(3) Introdução à Otimização de Redes Neurais

Redes Neurais e Aprendizado Profundo

Moacir Ponti

www.icmc.usp.br/~moacir — moacir@icmc.usp.br

Agenda

- Básico de Treinamento de Redes Neurais
  - Função de custo e gradiente
- Passo para frente e retro-propagação
- Checklists para o treinamento
- Otimizadores, tamanho do batch e taxa de aprendizado

Agenda

Básico de Treinamento de Redes Neurais

- Função de custo e gradiente

Passo para frente e retro-propagação

Checklists para o treinamento

Otimizadores, tamanho do batch e taxa de aprendizado

Como treinar? Otimização!

Machine Learning e Deep Learning depende de entender otimização e conhecer bem:

- Função de custo/perda e intuição de seus valores
- (Intuição) do gradiente da função
- Inicialização
- Algoritmo de otimização
- Taxa de aprendizado
- Tamanho do batch
- Convergência ao longo do treinamento

Função de custo/perda

Métrica que indique o custo de escolher o modelo atual

- Idealmente deve ser convexa e produzir um gradiente com boa magnitude
- Difícil, considerando todas as direções do hiper-espaço de parâmetros

Função de custo/perda

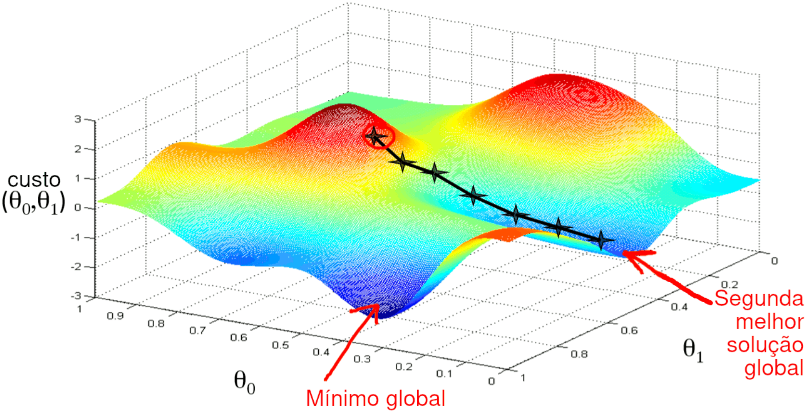
Destaques

- **Mean-squared-error**: erro médio quadrático/perda quadrática
  - utilizada para valores contínuos,
  - mede a divergência quadrática de cada valor de entrada com relação à saída
- **Cross-entropy**: entropia cruzada
  - mais comum e recomendada para probabilidades
  - teoria da informação
  - intuição: o numero de bits adicionais necessários para representar o evento de referência ao invés do predito.

O gradiente

Codifica as taxas de alteração no espaço de parâmetros

- queremos andar na direção do vale, em busca do mínimo global



Passo para frente

Forward-propagation ou *Forward pass*

- cálculo e armazenamento das variáveis intermediárias (incluindo saídas)
- na ordem da entrada para a saída da rede neural
- consideramos redes neurais com uma ou mais camadas ocultas/intermediárias e uma camada de saída

Passo para frente

- rede neural com uma camada oculta e uma camada de saída,
- desconsiderando termos bias de forma que cada camada tenha apenas pesos das conexões w,
- seja $\mathrm{x} \in R^d$ um exemplo de entrada:

Camada oculta com h neurônios

- a variável intermediária é:

$$\mathrm{z} = W^{(1)}\mathrm{x},$$

  sendo $W^{(1)} \in R^{h \times d}$
- o vetor de ativação da camada oculta de tamanho h é:

$$\mathrm{h} = \phi(\mathrm{z}),$$

Passo para frente

Camada de saída com q neurônios

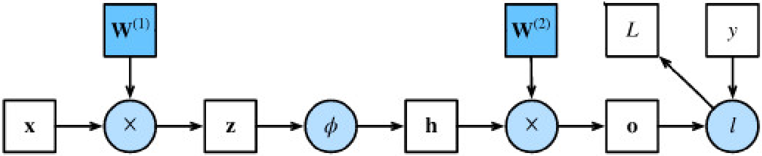
- a variável da camada de saída é computada a partir da entrada de tamanho h:

$$\mathrm{o} = W^{(2)}\mathrm{h},$$

  sendo $W^{(2)} \in R^{q \times h}$
- sendo o target y e uma função de custo ℓ calculamos o custo para o exemplo de entrada:

$$L = \ell(\mathrm{o}, y) = (y - \mathrm{o})^2$$

Grafo computacional da propagação para frente



Retro-propagação

Back-propagation

- calcular o gradiente dos parâmetros da rede neural
- atravessa a rede em ordem reversa, da saída para a entrada
- utiliza a **regra da cadeia**
- armazena as derivadas parciais das variáveis intermediárias com relação aos parâmetros

Derivadas

sejam funções $Y = f(X)$ e $Z = g(Y)$, sendo X, Y, Z tensores de tamanhos arbitrários

- a derivada de Z com relação à X é

$$\frac{\partial Z}{\partial X} \frac{\partial Z}{\partial Y} \times \frac{\partial Y}{\partial X}$$

$\times$ multiplica os argumentos após outras operações intermediárias necessárias.

Retro-propagação

- sejam os parâmetros da nossa rede $W^{(1)}$ e $W^{(2)}$
- o *backpropagation* calcula os gradientes $\frac{\partial L}{\partial W^{(1)}}$ e $\frac{\partial L}{\partial W^{(2)}}$

Gradiente da função de custo com relação à $W^{(2)}$

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial \mathrm{o}} \times \frac{\partial \mathrm{o}}{\partial W^{(2)}}$$

- resolvendo as derivadas parciais:

$$\frac{\partial L}{\partial \mathrm{o}} = \frac{\partial (y - \mathrm{o})^2}{\partial \mathrm{o}} = 2(y - \mathrm{o})$$
$$\frac{\partial \mathrm{o}}{\partial W^{(2)}} = \frac{\partial W^{(2)}\mathrm{h}}{\partial W^{(2)}} = \mathrm{h}$$

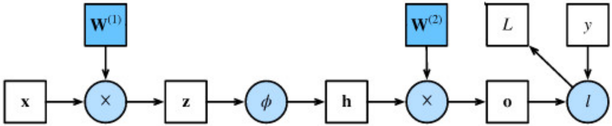
Retro-propagação

...

Gradiente da função de custo com relação à $W^{(2)}$

$$\begin{aligned}\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial \mathsf{o}} \times \frac{\partial \mathsf{o}}{\partial W^{(2)}} &= 2(y - \mathsf{o}) \times \mathsf{h} \\ &= 2(y - \mathsf{o}) \times \phi(\mathsf{z}) \\ &= 2(y - \mathsf{o}) \times \phi(W^{(1)}\mathsf{x})\end{aligned}$$

Retro-propagação



Gradiente da função de custo com relação à $W^{(1)}$

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial \mathsf{o}} \times \frac{\partial \mathsf{o}}{\partial \phi(\mathsf{z})} \times \frac{\partial \phi(\mathsf{z})}{\partial \mathsf{z}} \times \frac{\partial \mathsf{z}}{\partial W^{(1)}}$$

Retro-propagação

Gradiente da função de custo com relação à $W^{(1)}$

...

