Deep Learning

Sem 02 - Aula 04

Renato Assunção - DCC - UFMG

Backpropagation com momentum

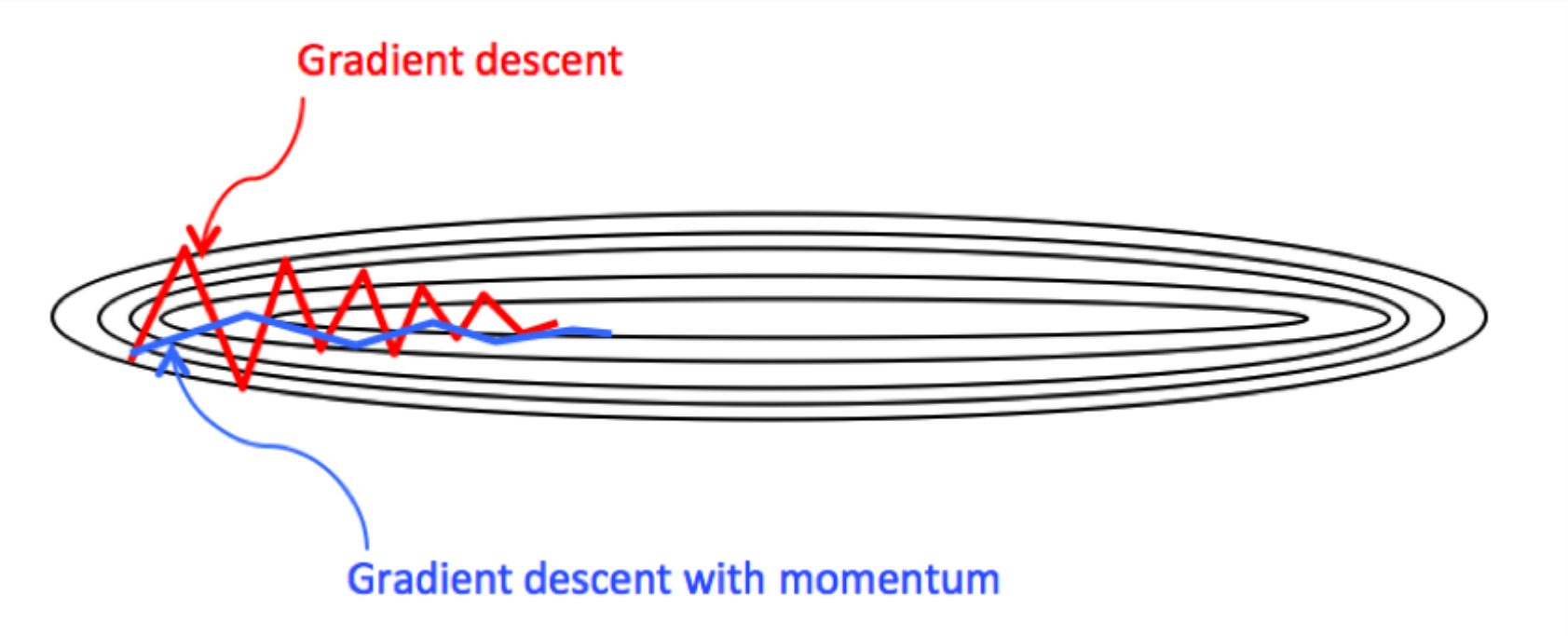
- Gradiente descendente:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\nabla \mathcal{L}(\mathbf{w}^{(t)})}_{\text{vetor gradiente}}$$

- Momentum: usando $\nabla \mathcal{L}(\mathbf{w}^{(0)}) = \mathbf{V}^{(0)}$

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

Resultado



Nesterov (proposto em Sutskever et al, 2013)

- Uma modificação adicional que melhora o desempenho: Nesterov
- Momentum:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta)\nabla\mathcal{L}(\mathbf{w}^{(t)}) + \beta\mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

- Não usa o gradiente em $\mathbf{w}^{(t)}$
- Vamos tentar obter uma posição projetada no próximo passo, ANTES de dar este passo.
- Temos uma "boa estimativa" da série do gradiente neste momento: $\mathbf{V}^{(t-1)}$
- Projetamos então para o próximo passo: $\mathbf{w}^{(t)} - \alpha\mathbf{V}^{(t-1)}$
- É neste ponto que calculamos o gradiente

Nesterov

- Momentum:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

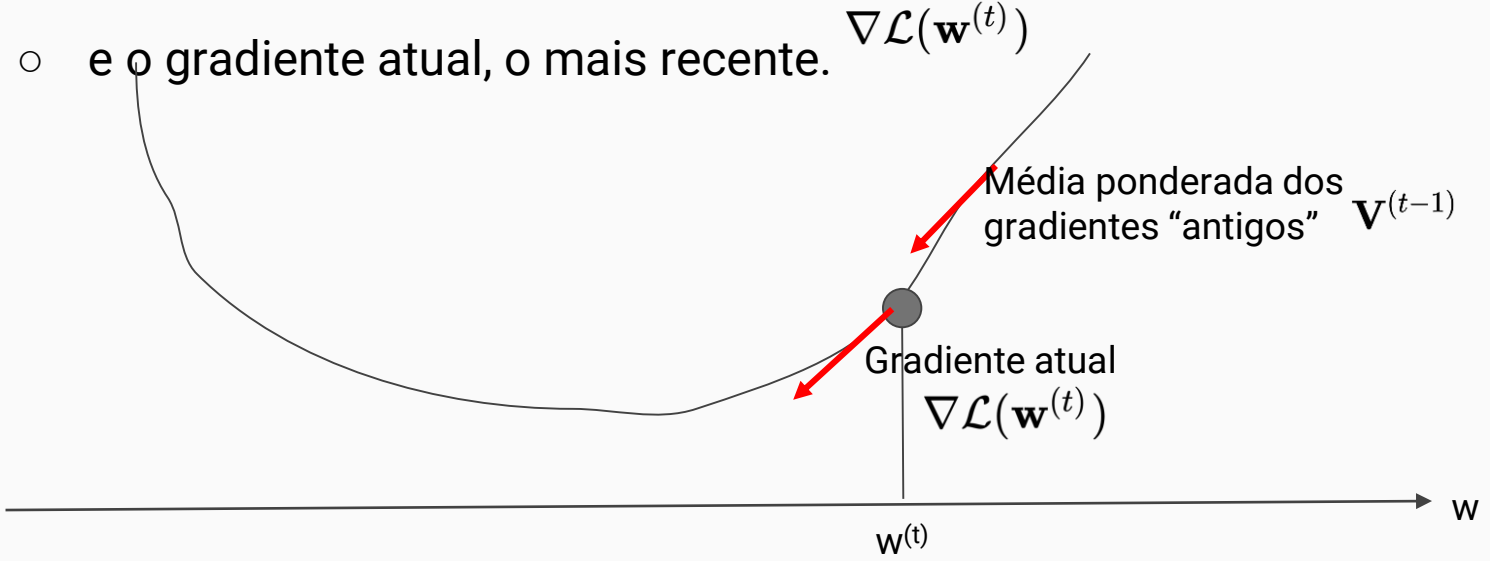
- Nesterov

Novidade

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)} - \alpha \mathbf{V}^{(t-1)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

