ATUALIZAÇÃO DE PESOS

ADAM / DECAIMENTO DA TAXA DE APRENDIZADO

Zenilton K. G. Patrocínio Jr
zenilton@pucminas.br

ADAM – *Adaptive Moment Estimation*

[Kingma and Ba, 2014]

(incompleto, mas bem próximo)

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

ADAM – *Adaptive Moment Estimation*

[Kingma and Ba, 2014]

(incompleto, mas bem próximo)

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

momentum

RMSProp-*like*

ADAM – *Adaptive Moment Estimation*

[Kingma and Ba, 2014]

(incompleto, mas bem próximo)

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

momentum

RMSProp-*like*

Semelhante ao RMSProp com "*momentum*"

```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

ADAM – *Adaptive Moment Estimation*

[Kingma and Ba, 2014]

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
  dx = # ... evaluate gradient
  m = beta1*m + (1-beta1)*dx # update first moment
  v = beta2*v + (1-beta2)*(dx**2) # update second moment
  mb = m/(1-beta1**t) # correct bias
  vb = v/(1-beta2**t) # correct bias
  x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

ADAM – *Adaptive Moment Estimation*

[Kingma and Ba, 2014]

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
  dx = # ... evaluate gradient
  m = beta1*m + (1-beta1)*dx # update first moment
  v = beta2*v + (1-beta2)*(dx**2) # update second moment
  mb = m/(1-beta1**t) # correct bias
  vb = v/(1-beta2**t) # correct bias
  x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

momentum

ADAM – *Adaptive Moment Estimation*

[Kingma and Ba, 2014]

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
  dx = # ... evaluate gradient
