Redes Profundas

Prof. Danilo Silva

EEL7514/EEL7513 - Tópico Avançado em Processamento de Sinais
EEL410250 - Aprendizado de Máquina

EEL / CTC / UFSC

Tópicos

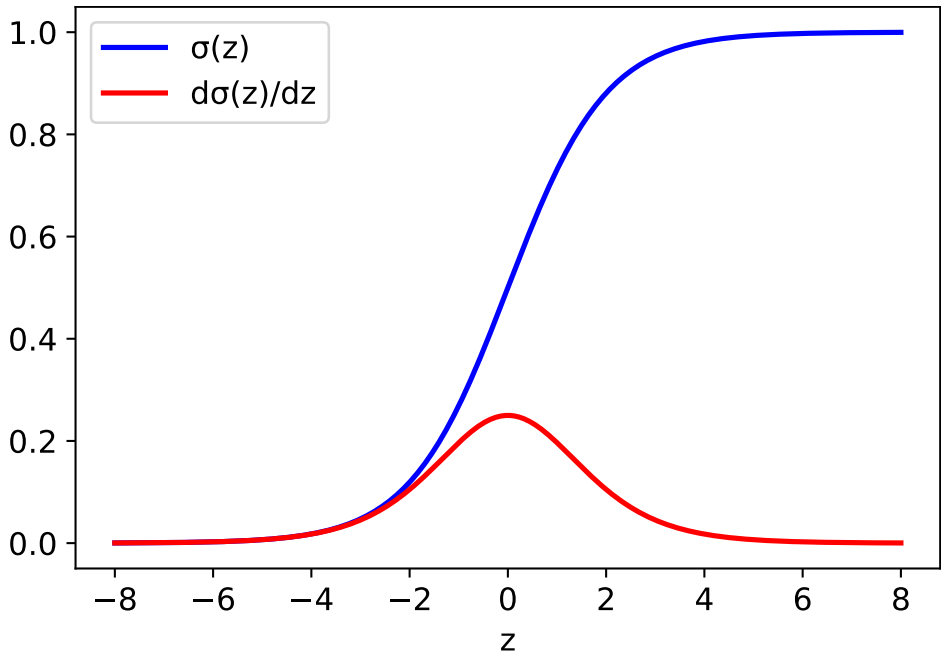
- Funções de ativação
- Inicialização de pesos
- Método do gradiente estocástico
- Variações do método do gradiente estocástico
- Outras técnicas de regularização
  - Early stopping
  - Dropout
  - Batch normalization
  - Data augmentation

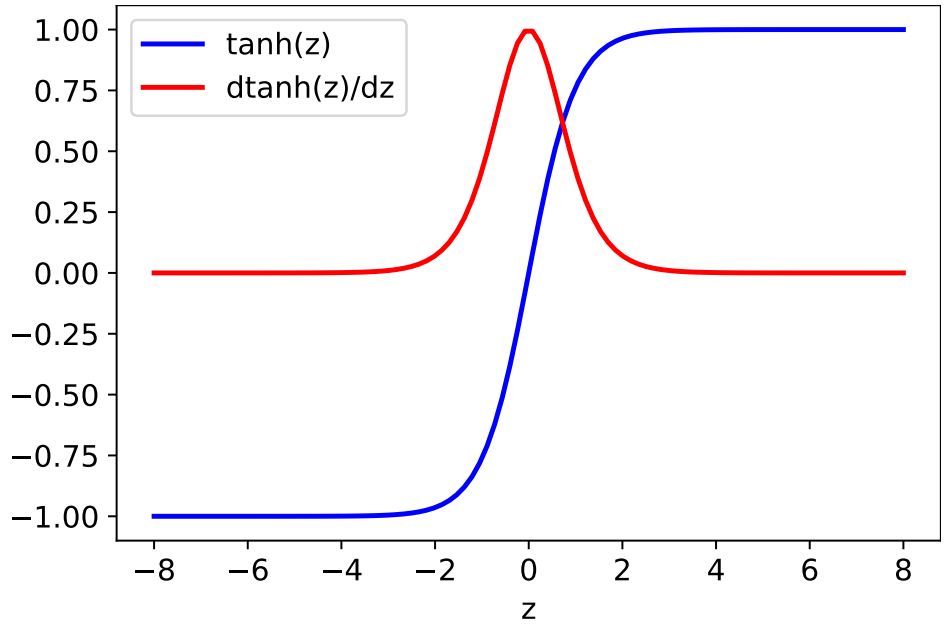
Introdução

- Por que usar redes profundas?
  - Complexidade $\times$ poder expressivo: Uma rede profunda é capaz de implementar a mesma função que uma rede de 1 camada oculta usando um número muito menor de unidades
  - Inspiração biológica:
    - O cérebro possui uma arquitetura profunda
    - Seres humanos costumam representar conceitos em níveis hierárquicos
- Desafio: Treinamento
  - Camadas inferiores (iniciais) tem um aprendizado mais lento
    - *Problem of vanishing (or exploding) gradients*
  - Aumento do poder expressivo exige:
    - Elevado poder computacional
    - Grande volume de dados

Funções de Ativação

Sigmóide Logística e Tangente Hiperbólica





- A função sigmóide logística $\sigma(z)$ satura para uma entrada $|z| \gg 0$, i.e., $g'(z) \approx 0$, tornando o aprendizado lento
- O mesmo problema ocorre para a tangente hiperbólica