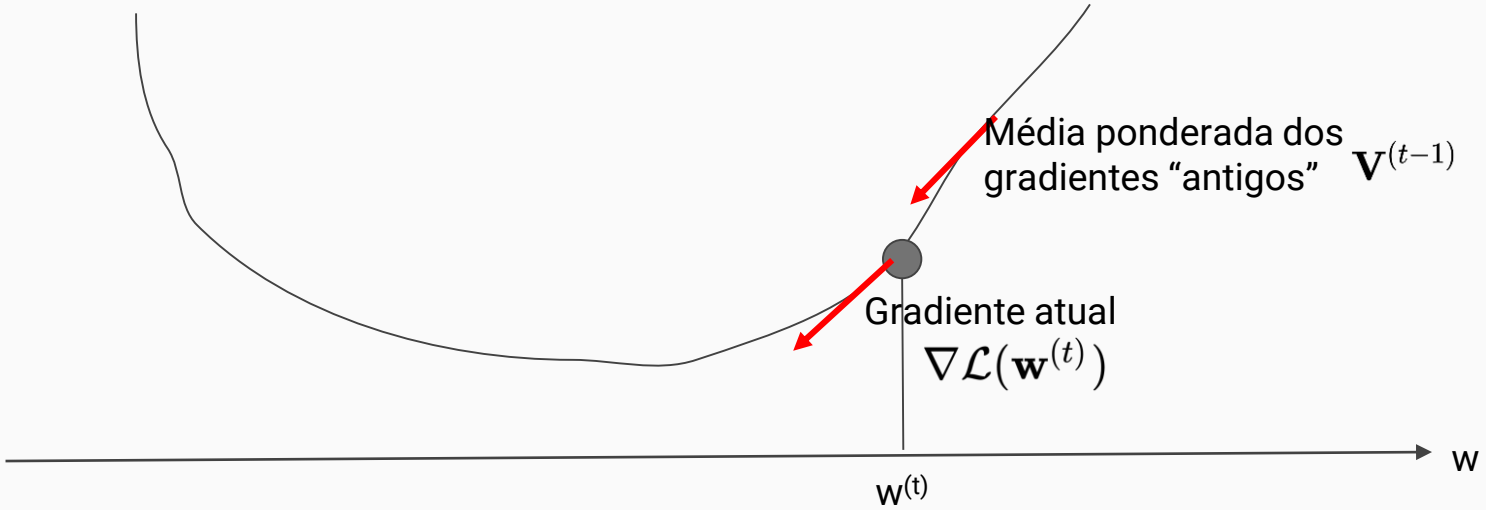
Qual a intuição de Nesterov? Runge-Kutta para ODE?

- Para calcular o tamanho do próximo passo, Momentum calcula uma média ponderada dos dois gradientes:
  - o “antigo”, $\mathbf{V}^{(t-1)}$
  - e o gradiente atual, o mais recente. $\nabla \mathcal{L}(\mathbf{w}^{(t)})$



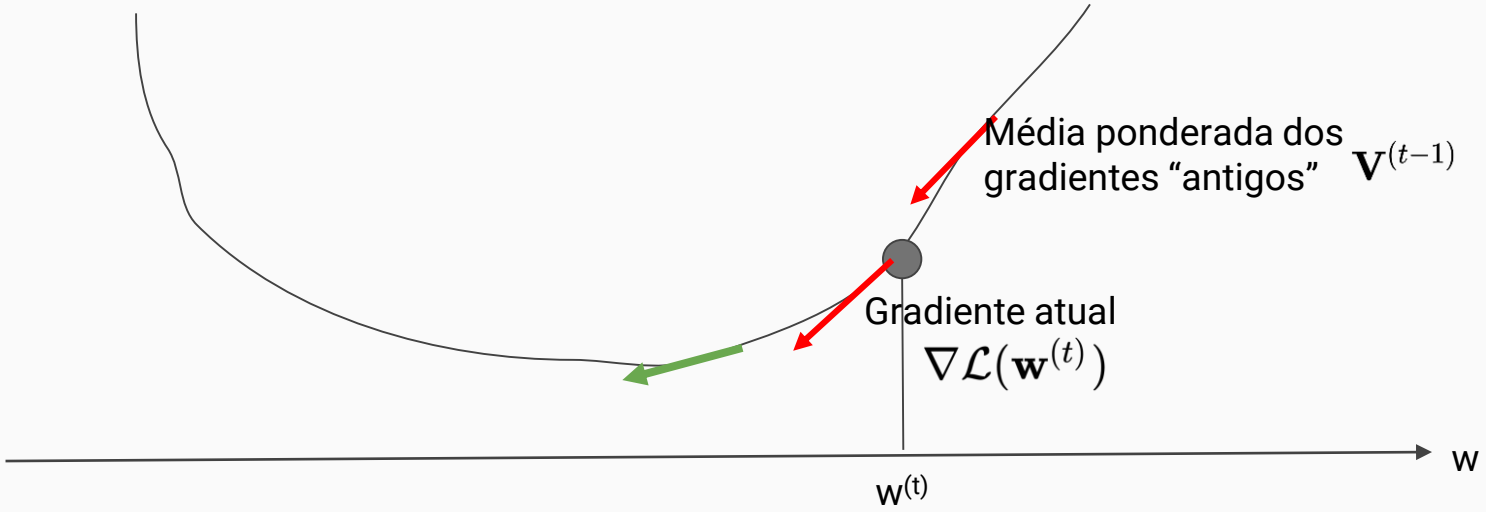
Qual a intuição de Nesterov? Runge-Kutta para ODE?

- Se ele pudesse ver um pouco mais à frente...
- Usaria um valor MAIS ATENUADO para o gradiente



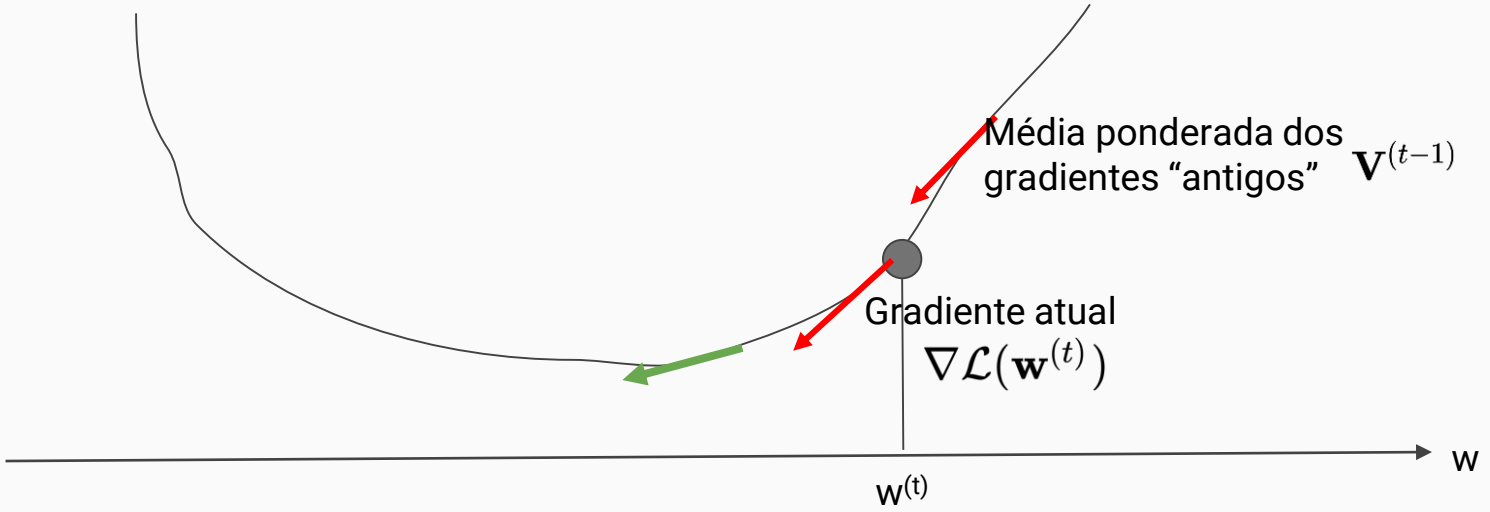
Qual a intuição de Nesterov? Runge-Kutta para ODE?

- Seria melhor usar uma média entre o gradiente antigo V_{t-1} e o gradiente FUTURO (seta em VERDE)



Qual a intuição de Nesterov? Runge-Kutta para ODE?