  m = beta1*m + (1-beta1)*dx # update first moment
  v = beta2*v + (1-beta2)*(dx**2) # update second moment
  mb = m/(1-beta1**t) # correct bias
  vb = v/(1-beta2**t) # correct bias
  x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

momentum

RMSProp-*like* (1)

RMSProp-*like* (2)

ADAM – *Adaptive Moment Estimation*

[Kingma and Ba, 2014]

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
  dx = # ... evaluate gradient
  m = beta1*m + (1-beta1)*dx # update first moment
  v = beta2*v + (1-beta2)*(dx**2) # update second moment
  mb = m/(1-beta1**t) # correct bias
  vb = v/(1-beta2**t) # correct bias
  x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

momentum

RMSProp-*like* (1)

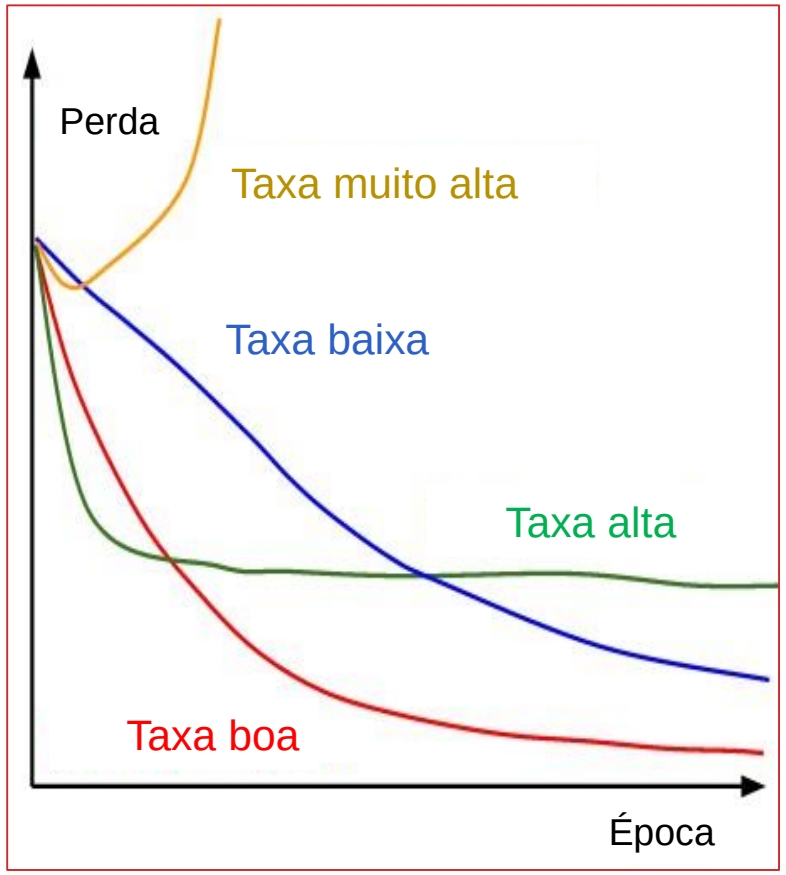
Coreção de viés
(só relevante nas primeiras iterações quando t é pequeno)

RMSProp-*like* (2)