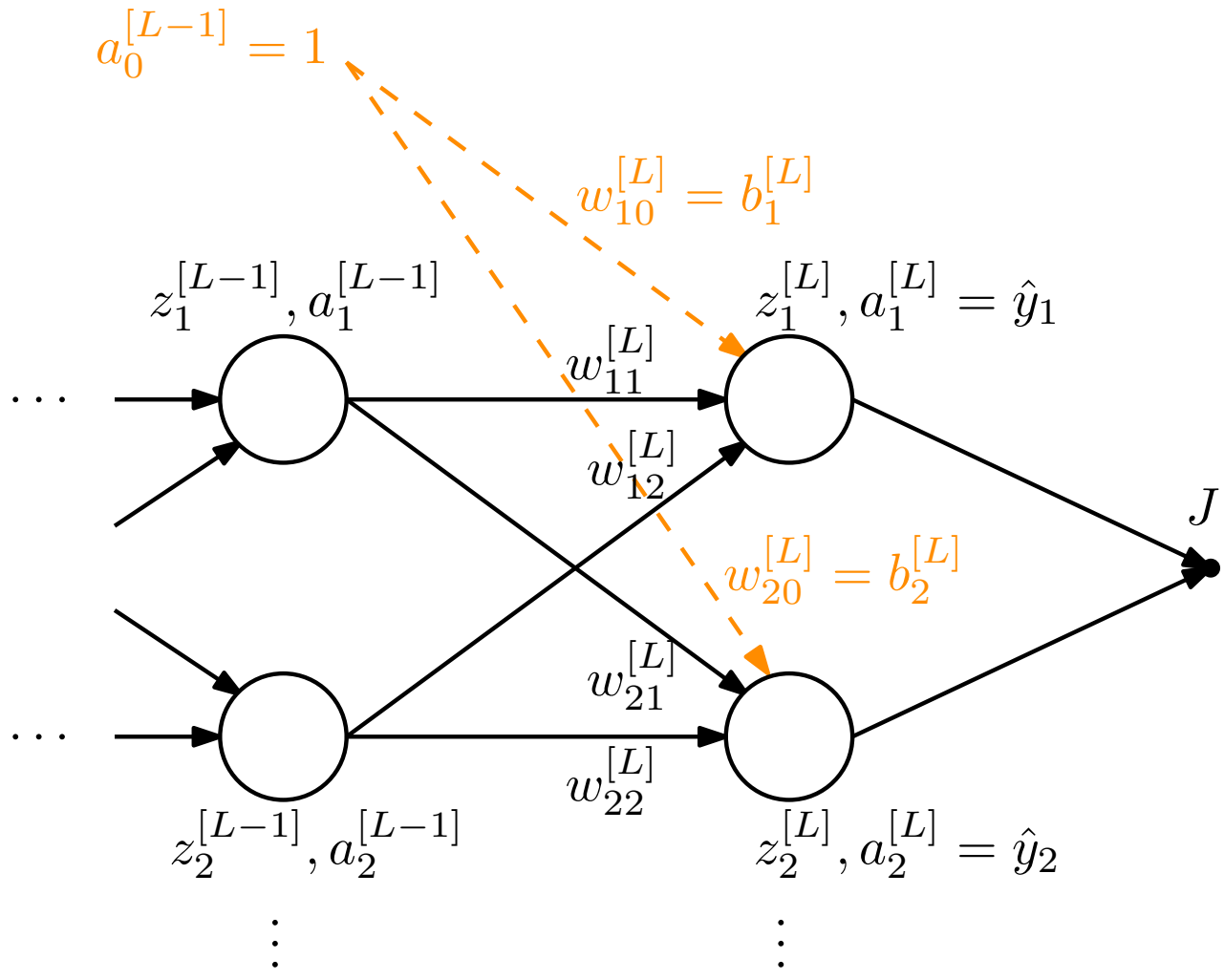
Vanishing Gradient Problem

- Propagações direta e reversa:

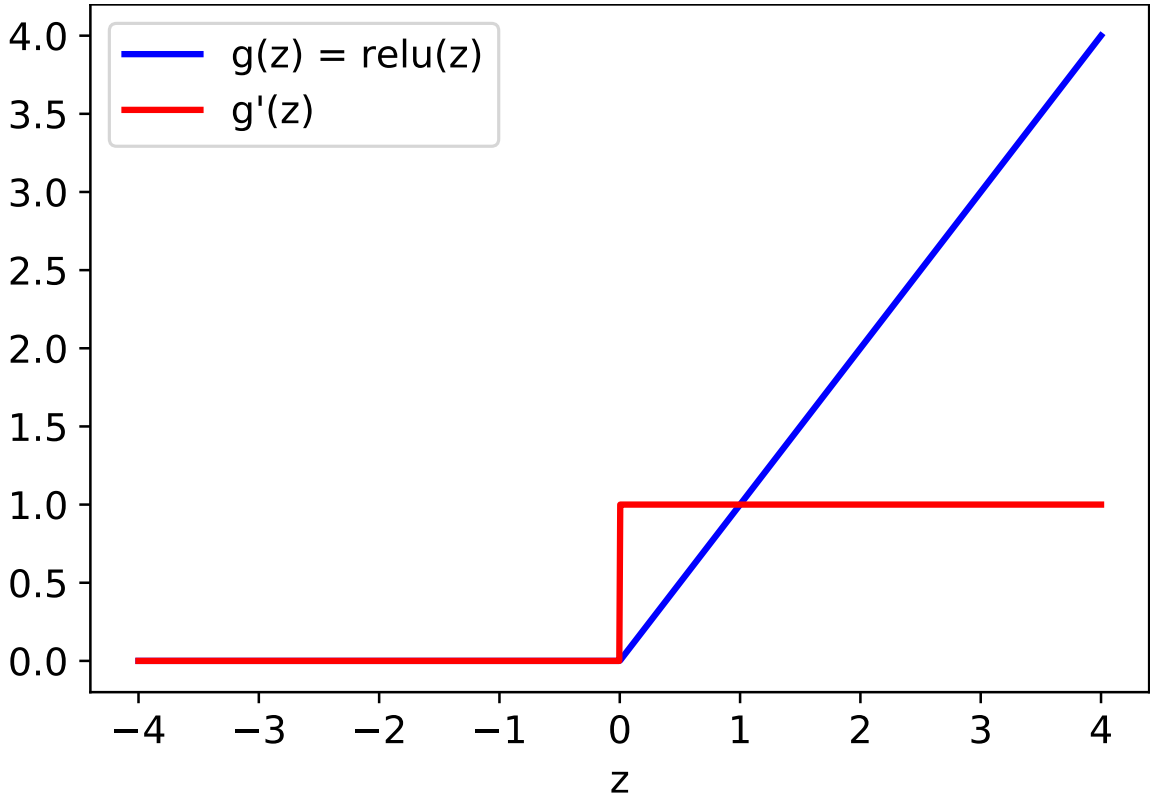
$$z_k^{[\ell]} = \sum_{j=0}^{n_{\ell-1}} w_{kj}^{[\ell]} a_j^{[\ell-1]}, \qquad \delta_j^{[\ell]} = g'_\ell(z_j^{[\ell]}) \sum_{k=1}^{n_{\ell+1}} w_{kj}^{[\ell+1]} \delta_k^{[\ell+1]}$$

- Uma forma de evitar que $|z_k^{[\ell]}| \gg 0$ seria fazer $w_{kj}^{[\ell]} \approx 0$, mas isso por sua vez provoca $\delta_j^{[\ell-1]} \approx 0$
  - Além disso, pesos muito pequenos podem não ser uma boa solução
- Conclusão: gradiente decai rapidamente de $\ell = L$ até $\ell = 1$
  - Difícil de treinar mais do que 1–2 camadas ocultas com $g(z) = \sigma(z)$
  - Argumento semelhante se aplica a $g(z) = \tanh(z)$

Redes Neurais: Notação



ReLU (*rectified linear unit*)



- Uma forma de amenizar esse problema é utilizar a função de ativação:

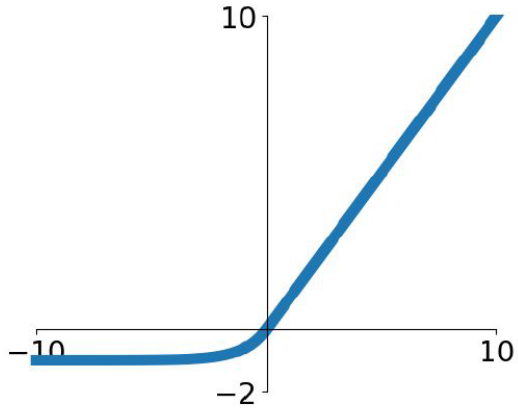
$$\text{relu}(z) = \max(0, z)$$

  - Permite treinar redes profundas (3+ camadas ocultas) mais facilmente

ReLU (*rectified linear unit*)

- Uma desvantagem da ReLU é sua saturação para $z < 0$
- Em particular, algumas unidades podem "morrer" durante o treinamento se em algum momento seus pesos forem tais que nenhuma amostra consegue ativar a unidade
- Motiva o uso de outras funções de ativação como a ELU (*exponential linear unit*)
  - No entanto, os benefícios são relativamente pequenos

$$g(z) = \begin{cases} z, & z > 0 \\ \alpha(e^z - 1), & x < 0 \end{cases}$$



- Outras funções de ativação: Leaky ReLU, Parametric ReLU, SELU, Swish, Maxout, ...

