

# SCC0270/SCC5809 - Redes Neurais

## Aula - Deep Learning

Profa. Dra. Roseli Aparecida Francelin Romero  
SCC - ICMC - USP

2018

# Sumário

- 1 Pré-treinamento não-supervisionado
  - Otimização X Regularização
  - Stacked Denoising

## Autoencoders (SDA)

- 2 Arquiteturas de CNN
- 3 Otimização para Deep Learning

# Pré-treinamento não-supervisionado

- Até 2006, arquiteturas profundas eram pouco discutidas na literatura de aprendizado de máquina.
  - Os experimentos com redes neurais *feed-forward* totalmente conectadas (e.g. MLP) realizados até então apontavam que, embora uma arquitetura de  $k$  camadas possa representar um mesmo modelo representado por uma arquitetura de  $k - 1$  camadas, redes mais profundas obtinham generalização pior do que com 1 ou 2 camadas ocultas.
- Descobriu-se, então, que resultados muito melhores podem ser obtidos por meio do pré-treinamento de cada camada por meio de aprendizado não-supervisionado [Hinton and Salakhutdinov, 2006].
  - Essa descoberta impulsionou uma nova onda de pesquisa na área de redes neurais.

## Pré-treinamento não-supervisionado

- **Exceção:** Redes Neurais Convolucionais (CNN - *Convolutional Neural Networks*) podem produzir boa generalização com várias intermediárias, ainda que seu treinamento seja estritamente supervisionado.
- Essas redes foram apresentadas pela primeira vez por Atlas et al. [1988], aprimoradas por LeCun et al. [1998] e generalizadas por Behnke [2003].
- Trata-se de redes **não** totalmente conectadas e nas quais há compartilhamento de pesos.
- Seriam redescobertas em 2012, conforme já citado.

## Greedy layer-wise unsupervised learning

- No caso das redes *feed-forward* totalmente conectadas, a maioria dos estudos mostrando o sucesso do pré-treinamento não-supervisionado fez uso de *Greedy layer-wise unsupervised learning* [Bengio et al., 2007]:
  - 1 Treinar primeiro a camada mais baixa com algum algoritmo de aprendizado não supervisionado, como Máquina de Boltzmann Restrita (RBM - *Restricted Boltzmann Machine*) ou algum auto-associador (*autoencoder*), gerando um conjunto inicial de parâmetros para a primeira camada da rede neural.
  - 2 Usar a saída da primeira camada (nova representação da entrada bruta) como entrada da próxima camada, e similarmente inicializar essa camada com um algoritmo de aprendizado não-supervisionado.
  - 3 Inicializado um conjunto de camadas, a rede neural completa pode ser ajustada com uso de um critério de treinamento supervisionado, como usual, usando o alg. *Back-propagation*.

# Pré-treinamento não-supervisionado

- Vantagens de pré-treinamento sobre inicialização aleatória são estatisticamente comprovadas em alguns trabalhos.
- **Layer-local unsupervised criteria:** introduzir um sinal de treinamento não supervisionado em cada camada pode ajudar a guiar os parâmetros daquela camada na direção de regiões melhores no espaço de parâmetros.

# Sumário

- 1 Pré-treinamento não-supervisionado
  - Otimização X Regularização
  - Stacked Denoising

## Autoencoders (SDA)

- 2 Arquiteturas de CNN
- 3 Otimização para Deep Learning

# Otimização X Regularização

- A melhora observada se deve a melhor otimização ou melhor regularização?
  - Como o erro no conjunto de treinamento de arquiteturas DNNs pode aproximar-se de zero mesmo sem etapa de pré-treinamento, tem-se que a melhora se deve a melhor **regularização**.
  - Pré-treinamento não-supervisionado pode ser considerado um **regularizador**, na medida em que restringe as regiões do espaço de parâmetros em que a solução se encontra.



# Otimização X Regularização

- Trabalhos constataam que o ajuste ruim das camadas mais baixas da arquitetura é o que leva a resultados ruins quando não há pré-treinamento.
- Quando a camada mais alta é pequena (i.e. poucas unidades), os resultados de redes neurais inicializadas aleatoriamente são ruins tanto para o conjunto de treinamento quanto para o conjunto de teste.
- Nos experimentos com parâmetros inicializados aleatoriamente em que o erro vai para zero, sempre é permitido que o número de unidades escondidas em cada camada possa ser arbitrariamente grande (minimizando o erro em um conjunto de validação).

