Deep Learning

Loo Tiang Kuan, Leonard

A0098795B

supervised by

Associate Professor Ji Hui

co-supervised by

Assistant Professor Alvina Goh Siew Wee

May 5, 2017

Today's Agenda

1. Initialization
2. Other Gradient Descents
3. Other Regulariser

Overview

Session 1
Activations
Cost
Stochastic Gradient Descent
Backpropagation
Overfitting
Regularization

Session 2
Initialisation
Hyper-parameters
Variants of SGD
More Regularisers
Intro to ConvNets

Session 3
Popular ConvNet Architectures
AlexNet
ZFNet
VGG
GoogLeNet
ResNet
Denoising

Session 4/5
Autoencoders
Transposed Convolution
Generative Network
Generative Adversarial Network

Recap: Stochastic Gradient Descent

Randomly sample into m mini-batches of size k:

$$\begin{aligned}\mathbf{x} &= \left\{\{\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, ..., \mathrm{x}^{(k)}\}, \{\mathrm{x}^{(k+1)}, \mathrm{x}^{(k+2)}, ..., \mathrm{x}^{(2k)}\}, ..., \{\mathrm{x}^{(n-k+1)}, ..., \mathrm{x}^{(n)}\}\right\} \\ &= \{M_1, M_2, ..., M_m\}\end{aligned}$$

$$\left.\frac{\partial C}{\partial w}\right|_{\mathbf{x}} \approx \left.\frac{\partial C}{\partial w}\right|_{\mathbf{x}_{M_j}} \quad , j = 1, ...m$$

In one Epoch, we perform the following with $j = 1, ..., m$ times:

$$\begin{aligned} w_{k+1} &\leftarrow w_k - \eta \left.\frac{\partial C}{\partial w_k}\right|_{\mathbf{x}_{M_j}} \\ b_{k+1} &\leftarrow b_k - \eta \left.\frac{\partial C}{\partial b_k}\right|_{\mathbf{x}_{M_j}} \end{aligned} \tag{1}$$

Outline

Weights Initialisation

Initially, we use

$$w \sim \mathcal{N}(0, 1)$$

But with large number of w, we have for $z = w^T x + b$

$$z \sim \mathcal{N}\left(0, \sqrt{N}\right)$$

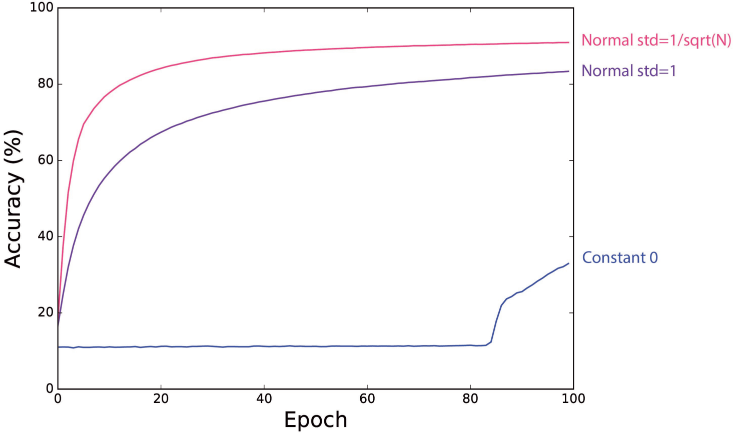
where z have higher chance of taking on large values in the first iteration. Therefore we let

$$w \sim \mathcal{N}\left(0, \frac{1}{\sqrt{N}}\right)$$

so that

$$z \sim \mathcal{N}(0, 1)$$

Weights Initialisation



Glorot, X., & Bengio, Y. (2010, May). *"Understanding the difficulty of training deep feedforward neural networks. "* In Aistats (Vol. 9, pp. 249-256).