Inicialização de Pesos

Inicialização de pesos

[Yann LeCun *et al.* 1998]

- Seja $\mathbf{W}$ uma matriz $n_{\text{out}} \times n_{\text{in}}$. Considere a multiplicação matricial

$$\mathbf{z} = \mathbf{W}\mathbf{a} \quad \Longleftrightarrow \quad z_k = \sum_{j=1}^{n_{\text{in}}} w_{kj} a_j, \quad k = 1, \ldots, n_{\text{out}}$$

- Assumindo que w_{kj} e a_j são i.i.d. com $E[w] = 0$, $\text{VAR}[w] = \sigma_w^2$, pode-se mostrar que:

$$E[z_k] = 0, \quad \text{VAR}[z_k] = n_{\text{in}} \sigma_w^2 E[a^2] = n_{\text{in}} \sigma_w^2 (E[a]^2 + \text{VAR}[a])$$

- Para uma ativação linear, a variância é preservada se e somente se

$$\sigma_w^2 = \frac{1}{n_{\text{in}}}$$

- Inicialização recomendada: $\mathcal{N}(0, \sigma_w^2)$ ou $U[-a, a]$ onde $a = \sqrt{3}\sigma_w$

Inicialização de pesos

[Xavier Glorot *et al.* 2010]

- Propagação reversa:

$$\delta_j^{[\ell]} = g'_\ell(z_j^{[\ell]}) \sum_{k=1}^{n_{\ell+1}} w_{kj}^{[\ell+1]} \delta_k^{[\ell+1]}$$

- Para uma ativação linear, a variância é preservada se e somente se

$$\sigma_w^2 = \frac{1}{n_{\text{out}}}$$

- Uma forma de balancear os dois objetivos é escolher

$$\sigma_w^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

- Inicialização recomendada: $\mathcal{N}(0, \sigma_w^2)$ ou $U[-a, a]$ onde $a = \sqrt{3}\sigma_w$

Inicialização de pesos

[Kaiming He *et al.* 2015]

- No caso específico da função de ativação ReLU, a análise linear da propagação direta é exata exceto pelos seguintes fatos:
  - Em média, 50% das ativações serão nulas
  - $E[a] \neq 0$, uma vez que $a = \mathrm{relu}(z) \geq 0$
- Pode-se mostrar que

$$E[a]^2 + \mathrm{VAR}[a] = E[a^2] = \mathrm{VAR}[z]/2$$

- Portanto, a variância é preservada se e somente se

$$\sigma_w^2 = \frac{2}{n_{\text{in}}} \text{ (na prop. direta)}, \qquad \sigma_w^2 = \frac{2}{n_{\text{out}}} \text{ (na prop. reversa)}$$

- Qualquer dos dois critérios é suficiente
- Inicialização recomendada: $\mathcal{N}(0, \sigma_w^2)$ ou $U[-a, a]$ onde $a = \sqrt{3}\sigma_w$

Método do Gradiente Estocástico (SGD)

Método do Gradiente

- Função custo e gradiente:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} J^{(i)}(\boldsymbol{\theta}), \quad J^{(i)}(\boldsymbol{\theta}) = L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}), \quad \hat{\mathbf{y}}^{(i)} = f(\mathbf{x}^{(i)}|\boldsymbol{\theta})$$

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla J^{(i)}(\boldsymbol{\theta})$$

- Cada iteração do método do gradiente requer o cálculo de $\nabla J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \nabla J(\boldsymbol{\theta}_{t-1})$$

  o qual por sua vez depende de todo o conjunto de treinamento

- Se m é muito grande, o número total de operações até a convergência pode ser muito elevado

Método do Gradiente Estocástico (SGD)

- Uma solução é aproximar $\nabla J(\boldsymbol{\theta})$ por $\nabla J^{(i)}(\boldsymbol{\theta})$, realizando a atualização dos pesos a cada novo exemplo de treinamento:

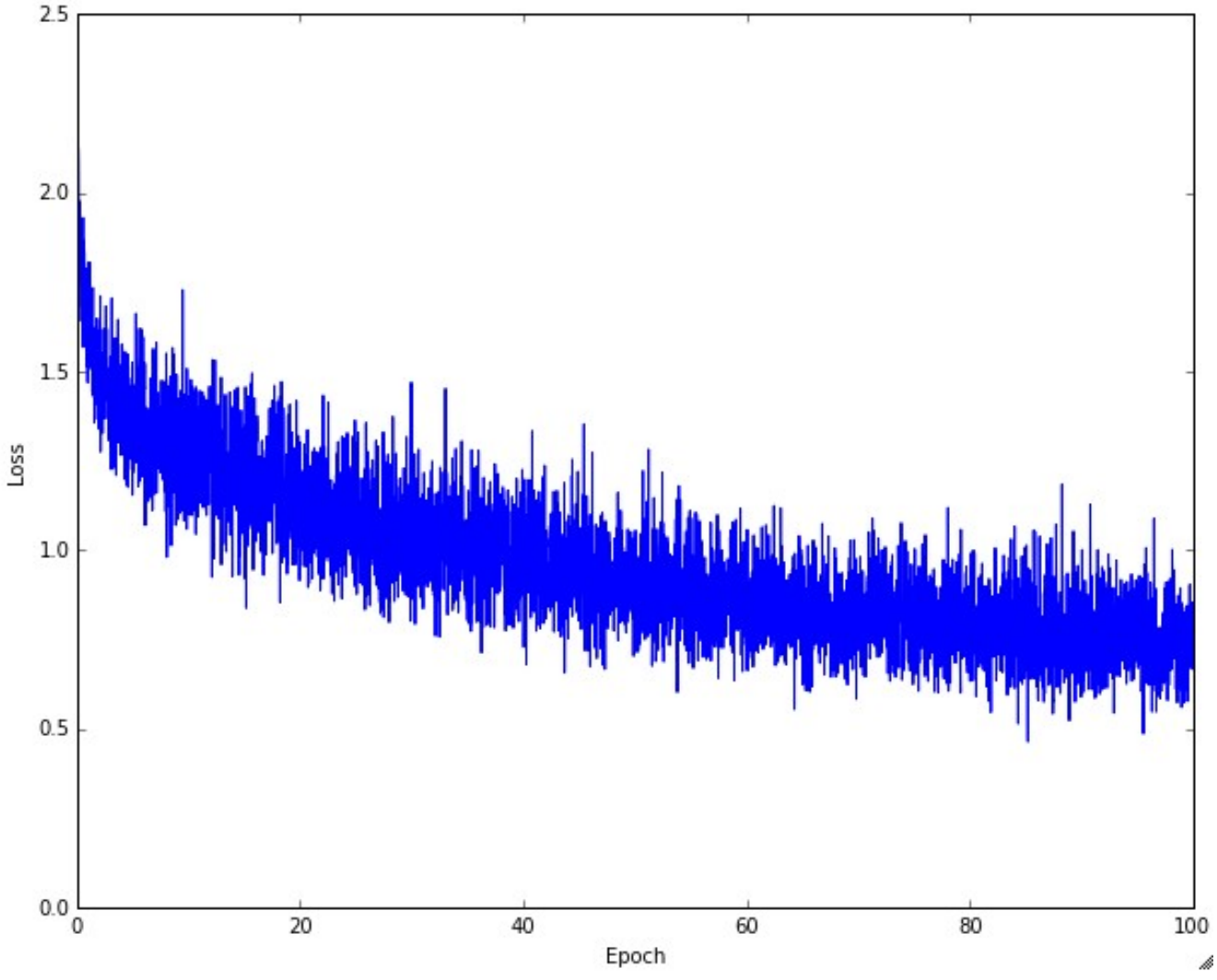
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \nabla J^{(i)}(\boldsymbol{\theta}_{t-1}), \quad i = t \bmod m$$

  - Chamado de método do gradiente estocástico pois $\nabla J^{(i)}(\boldsymbol{\theta})$ é uma variável aleatória que depende de exemplo escolhido $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$
  - Também chamado de *on-line gradient descent*, enquanto o método convencional é chamado *batch gradient descent*
- Cada passagem por todo o conjunto de treinamento é chamada de época (*epoch*)
  - Logo, m iterações são realizadas a cada época
  - Normalmente são necessárias múltiplas épocas até a convergência
  - É recomendável embaralhar o conjunto de treinamento a cada nova época

Método do Gradiente Estocástico (SGD)

- Vantagens:
  - Converge muito mais rapidamente que o método convencional em número de épocas $\implies$ menor custo computacional
  - Escalonável para m muito grande
  - Maior capacidade de escapar de mínimos locais
  - Assim como o método convencional, tem convergência garantida caso a taxa de aprendizagem seja reduzida apropriadamente a cada iteração
- Desvantagens:
  - Custo flutua significativamente a cada iteração

