Aprendizado Profundo 1

Conexões Residuais

Professor: Lucas Silveira Kupssinskü

Contexto

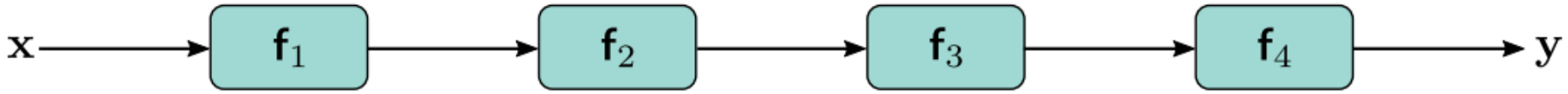
- Até agora vimos que:
  - MLPs são bons para resolver problemas de regressão e classificação
  - Camadas convolucionais são boas para trabalhar com imagens
  - Aumentar a profundidade da rede melhora os resultados
  - Gradiente que se dissipa e gradiente explosivo são problemas
    - Principalmente em redes neurais mais profundas

Contexto

- O que vamos ver agora
  - Modificações aditivas possibilitam treinar redes mais profundas
    - Isso: $h_2 = h_1 + f_2[h_1, \theta_2]$, $h_1 = x + f_1[x, \theta_1]$
    - Ao invés de: $h_2 = f_2[f_1[x, \theta_1], \theta_2]$
  - Esses “caminhos alternativos” recebem o nome de conexões residuais ou *skip connections*
  - Introduzimos *batch norm* para evitar gradiente explosivo

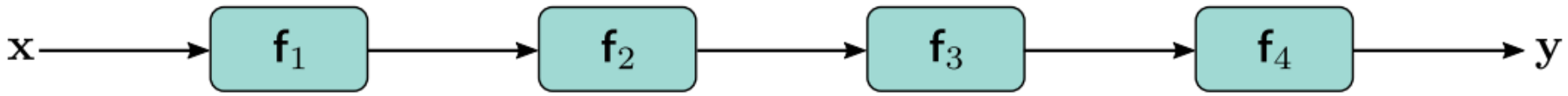
Processamento Sequencial

- Tanto as CNNs quanto os MLPs que vimos até agora processam sequencialmente cada uma das suas camadas
- $h_1 = f_1[x, \theta_1]$
- $h_2 = f_2[h_1, \theta_2]$
- $h_3 = f_3[h_2, \theta_3]$
- …
- $h_n = f_n[h_{n-1}, \theta_n]$



Processamento Sequencial

- Uma forma alternativa de pensar é escrever a mesma expressão como uma composição de funções
- $h_1 = f_1[x, \theta_1]$
- $h_n = f_n[\dots f_3[f_2[f_1[x, \theta_1], \theta_2], \theta_3], \theta_n]$



Processamento Sequencial

- AlexNet
  - 8 camadas
  - Acurácia 84.7%
- VGG-16
  - 16 camadas
  - Acurácia 92.7%
- Por que não aumentar a profundidade?
  - Pois a acurácia começa a piorar…

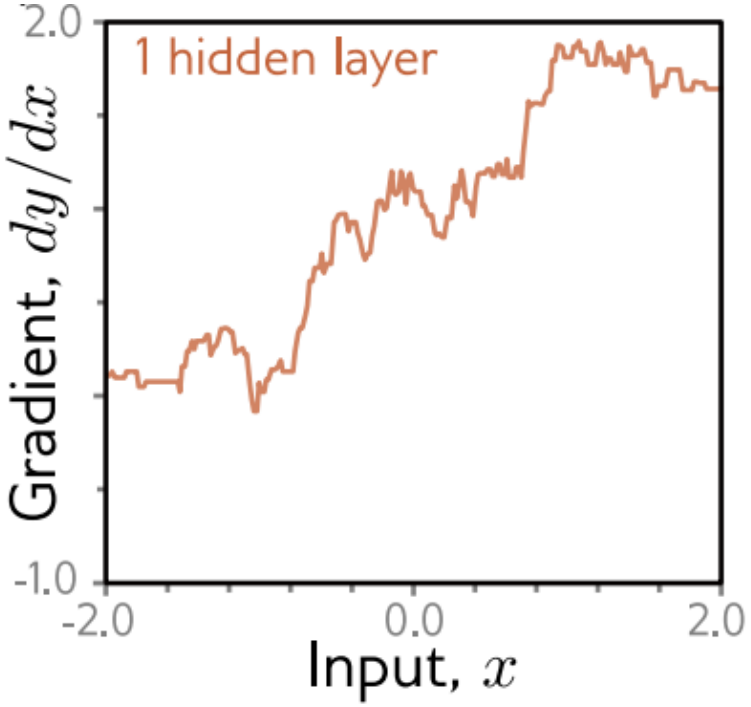
Processamento Sequencial

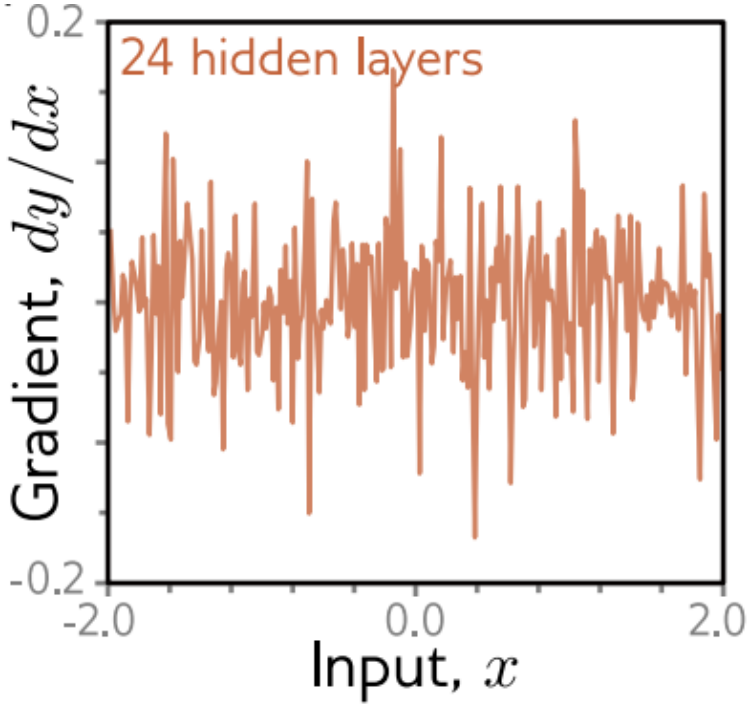
- Esse fenômeno não é completamente compreendido/explicado
- A hipótese mais aceita é a do “gradiente quebrado” (*shattered gradient*)
- Pequenas mudanças nos pesos causam mudanças erráticas nas camadas seguintes
  - A diferença do passo “infinitesimal” das equações para o passo utilizado fazem mais diferença

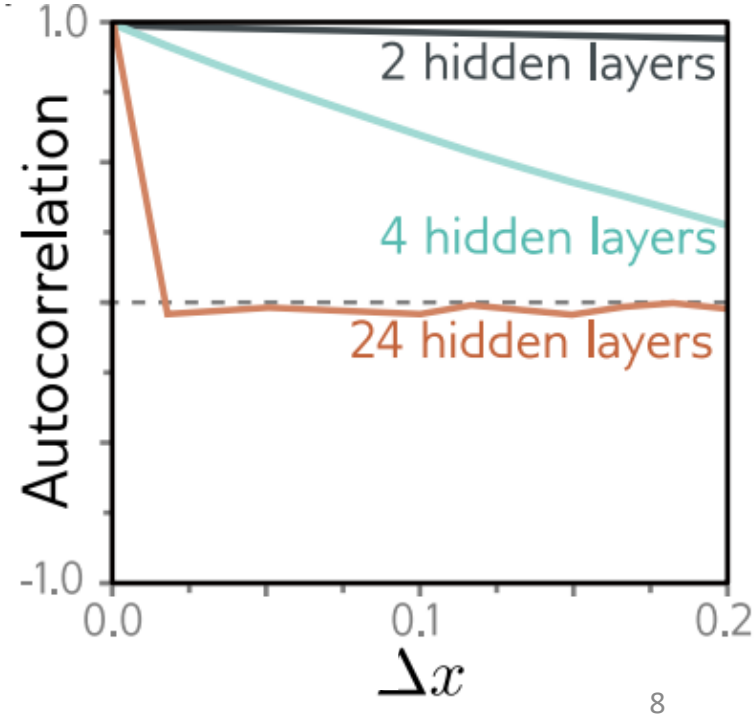
BALDUZZI, David et al. The shattered gradients problem: If resnets are the answer, then what is the question?. In: **International Conference on Machine Learning**. PMLR, 2017. p. 342-350.

Processamento Sequencial

- $\frac{\partial f_4}{\partial f_1} = \frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}$
- Repare que uma mudança em f_1 altera todas as funções subsequentes



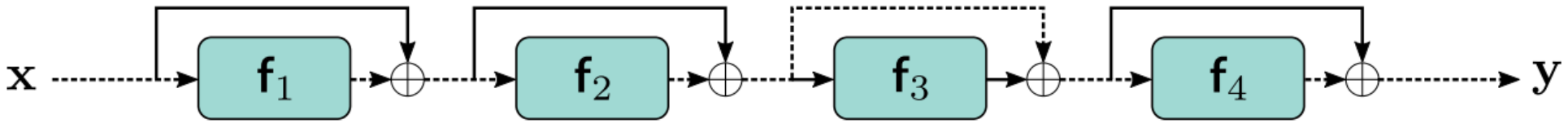




BALDUZZI, David et al. The shattered gradients problem: If resnets are the answer, then what is the question?. In: **International Conference on Machine Learning**. PMLR, 2017. p. 342-350.

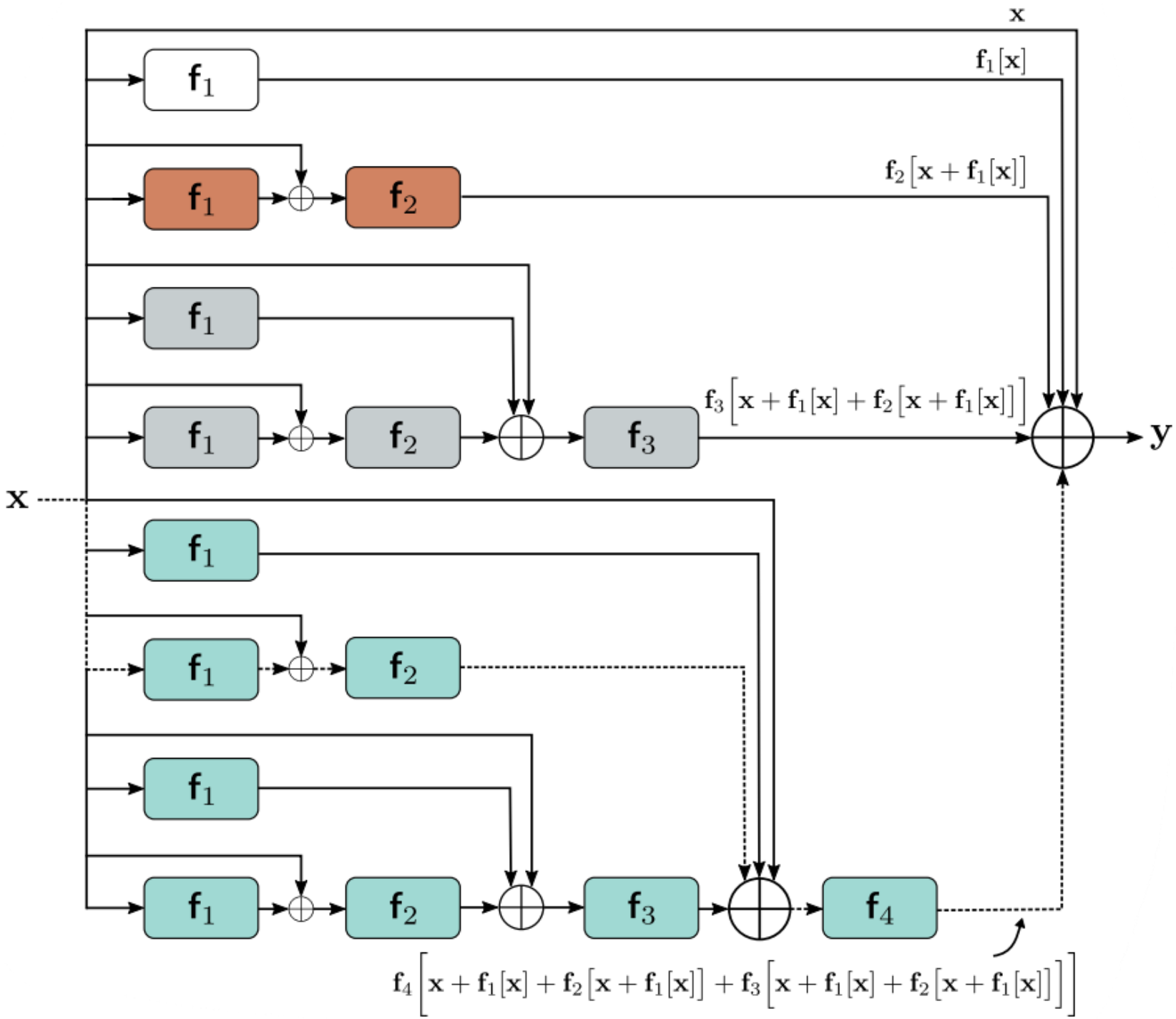
Conexões Residuais (*Residual* ou *Skip Connections*)

