Aula 22 - Aprendizado em Larga Escala

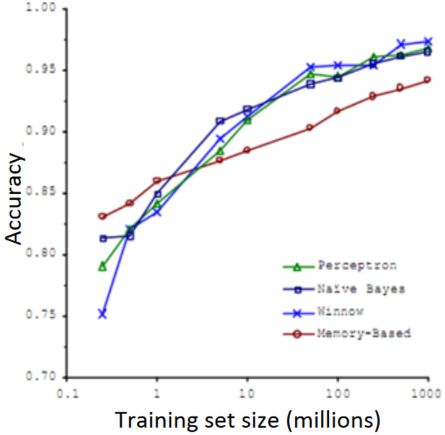
João Florindo

Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas - Brasil
florindo@unicamp.br

Outline

- HIPÓTESE: “O vencedor não é quem tem o melhor algoritmo, mas sim que tem mais dados!”

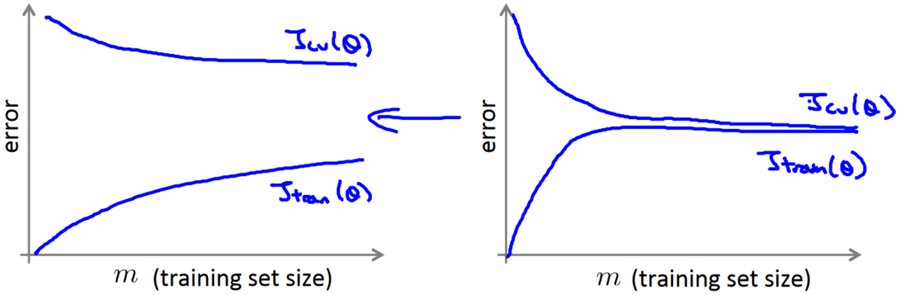
- HIPÓTESE: “O vencedor não é quem tem o melhor algoritmo, mas sim que tem mais dados!”



- PROBLEMA: Custo computacional.
- Gradiente descendente

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}.$$

  $m = 100000000$ é uma situação usual (p.ex., população de um país).
- Mas $m = 1000$ não seria suficiente? R.: Curva de aprendizado!

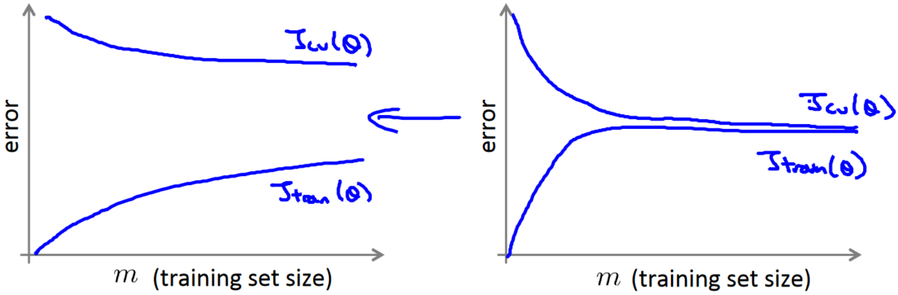


- PROBLEMA: Custo computacional.
- Gradiente descendente

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}.$$

$m = 100000000$ é uma situação usual (p.ex., população de um país).

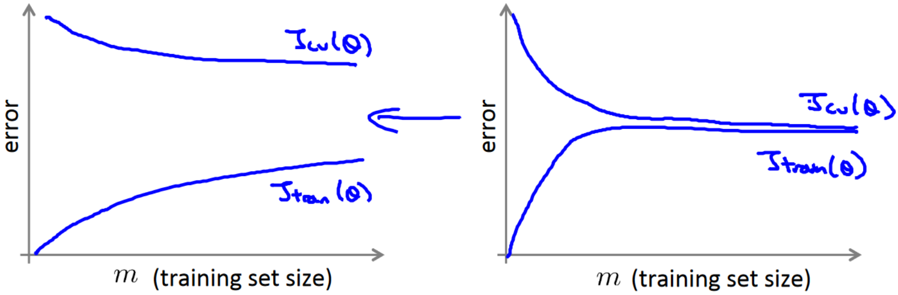
- Mas $m = 1000$ não seria suficiente? R.: Curva de aprendizado!



- PROBLEMA: Custo computacional.
- Gradiente descendente

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}.$$

  $m = 100000000$ é uma situação usual (p.ex., população de um país).
- Mas $m = 1000$ não seria suficiente? R.: Curva de aprendizado!