# Otimização X Regularização

## Hipótese

Quando não há restrições quanto ao tamanho das camadas, as duas camadas mais altas da rede neural são suficientes para representar o conjunto de treinamento, usando como entrada a representação obtida pelas camadas inferiores, ainda que seja uma representação ruim.

- Conclui-se que os efeitos do pré-treinamento não-supervisionado são mais marcantes para as camadas mais baixas de uma DNN.

# Otimização X Regularização

- Outra evidência de que funciona como regularizador:
  - Quando não há muita capacidade (i.e. arquitetura é mais simples), o pré-treinamento não-supervisionado tende a piorar a generalização.
  - Quando o conjunto de treinamento é pequeno (e.g., MNIST, com menos de 100 mil exemplos), ainda que haja melhora no erro de teste, o erro de treinamento aumenta.

# Otimização X Regularização

- Com pré-treinamento não-supervisionado, as camadas mais baixas da rede são restringidas, de modo a capturar regularidades na distribuição de entrada.
- Isso permite melhor generalização através de ajustes mais apropriados de todas as camadas, o que é uma forma de regularização.

## Treinamento não-supervisionado para DNNs

- PCA (*Principal Component Analysis*) e as variantes comuns de ICA (*Independent Component Analysis*) são inapropriados.
  - Não podem lidar com o caso *overcomplete*, em que o número de saídas da camada é maior que o número de entradas.
- Além disso, empilhar projeções lineares (ex. duas camadas de PCA) continua a ser uma transformação linear.
- Podem-se usar, entretanto, extensões do ICA para lidar com o caso *overcomplete*, assim como algoritmos relacionados a PCA e ICA, como *autoencoders* e RBMs.

# Treinamento não-supervisionado para DNNs

- Além de reduzir a dependência de um gradiente não confiável obtido de um critério supervisionado, o pré-treinamento não-supervisionado pode ser uma forma de **decompor** o problema em sub-problemas associados a **diferentes níveis de abstração**.
  - Algoritmos de aprendizado não supervisionado extraem informação sobre a distribuição da entrada.
  - Um treinamento não-supervisionado em uma única camada, devido à capacidade limitada da camada em questão, pode extrair *características de baixo nível*.
  - Uma segunda camada baseada no mesmo princípio, tomando a saída da primeira camada como entrada, poderia extrair *características de mais alto nível*.
  - O aprendizado, portanto, fica local a cada camada.

# Sumário

- 1 Pré-treinamento não-supervisionado
  - Otimização X Regularização
  - Stacked Denoising

## Autoencoders (SDA)

- 2 Arquiteturas de CNN
- 3 Otimização para Deep Learning

# Stacked Denoising Autoencoders (SDA)

- SDA é um auto-encoder que recebe um dado corrompido e é treinado para prever o original, um dado não corrompido como sua saída.



# Stacked Denoising Autoencoders (SDA)

- Fazem uso de *greedy layer-wise unsupervised training*.
- Pré-treinamento é realizado com *autoencoders* aplicados sucessivamente sobre cada camada, compondo os níveis de abstração.
- São mais fáceis de implementar do que as *Deep Belief Networks* (DBN), porque não fazem uso das Máquinas de Boltzmann Restritas (RBM).

# Auto-associadores (autoencoders)

- Recebem uma entrada  $x \in [0, 1]^d$  e a mapeiam (com um codificador, ou *encoder*) para uma representação oculta  $y \in [0, 1]^{d'}$  por meio de um mapeamento determinístico:

$$y = s(Wx + b)$$

onde  $s$  é uma não-linearidade, como uma função sigmoide.

- A representação  $y$ , ou **código** (*code*), é, então, mapeada de volta (com um **decodificador**, ou *decoder*) para uma reconstrução  $z$ , com o mesmo formato de  $x$ :

$$z = s(W'y + b')$$

## Auto-associadores (autoencoders)

- Os parâmetros desse modelo devem ser otimizados para minimizar o erro médio de reconstrução.
- Existem diferentes formas de se medir o erro de reconstrução, inclusive o tradicional *erro quadrático médio*:

$$L(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|^2$$

- Caso a entrada seja composta por vetores de bits ou vetores de probabilidades de bits, pode-se usar a entropia cruzada (*cross-entropy*) da reconstrução:

$$L_H(\mathbf{x}, \mathbf{z}) = - \sum_{k=1}^d [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)]$$

# Auto-associadores (autoencoders)

- Espera-se que  $\mathbf{y}$  seja uma representação que capture as coordenadas sobre os principais fatores de variância dos dados.
  - A representação  $\mathbf{y}$  pode não ser uma boa compressão para todo  $\mathbf{x}$ .
  - Devido ao aprendizado, deverá ser uma boa compressão, em particular, para os exemplos de treinamento, e espera-se que para outros exemplos também.
  - Nesse sentido, um *autoencoder* **GENERALIZA**.