- Projetamos grosseiramente a posição do próximo w com a info que temos neste momento: $\mathbf{w}^{(t+1)} \approx \mathbf{w}^{(t)} - \alpha \mathbf{V}^{(t-1)}$
- Calcule o gradiente nesta posição futura: $\nabla \mathcal{L}(\mathbf{w}^{(t)} - \alpha \mathbf{V}^{(t-1)})$



- **Tire a média ponderada entre este novo gradiente e o gradiente antigo**

Nesterov

- Momentum:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

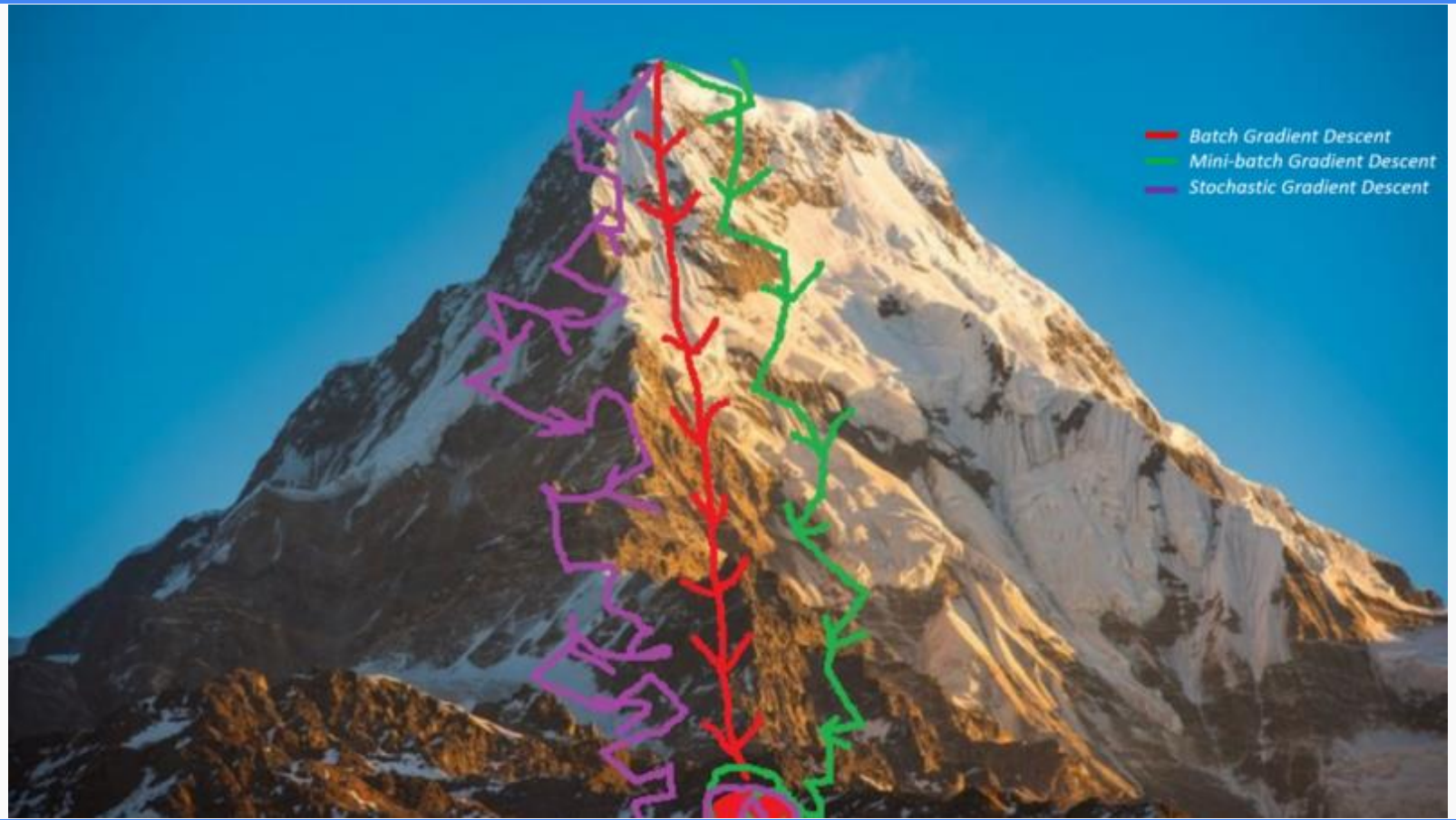
- Nesterov

Nesterov: futura posicao aprox

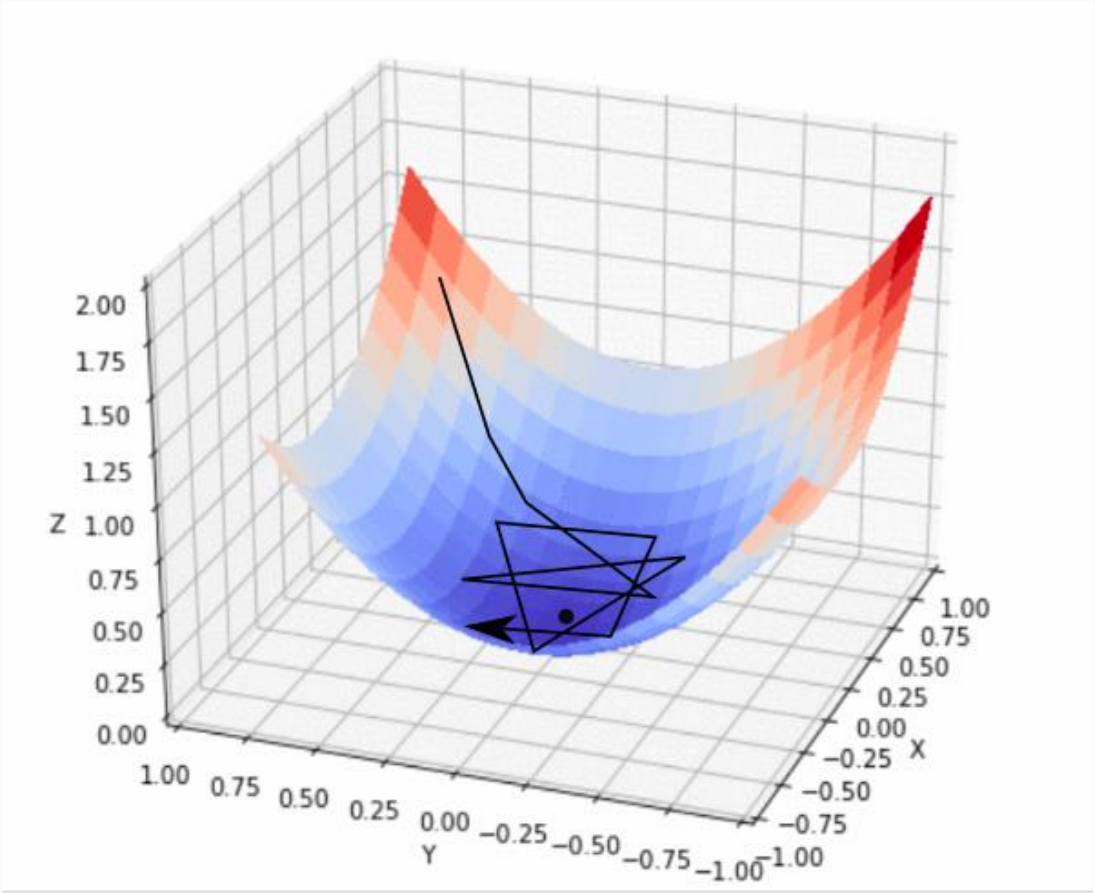
$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)} - \alpha \mathbf{V}^{(t-1)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