Exemplo



Mini-Batch SGD

- Estima o gradiente usando um subconjunto (*mini-batch*) de B exemplos
- Aproxima $\nabla J(\boldsymbol{\theta})$ na iteração t por

$$\nabla J_t(\boldsymbol{\theta}) \triangleq \frac{1}{B} \sum_{i=sB}^{sB+B-1} \nabla J^{(i)}(\boldsymbol{\theta}), \quad s = \lfloor (t \bmod m)/B \rfloor$$

- Atualização de pesos:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \nabla J_t(\boldsymbol{\theta}_{t-1})$$

- O hiperparâmetro B é conhecido como *batch size*
  - Cada época consiste de m/B iterações
- Além de possuir as mesmas vantagens do SGD:
  - Aumento de B reduz as flutuações
  - Permite paralelizar o cálculo do gradiente

Variações do Método do Gradiente Estocástico

Decaimento da taxa de aprendizado

(*Learning rate schedule*)

- Taxa de aprendizado constante: α
- Decaimento exponencial (a cada τ iterações):

$$\alpha_t = \alpha_0 r^{\lfloor t/\tau \rfloor}, \qquad 0 < r < 1$$

- Decaimento com o inverso do tempo (a cada τ iterações):

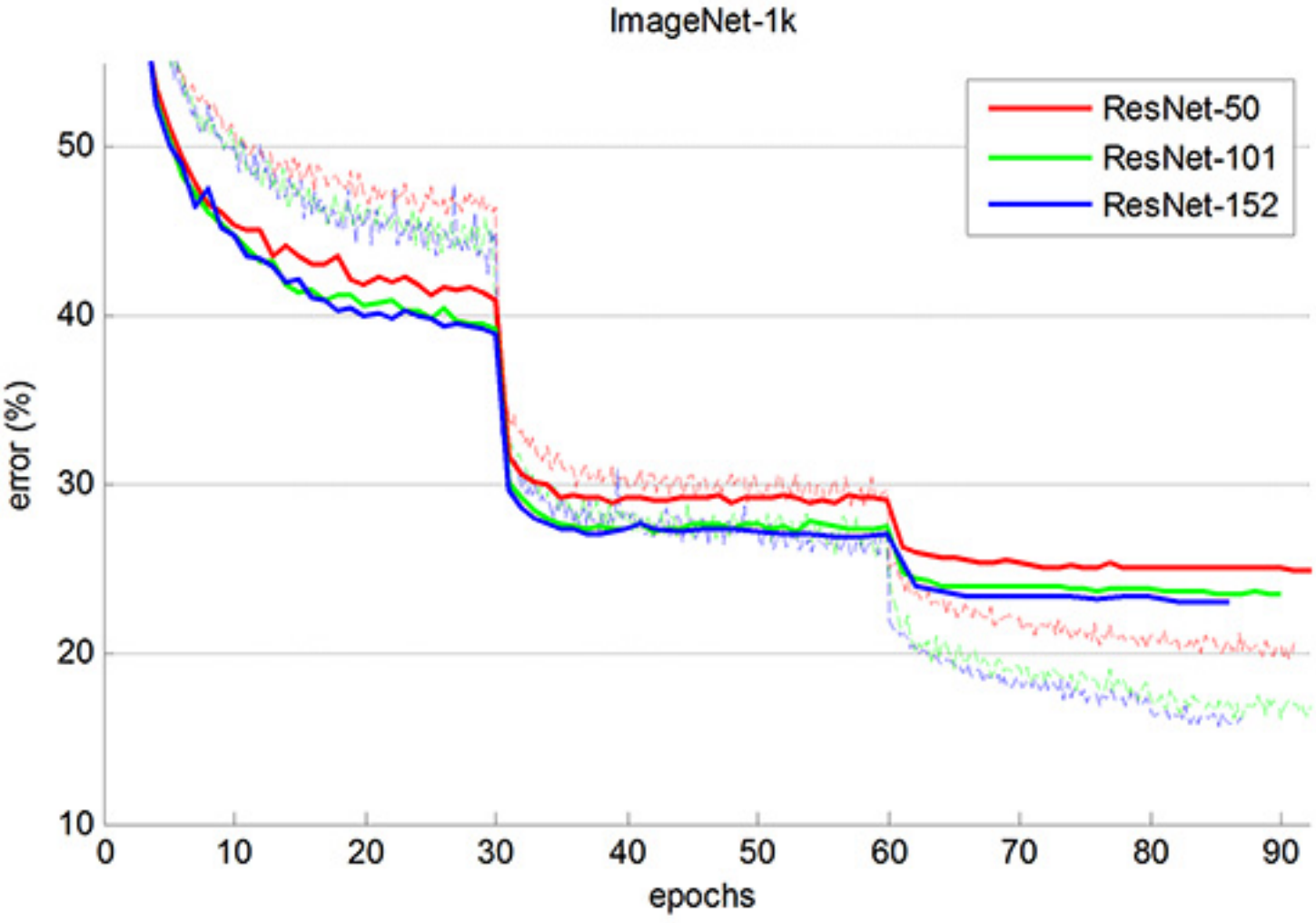
$$\alpha_t = \alpha_0 \frac{1}{1 + r\lfloor t/\tau \rfloor}, \qquad r = \textit{decay rate}$$

- Decaimento *invscaling* (sklearn):

$$\alpha_t = \alpha_0 / t^e, \qquad e \geq 0$$

- Decaimento adaptativo: decaimento exponencial sempre que a perda não se reduz após um certo número de iterações consecutivas

Exemplo


ImageNet-1k
ResNet-50
ResNet-101
ResNet-152
error (%)
epochs
10
20
30
40
50
0
10
20
30
40
50
60
70
80
90

SGD com Momento

- Utiliza o gradiente para atualizar não a posição, mas a velocidade da descida

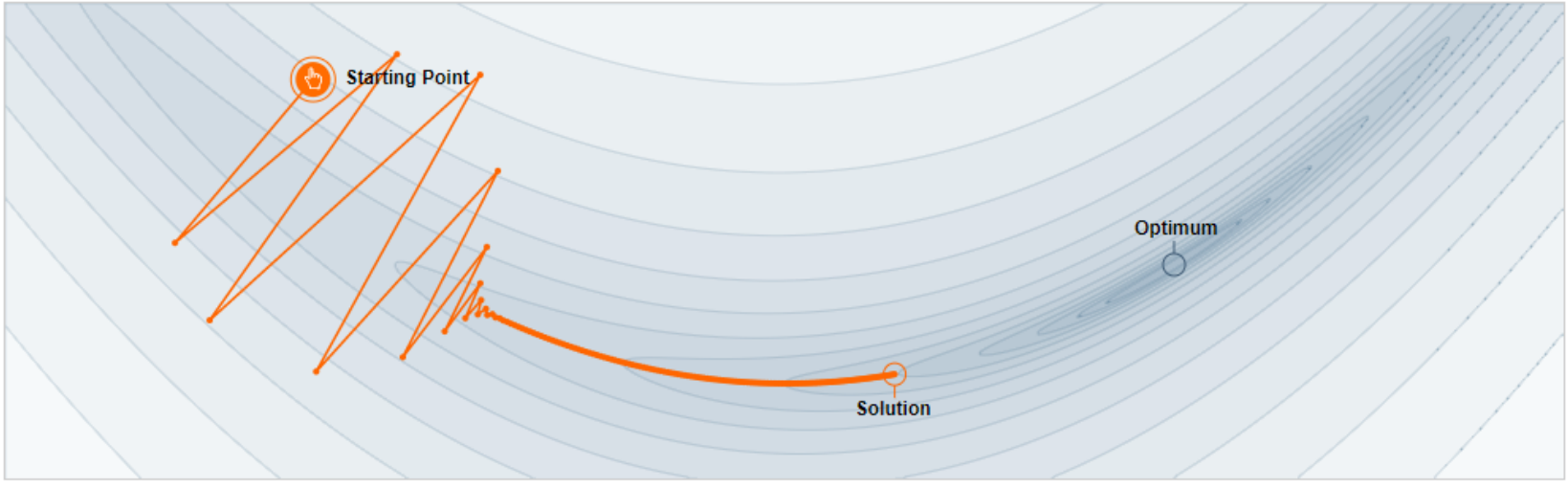
$$\mathbf{g}_t = \nabla J_t(\boldsymbol{\theta}_{t-1})$$
$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} - \alpha \mathbf{g}_t$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \mathbf{v}_t$$