- resolvendo as derivadas parciais (seja $\phi()$ a função sigmóide):

$$
\begin{aligned}
\frac{\partial \mathsf{o}}{\partial \phi(\mathsf{z})} &= W^{(2)} \\
\frac{\partial \phi(\mathsf{z})}{\partial \mathsf{z}} &= \mathsf{h}(1 - \mathsf{h}) \\
\frac{\partial \mathsf{z}}{\partial W^{(1)}} &= \mathsf{x}
\end{aligned}
$$

Retro-propagação

Gradiente da função de custo com relação à $W^{(1)}$

...

$$\frac{\partial L}{\partial W^{(1)}} = 2(y - \mathrm{o}) \times W^{(2)} \times \mathrm{h}(1 - \mathrm{h}) \times \mathrm{x}$$

Em resumo

Backpropagation

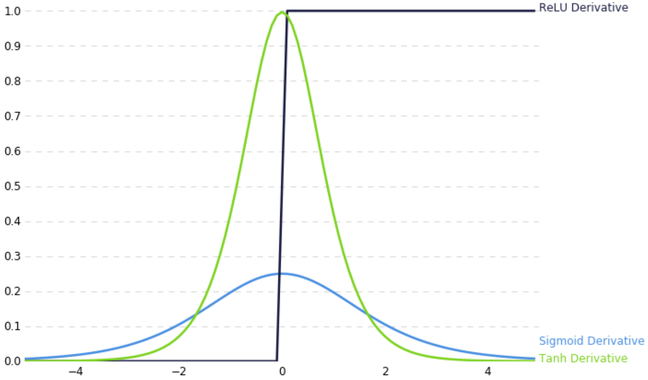
- utiliza a derivada ao longo das camadas para adaptar os pesos
- as funções de custo e de ativação tem que produzir derivada útil

Vanishing gradient

- se ativações geram valores muito baixos não é possível adaptar
- usar precisão dupla (double) e escalar as funções é uma possibilidade
- esse é um dos motivadores do uso de ReLU ao invés de Sigmóides como função de ativação

Uso de funções diferenciáveis

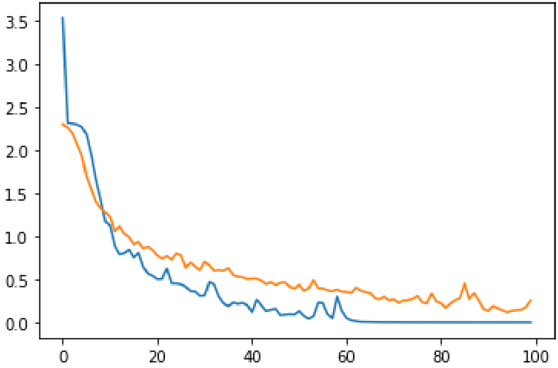
Derivada da sigmóide, ReLU e tangente hiperbólica



Agradecimentos a Harini suresh (`http://harinisuresh.com`) pelos gráficos

Valor da função de custo x gradiente

Ao longo do treinamento a rede adapta os pesos cada vez mais devagar, convergindo para uma solução



Inicialização

Aleatória portanto o resultado é diferente a cada execução.

Escolhas comuns

- Pesos: valor aleatório pela distribuição normal entre 0-1
- Bias: 0 (zero)

A complexidade do treinamento dificulta múltiplas execuções

- Importante fazer experimentos piloto em pequenos subconjuntos de dados

Check-list 1

- O valor da função de custo nos pesos aleatórios faz sentido?
- Ex. num problema de classificação com 10 classes com entropia cruzada calculada na saída softmax:
  - $-\ln(0.1) = 2.3026$

Agenda

Básico de Treinamento de Redes Neurais

Função de custo e gradiente

Passo para frente e retro-propagação

Checklists para o treinamento

Otimizadores, tamanho do batch e taxa de aprendizado

Otimizadores

Stochastic Gradient Descent (SGD)

Formulação original (atualização por instância)

$$\begin{aligned}\theta_{k+1} &= \theta_k - \alpha_k \nabla_\theta \ell(y, f(x; \theta_k)),\\ &= \theta_k - \alpha_k g(x, \theta_k),\end{aligned}$$

α_k é a taxa de aprendizado (learning rate) na iteração k

Batch Stochastic Gradient Descent (SGD)

computando o gradiente da função de custo de um lote de instâncias X_k na iteração k

$$\theta_{k+1} = \theta_k - \alpha_k g(X_k, \theta_k),$$

Tamanho de batch e Taxa de aprendizado

Há uma relação entre tamanho de batch e taxa de aprendizado.