A correção de viés compensa o fato de que *m* e *v* são inicializados em zero e precisam de algum tempo para “aquecer”, isto é, obter valores adequados

Atualização × Taxa de Aprendizado

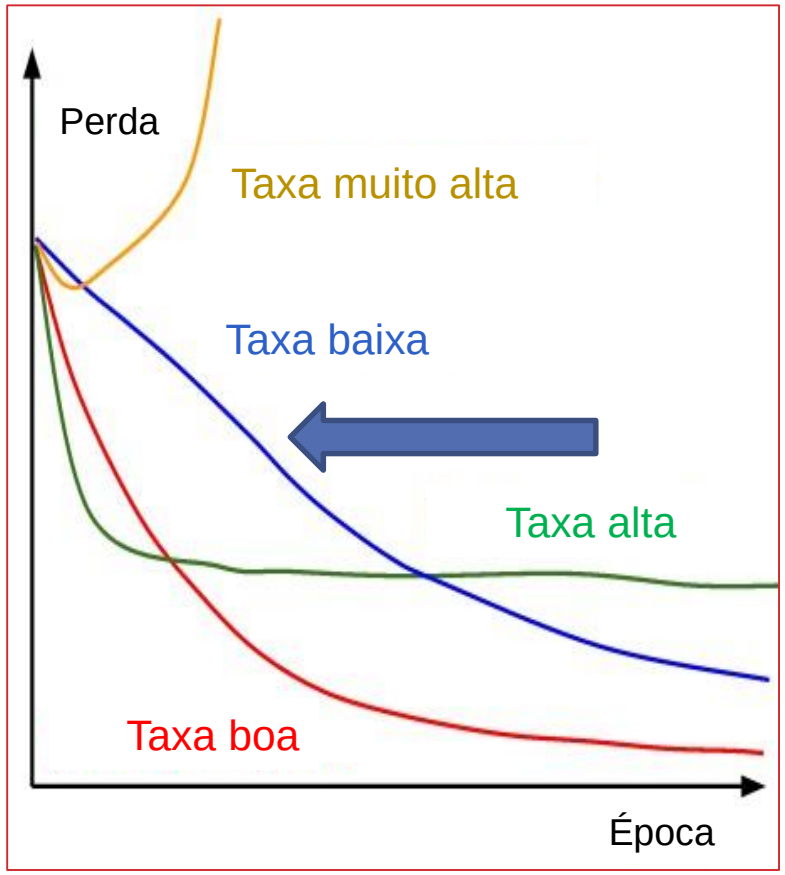
SGD, SGD + *Momentum*, SGD + NAG, ADAGRAD, RMSProp e ADAM possuem a **taxa de aprendizado** como um hiperparâmetro



P: Qual dessas é a melhor taxa de aprendizado a se utilizar?

Atualização × Taxa de Aprendizado

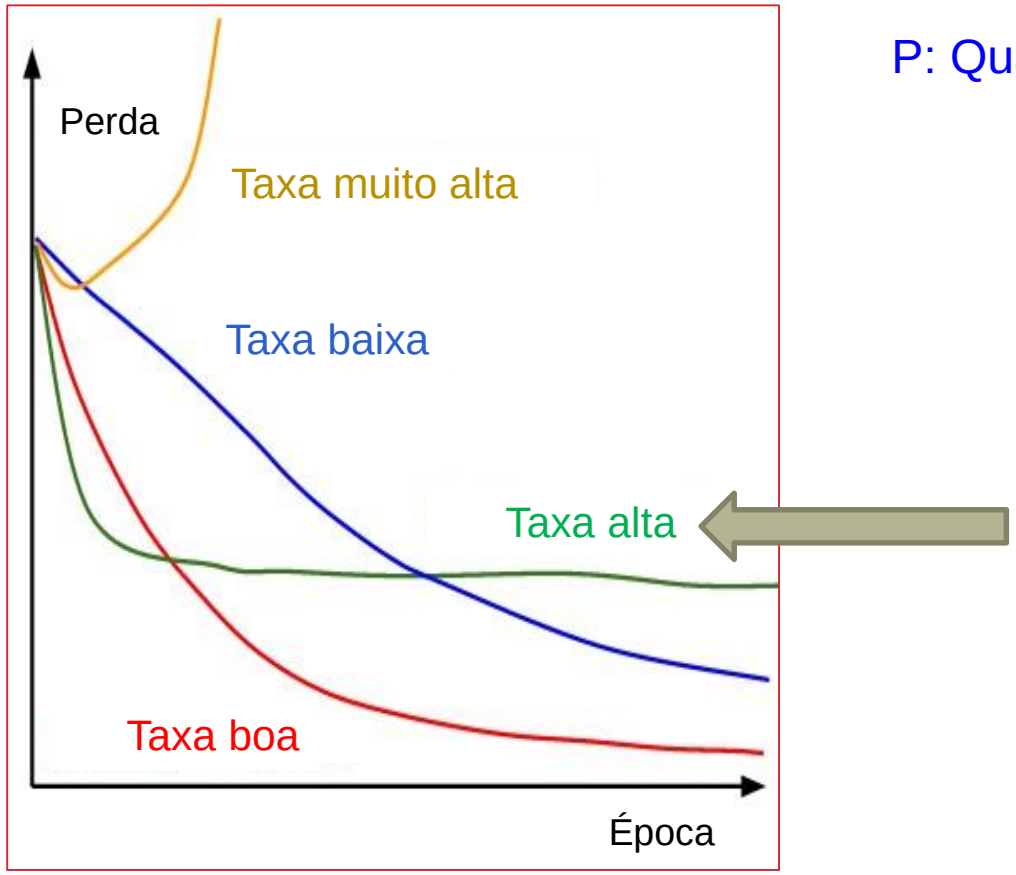
SGD, SGD + *Momentum*, SGD + NAG, ADAGRAD, RMSProp e ADAM possuem a **taxa de aprendizado** como um hiperparâmetro



P: Qual dessas é a melhor taxa de aprendizado a se utilizar?

Atualização × Taxa de Aprendizado

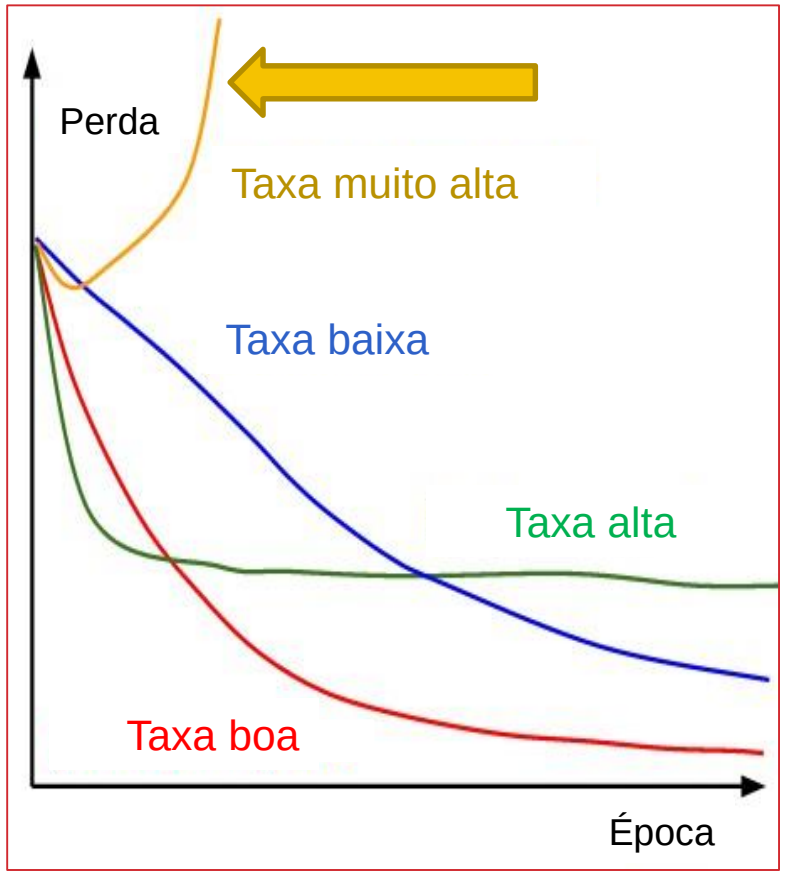
SGD, SGD + *Momentum*, SGD + NAG, ADAGRAD, RMSProp e ADAM possuem a **taxa de aprendizado** como um hiperparâmetro