- Se $J_{treino}(\theta)$ aumenta devagar e $J_{cv}(\theta)$ diminui devagar (esquerda), temos variância alta: aumentar m é conveniente.

- Mesmo à direita (viés alto), deve-se aumentar os atributos (ou unidades de uma rede neural): novamente requer mais dados de treinamento.

- Se $J_{treino}(\theta)$ aumenta devagar e $J_{cv}(\theta)$ diminui devagar (esquerda), temos variância alta: aumentar m é conveniente.

- Mesmo à direita (viés alto), deve-se aumentar os atributos (ou unidades de uma rede neural): novamente requer mais dados de treinamento.

Outline

- Gradiente descendente na regressão linear:

$$\boxed{\begin{array}{l} h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \\ J_{treino}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \\ \text{Repita } \{ \\ \qquad \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \\ \qquad \text{(para todo } j = 0, \cdots, n) \\ \} \end{array}}$$

- Cada iteração executada sobre todos os exemplos de treinamento DE UMA VEZ.
- Somatório representa a derivada

$$\frac{\partial}{\partial \theta_j} J_{treino}(\theta)$$

e a função de custo caminha diretamente para o mínimo.

- Gradiente descendente na regressão linear:

$$\boxed{\begin{array}{l} h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \\ J_{treino}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \\ \text{Repita } \{ \\ \qquad \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \\ \qquad \text{(para todo } j = 0, \cdots, n) \\ \} \end{array}}$$

- Cada iteração executada sobre todos os exemplos de treinamento DE UMA VEZ.

- Somatório representa a derivada

$$\frac{\partial}{\partial \theta_j} J_{treino}(\theta)$$

e a função de custo caminha diretamente para o mínimo.

- Gradiente descendente na regressão linear:

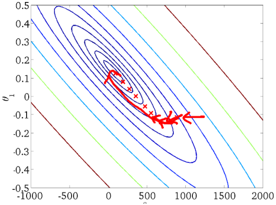
$$\boxed{\begin{array}{l} h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \\ J_{treino}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \\ \text{Repita } \{ \\ \qquad \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \\ \qquad \text{(para todo } j = 0, \cdots, n) \\ \} \end{array}}$$

- Cada iteração executada sobre todos os exemplos de treinamento DE UMA VEZ.

- Somatório representa a derivada

$$\frac{\partial}{\partial \theta_j} J_{treino}(\theta)$$

e a função de custo caminha diretamente para o mínimo.



- Este é o gradiente descendente em **lote** (*batch*).
- Alto custo para m grande.

- SOLUÇÃO: **gradiente descendente estocástico**.

- Parâmetros atualizados em cada iteração a partir de um ÚNICO exemplo de treino.

- Muito mais rápido!

1. Embaralhar aleatoriamente (reordenar) os exemplos de treinamento
2. Repita { % Usualmente repete $1 \sim 10\times$

   para $i := 1$ até m{

   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

   (para todo $j = 0, \cdots, n$)

   }

}

- SOLUÇÃO: **gradiente descendente estocástico**.

- Parâmetros atualizados em cada iteração a partir de um ÚNICO exemplo de treino.

- Muito mais rápido!

1. Embaralhar aleatoriamente (reordenar) os exemplos de treinamento
2. Repita { % Usualmente repete $1 \sim 10\times$

   para $i := 1$ até m{

   $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

   (para todo $j = 0, \cdots, n$)

   }

}

- SOLUÇÃO: **gradiente descendente estocástico**.
- Parâmetros atualizados em cada iteração a partir de um ÚNICO exemplo de treino.
- Muito mais rápido!

1. Embaralhar aleatoriamente (reordenar) os exemplos de treinamento
2. Repita { % Usualmente repete $1 \sim 10\times$

    para $i := 1$ até $m\{$

    $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

    (para todo $j = 0, \cdots, n$)

    }

}

- SOLUÇÃO: **gradiente descendente estocástico**.
- Parâmetros atualizados em cada iteração a partir de um ÚNICO exemplo de treino.
- Muito mais rápido!

1. Embaralhar aleatoriamente (reordenar) os exemplos de treinamento
2. Repita { % Usualmente repete $1 \sim 10\times$

    para $i := 1$ até m{

    $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

    (para todo $j = 0, \cdots, n$)

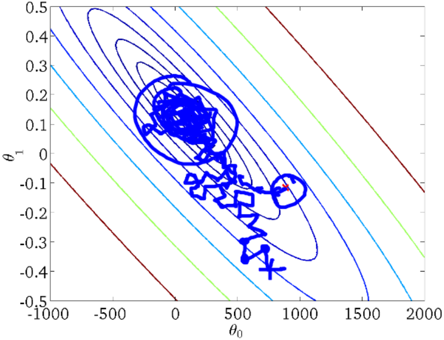
    }

}

- Caminhada rumo ao mínimo é mais errática:

- Pode não chegar no mínimo de fato, mas isso na prática não costuma ser problema.

