\right)},$$

o que motiva o uso da base $e = \exp(1)$ no critério de entropia cruzada, produzindo:

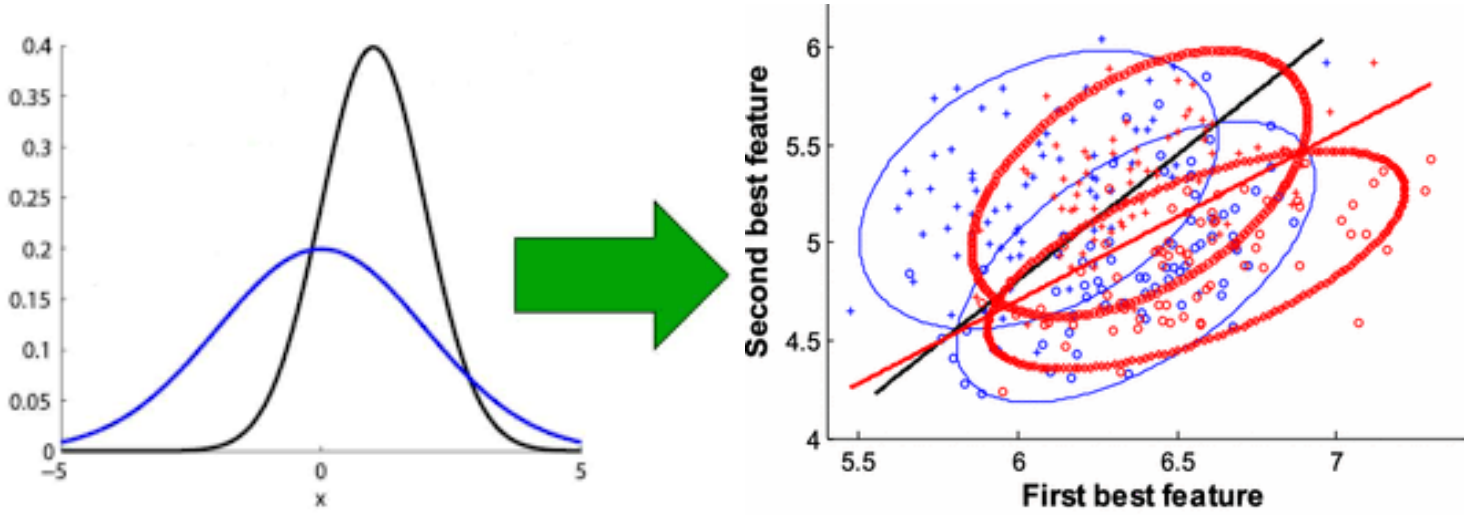
$$\frac{d}{dx}\log_e\left(x\right) = \frac{1}{x\ln\left(e\right)} = \frac{1}{x}$$

## 6. *Batch normalization* e *fixed-update initialization*

- Durante o treinamento, um fenômeno bem expressivo no caso de redes neurais profundas é a mudança da distribuição do sinal de entrada de cada camada, em virtude do ajuste dos pesos sinápticos das camadas anteriores.