# Auto-associadores (autoencoders)

- **Problema:** se não houver restrições, o *autoencoder* pode apenas aprender a função identidade, replicando as entradas que vier a receber (entrada  $n$ -dimensional codificada para dimensão de  $n$  ou maior).
  - Por meio de experimentos, verifica-se que, se treinados com gradiente descendente estocástico, *autoencoders* não-lineares com mais unidades escondidas do que entradas (*overcomplete*) geram representações úteis.

# Auto-associadores (autoencoders)

- Um *autoencoder* pode ser implementado como uma rede MLP cuja saída desejada é igual à entrada.
- Nesse caso, a camada intermediária é o código a ser decodificado na camada de saída.

# Denoising Autoencoders

- A ideia é forçar a camada escondida a descobrir características mais robustas.
- Um *denoising autoencoder* é uma versão **estocástica** do *autoencoder*.
  - Tenta codificar a entrada.
  - Tenta desfazer o efeito de um processo de corrompimento aplicado estocasticamente sobre a entrada.
    - Isso só pode ser feito capturando dependências estatísticas entre as entradas.

## Denoising Autoencoders (DAE)

- DAE é o processo no qual a entrada de cada camada é artificialmente corrompida, o que torna a entrada menos sensível a ruídos.
- Isto é útil nos casos em que somente um no. pequeno de amostras (entradas) é disponível.
- Em Vincent et al. [2008], o processo de corrompimento estocástico configura aleatoriamente algumas entradas (cerca de metade delas) para *zero*.
- Com isso, o *denoising autoencoder* tentará prever os valores faltantes (corrompidos) a partir dos valores não corrompidos, para subconjuntos dos padrões faltantes selecionados aleatoriamente.
- Note que ser capaz de prever qualquer subconjunto de variáveis a partir das restantes é condição suficiente para capturar a **distribuição conjunta de probabilidade** em um conjunto de variáveis.



# Stacked Denoising Autoencoders

- Introduzidos em Vincent et al. [2008], trata-se de DNNs com aprendizado não-supervisionado.
  - Posteriormente, serão vistas como MLPs e submetidas a um ajuste supervisionado com *backpropagation*.
- Consistem no empilhamento de *denoising autoencoders*.
- A **representação latente** (código) do *denoising autoencoder* de uma camada serve como entrada para a camada de cima.
  - Se o *autoencoder* é implementado como uma MLP, a representação latente é a sua **camada oculta**.

# Stacked Denoising Autoencoders

- O pré-treinamento não-supervisionado dessa arquitetura é realizado uma camada de cada vez (*layer-wise*).
- Cada camada é treinada como um *denoising autoencoder*, de modo a minimizar o erro de reconstrução da sua entrada.
- Uma vez que as  $k$  primeiras camadas tenham sido treinadas, podemos treinar a camada  $k + 1$ , porque torna-se possível computar o código (representação latente) da camada  $k$ .
- Somente os códigos dos *autoencoders* são mantidos.
  - As reconstruções de cada camada são úteis somente durante o seu treinamento não-supervisionado, devendo ser ignoradas assim que esse treinamento é finalizado.

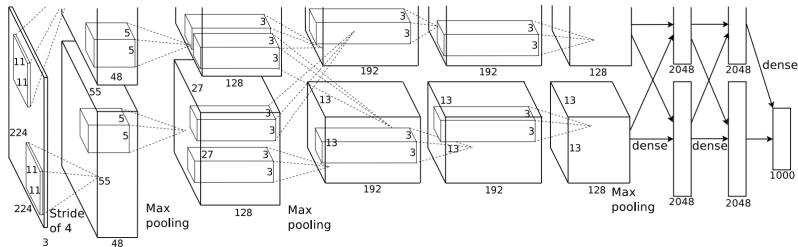
# Stacked Denoising Autoencoders

- Uma vez que todas as camadas tenham sido pré-treinadas, a rede vai para um segundo estágio, denominado ajuste fino (*fine-tuning*).
  - Ajuste fino supervisionado (*supervised fine-tuning*): minimizar o erro de predição em uma tarefa supervisionada.
- Para isso, pode-se acrescentar uma camada de *regressão logística* (aprendizado supervisionado) na camada no topo (código de saída da camada de saída) da rede.
- Em seguida, a rede deixa de ser vista como SDA, e passa a ser encarada como uma rede MLP.
  - Treinar a rede com o *backpropagation* → *fine-tuning*.
  - Obter resultados simulando-a como uma MLP comum.