P: Qual dessas é a melhor taxa de aprendizado a se utilizar?

Atualização × Taxa de Aprendizado

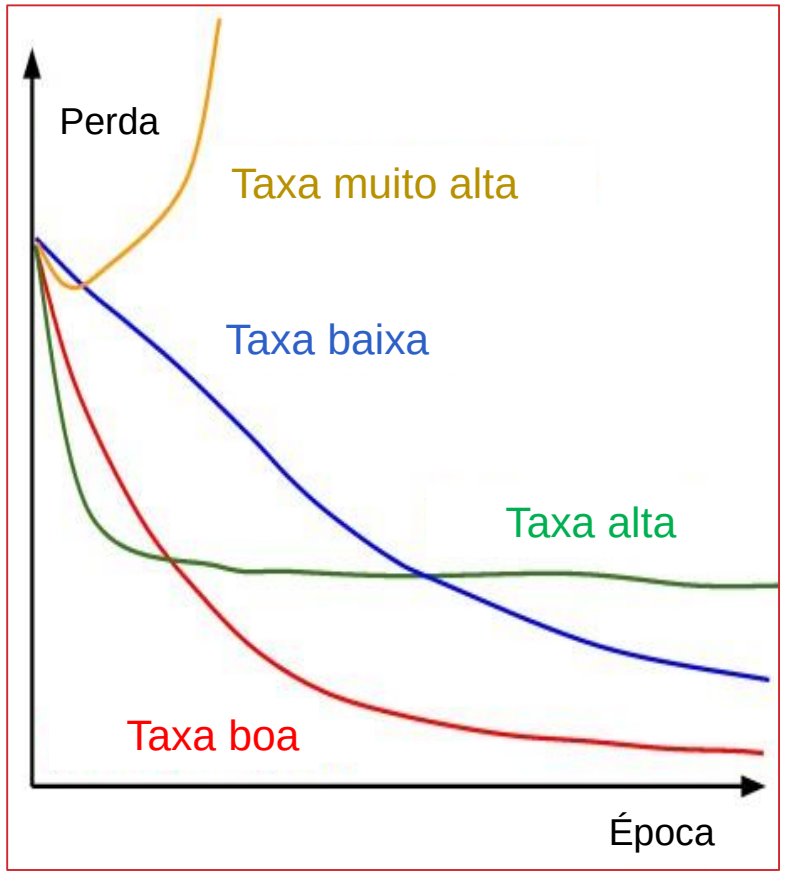
SGD, SGD + *Momentum*, SGD + NAG, ADAGRAD, RMSProp e ADAM possuem a **taxa de aprendizado** como um hiperparâmetro



P: Qual dessas é a melhor taxa de aprendizado a se utilizar?

Atualização × Taxa de Aprendizado

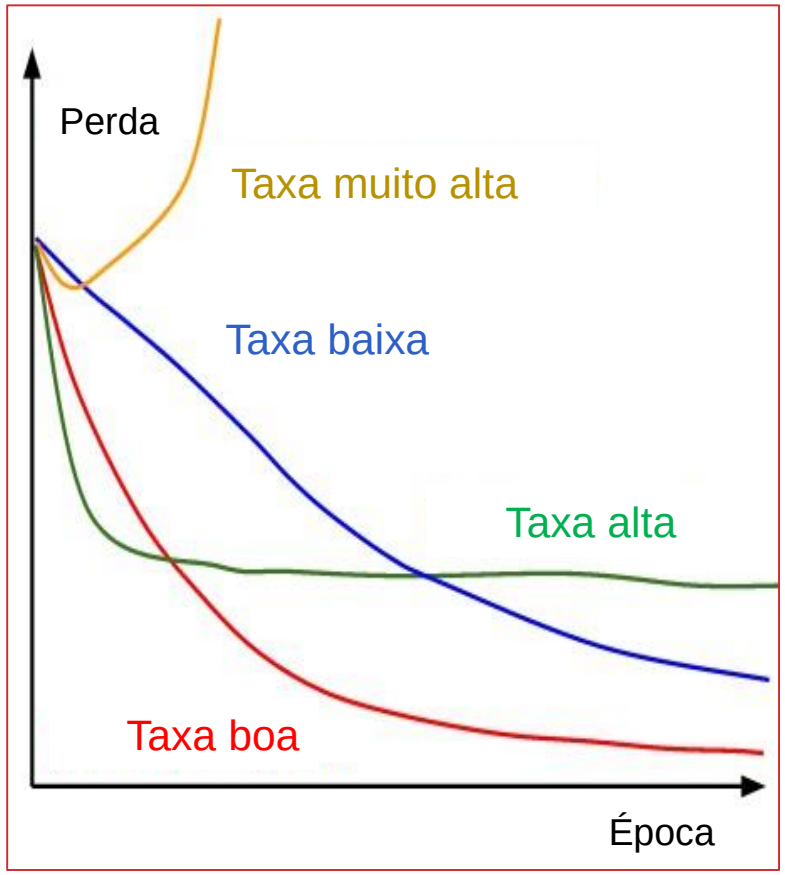
SGD, SGD + *Momentum*, SGD + NAG, ADAGRAD, RMSProp e ADAM possuem a **taxa de aprendizado** como um hiperparâmetro



P: Qual dessas é a melhor taxa de aprendizado a se utilizar?

Decaimento da Taxa de Aprendizado

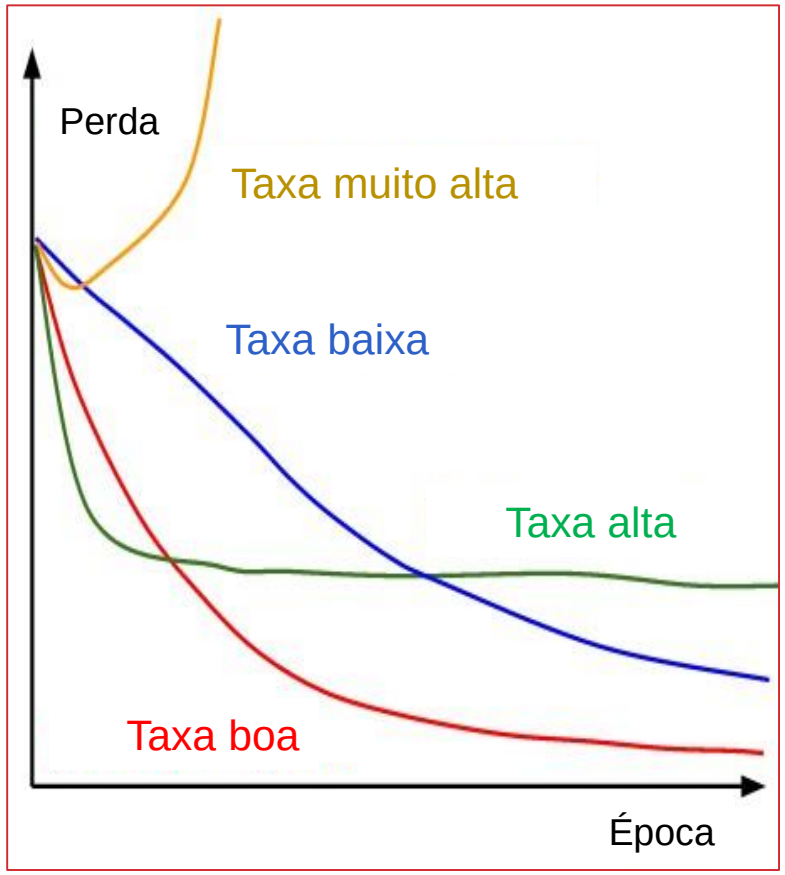
SGD, SGD + *Momentum*, SGD + NAG, ADAGRAD, RMSProp e ADAM possuem a **taxa de aprendizado** como um hiperparâmetro