Batch

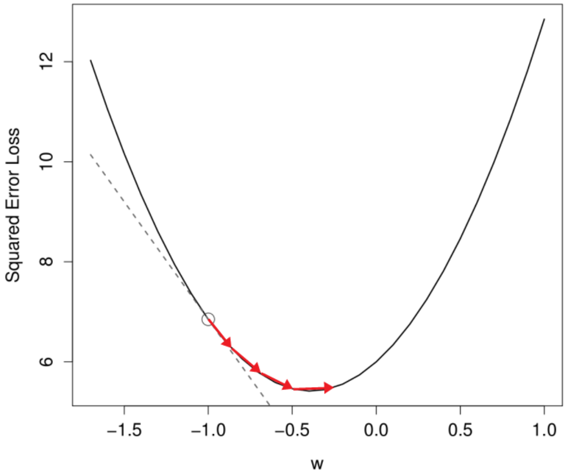
- Padrão é 32
  - batches maiores: estimativas mais suaves, difícil manter na memória, exige ajustar bem a taxa de aprendizado,
  - batches menores: estimativas mais ruidosas, mas que mostraram vantagens em encontrar melhores mínimos.

Tamanho de batch e Taxa de aprendizado

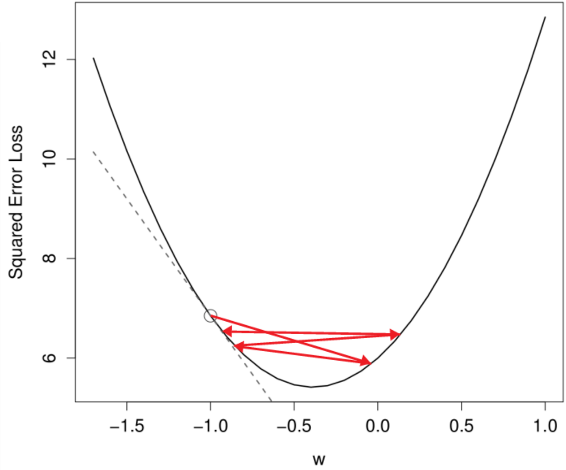
Taxa de aprendizado

- Padrão é 0.01
  - pode ser pouco adequado para alguns otimizadores
  - pode ser pouco adequado para batchs maiores (ou muito pequenos)
- É recomendado iniciar com um valor maior, e reduzir a taxa progressivamente (learning rate scheduling).

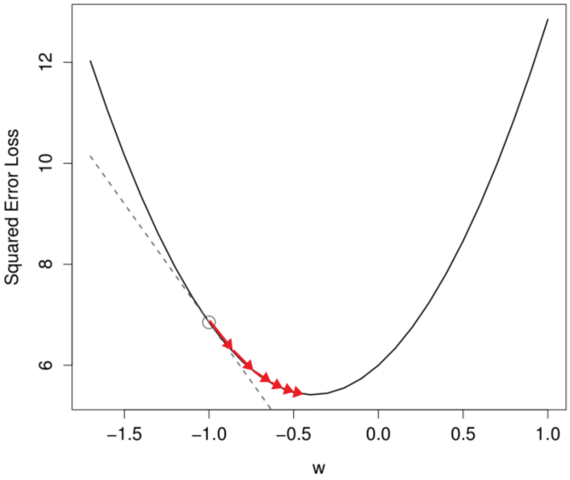
Taxa de aprendizado: intuição com valor pequeno



Taxa de aprendizado: intuição com valor excessivamente grande



Taxa de aprendizado: intuição com decaimento



Taxa de aprendizado

Check-list 2

- Utilize decaimento de taxa de aprendizado
- ... de forma fixa ou de acordo com métricas computadas no treinamento ou validação

Otimizadores

Momentum

Interpreta o custo como um terreno montanhoso.