# Sumário

- 1 Pré-treinamento não-supervisionado
- 2 Arquiteturas de CNN
- 3 Otimização para Deep Learning

# AlexNet (2012)



**Figura 1:** Arquitetura da CNN, mostrando explicitamente a separação entre as duas GPUs usadas no treinamento. As GPUs se comunicavam somente em algumas camadas.

# ZF Net (2013)

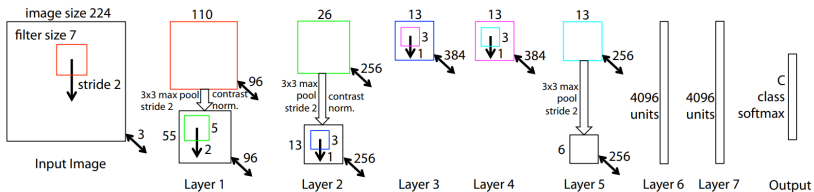


Figura 2: Arquitetura do modelo convolucional, em 8 camadas.

## VGG Net (2014)

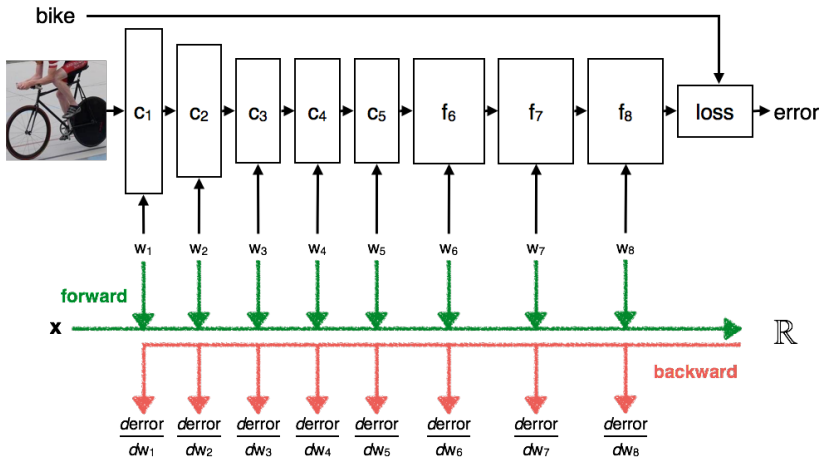


Figura 3: Ilustração de rede VGG.

# GoogLeNet (2015)

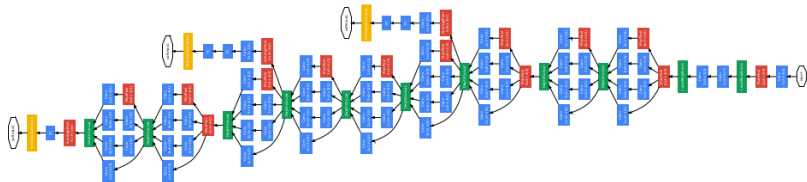


Figura 4: Visualização completa da rede GoogLeNet.