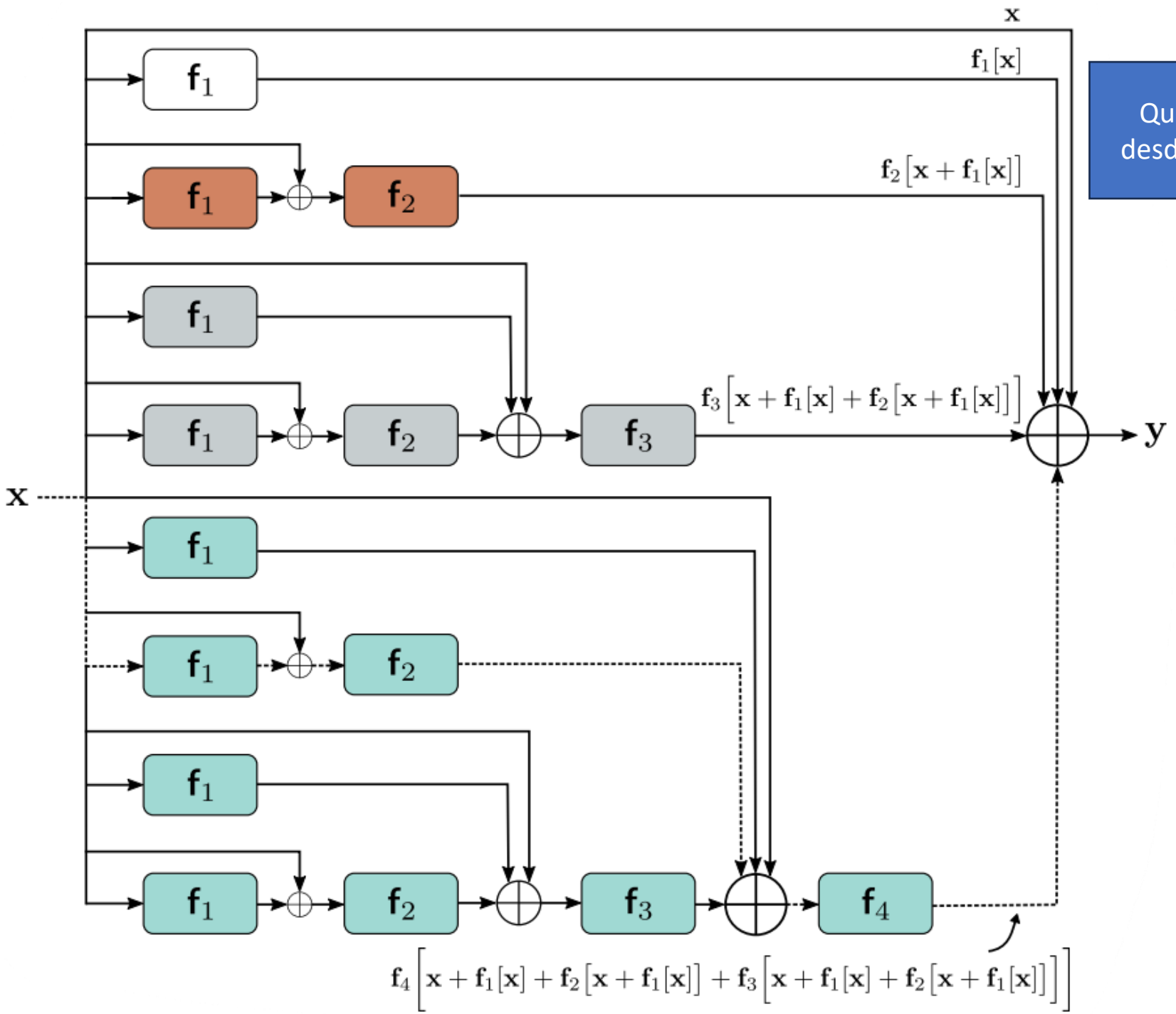
- Criamos ramos no grafo computacional
  - Adicionamos uma mudança a entrada
    - $h_4 = h_3 + f_4[h_3, \theta_4]$
    - $h_3 = h_2 + f_3[h_2, \theta_3]$
    - $h_2 = h_1 + f_2[h_1, \theta_2]$
    - $h_1 = x + f_1[x, \theta_1]$
  - O temo que está sendo adicionado é a conexão residual
    - Repare que agora temos a restrição da saída ter o mesmo tamanho da entrada
  - Cada mudança aditiva que realizamos é conhecida como *residual block* ou *residual layer*



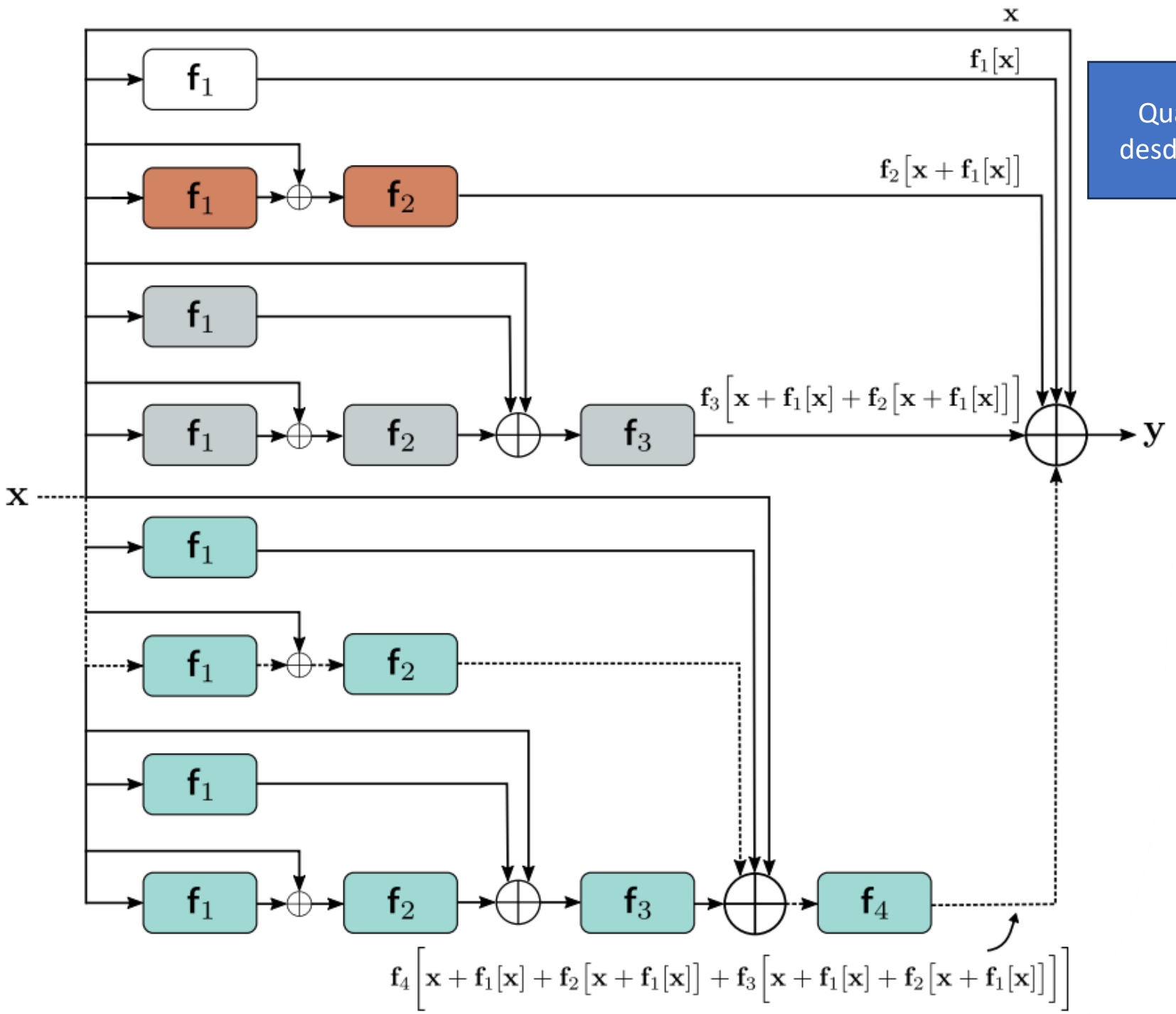
Conexões Residuais (*Residual* ou *Skip Connections*)

- Abrindo as expressões anteriores temos:
- $y = x + f_1[x]$
  $+f_2\left[x + f_1[x]\right]$
  $+f_3\left[x + f_1[x] + f_2\left[x + f_1[x]\right]\right]$
  $+f_4\left[x + f_1[x] + f_2\left[x + f_1[x]\right] + f_3\left[x + f_1[x] + f_2\left[x + f_1[x]\right]\right]\right]$


x
$f_1[x]$
$f_2[x + f_1[x]]$
$f_3[x + f_1[x] + f_2[x + f_1[x]]]$
y
x
$f_4[x + f_1[x] + f_2[x + f_1[x]] + f_3[x + f_1[x] + f_2[x + f_1[x]]]]$

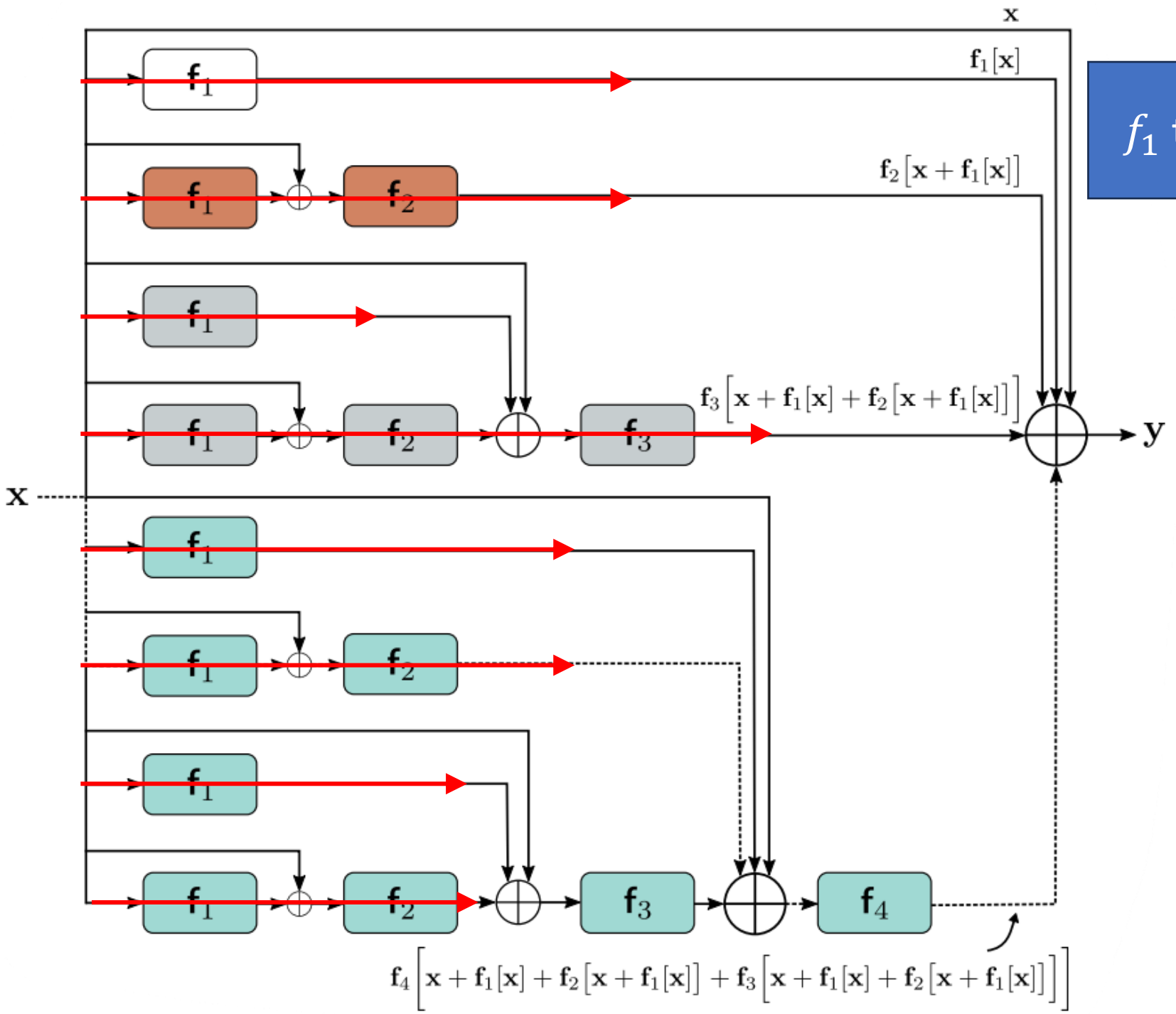


Quantos caminhos temos desde a entrada até a saída?