  - O hiperparâmetro $\gamma \in [0, 1)$ é um fator de decaimento da velocidade
  - Valores típicos: $\gamma = 0.5, 0.9, 0.99$

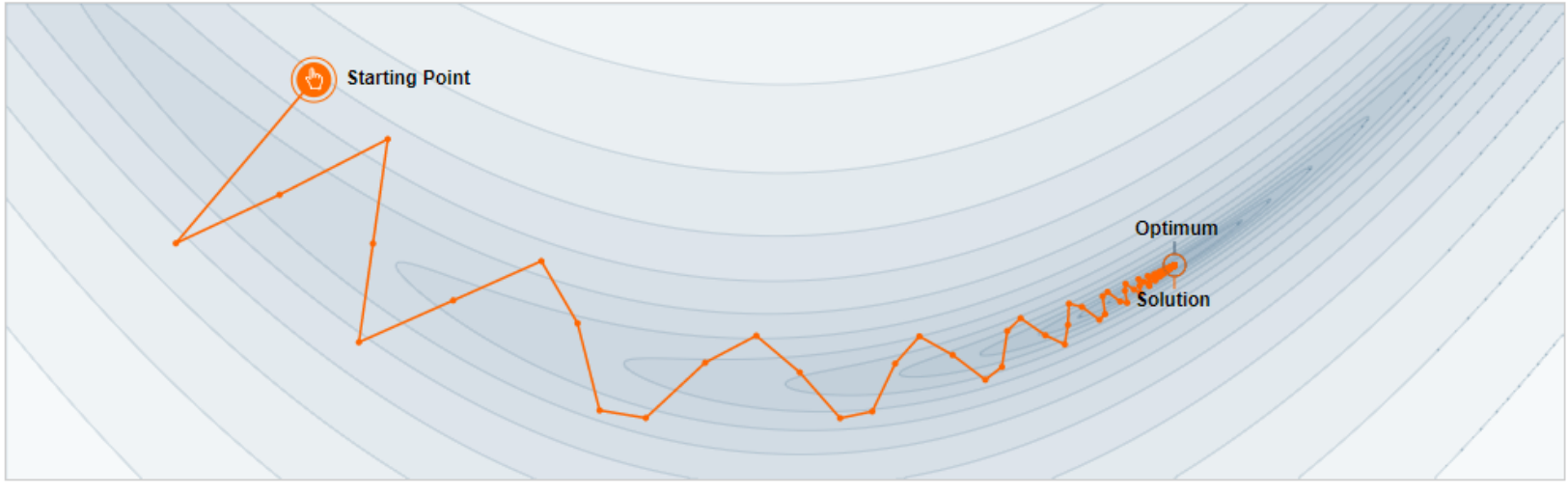
- Pode ser interpretado como o movimento de uma partícula descendo a superfície de erro mas sujeita à resistência do ar ($\gamma < 1$)

Exemplo

Sem momento:



Com momento ($\gamma = 0.86$):



Momento de Nesterov

$$\mathbf{g}_t = \nabla J_t(\boldsymbol{\theta}_{t-1} + \gamma \mathbf{v}_{t-1})$$
$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} - \alpha \mathbf{g}_t$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \mathbf{v}_t$$

AdaGrad

$$\mathbf{g}_t = \nabla J_t(\boldsymbol{\theta}_{t-1})$$
$$\mathbf{r}_t = \mathbf{r}_{t-1} + \mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \frac{1}{\sqrt{\mathbf{r}_t} + \epsilon} \odot \mathbf{g}_t$$

RMSProp

$$\mathbf{g}_t = \nabla J_t(\boldsymbol{\theta}_{t-1})$$
$$\mathbf{r}_t = \rho \mathbf{r}_{t-1} + (1-\rho)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \frac{1}{\sqrt{\mathbf{r}_t} + \epsilon} \odot \mathbf{g}_t$$

Adam

$$
\begin{aligned}
\mathbf{g}_t &= \nabla J_t(\boldsymbol{\theta}_{t-1}) \\
\mathbf{s}_t &= \rho_1 \mathbf{s}_{t-1} + (1-\rho_1)\mathbf{g}_t \\
\mathbf{r}_t &= \rho_2 \mathbf{r}_{t-1} + (1-\rho_2)\mathbf{g}_t \odot \mathbf{g}_t \\
\hat{\mathbf{s}}_t &= \frac{\mathbf{s}_t}{1-\rho_1^t} \\
\hat{\mathbf{r}}_t &= \frac{\mathbf{r}_t}{1-\rho_2^t} \\
\boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \alpha \frac{1}{\sqrt{\hat{\mathbf{r}}_t} + \epsilon} \odot \hat{\mathbf{s}}_t
\end{aligned}
$$

- Obs: $E[\mathbf{s}_t] = (1-\rho_1^t)E[\mathbf{g}_t]$ e $E[\mathbf{r}_t] = (1-\rho_2^t)E[\mathbf{g}_t \odot \mathbf{g}_t]$

Exemplos

Links:

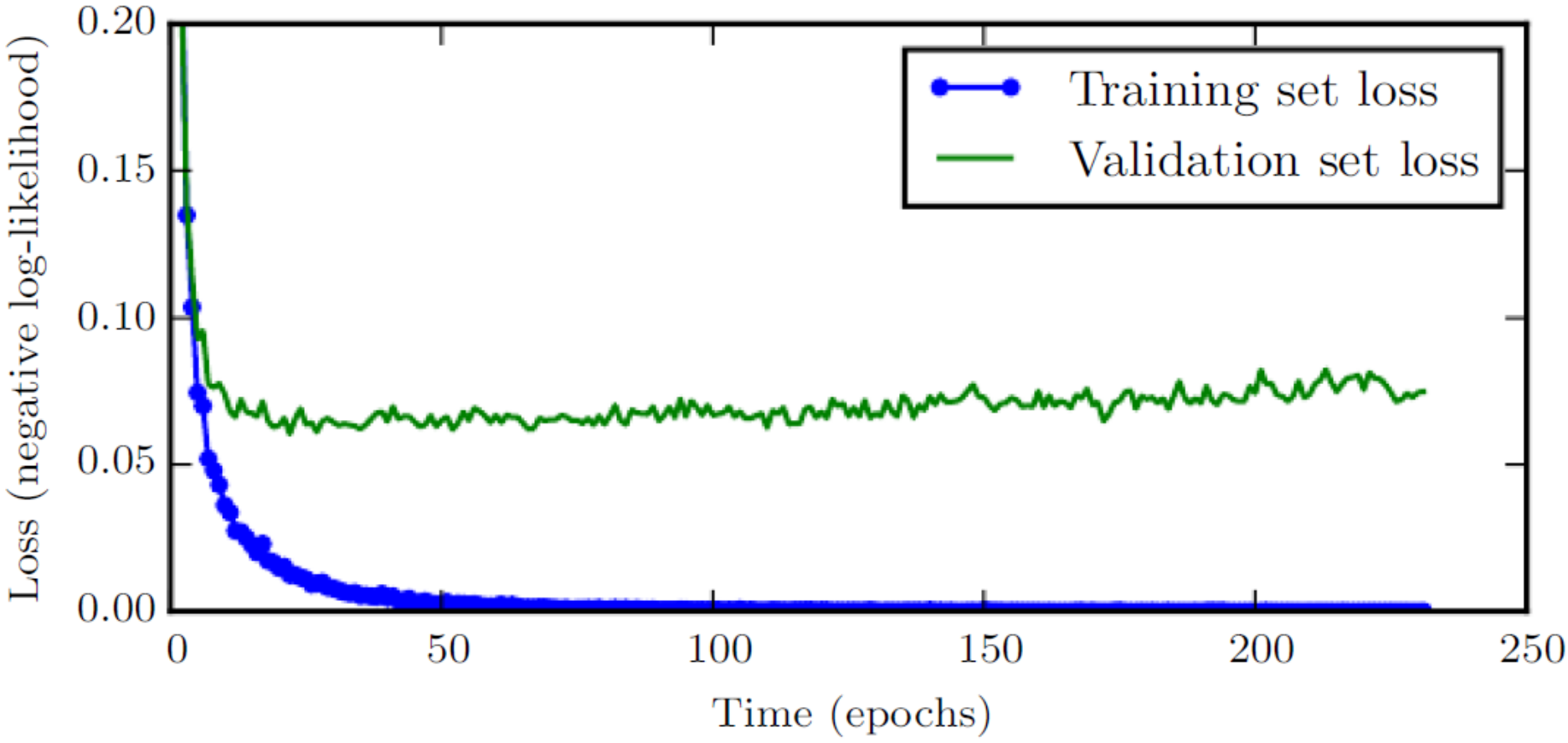
- A Visual Explanation of Gradient Descent Methods
- An overview of gradient descent optimization algorithms
- Why Momentum Really Works