# Microsoft ResNet (2015)

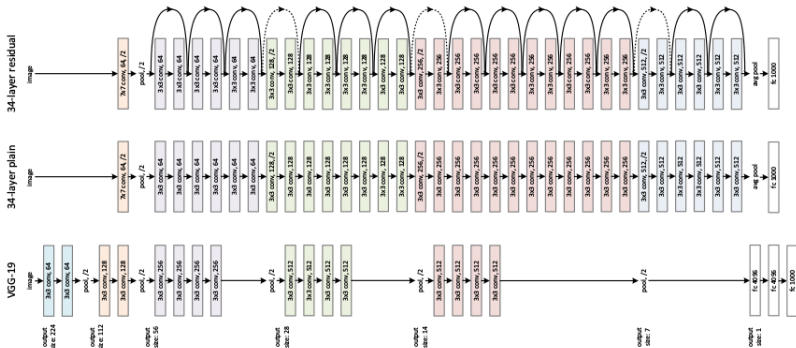
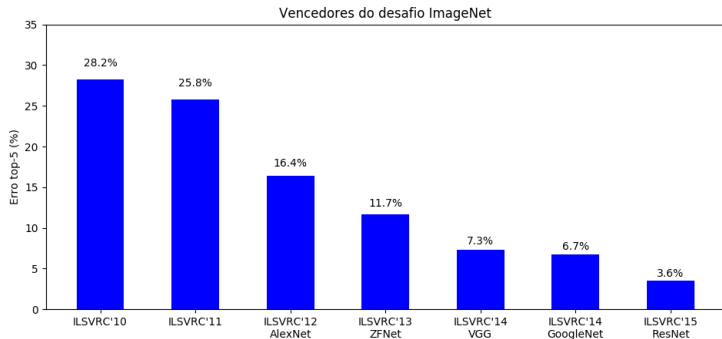
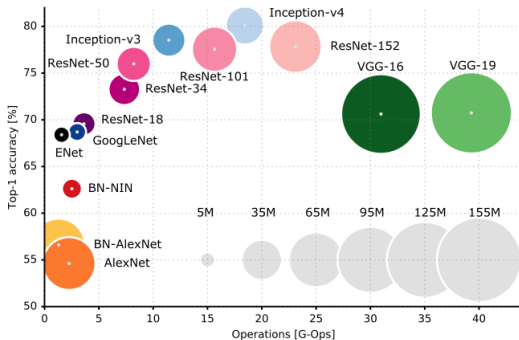


Figura 5: Ilustração de rede ResNet.

# Comparações



## Comparações



**Figura 6:** Acurácia top-1 *versus* número de G-ops (operações  $\times 10^9$ ) necessárias para realizar um único *forward* em várias arquiteturas populares de *deep learning*. O tamanho dos círculos é proporcional ao número de parâmetros da rede [Canziani et al. [2016]].

# Sumário

1 Pré-treinamento  
não-supervisionado

2 Arquiteturas de CNN

3 Otimização para Deep  
Learning

- Taxa de Aprendizado Adaptativa
- ADaptive GRADient - ADAGRAD
- Algoritmo ADADELTA

# Notação

- $w$  — pesos
- $f$  - função erro a ser minimizada
- $g$  - gradiente da  $f$

# Sumário

1 Pré-treinamento  
não-supervisionado

2 Arquiteturas de CNN

3 Otimização para Deep  
Learning

- Taxa de Aprendizado Adaptativa
- ADaptive GRADient - ADAGRAD
- Algoritmo ADADELTA

# Algoritmos para controlar a diminuição de $\eta$

- SGD + Momentum:



$$\Delta w_{t+1} = \eta \cdot g_t + \mu \cdot \Delta w_t$$



- como colocar valor para o parametro de aprendizado

$$\eta$$

- idealmente ele deveria adaptativo
- neuronios em diferentes cam. aprendem de forma diferente
- deveria-se usar taxas diferentes de aprendizado adaptativas

# RPROP - Propagação Resilient

- Riedmiller e Braun 1993
- Abordar o problema da taxa de aprendizagem adaptativa
- Aumentar a taxa de aprendizagem de um peso multiplicativamente se os sinais do último dois gradientes concordam
- Além disso, diminuir a taxa de aprendizagem multiplicativamente



## RPROP - Propagação Resilient

- Se  $g_t \cdot g_{t-1} > 0$  então aumenta  $\eta$
- Caso contrario, diminuir  $\eta$ ;
- considere um peso que teve atualizações de 0.1 em 9 minibatches
- e na decima, uma atualização de -0.9.
- Se fosse SGD, o peso voltaria para seu valor inicial
- no RPROP, ele receberia  $9\eta - \eta = 8\eta$

# Sumário

- 1 Pré-treinamento não-supervisionado
- 2 Arquiteturas de CNN
- 3 Otimização para Deep Learning
  - Taxa de Aprendizado Adaptativa
  - ADaptive GRADient - ADAGRAD
  - Algoritmo ADADELTA

## Métodos de Primeira Ordem por Dimensão: ADAGRAD

- O método chamado ADAGRAD tem se mostrado muito eficiente em tarefas de grande escala em um ambiente distribuído:

$$\Delta w_t = -\frac{\eta}{\text{sqrt}(t_{\tau=1} g_{\tau}^2)} \cdot g_t$$

onde a distancia euclidiana é calculada sobre todos os gradientes prévios em cada dimensão e  $\eta$  é a taxa de aprendizado global compartilhada em todas as dimensões.