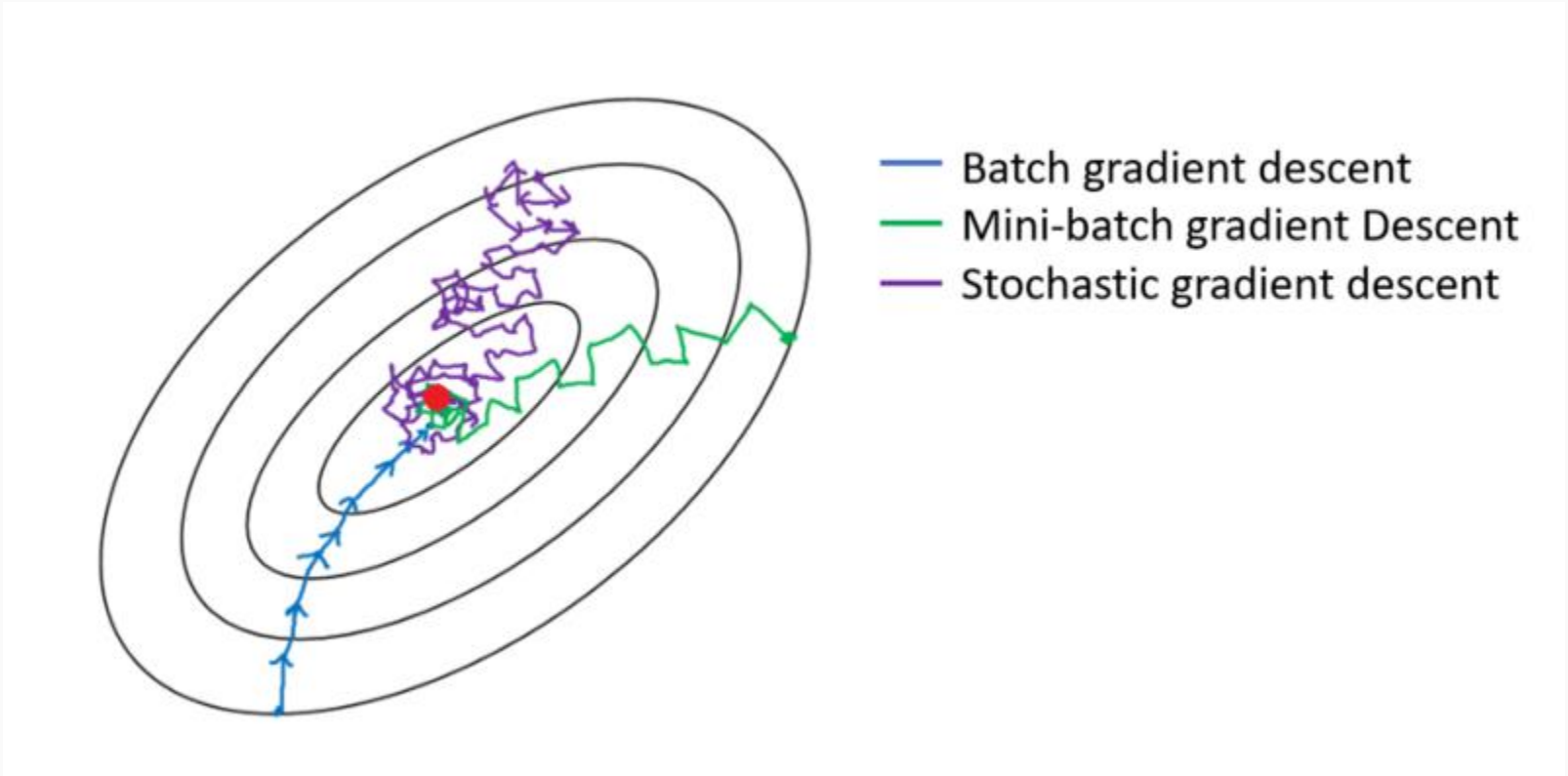
Stochastic gradient descent

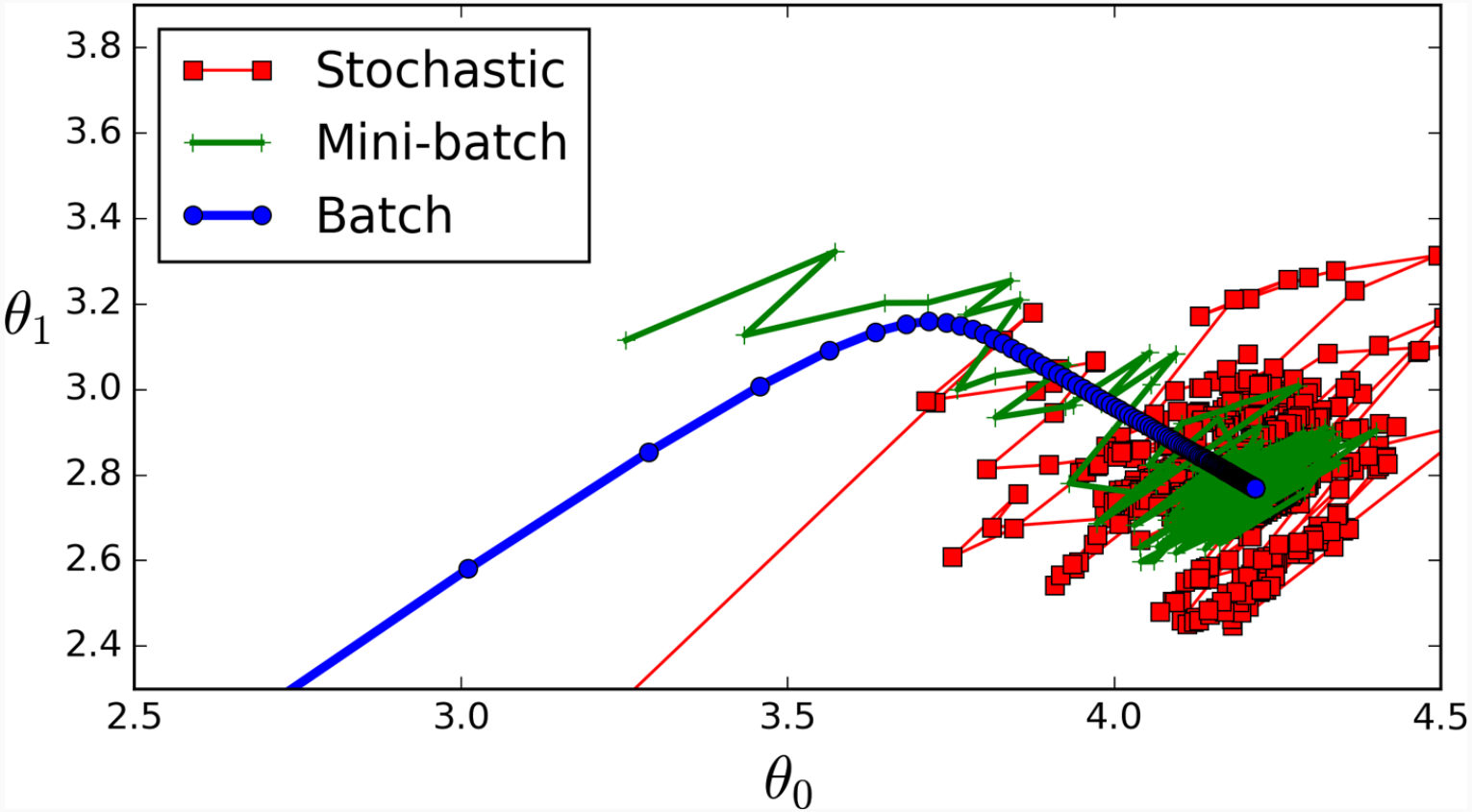
Stochastic Gradient Descent



Stochastic Gradient Descent




Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent


Stochastic
Mini-batch
Batch
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
θ_1
2.5
3.0
3.5
4.0
4.5
θ_0

Mini-Batch size

- Se conjunto de dados é pequeno (< 3000), use todos os dados.
- Se conjunto de dados não é pequeno, use batch size de acordo com o problema:
  - > 64 e < 1024
  - Algum valor intermediário entre estes é razoável: 64, 128, 256, 512, 1024
- Importante: escolha os dados do mini-batch de forma aleatória:
  - Não siga uma ordem temporal geográfica ou seguindo alguma característica que possa estar associada com a resposta

Terminologia

- Digamos que temos 2000 exemplos de treinamento
- Dividimos os 2000 exemplos em 4 lotes (batches) de tamanho 500.
- Precisamos de 4 iterações para completar 1 epoch

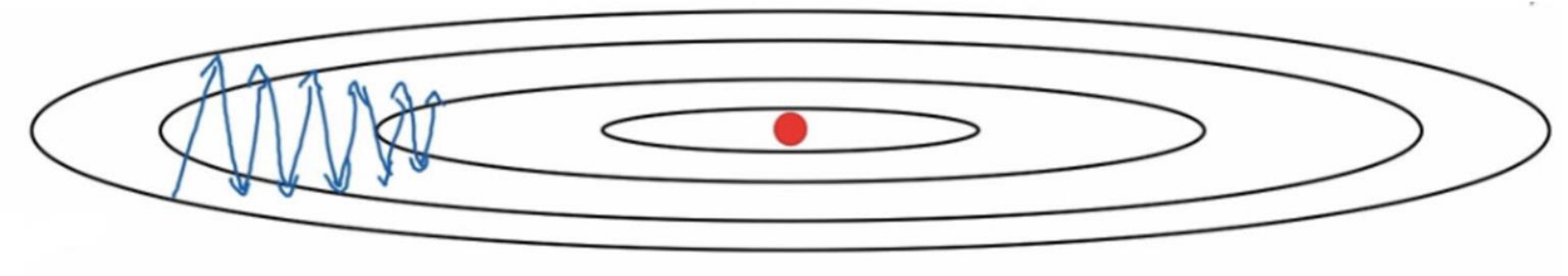
- UM EPOCH = o conjunto de dados é passado na rede neural UMA VEZ.
- Batch size = Número de exemplos em um batch
- Iterações para um epoch = número de batches necessários para completar um epoch.