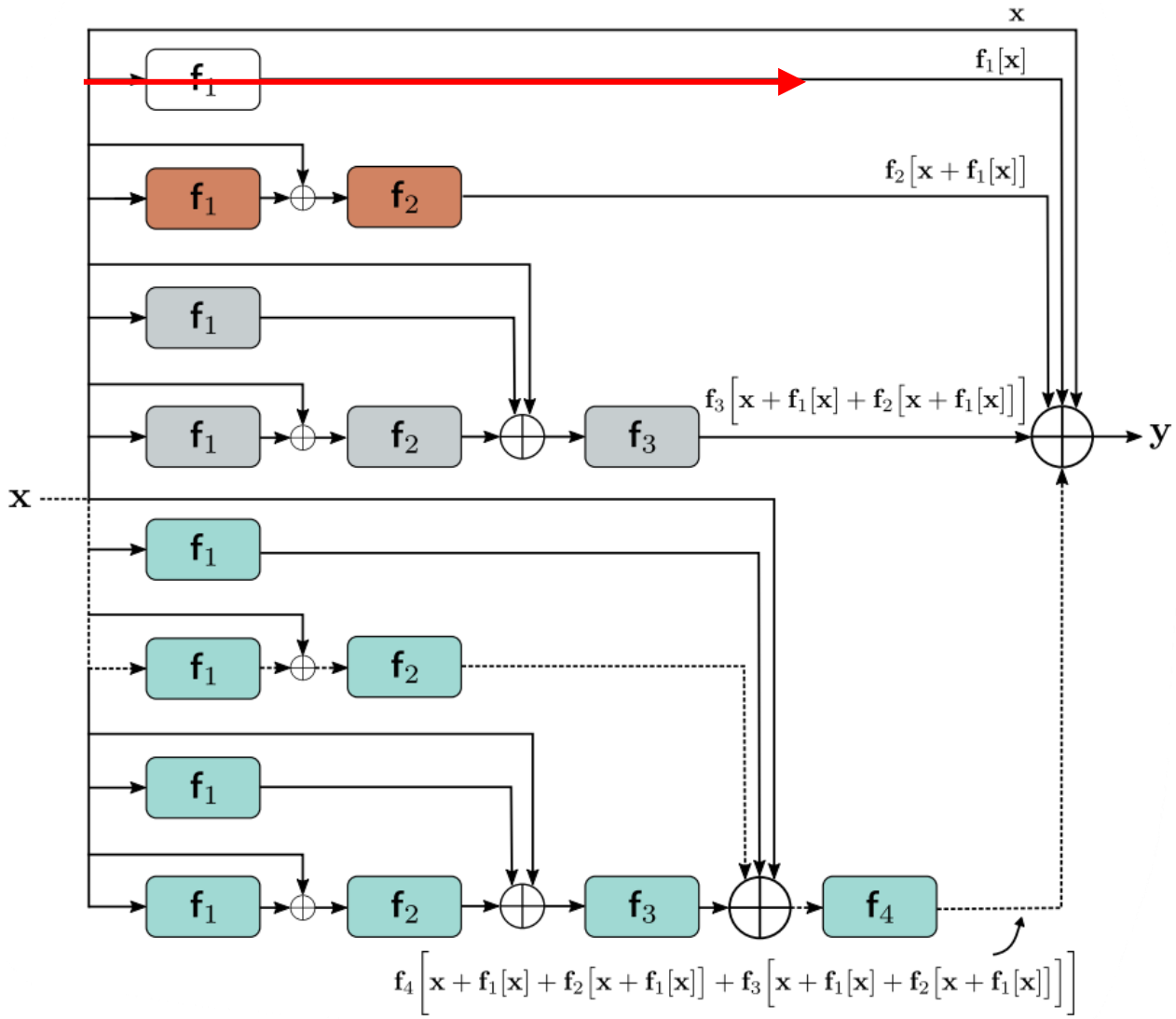
Quantos caminhos temos desde a entrada até a saída?

16



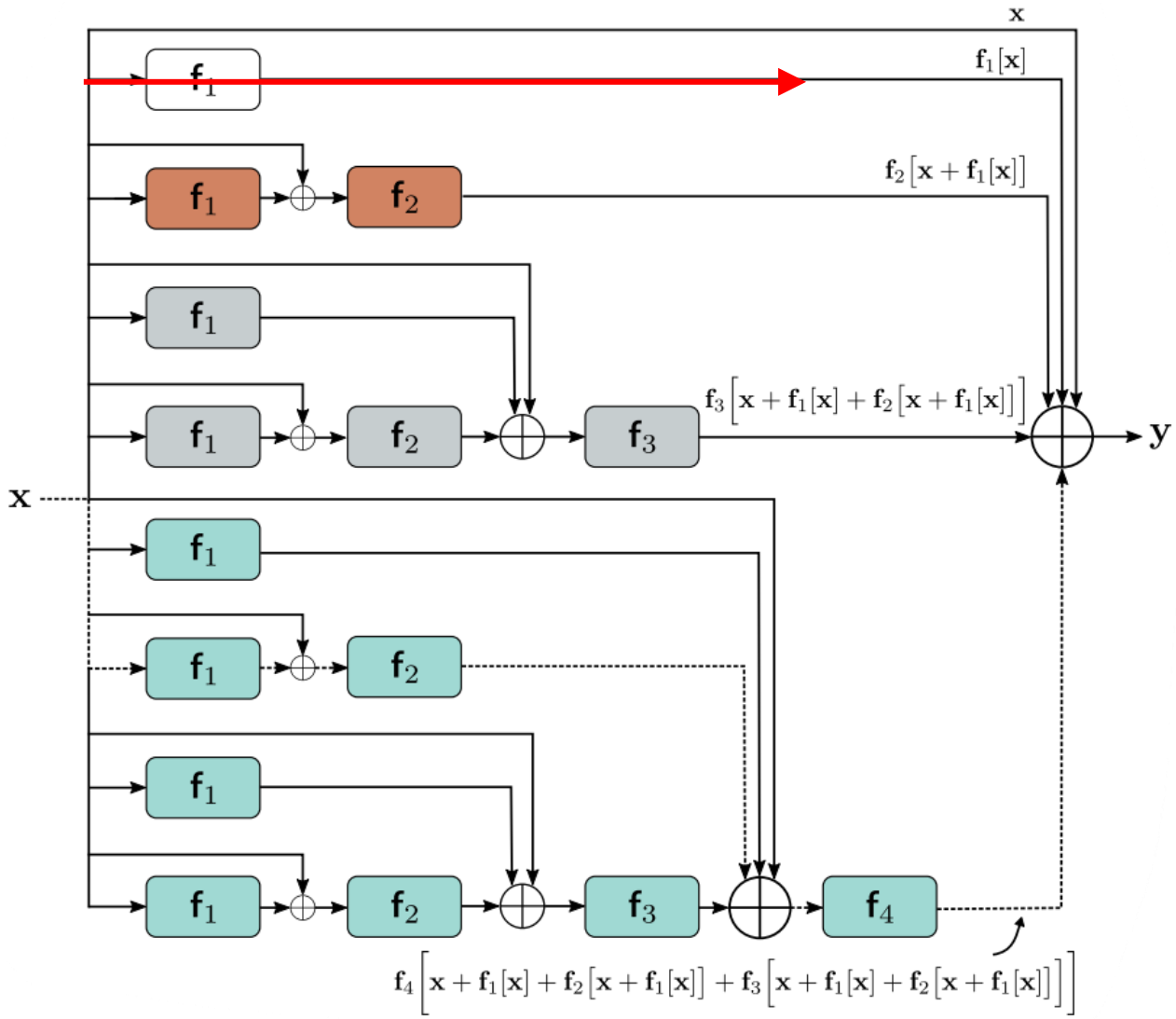
f_1 tem 8 caminhos

f_1 tem 8 caminhos



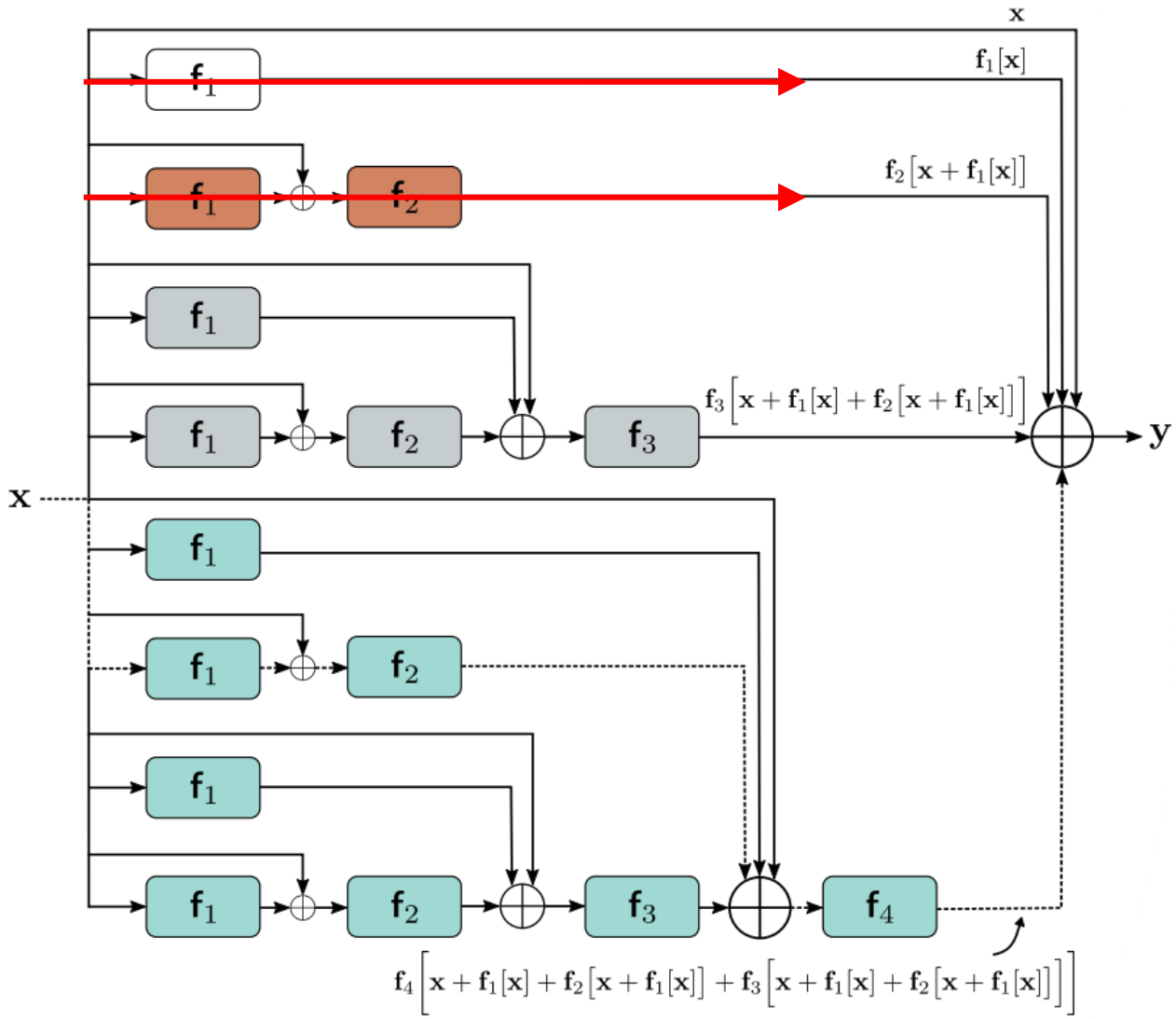
$$\frac{\partial y}{\partial f_1} = \frac{\partial f_1}{\partial f_1} + \cdots$$