- Caminhada rumo ao mínimo é mais errática:



- Pode não chegar no mínimo de fato, mas isso na prática não costuma ser problema.

Outline

- Meio-termo entre o gradiente em *batch* (m exemplos em cada iteração) e o gradiente estocástico ($m = 1$).
- b exemplos por iteração.
- b é o **tamanho do mini-lote** (usualmente com valores $2 \sim 100$).
- Exemplo com $b = 10$ e $m = 1000$:

$$\begin{array}{l} \text{Repita } \{ \\ \qquad \text{para } i = 1, 11, 21, 31, \cdots, 991 \ \{ \\ \qquad\qquad \theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)} \\ \qquad\qquad (\text{para todo } j = 0, \cdots, n) \\ \qquad \} \\ \} \end{array}$$

- O somatório pode ser vetorizado.

- Meio-termo entre o gradiente em *batch* (m exemplos em cada iteração) e o gradiente estocástico ($m = 1$).
- b exemplos por iteração.
- b é o **tamanho do mini-lote** (usualmente com valores $2 \sim 100$).
- Exemplo com $b = 10$ e $m = 1000$:

Repita {
    para $i = 1, 11, 21, 31, \cdots, 991$ {
        $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$
        (para todo $j = 0, \cdots, n$)
    }
}

- O somatório pode ser vetorizado.

- Meio-termo entre o gradiente em *batch* (m exemplos em cada iteração) e o gradiente estocástico ($m = 1$).
- b exemplos por iteração.
- b é o **tamanho do mini-lote** (usualmente com valores $2 \sim 100$).
- Exemplo com $b = 10$ e $m = 1000$:

Repita {
  para $i = 1, 11, 21, 31, \cdots, 991$ {
    $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$
    (para todo $j = 0, \cdots, n$)
  }
}

- O somatório pode ser vetorizado.

- Meio-termo entre o gradiente em *batch* (m exemplos em cada iteração) e o gradiente estocástico ($m = 1$).
- b exemplos por iteração.
- b é o **tamanho do mini-lote** (usualmente com valores $2 \sim 100$).
- Exemplo com $b = 10$ e $m = 1000$:

Repita {
  para $i = 1, 11, 21, 31, \cdots, 991$ {
    $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$
    (para todo $j = 0, \cdots, n$)
  }
}

- O somatório pode ser vetorizado.

- Meio-termo entre o gradiente em *batch* (m exemplos em cada iteração) e o gradiente estocástico ($m = 1$).

- b exemplos por iteração.

- b é o **tamanho do mini-lote** (usualmente com valores $2 \sim 100$).

- Exemplo com $b = 10$ e $m = 1000$:

Repita {
  para $i = 1, 11, 21, 31, \cdots, 991$ {
    $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$
    (para todo $j = 0, \cdots, n$)
  }
}

- O somatório pode ser vetorizado.

Outline

- Gradiente clássico tem convergência garantida.
- Debugamos plotando $J_{treino}(\theta)$.
- No gradiente estocástico temos o custo individual

$$custo(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h(x^{(i)}) - y^{(i)})^2$$

  e $J_{treino}(\theta)$ flutua.
- Plotar (p.ex. a cada 1000 iterações) a média de $custo(\theta, (x^{(i)}, y^{(i)}))$ sobre os últimos 1000 exemplos processados.

- Gradiente clássico tem convergência garantida.
- Debugamos plotando $J_{treino}(\theta)$.
- No gradiente estocástico temos o custo individual

$$custo(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h(x^{(i)}) - y^{(i)})^2$$

  e $J_{treino}(\theta)$ flutua.
- Plotar (p.ex. a cada 1000 iterações) a média de $custo(\theta, (x^{(i)}, y^{(i)}))$ sobre os últimos 1000 exemplos processados.