Adaptive learning rate

Adaptive Learning Rate

- O backpropagation mais comum tem sido backprop + momentum
- Existem duas modificações adicionais de backprop que também aparecem com frequência:
  - RMSProp
  - ADAM
- Outros métodos: AdaGrad
- Eles fazem com a learning rate alpha seja adaptativa.
- Usada quando trabalhamos com mini-batch sizes: adapativa a medida que os batches vão sendo usados.
- Vamos ver cada um deles a seguir.

Gradiente descendente com alpha adaptativo



- Outra tentativa de evitar o zig-zag de forma adaptativa
- Apropriado quando este zig-zag tem diferentes tamanhos ao longo das diferentes dimensões do vetor de parâmetros
    - Eixo horizontal: parâmetro de bias b
    - Eixo vertical: parâmetro de peso w
- Oscilação excessiva ao longo da direção vertical (w)
- Queremos amenizar esta oscilação vertical e aumentar o tamanho dos passos ao longo de b

AdaGrad

- Forme a soma acumulada dos gradientes ao quadrado

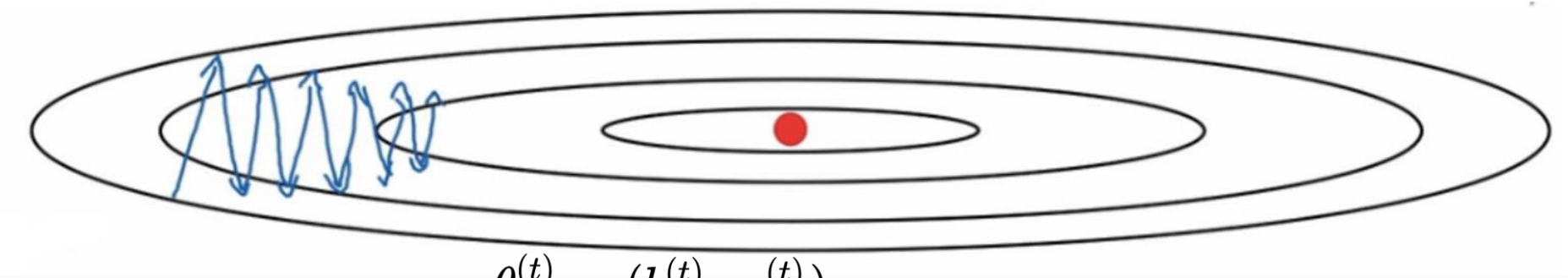
$$G = \sum_t \nabla \mathcal{L}(\mathbf{w}^{(t)}) \odot \nabla \mathcal{L}(\mathbf{w}^{(t)}) = \sum_t \left[\left(\frac{\partial \mathcal{L}}{\partial w_1} \right)^2, \ldots, \left(\frac{\partial \mathcal{L}}{\partial w_m} \right)^2 \right]^T$$

- Atualize cada coeficiente com uma learning rate que depende de G

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\alpha}{\sqrt{G_i + \varepsilon}} \frac{\partial \mathcal{L}}{\partial w_i}$$

- G cresce muito rápido. Hinton (2012) propôs RMSProp → a seguir

RMSProp:



$$\theta^{(t)} = (b^{(t)}, w^{(t)})$$

- Vetor de parâmetros
- Vetor gradiente $\nabla \mathcal{L}(\theta^{(t)}) = \left[\frac{\partial \mathcal{L}}{\partial b}, \frac{\partial \mathcal{L}}{\partial w}\right]$
- Variabilidade em cada coordenada

$$S_b^{(t+1)} = \gamma S_b^{(t)} + (1-\gamma)\left[\frac{\partial \mathcal{L}}{\partial b}\right]^2 \quad S_w^{(t+1)} = \gamma S_w^{(t)} + (1-\gamma)\left[\frac{\partial \mathcal{L}}{\partial w}\right]^2$$

- Atualização

$$w^{(t+1)} = w^{(t)} - \alpha \frac{1}{\sqrt{S_w^{(t+1)}} + \varepsilon} \frac{\partial \mathcal{L}}{\partial w}$$

$$b^{(t+1)} = b^{(t)} - \alpha \frac{1}{\sqrt{S_b^{(t+1)}} + \varepsilon} \frac{\partial \mathcal{L}}{\partial b}$$

$$\varepsilon = 10^{-6}$$

ADAM

- ADAM = Adaptive Moment Estimation
- Combina momentum e RMSProp
- Forma duas séries exponencialmente suavizadas:
    - do gradiente, como momentum

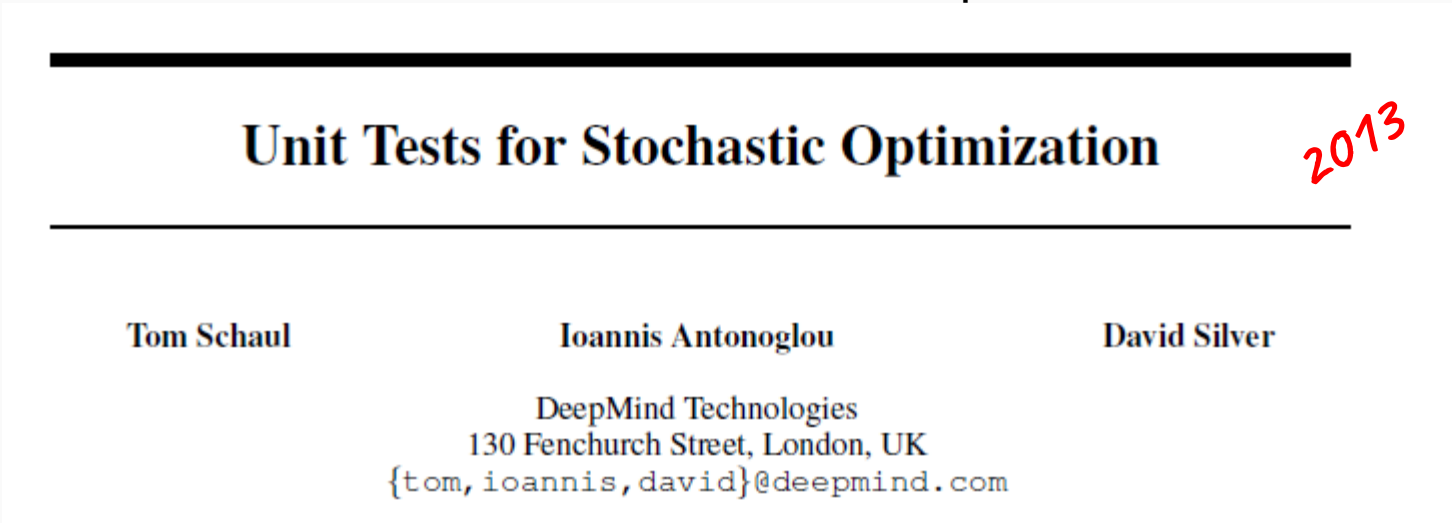
$$\mathbf{V}^{(t)} = (1-\beta)\nabla\mathcal{L}(\mathbf{w}^{(t)}) + \beta\mathbf{V}^{(t-1)}$$

    - do gradiente^2, como RMSProp

$$S_w^{(t+1)} = \gamma S_w^{(t)} + (1-\gamma)\left[\frac{\partial\mathcal{L}}{\partial w}\right]^2$$

- Atualize usando os dois valores:
- $$w^{(t+1)} = w^{(t)} - \frac{\alpha}{\sqrt{S_w^{(t+1)}}+\varepsilon}\mathbf{V}^{(t)}$$