⇒ **Decaimento da taxa de aprendizado ao longo do tempo!**

Decaimento da Taxa de Aprendizado

SGD, SGD + *Momentum*, SGD + NAG, ADAGRAD, RMSProp e ADAM possuem a **taxa de aprendizado** como um hiperparâmetro



⇒ **Decaimento da taxa de aprendizado ao longo do tempo!**

- **Constante:** p.ex., dividir por 2 a intervalos fixos
- **Exponencial:** $\alpha = \alpha_0 e^{-kt}$
- **1/t:** $\alpha = \alpha_0/(1+kt)$
- **$1/\sqrt{t}$:** $\alpha = \alpha_0/\sqrt{1+kt}$ (p.ex., ADAGRAD)

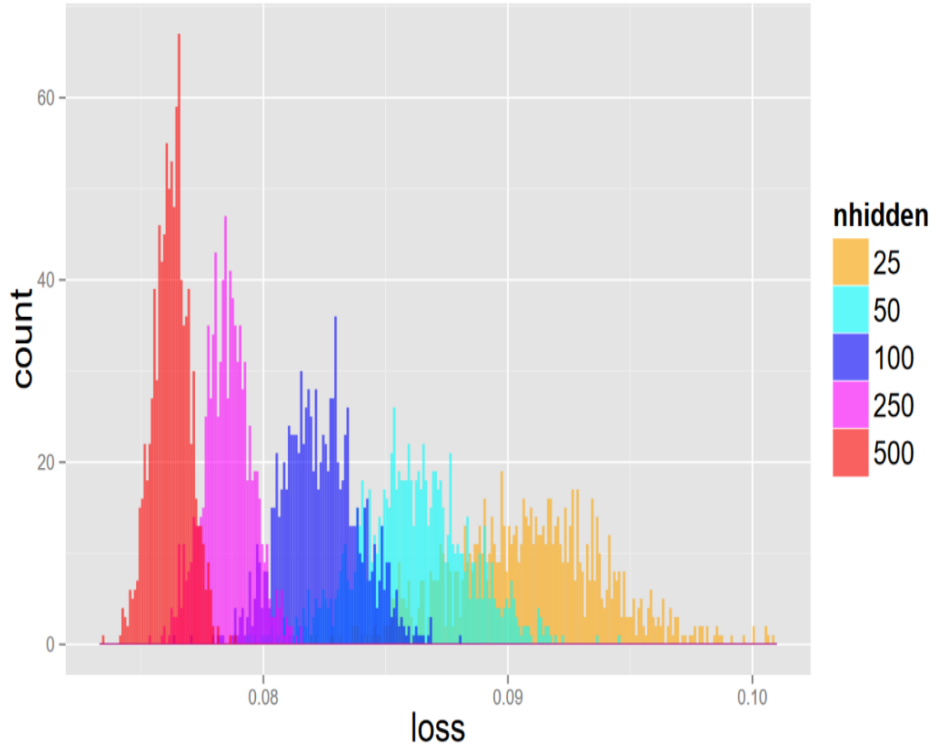
Múltiplos Mínimos Locais são um Problema?

Elas acontecem, mas a maioria dos mínimos locais tem perda semelhante ao mínimo global

Múltiplos Mínimos Locais são um Problema?

Elas acontecem, mas a maioria dos mínimos locais tem perda semelhante ao mínimo global