f_1 tem 8 caminhos



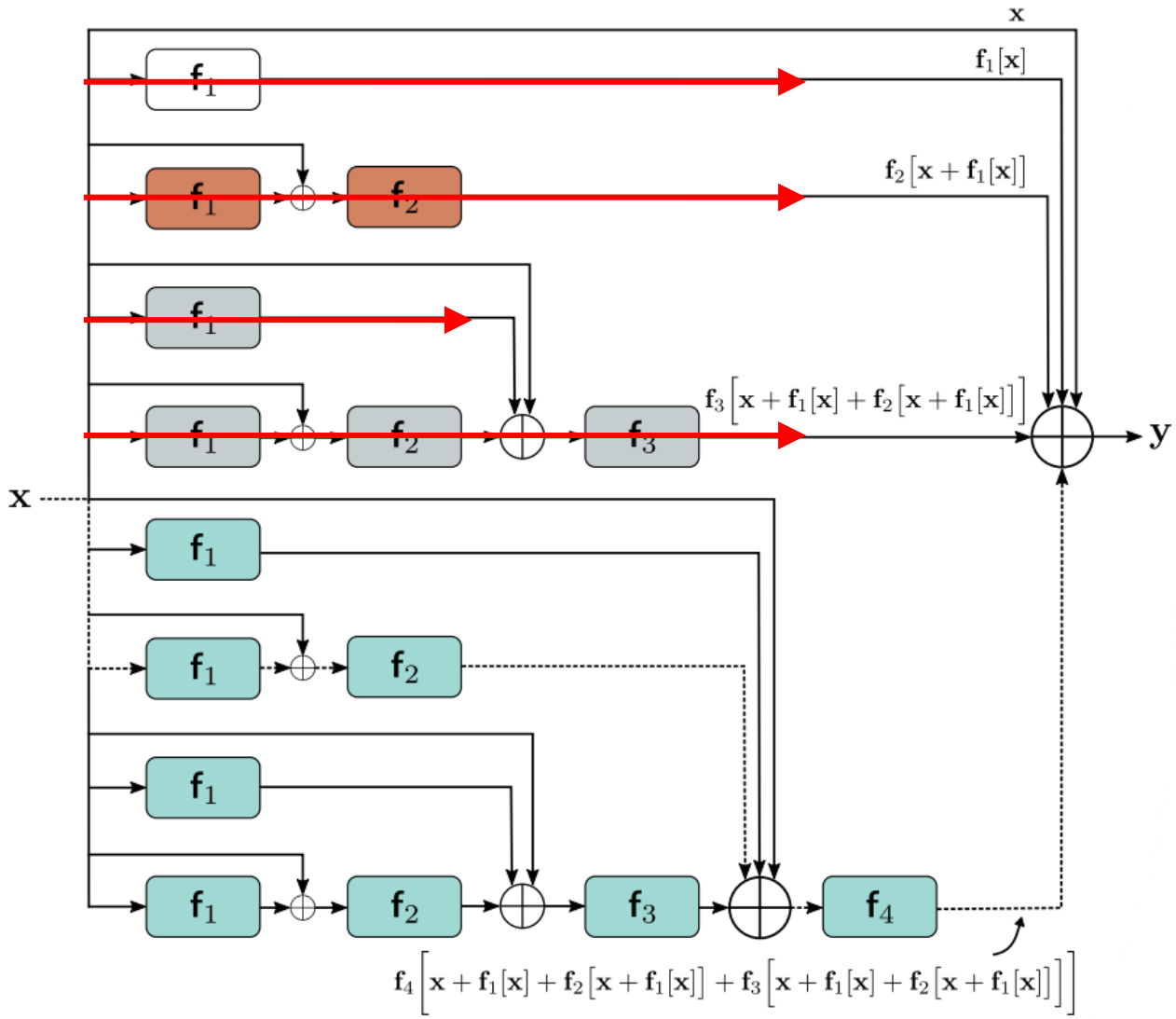
$$\frac{\partial y}{\partial f_1} = I + \cdots$$

f_1 tem 8 caminhos



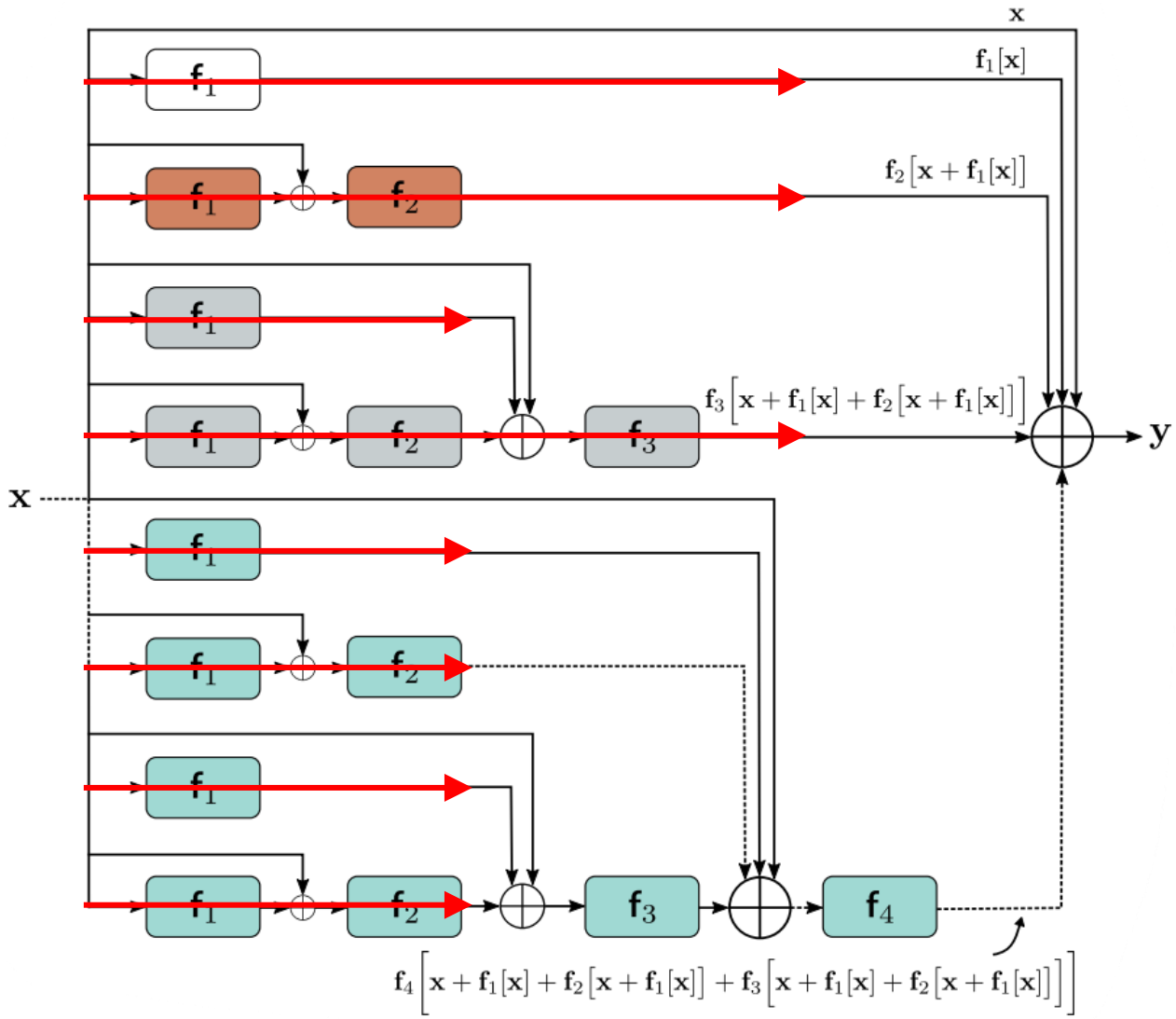
$$\frac{\partial y}{\partial f_1} = I + \frac{\partial f_2}{\partial f_1} + \cdots$$

f_1 tem 8 caminhos



$$\frac{\partial y}{\partial f_1} = I + \frac{\partial f_2}{\partial f_1} + \left(\frac{\partial f_3}{\partial f_1} + \frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\right) + \cdots$$

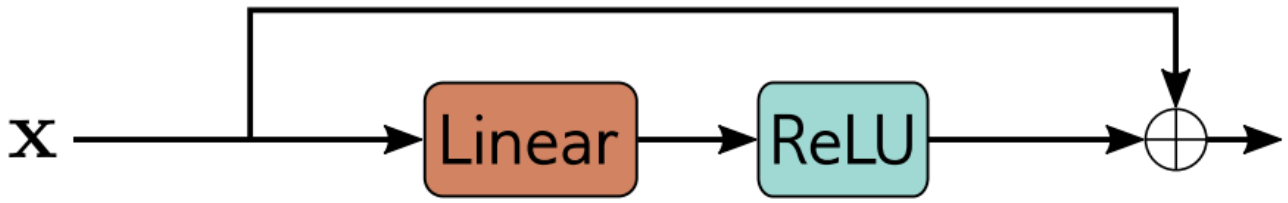
f_1 tem 8 caminhos



$$\frac{\partial y}{\partial f_1} = I + \frac{\partial f_2}{\partial f_1} + \left(\frac{\partial f_3}{\partial f_1} + \frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\right) + \left(\frac{\partial f_4}{\partial f_1} + \frac{\partial f_4}{\partial f_2}\frac{\partial f_2}{\partial f_1} + \frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial f_1} + \frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\right)$$

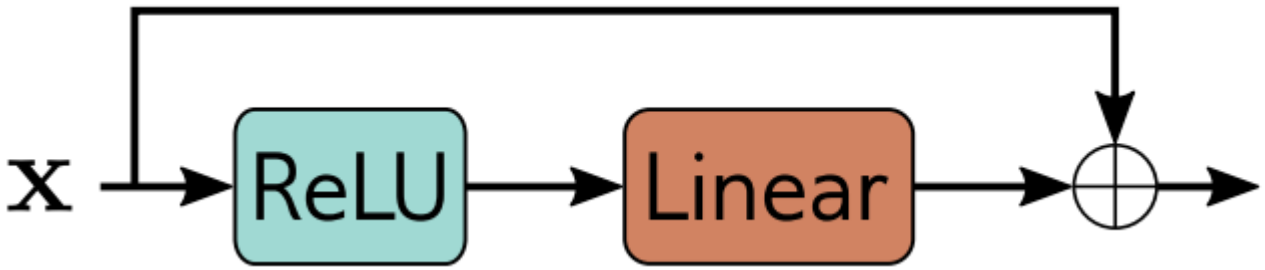
Bloco Residual

- Até agora parece que $f_i[x]$ poderia ser uma camada qualquer
  - Tecnicamente é verdade, mas na prática podemos fazer algumas mudanças para obter um resultado melhor
- Abaixo uma ilustração do que seria uma conexão residual em uma camada típica
  - Repare que nessa configuração a conexão residual só consegue aumentar o valor da representação recebida como entrada, nunca diminuir



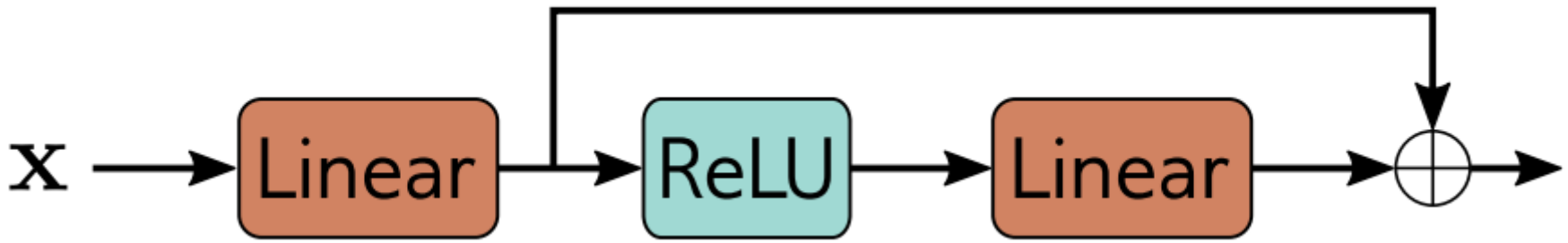
Bloco Residual

- Inverter a ordem da ativação e da transformação linear possibilita que a representação sofra adições e subtrações
  - Porém agora temos um outro problema, se a entrada chegar toda negativa vamos “matar” a propagação *forward*