- Gradiente clássico tem convergência garantida.
- Debugamos plotando $J_{treino}(\theta)$.
- No gradiente estocástico temos o custo individual

$$custo(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h(x^{(i)}) - y^{(i)})^2$$

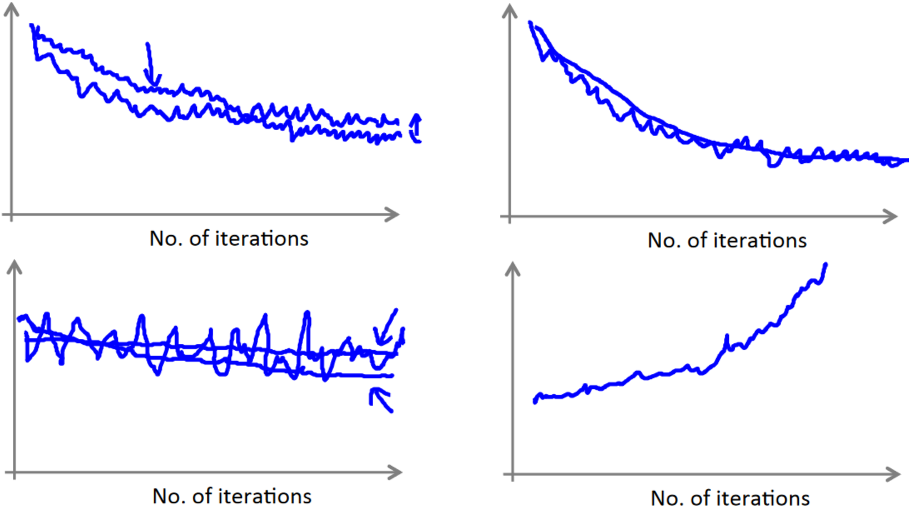
e $J_{treino}(\theta)$ flutua.

- Plotar (p.ex. a cada 1000 iterações) a média de $custo(\theta, (x^{(i)}, y^{(i)}))$ sobre os últimos 1000 exemplos processados.

- Gradiente clássico tem convergência garantida.
- Debugamos plotando $J_{treino}(\theta)$.
- No gradiente estocástico temos o custo individual

$$custo(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h(x^{(i)}) - y^{(i)})^2$$

  e $J_{treino}(\theta)$ flutua.
- Plotar (p.ex. a cada 1000 iterações) a média de $custo(\theta, (x^{(i)}, y^{(i)}))$ sobre os últimos 1000 exemplos processados.


No. of iterations
No. of iterations
No. of iterations
No. of iterations

- Se estiver correto, custo médio tende a cair.
- Com α pequeno, cai pouco, mas cai! (superior esquerda)
- Mais exemplos na média, p.ex., 5000 (superior direita) suaviza a curva, mas atrasa reflexo de alguma mudança de tendência.
- Flutuação em torno de constante (inferior esquerda) indica que não há aprendizado: diminuir α, mais atributos, etc.
- Média ascendente (inferior direita): diminuir α.

- Se estiver correto, custo médio tende a cair.
- Com α pequeno, cai pouco, mas cai! (superior esquerda)
- Mais exemplos na média, p.ex., 5000 (superior direita) suaviza a curva, mas atrasa reflexo de alguma mudança de tendência.
- Flutuação em torno de constante (inferior esquerda) indica que não há aprendizado: diminuir α, mais atributos, etc.
- Média ascendente (inferior direita): diminuir α.

- Se estiver correto, custo médio tende a cair.
- Com α pequeno, cai pouco, mas cai! (superior esquerda)
- Mais exemplos na média, p.ex., 5000 (superior direita) suaviza a curva, mas atrasa reflexo de alguma mudança de tendência.
- Flutuação em torno de constante (inferior esquerda) indica que não há aprendizado: diminuir α, mais atributos, etc.
- Média ascendente (inferior direita): diminuir α.

- Se estiver correto, custo médio tende a cair.
- Com α pequeno, cai pouco, mas cai! (superior esquerda)
- Mais exemplos na média, p.ex., 5000 (superior direita) suaviza a curva, mas atrasa reflexo de alguma mudança de tendência.
- Flutuação em torno de constante (inferior esquerda) indica que não há aprendizado: diminuir α, mais atributos, etc.
- Média ascendente (inferior direita): diminuir α.

- Se estiver correto, custo médio tende a cair.
- Com α pequeno, cai pouco, mas cai! (superior esquerda)
- Mais exemplos na média, p.ex., 5000 (superior direita) suaviza a curva, mas atrasa reflexo de alguma mudança de tendência.
- Flutuação em torno de constante (inferior esquerda) indica que não há aprendizado: diminuir α, mais atributos, etc.
- Média ascendente (inferior direita): diminuir α.