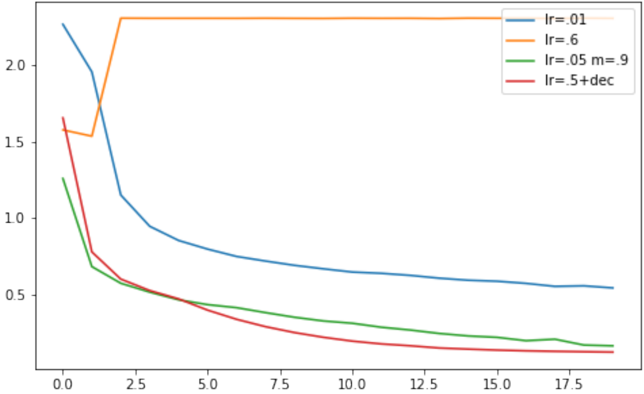
- Inicializar: posicionar partícula com velocidade zero no terreno
- Otimização: rolar partícula, considerando a aceleração.
- Consequência: a velocidade é ajustada considerando a magnitude de atualizações anteriores

$$\theta_{k+1} = \theta_k + m \cdot v - \alpha_k g(x, \theta_k),$$

v é o momentum, inicialmente 0; m peso: menor funciona como atrito que reduz a energia cinética do sistema (hiperparâmetro)

- Investigar $m \in [0.5, 0.9, 0.95, 0.99]$
- ou iniciar com valor menor e aumentar ao longo das épocas

Taxa de aprendizado: diferentes abordagens, caso real



Outros otimizadores

Adam

Utiliza momentos do gradiente: o segundo momento é usado para normalizar o primeiro, evitando outliers/pontos de inflexão

$$\theta_{k+1} = \theta_k - \alpha_k \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$$

$\hat{m}$ e $\hat{v}$ são estimativas corrigidas do primeiro e segundo momentos do gradiente.

- $\hat{m}_k$ é a soma do gradiente atual com o acúmulo de gradientes anteriores $\hat{m}_{k-1}$ (similar a momentum).
- $\hat{v}_k$ é a soma do quadrado do gradiente em k com o acúmulo de valores anteriores $\hat{v}_{k-1}$ (taxa de aprendizado adaptativa).
- Funciona melhor com taxa de aprendizado **menor**, quando comparado ao SGD

Otimizadores

Check-list 3

Utilizar:

- SGD (+ Momentum), ou
- Adam

Convergência ao longo do treinamento

O gráfico do custo diz **muito** sobre o aprendizado

Check-list 4

- Acompanhe o custo ao longo de épocas, se possível com conjunto de validação (idealmente não deve ser o teste!)
- Inicie com experimentos com poucos exemplos
  - explore os hiperparâmetros tentando obter "overfitting" para um subconjunto de exemplos, obtendo custo próximo a zero, e depois refine a busca num conjunto maior.

Bibliography I

Aline Becher, Moacir Ponti. **Optimization Matters: Guidelines to Improve Representation Learning with Deep Networks**
ENIAC, 2021. Book chapter.
https://arxiv.org/abs/1806.07908

Moacir A. Ponti, Fernando dos Santos, Leo Ribeiro, Gabriel Cavallari. **Training Deep Networks from Zero to Hero: avoiding pitfalls and going beyond**.
SIBGRAPI, 2021. Tutorial.
https://arxiv.org/abs/2109.02752

Moacir A. Ponti, Leo Ribeiro, Tiago Nazaré, Tu Bui, John Collomosse. **Everything You Wanted to Know About Deep Learning for Computer Vision but were Afraid to Ask**.
SIBGRAPI-T, 2017. Tutorial.

Bibliography II