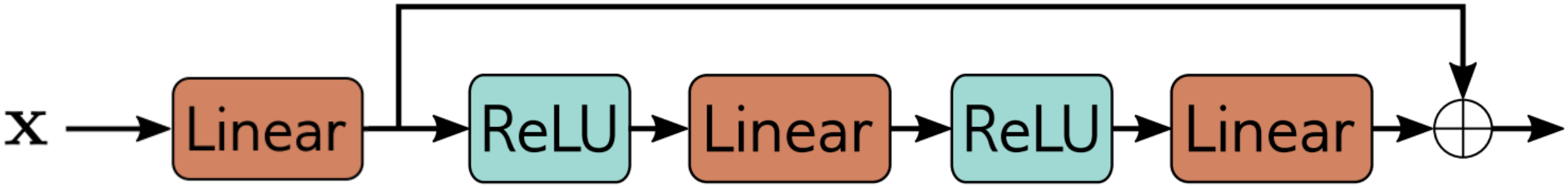
Bloco Residual

- Inverter a ordem da ativação e da transformação linear possibilita que a representação sofra adições e subtrações
  - Porém agora temos um outro problema, se a entrada chegar toda negativa vamos “matar” a propagação *forward*
  - Solução: Começamos a rede com uma transformação linear antes dos blocos residuais



Bloco Residual

- Podemos adicionar mais de uma transformação no mesmo bloco residual



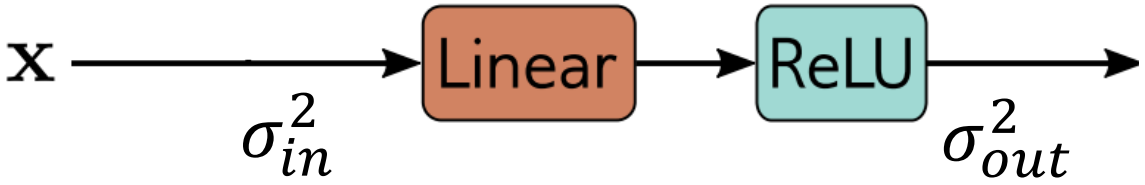
Aumentando a profundidade da rede

- Ao adicionar conexões residuais conseguimos dobrar a profundidade das redes convolucionais e ainda assim manter o aumento de performance
  - Mas ainda existem fenômenos que nos impedem de ir mais profundamente
  - Para compreender isso, precisamos estudar a variância das ativações em redes com *skip connections*

Variância das Ativações

- A inicialização He garante que a variância de uma variável aleatória após uma transformação linear + ReLU não será modificada

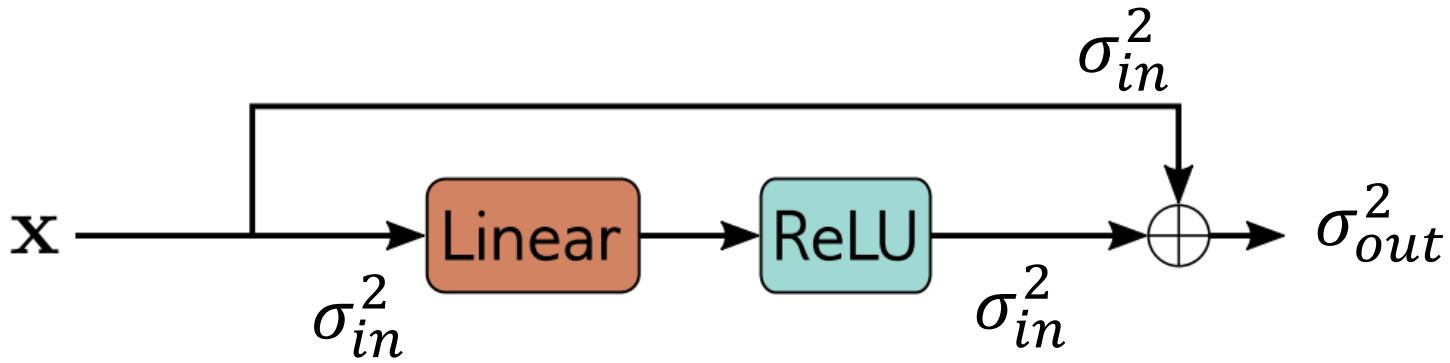
$$\sigma_{in}^2 = \sigma_{out}^2$$



Variância das Ativações

- A inicialização He garante que a variância de uma variável aleatória após uma transformação linear + ReLU não será modificada
  - Porém agora temos a *skip connection*
  - Como a variância da saída se comporta?

$$\sigma_{out}^2 = ?$$

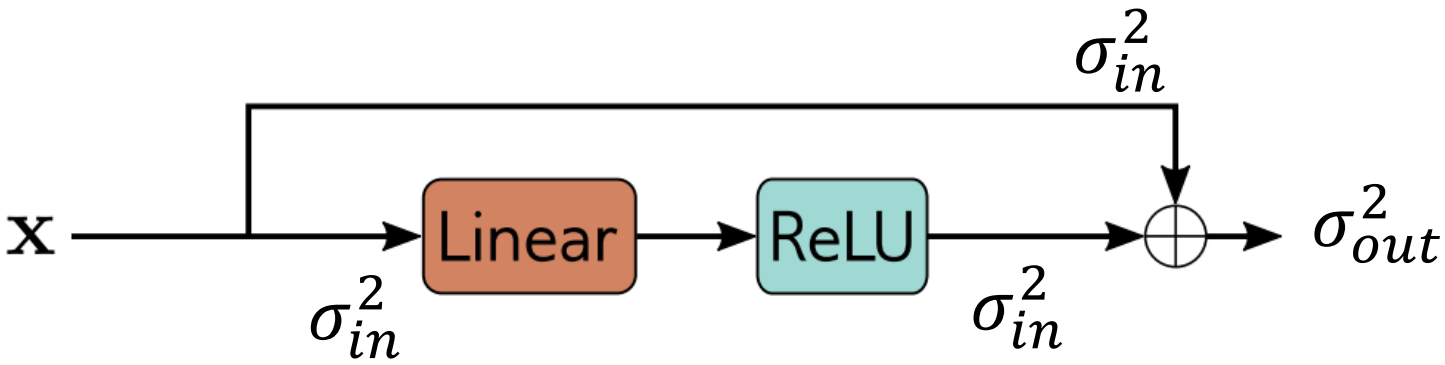


Variância das Ativações

- *Back to basics* ☺

$$Var[X] = E[(X - E[X])^2]$$

$$Var[X + Y] = E\left[\left((X + Y) - E[X + Y]\right)^2\right]$$



Variância das Ativações

- *Back to basics* ☺

$$Var[X] = E[(X - E[X])^2]$$

$$Var[X + Y] = E\left[\left((X + Y) - E[X + Y]\right)^2\right]$$

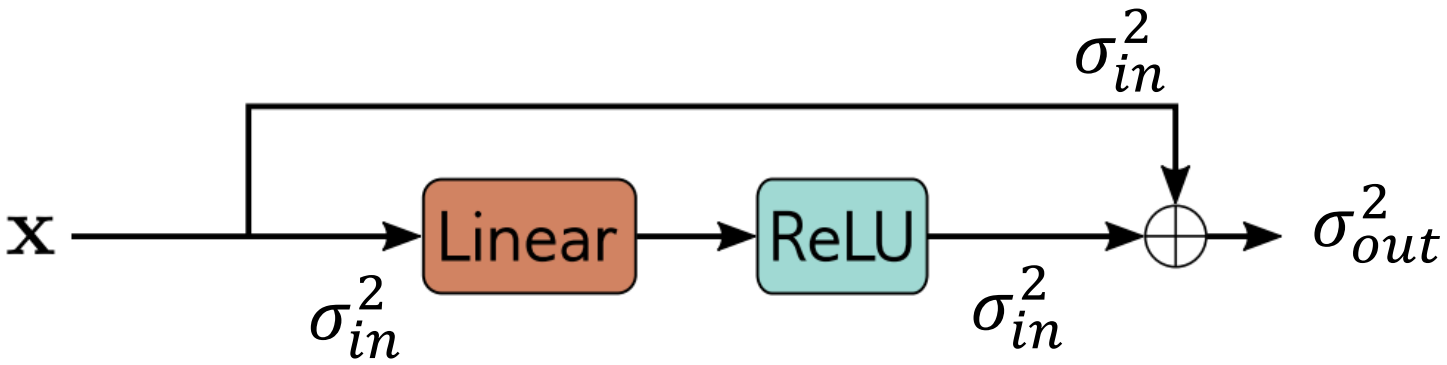
$$Var[X + Y] = E[(X + Y - E[X] - E[Y])^2]$$

$$Var[X + Y] = E\left[\left((X - E[X]) + (Y - E[Y])\right)^2\right]$$

$$Var[X + Y] = E[(X - E[X])^2 + 2(X - E[X])(Y - E[Y]) + (Y - E[Y])^2]$$

$$Var[X + Y] = E[(X - E[X])^2] + 2E[(X - E[X])(Y - E[Y])] + E[(Y - E[Y])^2]$$

$$Var[X + Y] = Var[X] + 2Cov(X, Y) + Var[Y]$$



Variância das Ativações

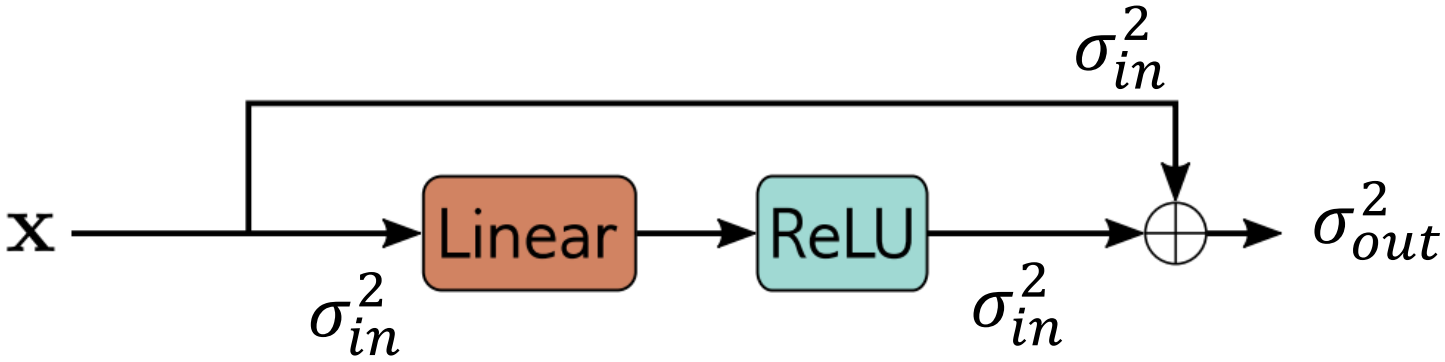
- *Back to basics* ☺

$$Var[X+Y] = Var[X] + \underbrace{2Cov(X,Y)}_{0} + Var[Y]$$

$$Var[X+Y] = Var[X] + Var[Y]$$

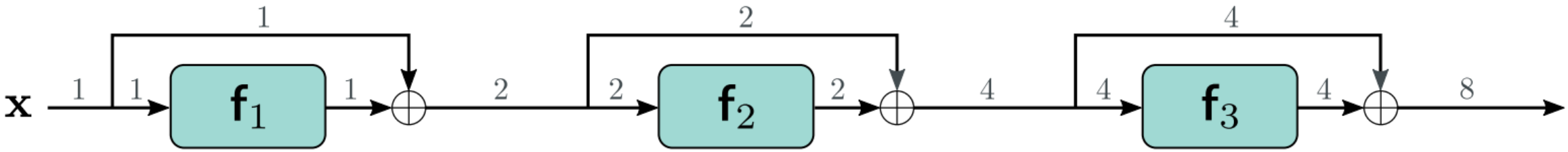
A variância vai dobrar depois de cada bloco residual

$$2\sigma_{in}^2 = \sigma_{out}^2$$



Variância das Ativações

- Em uma rede com muitas camadas rapidamente podemos exceder a precisão de ponto flutuante
  - Solução é usar *batch norm*



Batch Norm

- É uma normalização aplicada a ativações em camadas ocultas da rede
  - Desloca e reescala média e desvio padrão dos batchs para valores aprendidos durante o treinamento
- Necessita que utilizemos *batch size > 1*