Outline

1. Initialization
   - Weights Initialisation
   - Hyperparameters
     - Architecture
     - Learning Rate & Regularizer
     - Mini-batch Size
2. Other Gradient Descents
3. Other Regulariser

Outline

1. Initialization
   - Weights Initialisation
   - Hyperparameters
     - Architecture
     - Learning Rate & Regularizer
     - Mini-batch Size
2. Other Gradient Descents
3. Other Regulariser

Setting Up architecture

- Reduce problem to speed up learning
  - Classify from $\{0, 1\}$ instead of classifying $\{0, .., 9\}$
- Start with low number of hidden layers
- Monitor validation accuracy (start with 20% of training data)
- Overfit.

Eldan, R. and Shamir, O., 2015. *"The Power of Depth for Feedforward Neural Networks."* arXiv preprint arXiv:1512.03965.

Hyperparameters

$$
\begin{aligned}
w^l_{i,j} &: \quad \text{Weights} \\
b^l_{i,j} &: \quad \text{Biases} \\
m &: \quad \text{Mini-Batch Size} \\
\eta &: \quad \text{Learning Rate} \\
\lambda &: \quad \text{Regularizer}
\end{aligned}
$$

Outline

Learning Rate & Regularizer

- Start with setting $\lambda = 0$
- Increase/Decrease η by a factor of 10 to find out a good order of magnitude
- Fine-tune both λ & η once a good order has been established

Outline

Mini-batch Size

- If too small, GPU not sufficiently used
- If too large, weights not updated often enough

Outline

1. Initialization
2. Other Gradient Descents
   - SGD
   - Momentum
   - Nesterov Momentum
   - AdaGrad
   - RMSProp
   - Adam
3. Other Regulariser

Stochastic Gradient Descent

Randomly sample into m mini-batches of size k:

$$\mathbf{x} = \left\{ \{\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, ..., \mathrm{x}^{(k)}\}, \{\mathrm{x}^{(k+1)}, \mathrm{x}^{(k+2)}, ..., \mathrm{x}^{(2k)}\}, ..., \{\mathrm{x}^{(n-k+1)}, ..., \mathrm{x}^{(n)}\} \right\}$$
$$= \{M_1, M_2, ..., M_m\}$$

$$\left.\frac{\partial C}{\partial w}\right|_{\mathbf{x}} \approx \left.\frac{\partial C}{\partial w}\right|_{\mathbf{x}_{M_j}} \quad , j = 1, ...m$$

In one Epoch, we perform the following with $j = 1, ..., m$ times:

$$\begin{aligned} w_{k+1} &\leftarrow w_k - \eta \left.\frac{\partial C}{\partial w_k}\right|_{\mathbf{x}_{M_j}} \\ b_{k+1} &\leftarrow b_k - \eta \left.\frac{\partial C}{\partial b_k}\right|_{\mathbf{x}_{M_j}} \end{aligned} \tag{2}$$

Outline

Momentum

Idea: Introduce friction/velocity component.

$$
\begin{aligned}
v_{k+1} &= \mu v_k - \eta \frac{\partial C}{\partial w} \\
w_{k+1} &= w_k + v_{k+1} \\
&= (w_k + \mu v_k) - \eta \frac{\partial C}{\partial w}
\end{aligned}
$$

where $0 < \mu < 1$ correspond to friction/velocity magnitude.

Outline

1. Initialization
2. Other Gradient Descents
   - SGD
   - Momentum
   - Nesterov Momentum
   - AdaGrad
   - RMSProp
   - Adam
3. Other Regulariser

Nesterov Momentum

Idea: Use gradient ONLY after travelling along μv_k,

$$
\begin{aligned}
v_{k+1} &= \mu v_k - \eta \left.\frac{\partial C}{\partial w}\right|_{w = w_k + \mu v_k} \\
w_{k+1} &= w_k + v_{k+1} \\
&= (w_k + \mu v_k) - \eta \left.\frac{\partial C}{\partial w}\right|_{w = w_k + \mu v_k}
\end{aligned}
$$