## Métodos de Primeira Ordem por Dimensão: ADAGRAD

- Uma vez que esta taxa dinâmica cresce com o inverso da inclinação, as magnitudes grandes dos gradientes têm taxas de aprendizagem menores e pequenos gradientes têm grandes taxas de aprendizagem.
- Isto tem a propriedade importante, como nos métodos de segunda ordem, que o progresso ao longo de cada dimensão nivela ao longo do tempo.
- Isto é muito benéfico para as redes DNNs, uma vez que a escala dos gradientes em cada camada é muitas vezes diferentes em várias ordens de grandeza, de modo que a taxa ótima de aprendizagem deve levar isso em conta.

## Métodos de Primeira Ordem por Dimensão: ADAGRAD

- Uma vez que as magnitudes de gradientes são consideradas em ADAGRAD, este método pode ser sensível às condições iniciais dos parâmetros e aos gradientes correspondentes
- Se os gradientes iniciais são grandes, as taxas de aprendizagem serão baixas para o restante do treinamento.
- Isto pode ser evitado pelo aumento da taxa de aprendizagem global, porém isto torna o método ADAGRAD sensível a escolha da taxa de aprendizagem.

## Métodos de Segunda Ordem por Dimensão: ADADELTA

- Devido ao contínuo acúmulo de gradientes ao quadrado no denominador do ADAGRAD, a taxa de aprendizagem continuará a diminuir ao longo do treinamento, e eventualmente, chegar a zero e parar o treinamento completamente.
- O método ADADELTA surgiu para evitar este problema, pois, faz uma aprox. de 2a. ordem.
- Métodos de segunda ordem como o Método de Newton ou Quase Newton consideram a matriz Hessiana na determinação do mínimo
- Como o cálculo da matriz Hessiana é custoso computacionalmente para grandes base de dados, Becker and LecCun [1988] propuseram uma aproximação da diagonal da Hessiana.

## Métodos de Segunda Ordem por Dimensão: ADADELTA

- Esta aproximação diagonal pode ser calculada com um cálculo adicional para **feedforward** e **back-propagation** através da rede, dobrando o cálculo sobre SGD.
- $\Delta w_t = -\frac{1}{|diag(H_t)| + \mu} \cdot g_t$  onde  $\mu$  é uma constante pequena para melhorar o condicionamento da matriz hessiana em regiões de pequena curvatura.

# Sumário

- 1 Pré-treinamento não-supervisionado
- 2 Arquiteturas de CNN
- 3 Otimização para Deep Learning
  - Taxa de Aprendizado Adaptativa
  - ADaptive GRADient - ADAGRAD
  - Algoritmo ADADELTA



# Algoritmo ADADELTA

- Em vez de acumular a soma de gradientes quadrados de todas as iterações, apenas uma janela dos últimos gradientes são acumulados, que para ser de algum tamanho fixo  $L$  (em vez de  $T$ , onde o tamanho  $t$  é a iteração atual, como em ADAGRAD).
- Com esta janela de acumulação, o denominador da ADAGRAD não pode acumular até ao infinito e, em vez torna-se estimativas locais usando gradientes recentes.
- Isso garante que a aprendizagem continua a fazer progressos, mesmo depois de muitas iterações de atualizações.

# Algoritmo ADADELTA

- Uma vez que armazenar  $\mathbf{L}$  gradientes quadrados anteriores é ineficiente, o método ADADELTA implementa esta acumulação como uma **forma exponencial da média dos gradientes quadrados**.
- **Assumindo que no tempo  $t$   $E[g_t^2]$  é dado por:**  
 $E[g^2]_t = \rho \cdot E[g^2]_{t-1} + (1 - \rho)g_t^2$  **onde  $\rho$  é uma constante de decaimento semelhante ao utilizado no termo momentum.**
- **Desde que exigem a raiz quadrada desta quantidade nas atualizações de parâmetros, isso se torna efetivamente o RMS de gradientes quadrados anteriores até o tempo  $t$ :**  
 $RMS[g_t] = \text{sqrt}([g^2]_t + \epsilon)$  **onde  $\epsilon$  é uma constante.**
- **O parâmetro resultante é dado por:**  $\Delta w_t = -\frac{\eta}{RMS[g_t]} \cdot g_t$
- **este metodo é conhecido como o RMS PROP se  $\rho = 0.9$**

# Algoritmo ADADELTA

- Entrada: Taxa de decaimento  $\rho$  e Constante:  $\epsilon$  e Parâmetro inicial  $x_0$
- ❶ Inicializar variáveis acumulativas  $E[g^2]_0 = 0, E[w^2]_0 = 0$
- ❷ Para  $t = 1$  até  $T$  faça
- ❸ **Compute Gradiente:**  $g_t$
- ❹ **Acumule Gradiente:**  $E[g^2]_t = \rho E[g^2]_{t-1} + (1 - \rho)g_t^2$
- ❺ **Atualização:**  $\Delta w_t = \frac{RMS\Delta[w]_{t-1}}{RMS[g_t]} g_t$