- α pode diminuir a cada iteração:

$$\alpha = \frac{\mathit{const}1}{\text{número da iteração} + \mathit{const}2}.$$

- Menos chance de ficar flutuando em torno de um mínimo sem chegar nele.

- MAS: implica em mais dois hiperparâmetros para ajustar.

- Uma aproximação do mínimo já costuma ser sificiente.

- α pode diminuir a cada iteração:

$$\alpha = \frac{\textit{const}1}{\text{número da iteração} + \textit{const}2}.$$

- Menos chance de ficar flutuando em torno de um mínimo sem chegar nele.

- MAS: implica em mais dois hiperparâmetros para ajustar.

- Uma aproximação do mínimo já costuma ser sificiente.

- α pode diminuir a cada iteração:

$$\alpha = \frac{\textit{const}1}{\text{número da iteração} + \textit{const}2}.$$

- Menos chance de ficar flutuando em torno de um mínimo sem chegar nele.

- MAS: implica em mais dois hiperparâmetros para ajustar.

- Uma aproximação do mínimo já costuma ser sificiente.

- α pode diminuir a cada iteração:

$$\alpha = \frac{\mathit{const}1}{\text{número da iteração} + \mathit{const}2}.$$

- Menos chance de ficar flutuando em torno de um mínimo sem chegar nele.

- MAS: implica em mais dois hiperparâmetros para ajustar.

- Uma aproximação do mínimo já costuma ser sificiente.

Outline

- Aprende com cada novo exemplo de treino que chega e em seguida o descarta.

- EX. 1 - Correios:
  - Usuário entra origem e destino e o sistema calcula um preço para a entrega.
  - Usuário contrata ($y = 1$) ou não ($y = 0$) o serviço.
  - Atributos: propriedades do usuário, origem/destino, preço oferecido, etc.
  - Aprender $p(y = 1|x; \theta)$ para otimizar o preço.

```
Repita "para sempre" {
        Obter (x, y) do usuário correspondente
        Atualizar θ usando (x, y):
                θ_j := θ_j − α(h_θ(x) − y)x_j   (j = 0, ···, n)
}
```

NOTAS

- Não temos mais $(x^{(i)}, y^{(i)})$ pois só o usuário atual interessa.
- ADAPTA-SE a mudanças de preferência dos clientes, p.ex., uma crise.

- Aprende com cada novo exemplo de treino que chega e em seguida o descarta.

- EX. 1 - Correios:
  - Usuário entra origem e destino e o sistema calcula um preço para a entrega.
  - Usuário contrata ($y = 1$) ou não ($y = 0$) o serviço.
  - Atributos: propriedades do usuário, origem/destino, preço oferecido, etc.
  - Aprender $p(y = 1|x; \theta)$ para otimizar o preço.

Repita "para sempre" {
    Obter (x, y) do usuário correspondente
    Atualizar θ usando (x, y):
        $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j \; (j = 0, \cdots, n)$
}

NOTAS

- Não temos mais $(x^{(i)}, y^{(i)})$ pois só o usuário atual interessa.
- ADAPTA-SE a mudanças de preferência dos clientes, p.ex., uma crise.

- Aprende com cada novo exemplo de treino que chega e em seguida o descarta.

- EX. 1 - Correios:
  - Usuário entra origem e destino e o sistema calcula um preço para a entrega.
  - Usuário contrata ($y = 1$) ou não ($y = 0$) o serviço.
  - Atributos: propriedades do usuário, origem/destino, preço oferecido, etc.
  - Aprender $p(y = 1|x; \theta)$ para otimizar o preço.

    Repita "para sempre" {
    Obter (x, y) do usuário correspondente
    Atualizar θ usando (x, y):
    $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j \; (j = 0, \cdots, n)$
    }

NOTAS

- Não temos mais $(x^{(i)}, y^{(i)})$ pois só o usuário atual interessa.
- ADAPTA-SE a mudanças de preferência dos clientes, p.ex., uma crise.

- Aprende com cada novo exemplo de treino que chega e em seguida o descarta.

- EX. 1 - Correios:
  - Usuário entra origem e destino e o sistema calcula um preço para a entrega.
  - Usuário contrata ($y = 1$) ou não ($y = 0$) o serviço.
  - Atributos: propriedades do usuário, origem/destino, preço oferecido, etc.
  - Aprender $p(y = 1|x; \theta)$ para otimizar o preço.

```
Repita "para sempre" {
      Obter (x, y) do usuário correspondente
      Atualizar θ usando (x, y):
            θ_j := θ_j − α(h_θ(x) − y)x_j  (j = 0, ··· , n)
}
```

NOTAS

- Não temos mais $(x^{(i)}, y^{(i)})$ pois só o usuário atual interessa.
- ADAPTA-SE a mudanças de preferência dos clientes, p.ex., uma crise.