Reduzindo Overfitting

Lidando com Underfitting e Overfitting

- Para reduzir underfitting:
  - Aumentar a capacidade do modelo
    - Ex: Aumentar o número de camadas e/ou número de unidades por camada
  - Treinar por um número maior de épocas e/ou com métodos diferentes
  - Escolher uma representação mais eficiente dos dados
    - Análogo ao desenvolvimento manual de bons atributos
- Para reduzir overfitting:
  - Regularização L1/L2
  - Early stopping
  - Dropout
  - Batch normalization
  - Aumentar o conjunto de treinamento
    - Novas amostras
    - Transformação de amostras existentes *(data augmentation)*

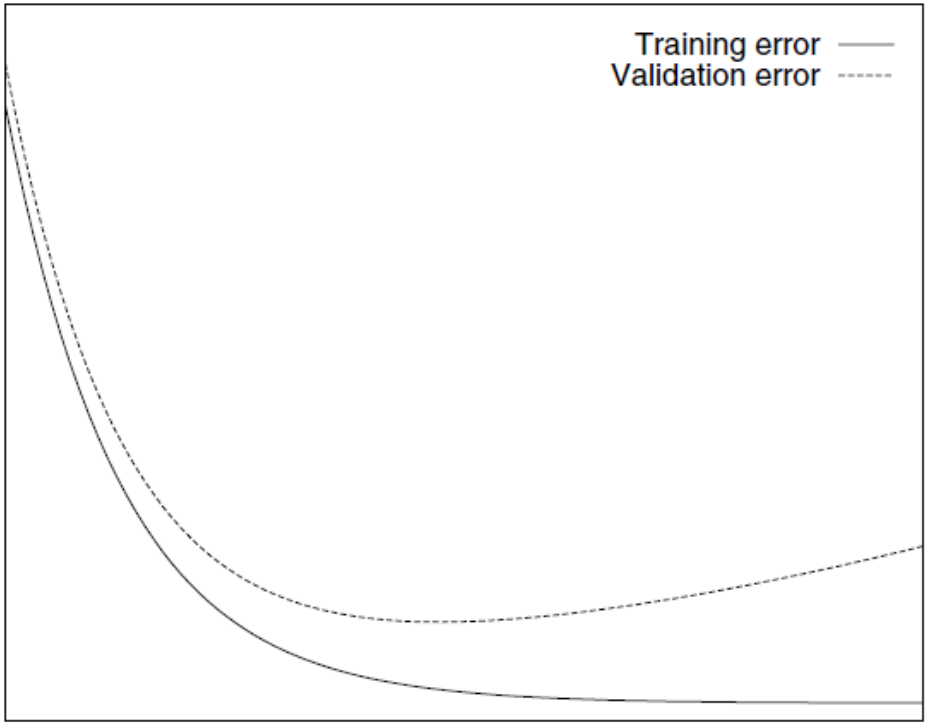
Early Stopping



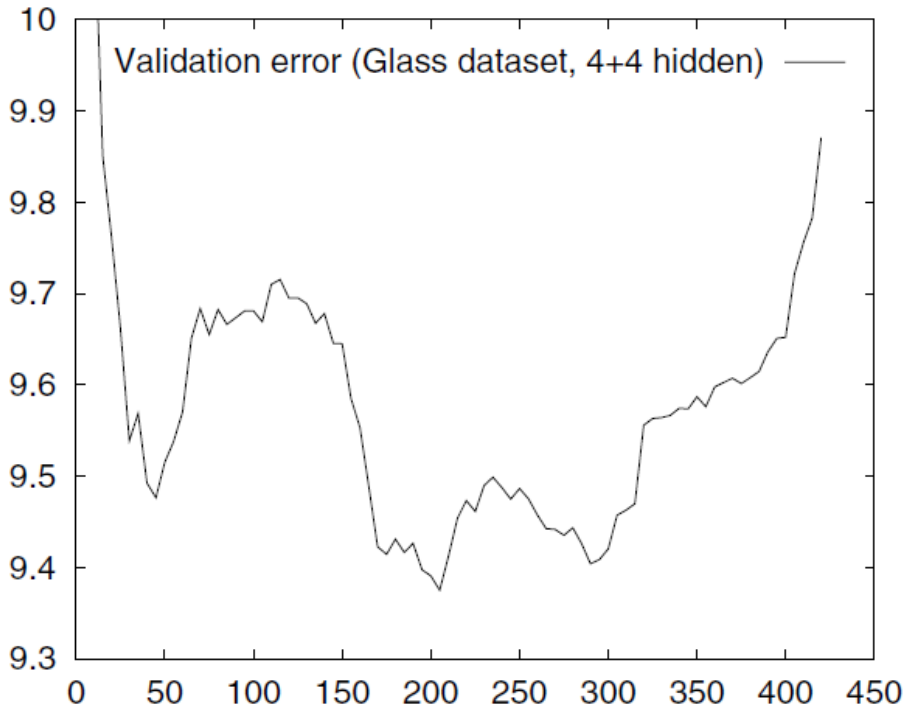
- Princípio básico:

  Parar o treinamento quando o erro de validação começar a crescer

Early Stopping



Expectativa

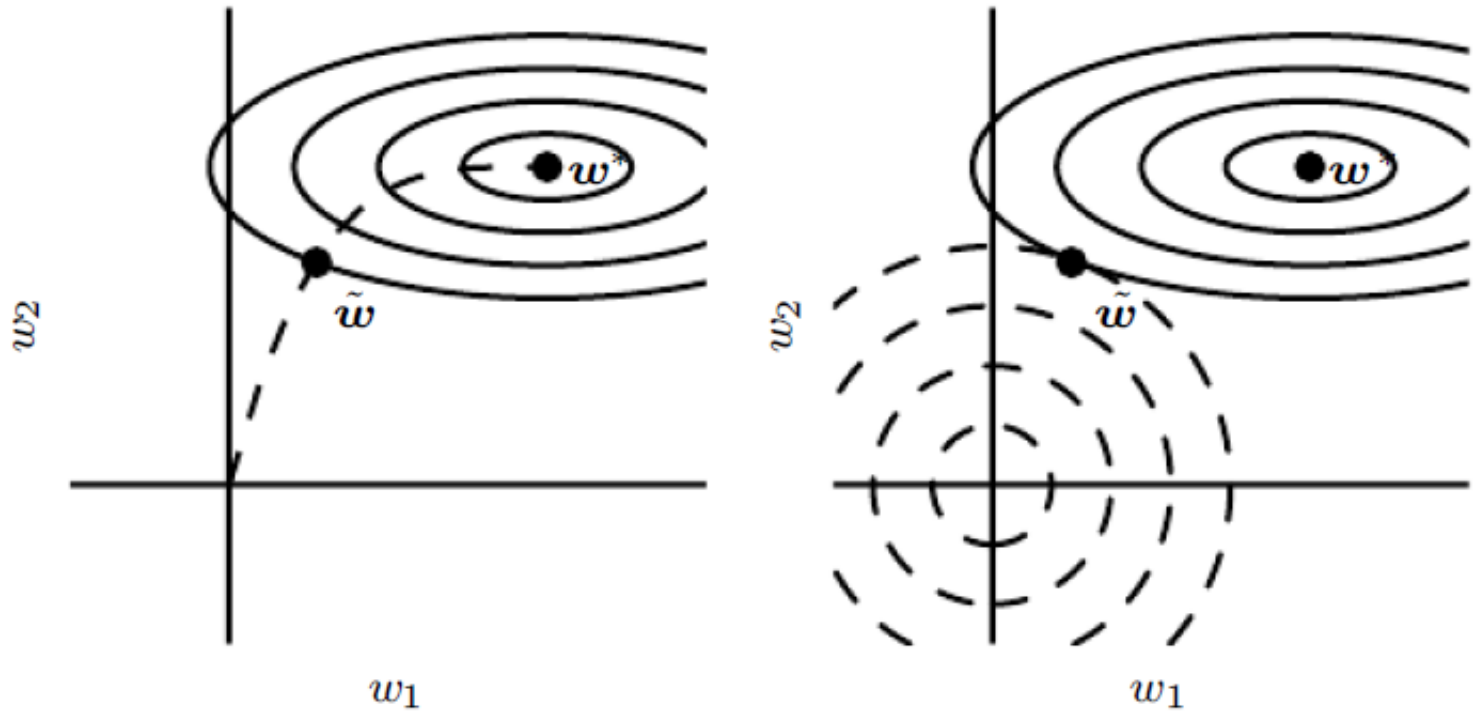


Realidade

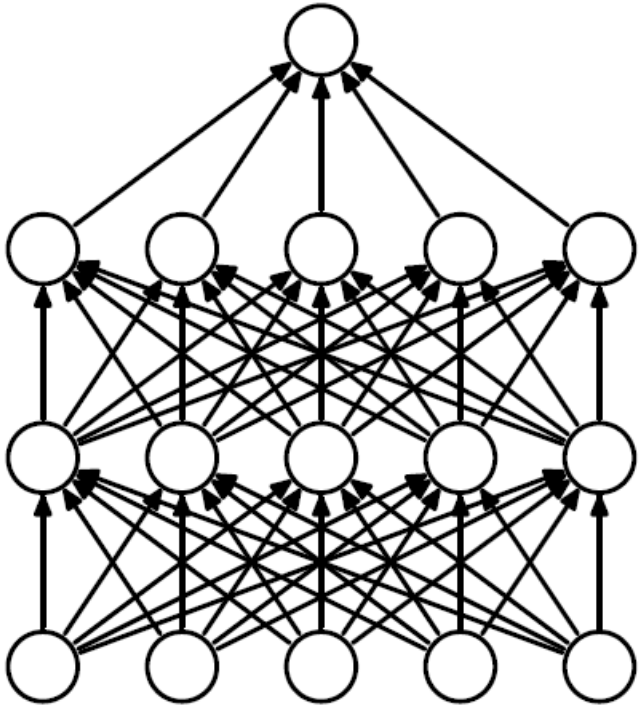
Early Stopping com Paciência - Algoritmo

- Separe um conjunto de validação (por exemplo, uma parte do conjunto de treinamento que não será usada para treinamento)
- Monitore o erro de validação a cada época durante o treinamento
  - O erro de validação irá (erraticamente) diminuir e depois subir
- Sempre que houver uma melhoria no erro de validação, armazene o erro de validação, os parâmetros do modelo e a época correspondente
- Interrompa o treinamento quando passarem-se p épocas sem que haja uma melhoria no menor erro de validação
  - p representa a "paciência" do algoritmo
- Retorne os parâmetros do melhor modelo obtido e o número de épocas
- (Opcional) Treine novamente pelo mesmo número de épocas usando agora todos dados disponíveis (treinamento + validação)

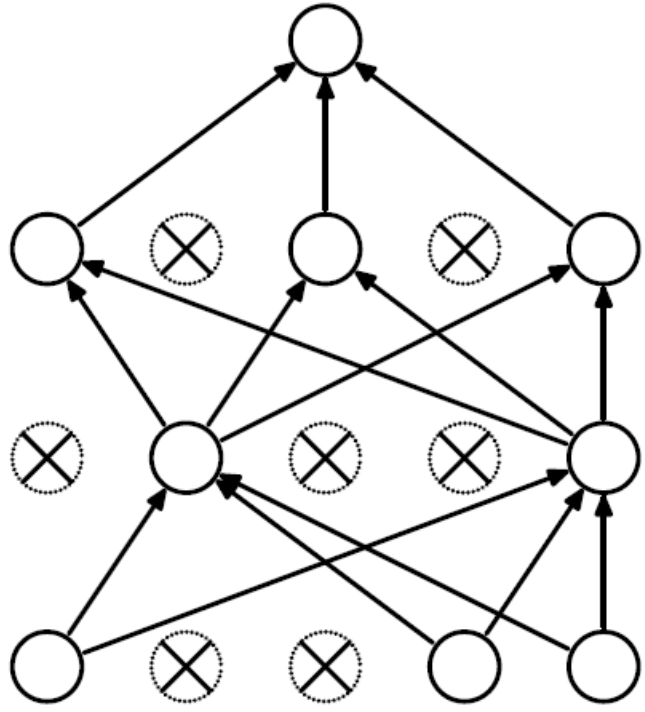
Early Stopping - Interpretação



Dropout



(a) Standard Neural Net



(b) After applying dropout.

- Consiste em zerar aleatoriamente a saída de algumas unidades durante o treinamento para prevenir co-adaptação
  - Força cada unidade a ser individualmente útil ao invés de “se escorar” em outras unidades

Dropout - Implementação

- A cada iteração do treinamento, durante a etapa de propagação direta, multiplica-se a ativação de cada unidade por uma máscara binária aleatória e independente $u_j^{[\ell]}$:

$$a_j^{[\ell]} \leftarrow a_j^{[\ell]} u_j^{[\ell]}$$

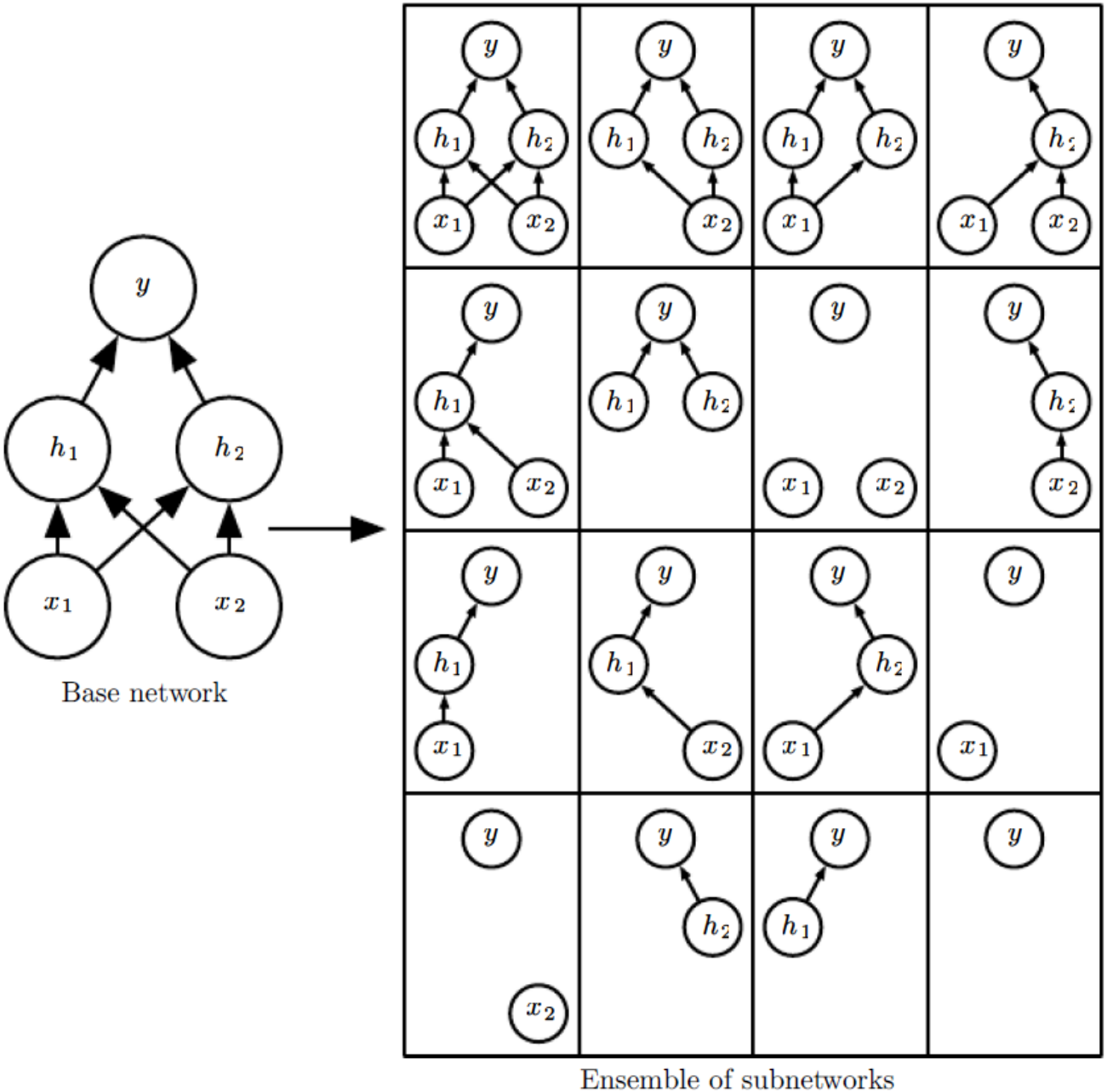
$$u_j^{[\ell]} = \begin{cases} 0, & \text{com probabilidade } p \text{ (\textit{drop})} \\ 1/(1-p), & \text{com probabilidade } 1-p \text{ (\textit{keep})} \end{cases}$$

- Uma nova realização de $\{u_j^{[\ell]}\}$ é feita a cada iteração do treinamento
- Na avaliação (no conjunto de teste), a máscara não é aplicada, i.e., $u_j^{[\ell]}$ é fixado em 1
- A normalização por $1/(1-p)$ garante $E[u_j^{[\ell]}] = 1$ no treinamento, preservando o valor médio sobre todas as ativações

Dropout - Interpretação

- O desempenho de um modelo tipicamente pode ser melhorado combinando as predições de diversos modelos diferentes
  - Também podem ser várias versões do mesmo modelo treinado de formas ligeiramente diferentes
  - Técnicas desse tipo são conhecidas como *ensemble methods*
- A técnica dropout pode ser interpretada como um método de *ensemble*, pois combina (durante o treinamento) as predições de todas as sub-redes que podem ser obtidas removendo-se uma ou mais unidades ocultas da rede original

Dropout - Exemplo



Batch Normalization

- Batch normalization é uma operação que normaliza a variável de entrada (equivalente ao *StandardScaler*), subtraindo a média e dividindo pelo desvio padrão:

$$\hat{x} = \frac{x - \mu_x}{\sigma_x}$$

onde μ_x e σ_x são calculados sobre um mini-batch

$$\mu_x = \frac{1}{B}\sum_{i=1}^{B} x^{(i)}, \qquad \sigma_x^2 = \frac{1}{B}\sum_{i=1}^{B}(x^{(i)} - \mu_x)^2$$

- Assim, $\hat{x}$ possui média nula e variância unitária
- Em seguida, é combinada com uma unidade linear para produzir uma saída com média β e desvio padrão γ:

$$y = \gamma\hat{x} + \beta = \mathrm{BN}(x)$$

  - Estes pesos são aprendidos durante o treinamento

Batch Normalization

- Batch normalization ajuda a estabilizar a distribuição das ativações internas da rede durante o treinamento, o que facilita a convergência do método do gradiente
- Normalmente aplicada antes da função de ativação (não-linearidade)

$$y_j^{[\ell]} = \mathsf{BN}(z_j^{[\ell]})$$

$$a_j^{[\ell]} = g(y_j^{[\ell]}) = g(\mathsf{BN}(z_j^{[\ell]}))$$

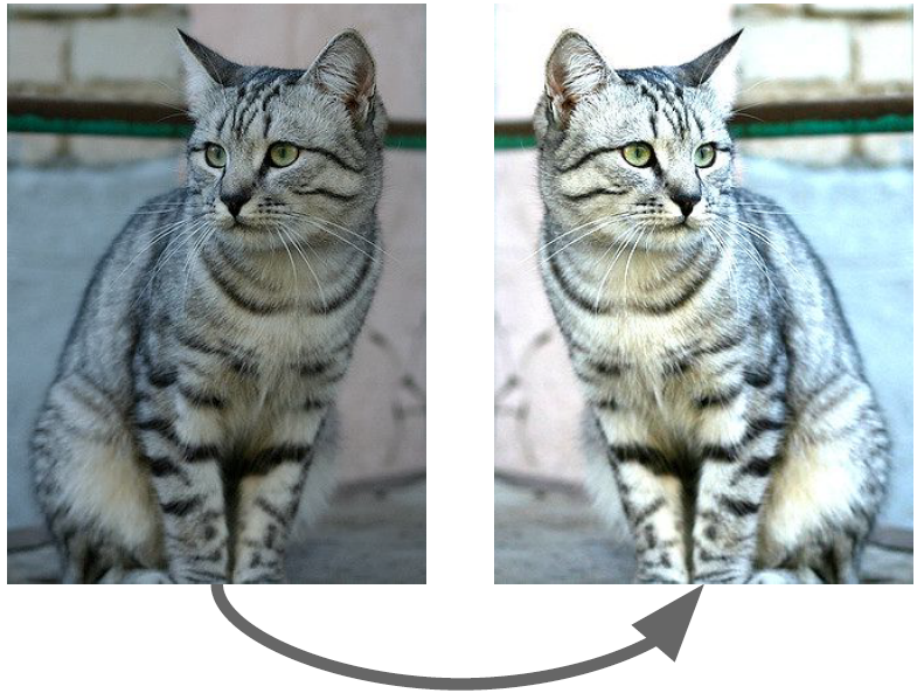
  - Alternativamente, alguns autores aplicam *após* a função de ativação

- Embora o objetivo seja facilitar o treinamento, resultados experimentais tem mostrado que possui também algum efeito de regularização, tornando em alguns casos desnecessário o uso de *dropout*

Exemplo

```
from tensorflow.keras import Sequential, Input
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.optimizers.schedules import InverseTimeDecay
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Input(shape=(16,)))

model.add(Dense(100, kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(100, kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

model.summary()

model.compile(loss='binary_crossentropy', metrics=['accuracy'],
              optimizer=SGD(learning_rate=0.1, # or InverseTimeDecay(0.1, 1, 0.5, staircase=True),
                            momentum=0.9, nesterov=True))

model.fit(x_train, y_train, batch_size=64, epochs=20, validation_split=0.2,
          callbacks=[EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)])
```

Data Augmentation



- Adicionar dados sintéticos ao conjunto de treinamento, possivelmente construídos através de transformações do conjunto original
  - Rotação
  - Translação
  - Alongamento
  - *Cropping*
  - Adição de ruído
  - etc