- Algum estudo sobre o desempenho desses métodos: inconclusivo
- Nenhum método domina em todas as situações
- Mais usados: SGD + momentum ou RMSProp + momentum = ADAM



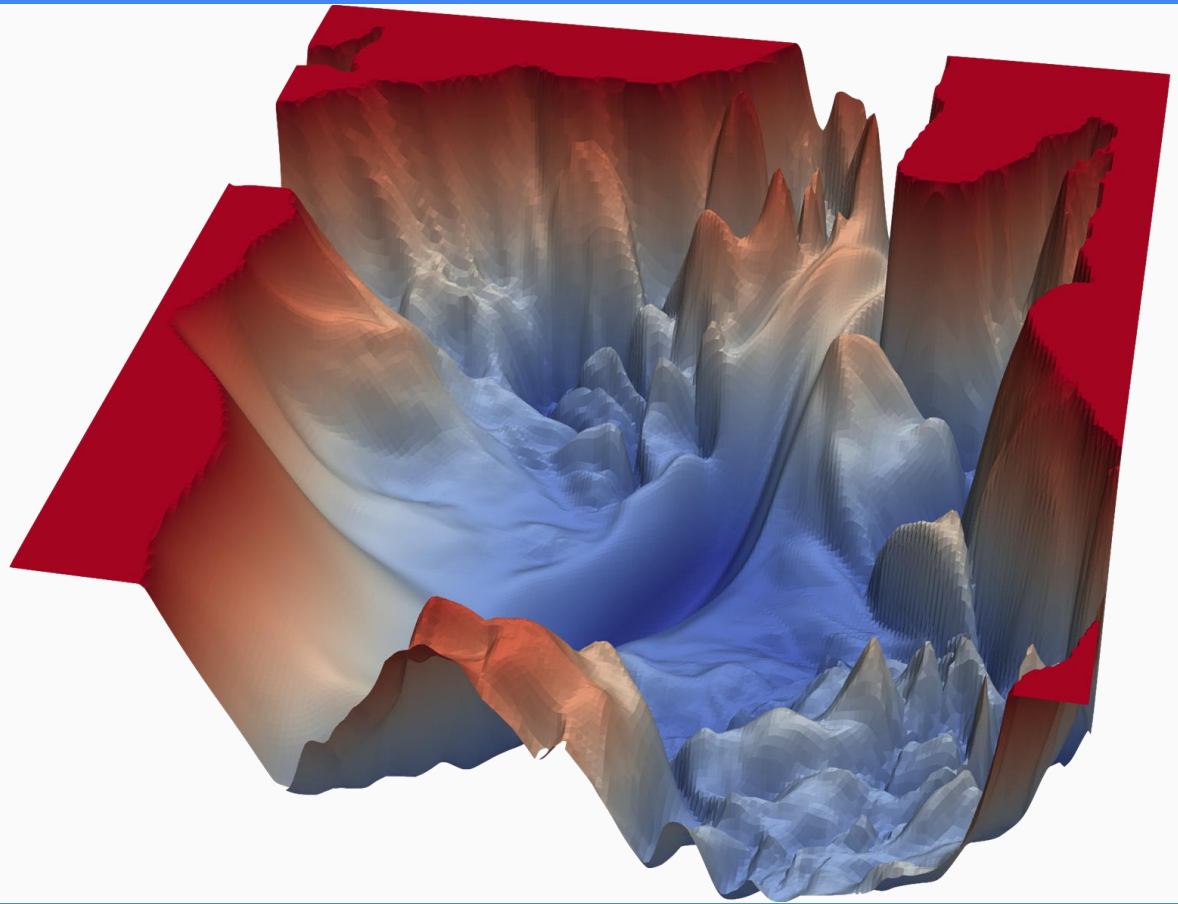
Unit Tests for Stochastic Optimization

2013

Tom Schaul **Ioannis Antonoglou** **David Silver**

DeepMind Technologies
130 Fenchurch Street, London, UK
{tom,ioannis,david}@deepmind.com

Esteja ciente de que a função de custo pode ser complexa

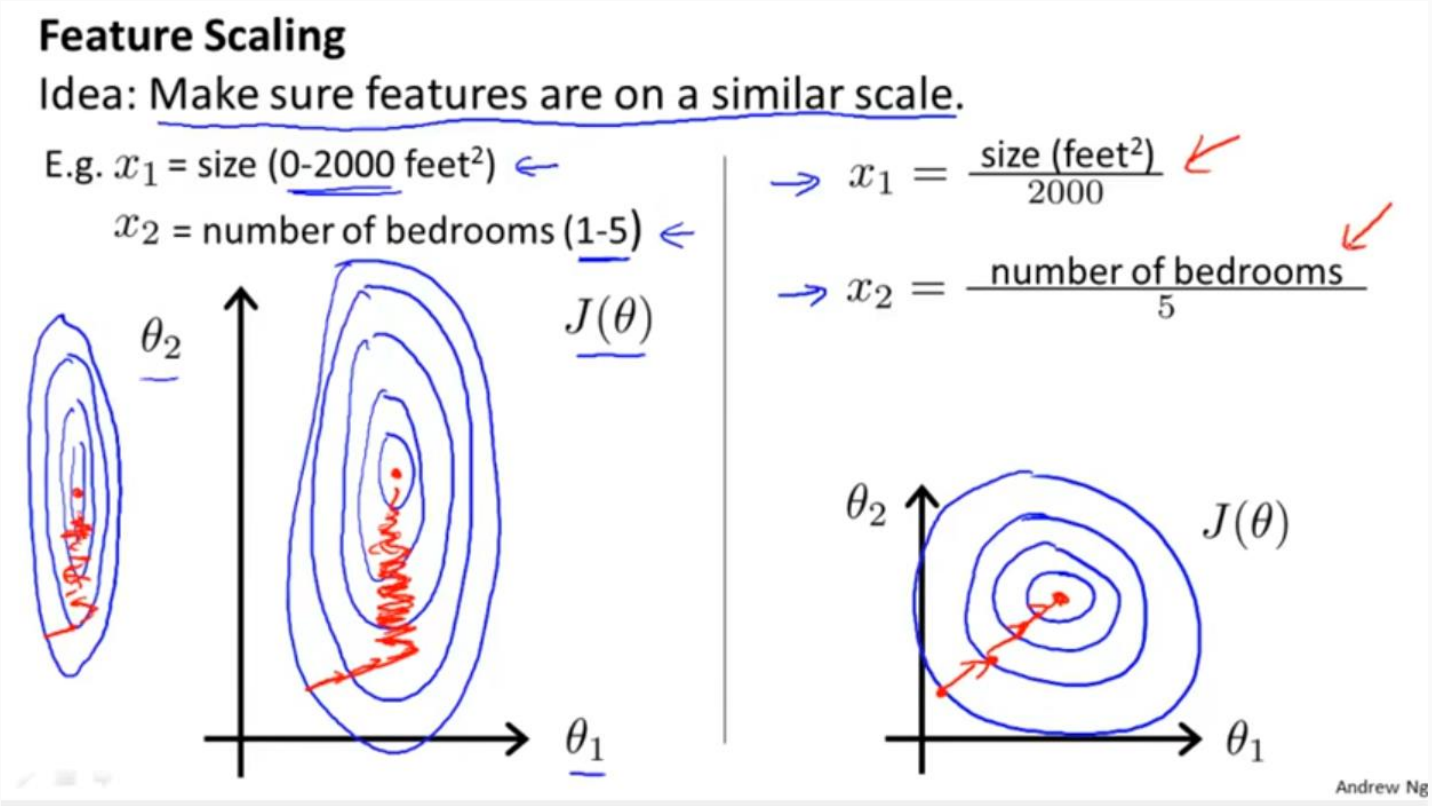


Normalização de features

Normalizando features

- Features aparecem com ordens de grandeza muito diferentes.
- Por exemplo, idade varia de 0 a 100 anos
- A renda mensal R$ 800 a R$ 100.000 (e mais).
- Usar os dados crus como entrada, não normalizados, pode frear o gradiente descendente:
  - dados crus distorcem a função de custo tornando o ponto mínimo difícil de alcançar.
- Um truque importante em ML: garantir que todos as features estejam em uma escala similar.
- Esta é uma etapa de pré-processamento dos dados.