- Aprende com cada novo exemplo de treino que chega e em seguida o descarta.

- EX. 1 - Correios:
  - Usuário entra origem e destino e o sistema calcula um preço para a entrega.
  - Usuário contrata ($y = 1$) ou não ($y = 0$) o serviço.
  - Atributos: propriedades do usuário, origem/destino, preço oferecido, etc.
  - Aprender $p(y = 1|x; \theta)$ para otimizar o preço.

    Repita "para sempre" {
    Obter (x, y) do usuário correspondente
    Atualizar θ usando (x, y):
    $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j \; (j = 0, \cdots, n)$
    }

NOTAS

- Não temos mais $(x^{(i)}, y^{(i)})$ pois só o usuário atual interessa.
- ADAPTA-SE a mudanças de preferência dos clientes, p.ex., uma crise.

- Aprende com cada novo exemplo de treino que chega e em seguida o descarta.

- EX. 1 - Correios:
  - Usuário entra origem e destino e o sistema calcula um preço para a entrega.
  - Usuário contrata ($y = 1$) ou não ($y = 0$) o serviço.
  - Atributos: propriedades do usuário, origem/destino, preço oferecido, etc.
  - Aprender $p(y = 1|x; \theta)$ para otimizar o preço.

Repita "para sempre" {
  Obter (x, y) do usuário correspondente
  Atualizar θ usando (x, y):
    $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j \; (j = 0, \cdots, n)$
}

NOTAS

- Não temos mais $(x^{(i)}, y^{(i)})$ pois só o usuário atual interessa.
- ADAPTA-SE a mudanças de preferência dos clientes, p.ex., uma crise.

- Aprende com cada novo exemplo de treino que chega e em seguida o descarta.

- EX. 1 - Correios:
  - Usuário entra origem e destino e o sistema calcula um preço para a entrega.
  - Usuário contrata ($y = 1$) ou não ($y = 0$) o serviço.
  - Atributos: propriedades do usuário, origem/destino, preço oferecido, etc.
  - Aprender $p(y = 1|x; \theta)$ para otimizar o preço.

Repita "para sempre" {

Obter (x, y) do usuário correspondente

Atualizar θ usando (x, y):

$$\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j \ (j = 0, \cdots, n)$$

}

NOTAS

- Não temos mais $(x^{(i)}, y^{(i)})$ pois só o usuário atual interessa.
- ADAPTA-SE a mudanças de preferência dos clientes, p.ex., uma crise.

Outro Exemplo

- Usuário busca por "celular Android com câmera 4k".
- Existem 100 celulares na loja e essa busca retorna 10.
- $x =$ atributos do celular, quantas palavras na busca casam com o nome ou a descrição do celular, etc.
- $y = 1$ se o usuário clica no link e $y = 0$ caso contrário.

Outro Exemplo

- Usuário busca por "celular Android com câmera 4k".
- Existem 100 celulares na loja e essa busca retorna 10.
- x = atributos do celular, quantas palavras na busca casam com o nome ou a descrição do celular, etc.
- $y = 1$ se o usuário clica no link e $y = 0$ caso contrário.

Outro Exemplo

- Usuário busca por "celular Android com câmera 4k".
- Existem 100 celulares na loja e essa busca retorna 10.
- $x =$ atributos do celular, quantas palavras na busca casam com o nome ou a descrição do celular, etc.
- $y = 1$ se o usuário clica no link e $y = 0$ caso contrário.

Outro Exemplo

- Usuário busca por "celular Android com câmera 4k".
- Existem 100 celulares na loja e essa busca retorna 10.
- $x =$ atributos do celular, quantas palavras na busca casam com o nome ou a descrição do celular, etc.
- $y = 1$ se o usuário clica no link e $y = 0$ caso contrário.

Outro Exemplo

- Aprender $p(y = 1|x; \theta)$: previsão de CTR (*Click True Rate*).
- Sistema poderá mostrar os 10 celulares mais susceptíveis de serem clicados.
- Outros exemplos: escolha de ofertas especiais para mostrar ao usuário, seleção customizada de artigos em um jornal/revista, recomendação de produtos (filtragem colaborativa pode gerar atributos), etc.