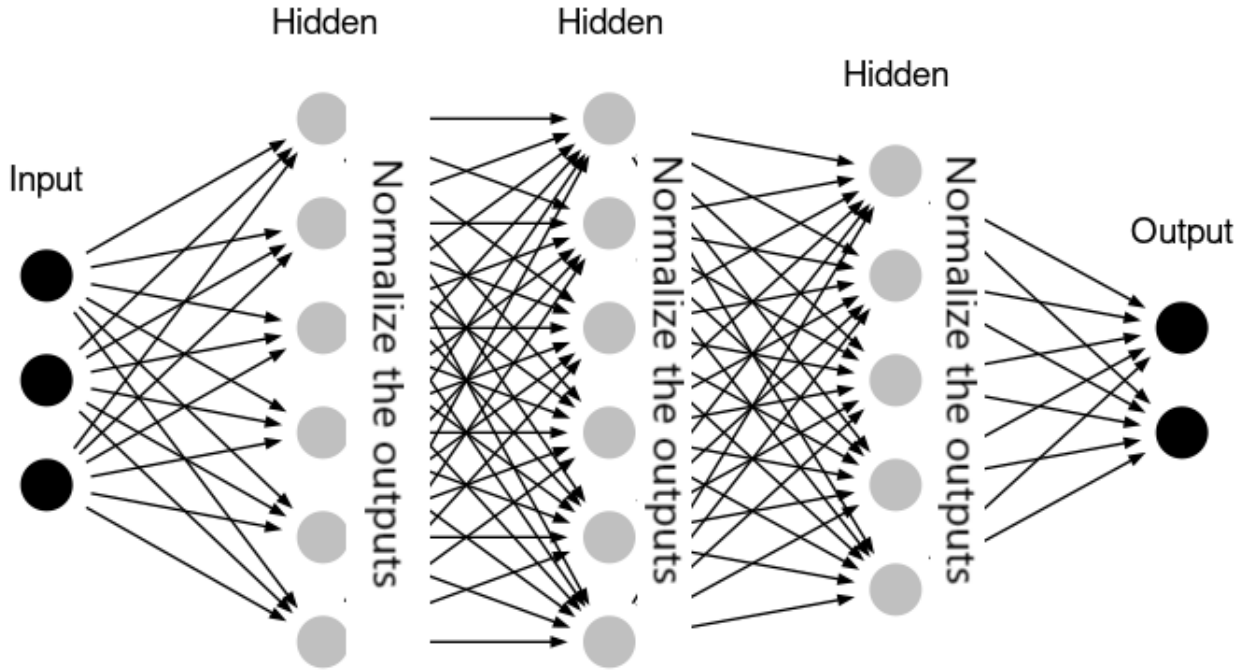



Fonte: https://mc.ai/batch-normalization-speed-up-neural-network-training/




- Esse fenômeno é chamado de *internal covariate shift* e sua eliminação ou atenuação tende a acelerar significativamente o processo de treinamento.
- Isso pode ser conquistado ao se adotar uma técnica denominada *batch normalization* (IOFFE & SZEGEDY, 2015), que, além de acelerar o treinamento, conduz a modelos de aprendizado que regularizam melhor, reduzindo a necessidade de *dropout*.

- A técnica é robusta o suficiente para executar a normalização sem criar dificuldades para o cálculo do vetor gradiente e sem requerer uma análise para todo o conjunto de treinamento toda vez que pesos sinápticos forem ajustados.

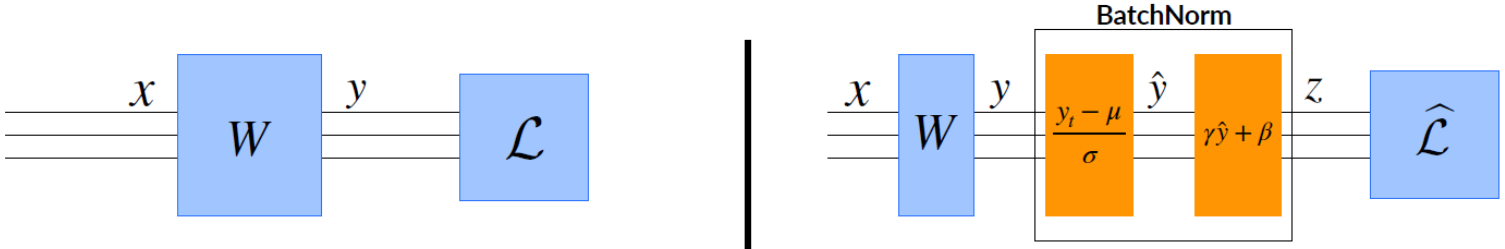



Fonte: https://mc.ai/batch-normalization-speed-up-neural-network-training/

- Curiosamente, embora IOFFE & SZEGEDY (2015) defendessem que o motivo do sucesso da técnica estava na atenuação do efeito de *covariate shift*, Santurkar et al.