Para uma simples rede de imagens, na medida que a rede fica mais complexa (mais unidades ocultas), existem mais mínimos locais agrupados próximos da perda global mínima



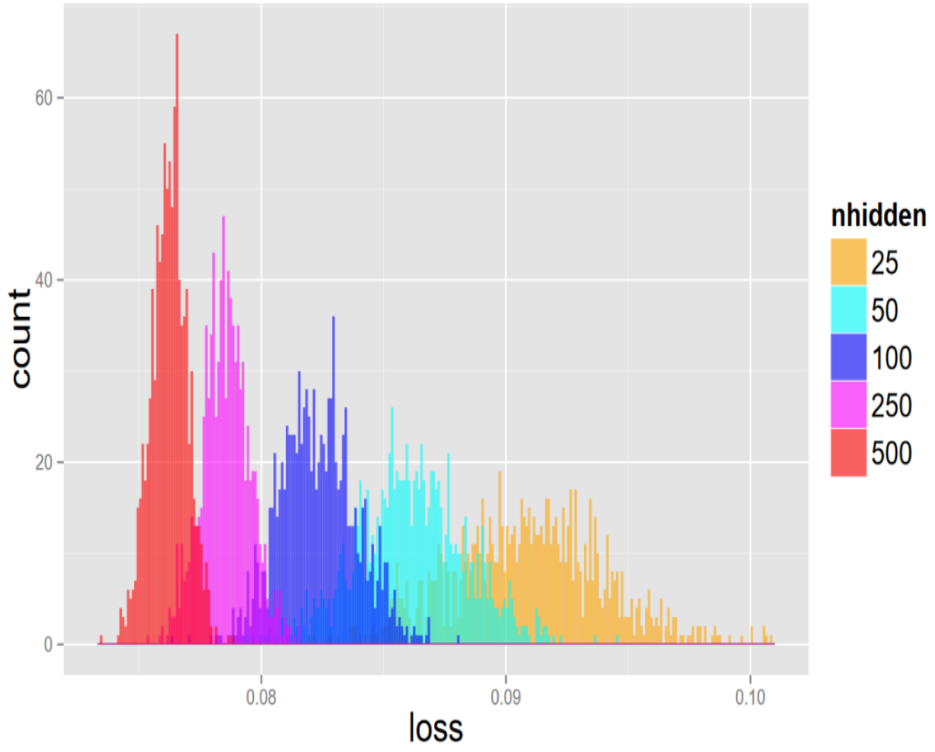
Veja Choromanska, Henaff, Mathieu, Ben Arous and Lecunn “The Loss Surface of Multilayer Networks” arXiv 1412.0233, 2014

Múltiplos Mínimos Locais são um Problema?

Elas acontecem, mas a maioria dos mínimos locais tem perda semelhante ao mínimo global

Para uma simples rede de imagens, na medida que a rede fica mais complexa (mais unidades ocultas), existem mais mínimos locais agrupados próximos da perda global mínima

Portanto, é bom, e até mesmo desejável, projetar redes muito complexas para aliviar o problema dos mínimos locais



Veja Choromanska, Henaff, Mathieu, Ben Arous and Lecunn “The Loss Surface of Multilayer Networks” arXiv 1412.0233, 2014