Outro Exemplo

- Aprender $p(y = 1|x; \theta)$: previsão de CTR (*Click True Rate*).
- Sistema poderá mostrar os 10 celulares mais susceptíveis de serem clicados.
- Outros exemplos: escolha de ofertas especiais para mostrar ao usuário, seleção customizada de artigos em um jornal/revista, recomendação de produtos (filtragem colaborativa pode gerar atributos), etc.

Outro Exemplo

- Aprender $p(y = 1|x; \theta)$: previsão de CTR (*Click True Rate*).
- Sistema poderá mostrar os 10 celulares mais susceptíveis de serem clicados.
- Outros exemplos: escolha de ofertas especiais para mostrar ao usuário, seleção customizada de artigos em um jornal/revista, recomendação de produtos (filtragem colaborativa pode gerar atributos), etc.

Outline

- Estratégia, proposta por Jefrey Dean e Sanjay Ghemawat.
- Divide uma tarefa muito grande em tarefas menores independentes e distribui entre máquinas.
- EX.: Gradiente em *batch*, $m = 400$ (no mundo real seria 400 milhões!).

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- Se tivermos 4 máquinas, dividimos o treino em 4 partes.

- Estratégia, proposta por Jefrey Dean e Sanjay Ghemawat.
- Divide uma tarefa muito grande em tarefas menores independentes e distribui entre máquinas.
- EX.: Gradiente em *batch*, $m = 400$ (no mundo real seria 400 milhões!).

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- Se tivermos 4 máquinas, dividimos o treino em 4 partes.

- Estratégia, proposta por Jefrey Dean e Sanjay Ghemawat.
- Divide uma tarefa muito grande em tarefas menores independentes e distribui entre máquinas.
- EX.: Gradiente em *batch*, $m = 400$ (no mundo real seria 400 milhões!).

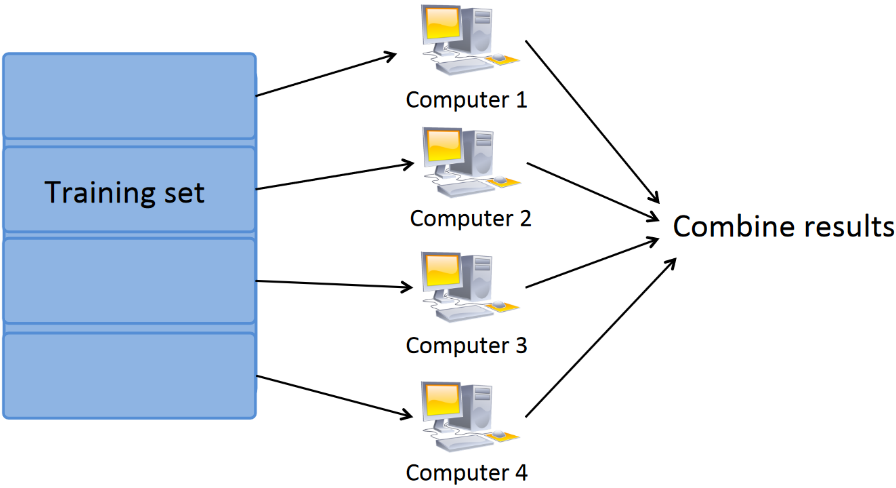
$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- Se tivermos 4 máquinas, dividimos o treino em 4 partes.

- Estratégia, proposta por Jefrey Dean e Sanjay Ghemawat.
- Divide uma tarefa muito grande em tarefas menores independentes e distribui entre máquinas.
- EX.: Gradiente em *batch*, $m = 400$ (no mundo real seria 400 milhões!).

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- Se tivermos 4 máquinas, dividimos o treino em 4 partes.


Training set
Computer 1
Computer 2
Computer 3
Computer 4
Combine results

- Máquina 1: Usa $(x^{(1)}, y^{(1)}), \cdots, (x^{(100)}, y^{(100)})$ e calcula

$$temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

- Máquina 2: Usa $(x^{(101)}, y^{(101)}), \cdots, (x^{(200)}, y^{(200)})$ e calcula

$$temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

- Máquina 3: Usa $(x^{(201)}, y^{(201)}), \cdots, (x^{(300)}, y^{(300)})$ e calcula

$$temp_j^{(3)} = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

- Máquina 4: Usa $(x^{(301)}, y^{(301)}), \cdots, (x^{(400)}, y^{(400)})$ e calcula

$$temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Finalmente, uma máquina mestre combina os resultados parciais:

$$\theta_j := \theta_j - \alpha \frac{1}{400}(temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)}), \qquad (j = 0, \cdots, n).$$

- Pode ser aplicado sobre qualquer algoritmo que possa ser reescrito como uma soma de partes.