Necessidade de normalizar as features



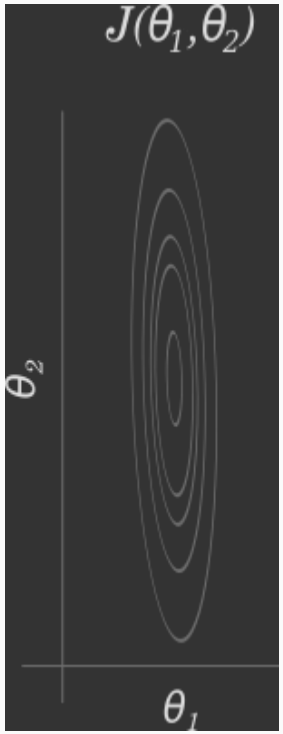
Modelo de preço (em milhares)

Preço = -1000 +
+ 10*Area (m^2)
+ 1000*Quartos

Coef de area não pode mudar de 10 para 110

O de quarto pode mudar para 1100 facilmente

- Suponha que você tenha duas features em indivíduos:
  - x1 como a renda mensal (800 - 100.000);
  - x2 como a idade (0-100).
- Gráfico ao lado mostra a função de custo típica que vai aparecer.
  - uma pequena mudança no coeficiente θ_1 da variável x1 faz o preditor (escore) linear variar muito: $\theta_0 + \theta_1 x_1 + \theta_2 x_2$
  - O escore mudando muito, a função de custo $J(\theta_1, \theta_2)$ também muda muito.
- Já uma pequena mudança no coeficiente θ_2 não afeta muito o escore e nem a função de custo $J(\theta_1, \theta_2)$
- A função de custo é um "monte" muito fino e esticada.
- Gradiente descendente vai oscilar muito para frente e para trás, demorando muito para encontrar o caminho até o ponto mínimo.



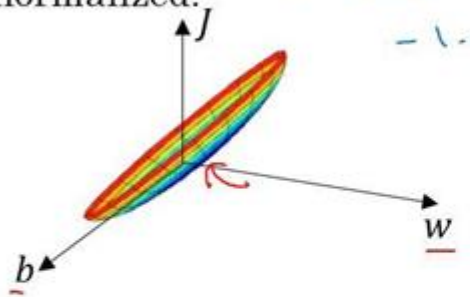
Why normalize inputs?

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

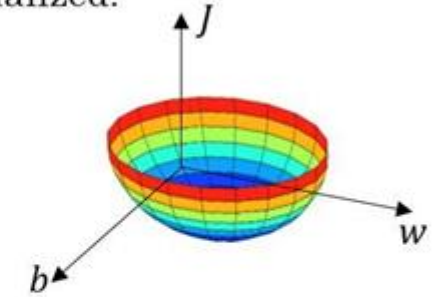
w_1 x_1: 1 ... 1000 ←

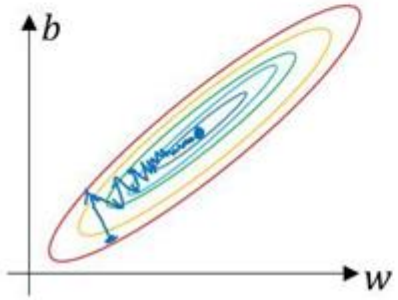
w_2 x_2: 0 ... 1 ←

Unnormalized:



Normalized:

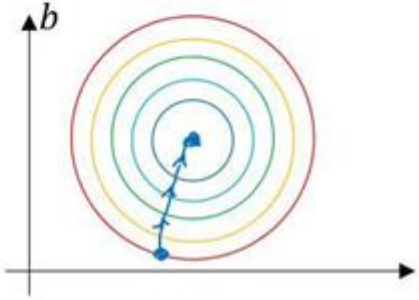




x_1: 0 ... 1

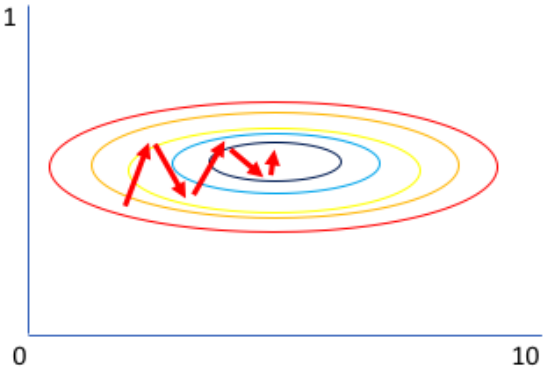
x_2: −1 .. 1

x_3: 1 .. 2

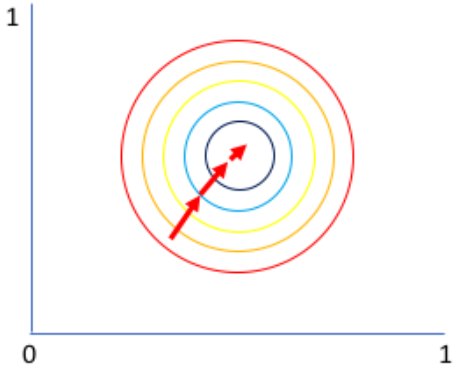


Andrew Ng

Why normalize?



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

- Normalizando as features, terminamos com $J(\theta_1, \theta_2)$ numa forma de "monte" mais esférico
- Gradiente descendente vai ser mais eficiente.
- Como normalizar?
  - min-max
    - por todas as features no intervalo [−1,1].
      - x* ← (x-min(x)) / (max(x) - min(x))
      - Renda* ← (Renda - 800) / (100000 - 800)
  - padronização estatística (ou z-escore):
    - features têm média zero e desvio-padrão 1
      - x* ← (x - mean(x)) / sd(x)
      - Renda* ← (Renda - 2500) / 22400
- Regra geral: quando em dúvida, padronize os dados. Mal não vai fazer e pode ajudar bastante.

Inicialização

On the importance of initialization and momentum in deep learning

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

Como inicializar

- Uma regra fundamental: break the symmetry, otherwise backprop will not converge.
- Recomendação:
  - inicialize todos os termos de bias iguais a ZERO
  - inicialize os pesos w’s de forma aleatória
  - Os w’s devem ser distintos (break the symmetry)
  - Os w’s devem ser pequenos (regularização)

Como inicializar

- Camada com M inputs e N unidades (ou neurônios)
- Tome os w's como variáveis aleatórias independentes
- Duas recomendações mais populares:
  - $W \sim U(\frac{-1}{\sqrt{M}}, \frac{1}{\sqrt{M}})$
  - Se $U = W_1 x_1 + \ldots W_m x_m$ com $X_i \sim (0,1)$ então
  - $\mathbb{E}(U) = 0$ e $\mathbb{V}(U) = m \times 1 \times \frac{1}{12}\left[\frac{2}{\sqrt{m}}\right]^2 = \frac{1}{3}$ = constante por camada
  - Outra opção: $W \sim U(-\sqrt{\frac{6}{M+N}}, \sqrt{\frac{6}{M+N}})$

Cross entropy as cost function

Entropia

- Veremos que cross-entropy é a log-verossimilhança...