Batch Norm

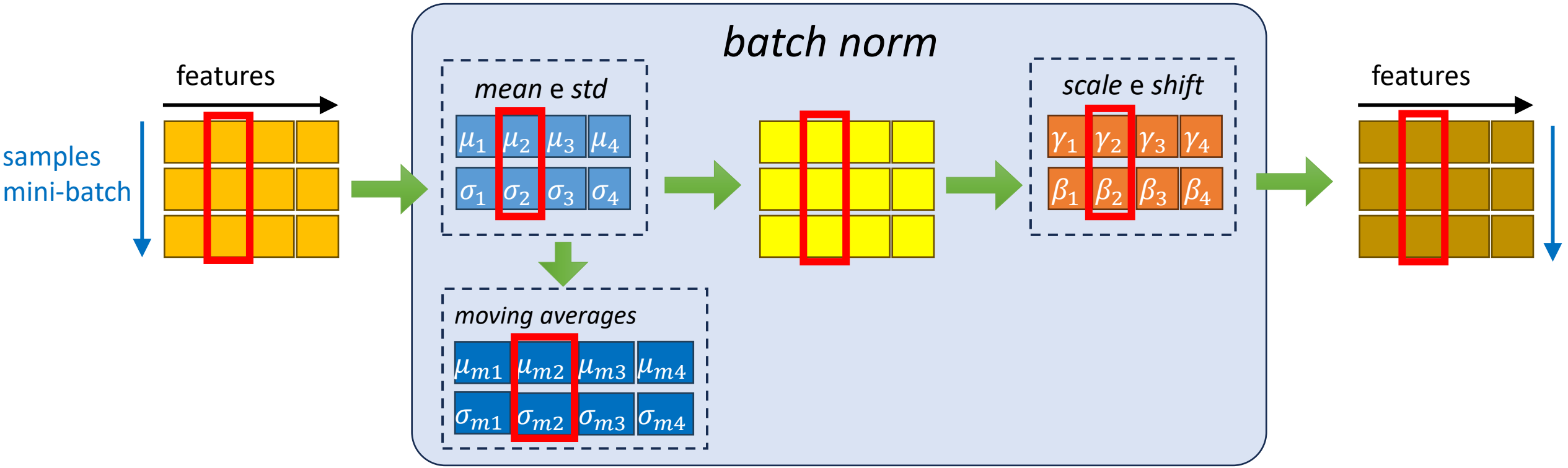
- Como funciona:
- Computar a média e o desvio padrão empíricos do *mini batch* atual
  - $\mu_h = \frac{1}{|B|}\sum_{i \in B} h_i$
  - $\sigma_h = \sqrt{\frac{1}{|B|}\sum_{i \in B}(h_i - \mu_h)^2}$
  - Repare que todas as quantidades aqui são escalares

Batch Norm

- Como funciona:
- Padronizamos as ativações para ter media 0 e desvio padrão igual a 1
  - $h_i \leftarrow \frac{h_i - \mu_h}{\sigma_h + \epsilon}, \forall i \in B$
- Por último reescalamos as ativações por γ e deslocamos por β
  - $h_i \leftarrow \gamma h_i + \beta, \forall i \in B$

Batch Norm

- Existe um γ, um β, um μ e um σ por unidade oculta em uma camada densa
  - 4 parâmetros por unidade oculta, dois deles são treináveis



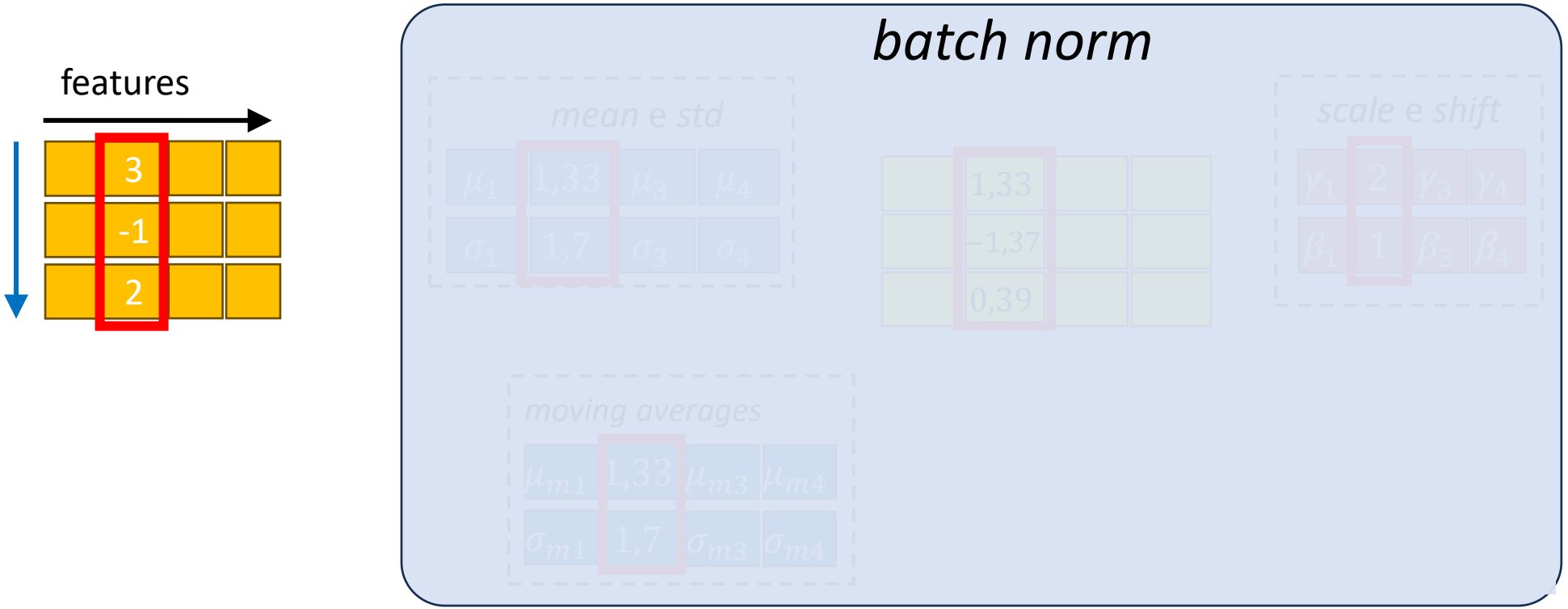
Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Batch Norm

- Como funciona:
- Importante:
  - existe um γ, um β, um μ e um σ por unidade oculta em uma camada densa
    - 4 parâmetros por unidade oculta
  - existe um γ, um β, um μ e um σ por canal em uma camada convolucional
  - Apenas γ e δ são parâmetros aprendidos, μ e σ são calculados
    - γ e δ são inicializados com 1 e 0 respectivamente

Batch Norm - Exemplo

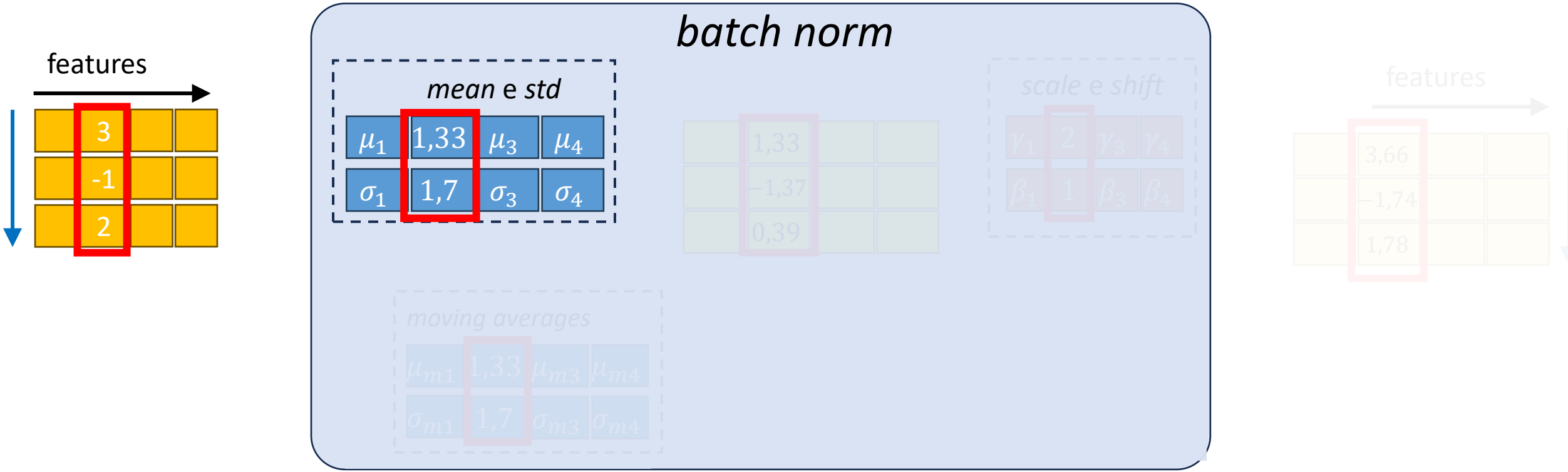
- Vamos começar com um *mini batch*



Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Batch Norm - Exemplo

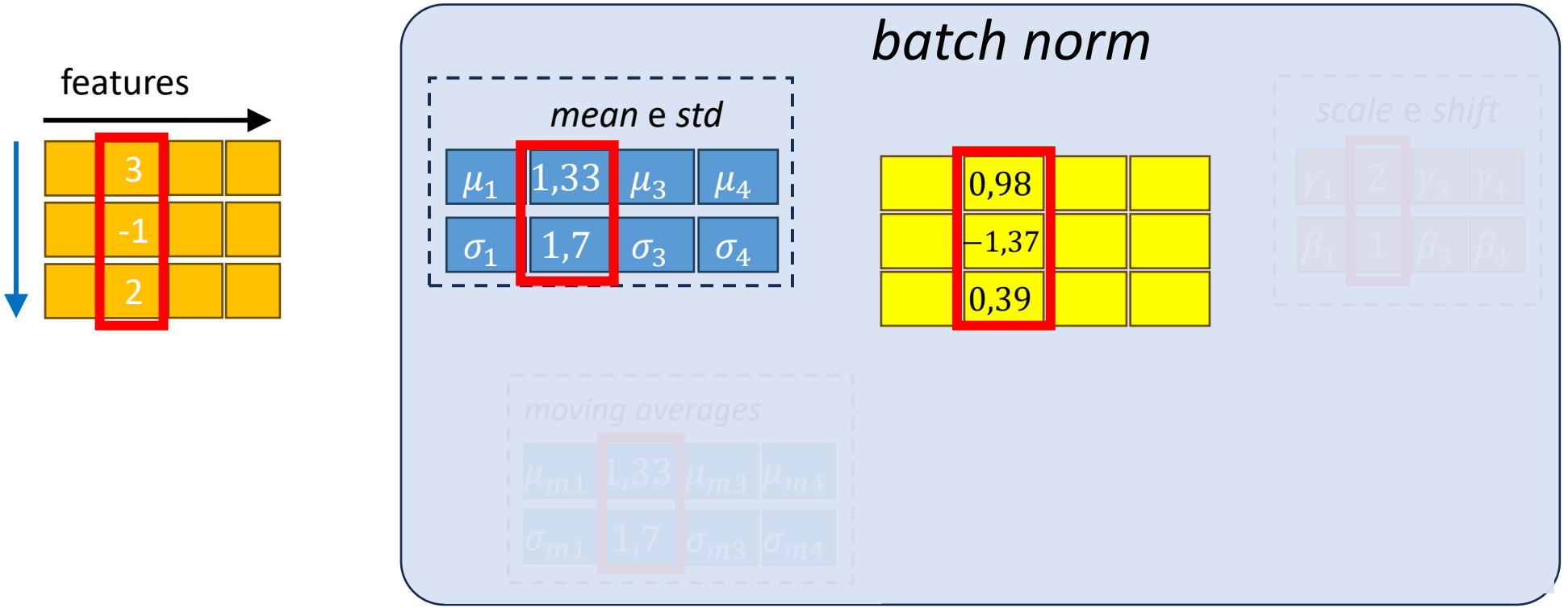
- Vamos começar com um *mini batch*
  - A media μ *e o* desvio padrão de cada *feature* do *mini batch* σ



Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Batch Norm - Exemplo

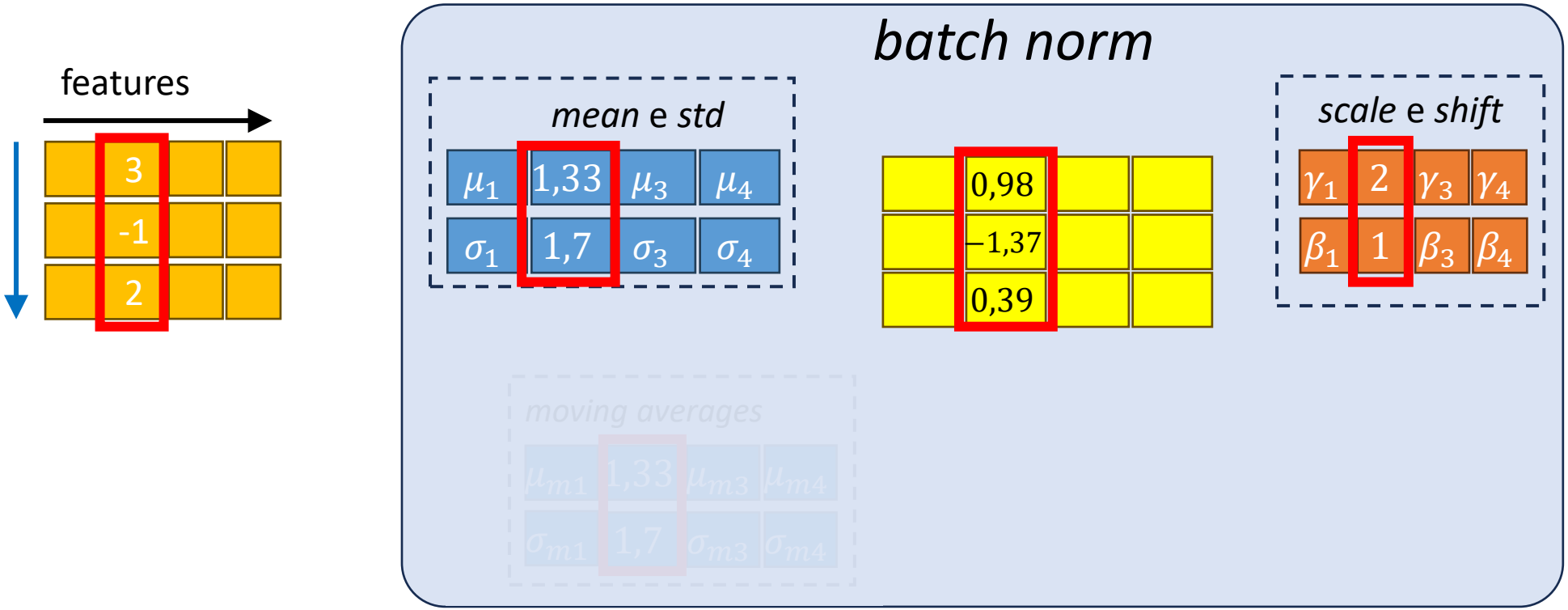
- Vamos começar com um *mini batch*
  - μ e σ são usados para padronizar os dados do *minibatch* $h = \frac{h-\mu}{\sigma+\epsilon}$



Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Batch Norm - Exemplo

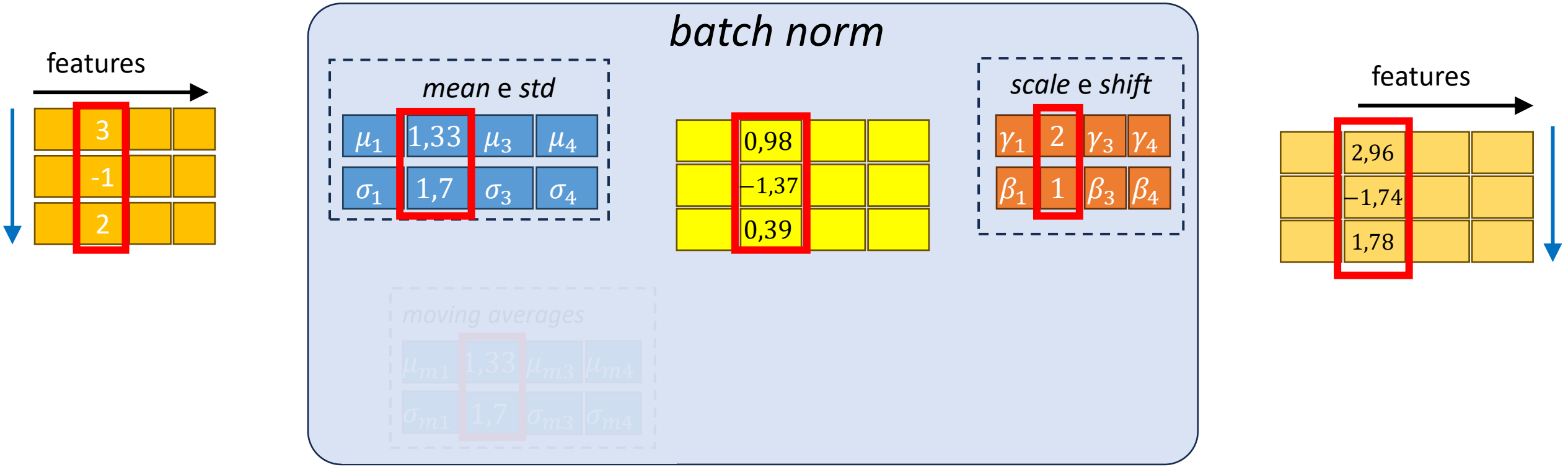
- Existem dois parâmetros treináveis na camada de *batch norm* que são:
  - a nova média β e o novo desvio padrão γ



Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Batch Norm - Exemplo

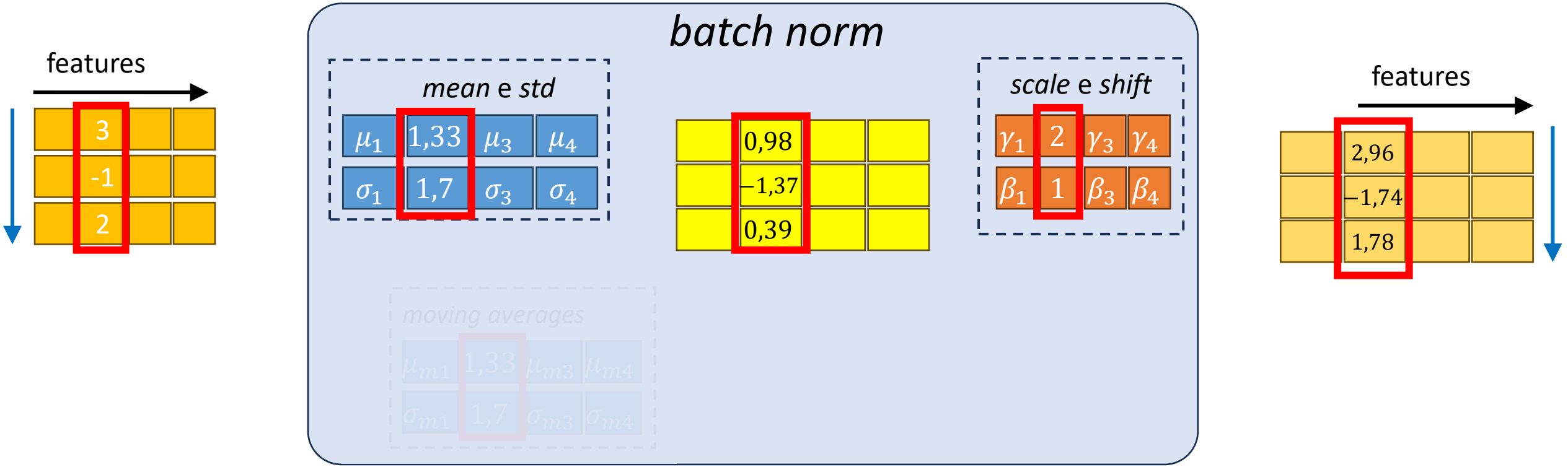
- Os dados padronizados são atualizados conforme a media e desvio padrão aprendidos $h = h\gamma + \beta$



Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Batch Norm - Exemplo

- Isso funciona bem em treinamento, quando temos *mini batchs*
  - Em inferência (sem *mini batchs*), precisamos ter uma média e um desvio padrão para atualizar os dados

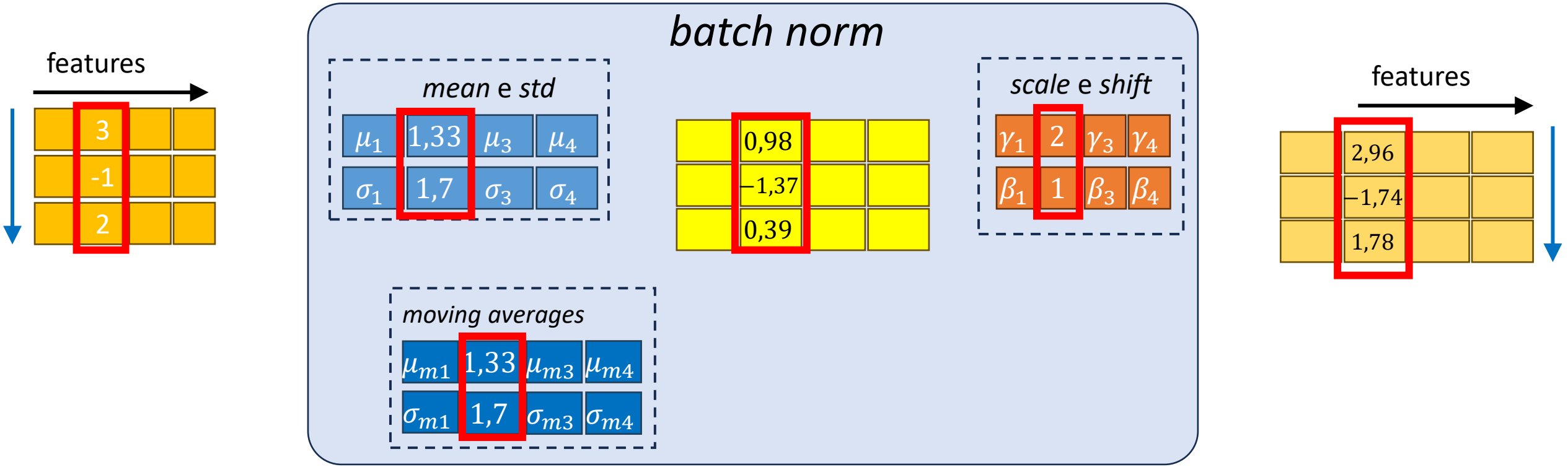


Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Batch Norm - Exemplo

- Mantemos uma média móvel das médias e dos desvios padrão calculados durante o treino
  - Mais precisamente seria uma média exponencial móvel (EMA)

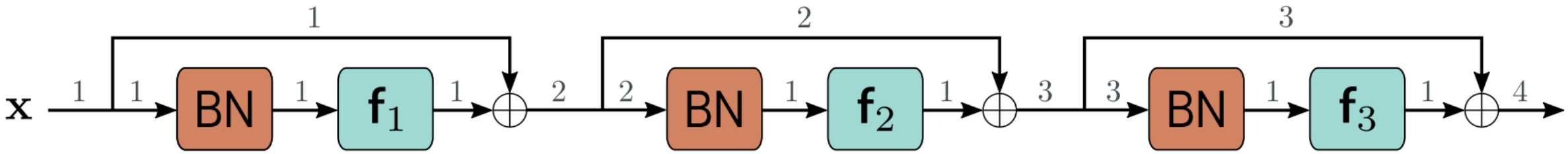
$$\mu_{mov} = \alpha\mu_{mov} + (1-\alpha)\mu \qquad \text{e} \qquad \sigma_{mov} = \alpha\sigma_{mov} + (1-\alpha)\sigma$$



Fonte imagem: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

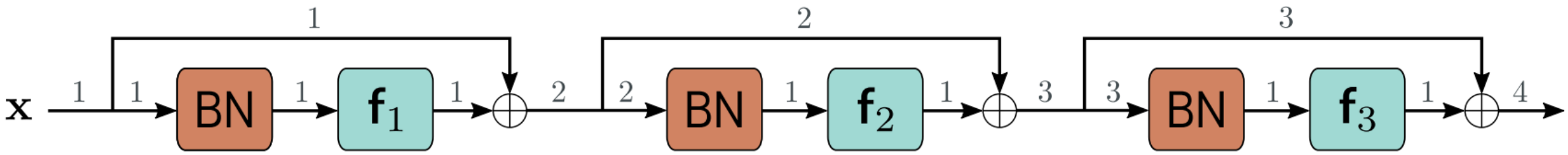
Batch Norm

- Com *Batch Norm* ativo, conseguimos treinar redes muito mais profundas sem explosão nem dissipação de gradiente