## Algoritmo ADADELTA Cont

- 6 Acumule Atualizações:  
$$E[\Delta w^2]_t = \rho E[\Delta w^2]_{t-1} + (1 - \rho) \Delta w_t^2$$
- 7 Aplicar Atualização:  $w_{t+1} = w_t + \Delta w_t$
- 8 Fim para

## Método de Primeira Ordem: AdamOptimizer

- Adam é um método para otimização estocástica eficiente que apenas requer gradientes de primeira ordem com pouco requisito de memória.
- O método calcula taxas de aprendizado adaptativas para diferentes parâmetros de estimativas de primeiro e segundo momentos dos gradientes;
- O nome ADAM é obtido de: "ADaptive Moment estimation.
- combina as vantagens de dois alg. conhecidos: ADAGRAD (Duchi et al., 2011), que trabalha com "sparse gradients", e RMSProp (Tieleman & Hinton, 2012), que trabalha bem com pontos não-estacionário e on-line;

# ADAMOPTIMIZER

- Seja  $\alpha$ : tamanho do passo
- Sejam:  $\beta_1, \beta_2 \in [0, 1)$ : Taxas de decaimento Exponencial para estimativa do momento
- Seja  $f(w)$ : função objetivo estocástica com parametros  $w$
- $m_0$ : vetor parâmetro inicial  $m_0 \leftarrow 0$  (Inicialize primeiro vetor momento)
- $v_0 \leftarrow 0$  (Inicialize 2nd vetor momento)
- $t \leftarrow 0$  (Inicialize o tempo  $t$ )

## ADAMOPTIMIZER - Cont.

- **while**  $w_t$  não convergiu **do**

$$t \leftarrow t + 1$$

$$g_t \leftarrow \Delta_w(w_{t-1}) \text{ (calcule gradientes w.r.t no tempo } t)$$

$$m_t \leftarrow \beta_1 \Delta m_{t-1} + (1 - \beta_1) \Delta g_t \text{ (Atualize estimativa primeiro momento)}$$

$$v_t \leftarrow \beta_2 \Delta v_{t-1} + (1 - \beta_2) \Delta g_t^2 \text{ (Atualize estimativa segundo momento)}$$

$$\hat{m}_t \leftarrow \frac{m_t}{(1 - \beta_1^t)} \text{ (Calcula estimativa bias primeiro momento)}$$

## ADAMOPTIMIZER - Cont.

$\hat{v}_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$  (Calcula estimativa bias segundo momento)

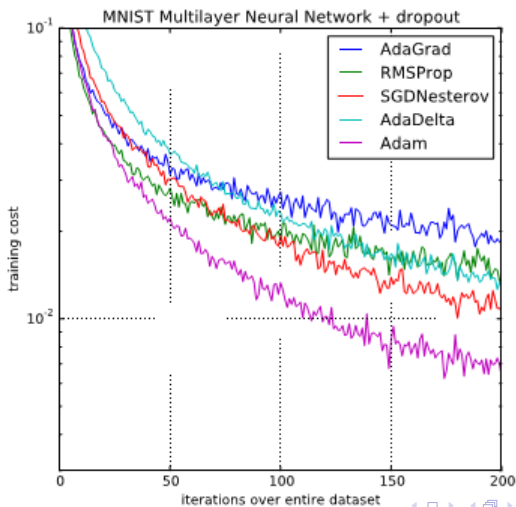
$w_t \leftarrow w_{t-1} - \alpha \times \frac{\hat{m}_t}{(\text{sqrt}(\hat{v}_t)+\epsilon)}$  (Atualize parametros)

**end while**

return  $w_t$  (Parametros resultantes)