- Entropia é uma medida da incerteza de uma variável aleatória.
- Se temos uma variável aleatória X com função de massa de probabilidade p (x) = Pr [X = x], definimos a Entropia H (X) da variável aleatória X como

$$H(X) = -\sum_{x} p(x) \log p(x) \qquad (1)$$

$$
\begin{aligned}
H(X) &= -\sum_x p(x) \log p(x) \\
&= \sum_x -p(x) \log p(x) \\
&= \sum_x p(x) \log \frac{1}{p(x)} \\
&= \mathbb{E}_{X \sim p(x)} \left[\log \frac{1}{p(x)} \right]
\end{aligned}
\tag{3}
$$

Podemos escrever H(p)

Entropia relativa

- verdadeira distribuição dos dados = p(x) → desconhecida
- Modelamos usando uma aproximação q(x)
- Qual é a ineficiência de assumir que a distribuição verdadeira é q(x), e não p(x)?
- Medimos com entropia-relativa ou distância de Kullback-Leibler.
- Entropia relativa é uma medida da distância entre duas distribuições.
- Denotada D(p || q) ou KL(p || q), é definida como

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{4}$$

$$= \mathbb{E}_{X \sim p(x)} \left[\log \frac{p(x)}{q(x)} \right]$$

- Se expandirmos log (p (x) / q (x)), temos

$$
\begin{aligned}
D(p||q) &= \mathbb{E}_{X\sim p(x)}\left[\log \frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{X\sim p(x)}\left[\log \frac{1}{q(x)} - \log \frac{1}{p(x)}\right] \\
&= \mathbb{E}_{X\sim p(x)}\left[\log \frac{1}{q(x)}\right] - \mathbb{E}_{X\sim p(x)}\left[\log \frac{1}{p(x)}\right]
\end{aligned}
\tag{5}
$$

- o 2o termo é a entropia da distribuição p(x)
- o 1o termo é chamado de entropia cruzada (cross-entropy)

- A entropia cruzada está intimamente relacionada com a entropia relativa.
- Podemos definir a entropia cruzada, denotada por H (p, q), como

$$
\begin{aligned}
H(p,q) &= H(p) + D(p||q) \\
&= \mathbb{E}_{X\sim p(x)}\left[\log \frac{1}{p(x)}\right] + \mathbb{E}_{X\sim p(x)}\left[\log \frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{X\sim p(x)}\left[\log \frac{1}{q(x)}\right]
\end{aligned}
\tag{6}
$$

Máxima verossimilhança

- Dados $X_{example}$ = {x_1, x_2,..., x_m} i.i.d distribuição desconhecida $p_{dados}(x)$.
- Usamos o modelo paramétrico $p_{model}(x; \theta)$ com θ como parâmetro.
- Para obter o melhor modelo, precisamos encontrar θ que produza uma distribuição $p_{model}(x, \theta)$ o mais similar (verossímil) a $p_{dados}(x)$.
- MLE é este valor de θ:

$$\theta_{\mathrm{ML}} = \underset{\theta}{\operatorname{argmax}}\, p_{\mathrm{model}}\left(X_{\mathrm{example}}; \theta\right) \tag{7}$$

- Na verdade, usamos o log:

$$\theta_{\mathrm{ML}} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{\mathrm{model}}\left(x_i; \theta\right) \tag{9}$$

- Não altera se dividirmos por m. p_example(x) = distribuição empírica discreta

$$\begin{aligned}\theta_{\mathrm{ML}} &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \frac{1}{m} \log p_{\mathrm{model}}\left(x_i; \theta\right) \\ &= \underset{\theta}{\operatorname{argmax}}\ \mathbb{E}_{X \sim p_{\mathrm{example}}(x)} \left[\log p_{\mathrm{model}}\left(x; \theta\right)\right]\end{aligned} \tag{10}$$

- Se tomarmos p_example(x) como p(x) e p_model(x; θ) como q(x), temos

$$\begin{aligned}\theta_{\mathrm{ML}} &= \underset{\theta}{\operatorname{argmax}}\ \mathbb{E}_{X \sim p(x)} \left[\log q(x)\right] \\ &= \underset{\theta}{\operatorname{argmin}}\ -\mathbb{E}_{X \sim p(x)} \left[\log q(x)\right] \\ &= \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{X \sim p(x)} \left[\log \frac{1}{q(x)}\right]\end{aligned} \tag{11}$$

entropia cruzada

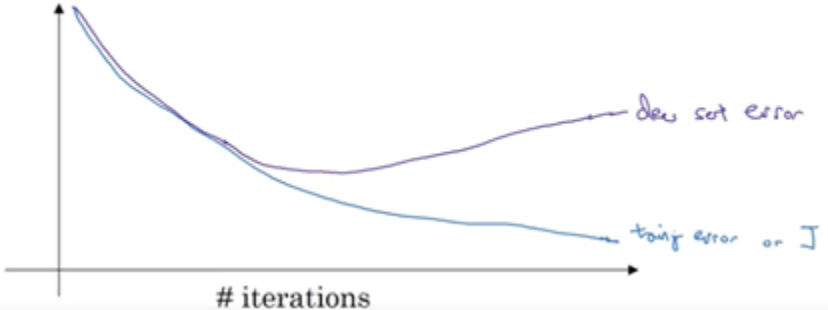
$$\begin{aligned}\theta_{\mathrm{ML}} &= \underset{\theta}{\operatorname{argmax}}\ \mathbb{E}_{X\sim p(x)}\left[\log q(x)\right] \\ &= \underset{\theta}{\operatorname{argmin}}\ -\mathbb{E}_{X\sim p(x)}\left[\log q(x)\right] \\ &= \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{X\sim p(x)}\left[\log \frac{1}{q(x)}\right]\end{aligned} \tag{11}$$

- Obter o MLE é o mesmo que minimizar a entropia cruzada entre nosso modelo paramétrico $p\_{model}(x; \theta)$ e a distribuição empírica $p\_{example}(x)$.

Outras maneiras de regularizar: early stopping

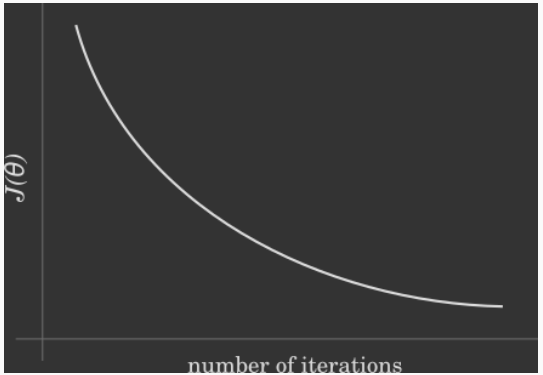
- Divida os dados de treinamento em um conjunto de treinamento e um conjunto de validação.
- Treinar apenas no conjunto de treinamento.
- Avaliar o erro no conjunto de validação de vez em quando (a cada 5 iterações, por exemplo)
- Pare o treinamento assim que o erro no conjunto de validação subir.
- Use os pesos que a rede tinha na etapa anterior como resultado da execução do treinamento.

Early stopping



Monitore a função de custo $J(\theta_0, \theta_1, \ldots, \theta_p)$

- Verifique se o gradiente descendente está funcionando corretamente.
- Queremos o valor dos THETAs que minimizam a função de custo
- Plotamos a função de custo J versus a iteração do algoritmo para ver se estamos diminuindo J a cada passo.
- O número de iterações no eixo horizontal, a função de custo J na vertical.
    - Em cada iteração, o gradiente descendente produz novos valores de θs
    - Com esses novos valores, avaliamos a função de custo J(θ).
    - Devemos ter uma curva decrescente se o algoritmo se comportar bem
    - Significa que ele está minimizando o valor dos θs corretamente.



Checando convergência com plot de J(θ)

- Plotar J(θ) também informa se o gradiente descendente convergiu ou não.
- Não existe um único número de iterações até a convergência. Isto varia em função de:
  - qualidade do valor inicial
  - tamanho do problema (número de parâmetros)
  - grau de concentração da superfície J(θ) em torno do ponto de mínimo
- Em geral, você pode assumir que o algoritmo encontrou um mínimo quando J(θ) diminui menos que algum valor pequeno ϵ em uma iteração.
  - Escolher um valor adequado ϵ não é uma tarefa fácil.
  - Algumas pessoas tomam ϵ = 10^{−3} e usam um teste de convergência automática:
    - convergiu se J(θ) diminuir menos que ϵ em uma iteração.
- Pode declarar convergência, pode exigir também uma diferença relativa pequena para valores sucessivos de J(θ)

Escolha bem a taxa de aprendizagem alpha

- O plot de J(θ) pode ficar estranho:
  - J(θ) crescendo
  - J(θ) diminuindo muuuuuuito lentamente
  - J(θ) oscilando: subindo e descendo, sem diminuir de forma muito consistente
- Uma solução é trocar a learning rate α.
- É provado matematicamente que, para α suficientemente pequeno, J(θ) diminui em cada iteração.
- Por outro lado, se α é muito pequeno, gradient descent pode demorar a convergir (pois theta quase não muda de iteração para iteração).
- Regra geral: tentar um intervalo de valores α.
- Comece com α = 0,001 e observe o gráfico J (θ). Diminui de forma adequada e rápida? Done.
- Caso contrário, mude para α = 0,01 (escala logarítmica) e repita até que o algoritmo funcione bem.

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj

- jj
- jj