- Verdade para vários algoritmos de aprendizado.

- Inclui algoritmos avançados de otimização. EX.: Na regressão logística:

$$J_{treino}(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

e algoritmos avançados precisam da derivada parcial

$$\frac{\partial}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}.$$

Este somatório pode ser quebrado em partes e o *MapReduce* ser aplicado.

- Pode ser aplicado sobre qualquer algoritmo que possa ser reescrito como uma soma de partes.
- Verdade para vários algoritmos de aprendizado.
- Inclui algoritmos avançados de otimização. EX.: Na regressão logística:

$$J_{treino}(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

e algoritmos avançados precisam da derivada parcial

$$\frac{\partial}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}.$$

Este somatório pode ser quebrado em partes e o *MapReduce* ser aplicado.

- Pode ser aplicado sobre qualquer algoritmo que possa ser reescrito como uma soma de partes.

- Verdade para vários algoritmos de aprendizado.

- Inclui algoritmos avançados de otimização. EX.: Na regressão logística:

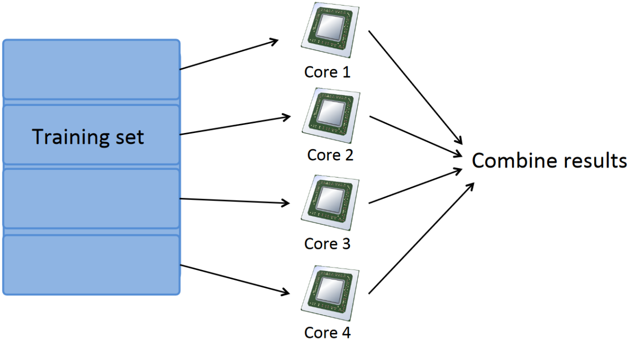
$$J_{treino}(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

e algoritmos avançados precisam da derivada parcial

$$\frac{\partial}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}.$$

Este somatório pode ser quebrado em partes e o *MapReduce* ser aplicado.

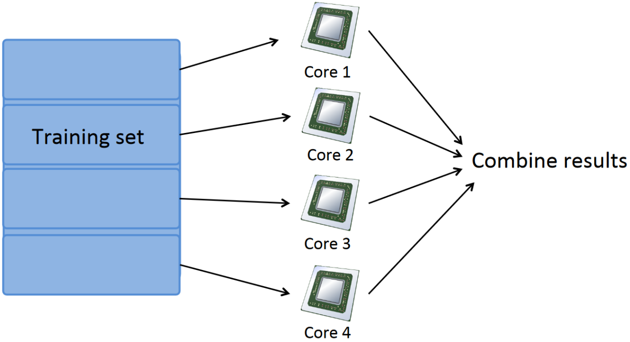
- Mesmo para uma única máquina, mas com vários *cores*, o *MapReduce* se aplica de forma idêntica.



- Neste caso praticamente não há latência/*overhead* na comunicação entre os nós.

- Maioria das bibliotecas de álgebra linear já fazem também essa divisão de tarefas entre os núcleos automaticamente.

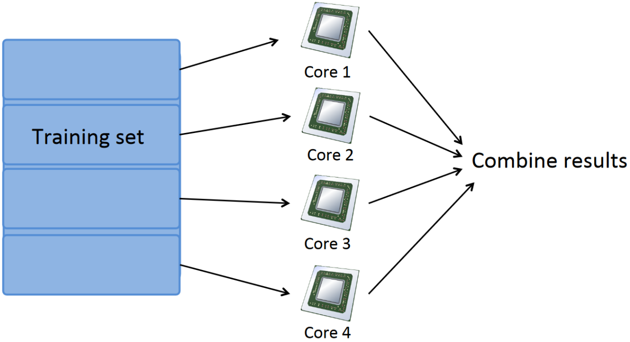
- Mesmo para uma única máquina, mas com vários *cores*, o *MapReduce* se aplica de forma idêntica.



- Neste caso praticamente não há latência/*overhead* na comunicação entre os nós.

- Maioria das bibliotecas de álgebra linear já fazem também essa divisão de tarefas entre os núcleos automaticamente.

- Mesmo para uma única máquina, mas com vários *cores*, o *MapReduce* se aplica de forma idêntica.



- Neste caso praticamente não há latência/*overhead* na comunicação entre os nós.

- Maioria das bibliotecas de álgebra linear já fazem também essa divisão de tarefas entre os núcleos automaticamente.