# Taxa de aprendizagem adaptativa - MNIST



# Taxa de aprendizado adaptativa - Regressão Logística

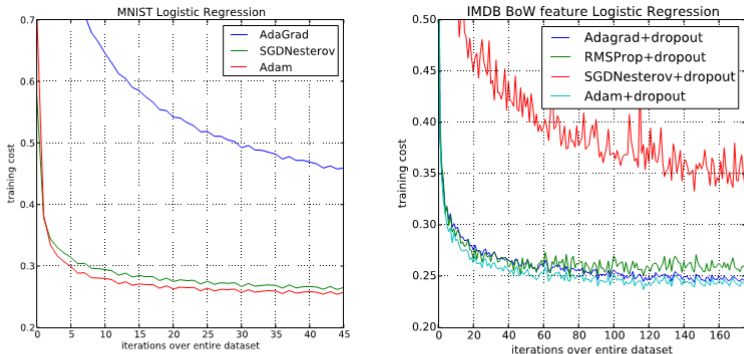
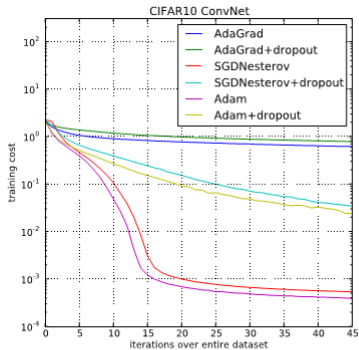
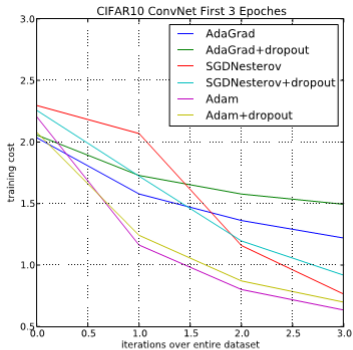


Figure 1: Logistic regression training negative log likelihood on MNIST images and IMDB movie reviews with 10,000 bag-of-words (BoW) feature vectors.

# Taxa de aprendizado adaptativa - CIFAR



## Usando dif. tec. param. de velocidade - MLP

Épocas	ADADelta		Adam	
	loss	acc	loss	acc
1a.	0.231	0.9303	0.2209	0.9355
2a.	0.1064	0.9683	0.0969	0.9698
3a.	0.0773	0.99768	0.0683	0.9787
4a.	0.0616	0.9814	0.0525	0.9828
5a.	0.0506	0.9847	0.0448	0.9850
	1s 63us/step		1s 64us/step	

## Artigos Importantes

- G.E. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks.” Science 313.5786 (2006): 504-507.
- P. Baldi and P. Sadowski, “The dropout learning algorithm”, Artificial Intelligence, Volume 210, Pages 78-122, May 2014.
- Y. Bengio, A. Courville and P. Vincent, “Representation Learning: A Review and new Perspectives”, Arxiv, 2012.
- Y. Bengio, “Practical recommendations for gradient-based training of deep architectures,” ArXiv, 2012.
- S. Becker and Y. LeCun, “Improving the convergence of back-propagation learning with second order methods,” Tech. Rep., Department of Computer Science, University of Toronto, Toronto, ON, Canada, 1988.

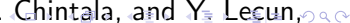
## Artigos Importantes

- M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in Computer Vision – ECCV 2014, 2014, pp. 818–833.
- K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” CoRR, vol. abs/1409.1556, 2014.
- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.
- Y. Wei and Q. Yuan, “Deep residual learning for remote sensed imagery pansharpening,” in 2017 International Workshop on Remote Sensing with Intelligent Processing (RSIP), 2017, pp. 1–4.

## Artigos Importantes

- R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC, USA, 2014, pp. 580–587.
- R. Girshick, “Fast R-CNN,” in 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440–1448.
- S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
- J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition

## Artigos Importantes

- Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” in 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1701–1708.
- Z. Yu and C. Zhang, “Image based Static Facial Expression Recognition with Multiple Deep Network Learning,” Microsoft Research, Nov. 2015.
- D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” Neural Networks, vol. 32, pp. 333–338, Aug. 2012.
- H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, “A convolutional neural network cascade for face detection,” in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5325–5334.
- P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. Lecun, 



## Artigos Importantes

- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, Jan. 2014.
- Y. Bengio, "Deep Learning of Representations for Unsupervised and Transfer Learning," in PMLR, 2012, pp. 17–36.
- J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3320–3328.
- J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition," in

## Artigos Importantes

- J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, “Efficient object localization using Convolutional Networks,” in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 648–656.
- Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, May 2015.

# Referências I

- Atlas, L. E., Homma, T., and Marks II, R. J. (1988). An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification. In *Proc. Neural Information Processing Systems (NIPS)*, page 31.
- Behnke, S. (2003). *Hierarchical neural networks for image interpretation*, volume 2766. Springer Science & Business Media.
- Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153.
- Canziani, A., Paszke, A., and Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

## Referências II

- Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.