(2019) mostraram que os benefícios da técnica advêm de outra motivação: o emprego de *batch normalization* promove uma suavização da superfície de erro, o que conduz a gradientes mais estáveis e possibilita a adoção de passos de treinamento maiores, os quais levam efetivamente a uma redução do erro.

- Com um gradiente mais estável, adotar passos mais largos na direção do gradiente atual conduz a um novo vetor gradiente que tende a exibir alta correlação com o gradiente anterior, o que torna o progresso acentuado do treinamento mais verossímel, reduzindo inclusive a influência do conjunto inicial de pesos sinápticos no desempenho final do modelo de aprendizado.
- Além disso, Santurkar et al. (2019) também mostraram que *batch normalization* não é a única técnica capaz de promover essa suavização da superfície de erro, podendo inclusive nem ser a técnica mais eficiente. A inicialização FIXUP (Zhang et al., 2019), *fixed-update initialization*, abre mão de *batch normalization* ao adotar um reescalamento adequado de inicializações convencionais de pesos.




(a) Vanilla Network

(b) Vanilla Network + BatchNorm Layer

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

**Algorithm 1:** Batch Normalizing Transform, applied to activation $x$ over a mini-batch.

Extraído de Santurkar et al. (2019).

# 7. Referências

BENGIO, Y. “Practical Recommendations for Gradient-Based Training of Deep Architectures”, arXiv:submit/0500581, 2012.

BOTTOU, L., CURTIS, F.E., NOCEDAL, J. “Optimization Methods for Large-Scale Machine Learning”, arXiv:1606.04838v3, 2018.

DAUPHIN, Y.N., PASCANU, R., GULCEHRE, C., CHO, K., GANGULI, S., BENGIO, Y. “Identifying and attacking the saddle point problem in high-dimensional nonconvex optimization”, arXiv:1406.2572v1, pp. 1–14, 2014.

DUCHI, J., HAZAN, E., SINGER, Y. “Adaptive subgradient methods for online learning and stochastic optimization”, *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

IOFFE, S., SZEGEDY, C. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, arXiv:1502.03167v3, 2015.

KINGMA, D.P., BA, J.L. “Adam: A Method for Stochastic Optimization. International Conference on Learning Representations”, pp. 1-13, 2015.

LI, M., ZHANG, T., CHEN, Y., SMOLA, A.J. "Efficient Mini-batch Training for Stochastic Optimization", Proceedings of the 20th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (KDD'2014), pp. 661-670, 2014.

MARTENS, J. "Deep learning via Hessian-free optimization", Proceedings of the 27th International Conference on Machine Learning (ICML'2010), pp. 735-742, 2010.

MARTENS, J., SUTSKEVER, I. "Training Deep and Recurrent Networks with Hessian-Free Optimization", in Neural Networks: Tricks of the Trade, pp. 479-535, 2012.

QIAN, N. "On the momentum term in gradient descent learning algorithms", Neural Networks, vol. 12, pp. 145-151, 1999.

RUDER, S. "An overview of gradient descent optimization algorithms", arXiv: 1609.04747v2, 2017.

SANTURKAR, S. TSIPRAS, D., ILYAS, A., MADRY, A. "How Does Batch Normalization Help Optimization?", arXiv:1805.11604v5, 2019.

WILSON, A.C., ROELOFS, R., STERN, M., SREBRO, N., RECHT, B. "The Marginal Value of Adaptive Gradient Methods in Machine Learning", arXiv: 1705.08292v2, 2018.

ZEILER, M.D. "ADADELTA: An Adaptive Learning Rate Method", arXiv: 1212.5701, 2012.

ZHANG, H., DAUPHIN, Y.N., MA, T. "Fixup Initialization: Residual Learning Without Normalization", International Conference on Learning Representations (ICLR'2019), pp. 1-16, 2019.