Batch Norm

- Torna a rede invariante a mudanças de escala nos parâmetros
  - Se os parâmetros dobrarem de magnitude as ativações também dobram e a variância também dobra (a primeira etapa do batch norm compensa esse aumento)
- Forward mais estável
- Taxas de Aprendizado (η) mais elevadas
- Regulariza o processo de treinamento

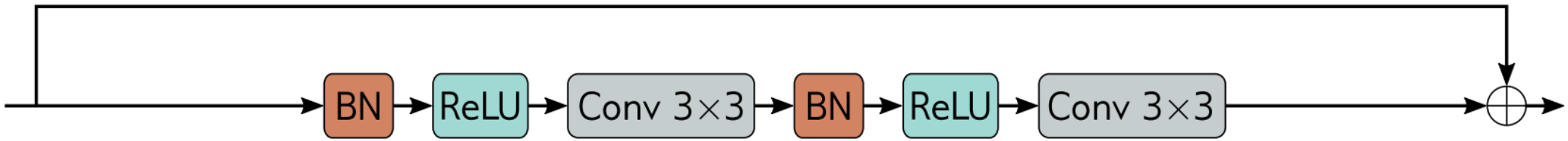


Skip Connections + Batch Norm

- Usando *skip connections* e *batch norm* temos redes convolucionais com aplicações em praticamente todas as áreas de visão computacional
  - Permitem treinamento de redes com ~1000 camadas ocultas
- Ainda hoje são o padrão para a maioria das atividades de visão computacional, apesar do recente sucesso de *transformers*
- Vamos as arquiteturas

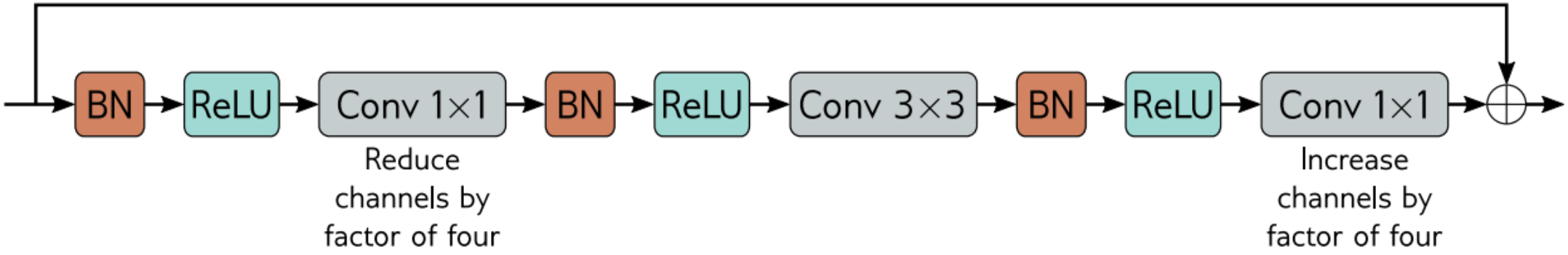
ResNet

- Redes convolucionais constituídas de blocos residuais
  - Cada bloco foi projetado por “tentativa e erro”
  - Apresenta resultados bons, mas tem um elevado número de parâmetros



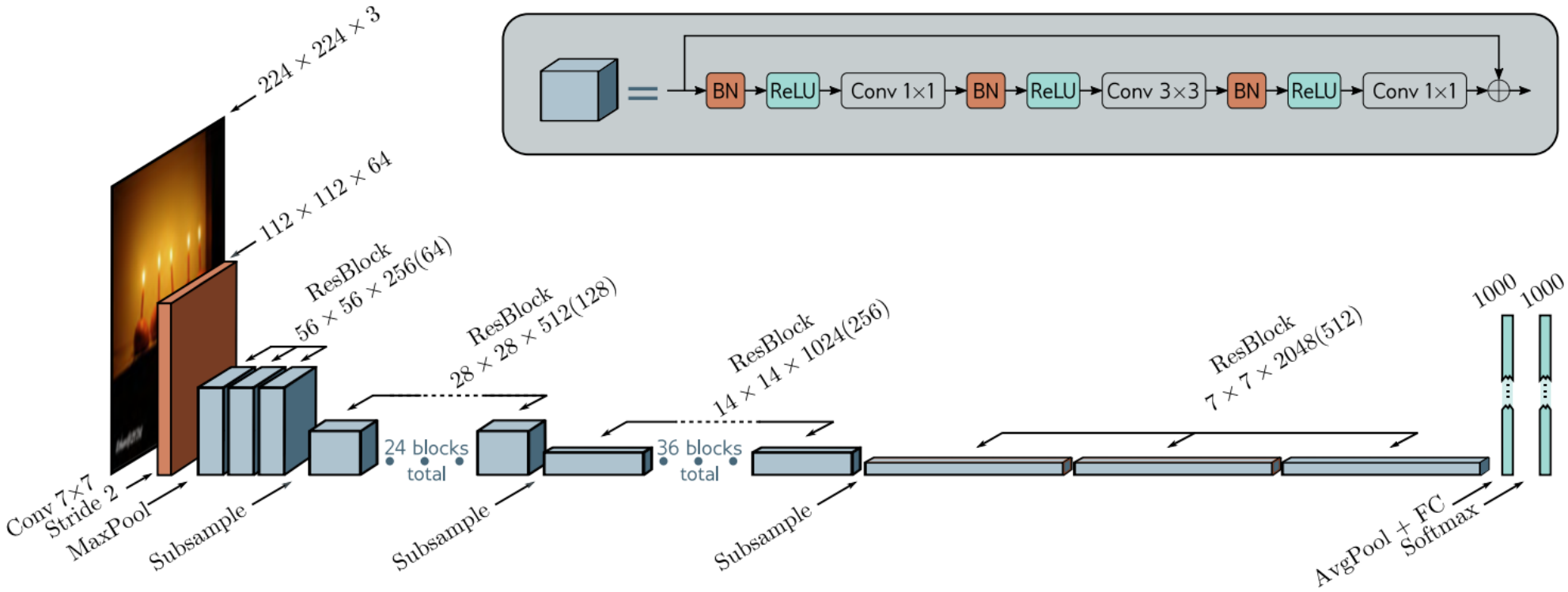
ResNet

- Bloco residual com gargalo
  - *Bottleneck Residual Block*
- Foi introduzido para diminuir o número de parâmetros treináveis
  - A primeira conv 1x1 reduz o número de canais
  - A segunda conv 1x1 aumenta o número de canais para possibilitar a adição no final do bloco



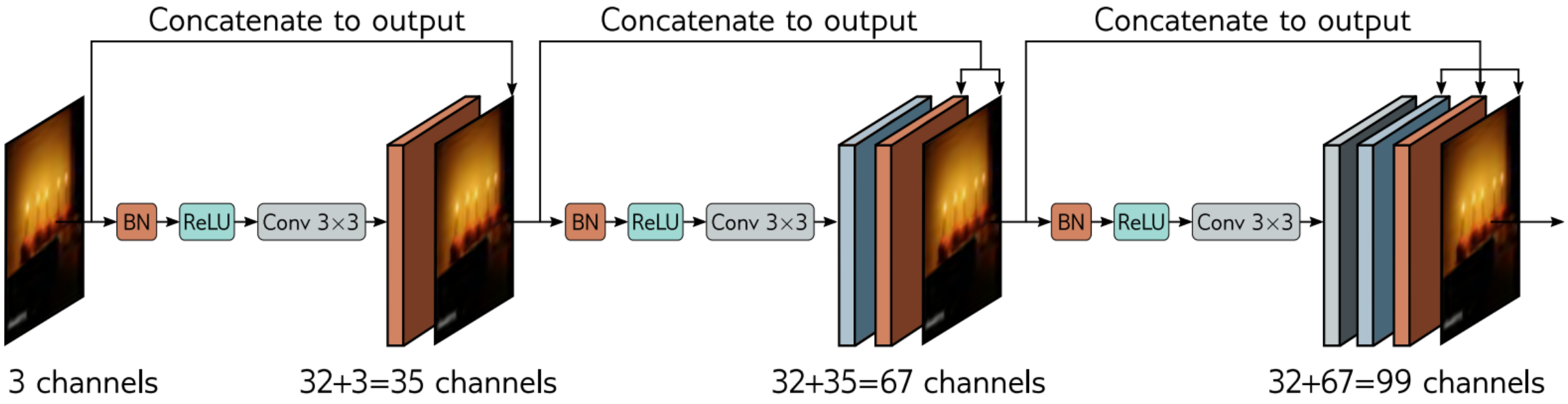
ResNet-200

- 4.8% de erro no ImageNet
- Números entre parênteses representam a quantidade de canais depois da conv 1x1
- Utiliza o bloco residual com gargalo



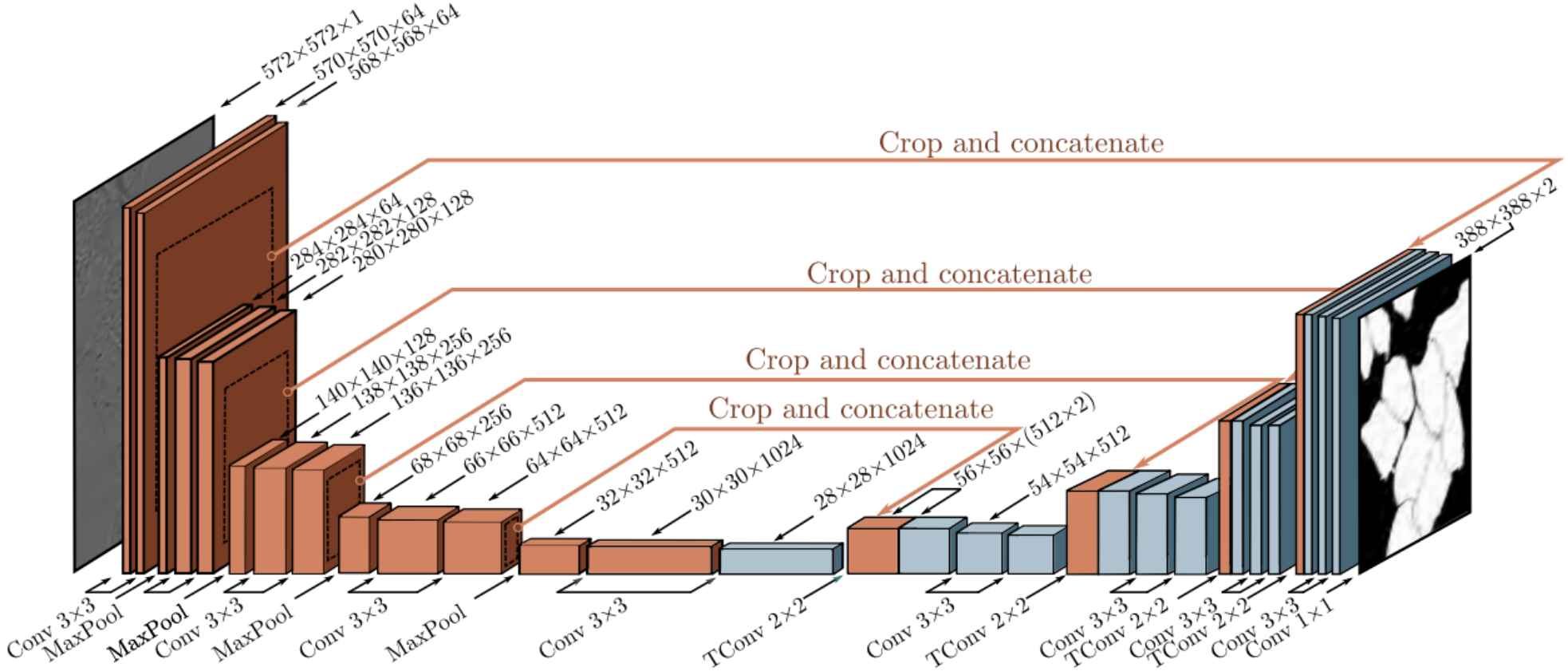
DenseNet

- Ao invés de adicionar a representação da entrada com a saída podemos fazer a concatenação dos canais de entrada aos canais de saída
  - Menos comum que ResNet mas atinge resultados similares
  - Quando ocorre *downsampling* a representação não é concatenada



U-Net

- Arquitetura *encoder-decoder* com *skip connections*
  - Como a convolução foi feita sem *padding* foi necessário cortar a imagem original antes de concatenar



Referências:

- Sugere-se ***fortemente*** a leitura de:
  - Capítulo 11 de Understanding Deep Learning
    - https://udlbook.github.io/udlbook/
- Sugere-se também ler o artigo abaixo sobre conexões residuais:
  - HE, Kaiming et al. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 770-778.