Let $\widehat{w_k} = w_k + \mu v_k$,

$$
\widehat{w_{k+1}} = \widehat{w_k} - \mu v_k + (1 + \mu) v_{k+1}
$$

Outline

AdaGrad

Idea: Store all past gradients,

$$
\begin{aligned}
c_{k+1} &= c_k + \left(\frac{\partial C}{\partial w_k}\right)^2 = \sum_{i=0}^{k} \left(\frac{\partial C}{\partial w_k}\right)^2 \quad \text{(sum of squares: second moment)} \\
w_{k+1} &= w_k - \frac{\eta}{\sqrt{c_{k+1}}} \frac{\partial C}{\partial w_k}, \quad \text{Learning rate decay} \\
&= w_k - \eta \frac{\partial C}{\partial w_k} \cdot \frac{1}{\sqrt{c_{k+1}}}, \quad \text{Entry-wise division}
\end{aligned}
$$

- (+) Helps in learning rate decay.
- (+) Control amplitude of descent (not direction!).
- (-) Effective gradient converges to 0 too early.

Outline

RMSProp

Idea: Introduce decaying rate α in storing of past gradients,

$$
\begin{aligned}
c_{k+1} &= \alpha c_k + (1-\alpha)\left(\frac{\partial C}{\partial w_k}\right)^2 \\
&= (1-\alpha)\sum_{i=0}^{k}\alpha^{k-i}\left(\frac{\partial C}{\partial w_k}\right)^2 \\
w_{k+1} &= w_k - \eta\frac{\partial C}{\partial w_k}\cdot\frac{1}{\sqrt{c_{k+1}}}
\end{aligned}
$$

- Does not lead to zero gradient update in the long run.
- Default value: $\alpha = 0.9$.

Outline

1. Initialization
2. Other Gradient Descents
   - SGD
   - Momentum
   - Nesterov Momentum
   - AdaGrad
   - RMSProp
   - Adam
3. Other Regulariser

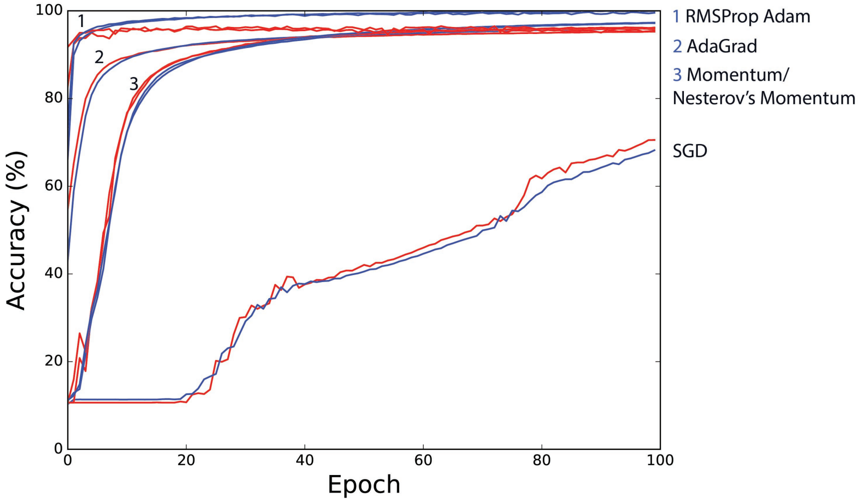
Adam

Idea: Incoporate first moments v_{k+1} and second moments c_{k+1},

$$
\begin{aligned}
v_{k+1} &= \beta_1 v_k + (1-\beta_1)\frac{\partial C}{\partial w_k} = (1-\beta_1)\sum_{i=0}^{k}\beta_1^{k-i}\frac{\partial C}{\partial w_k},\\
c_{k+1} &= \beta_2 c_k + (1-\beta_2)\left(\frac{\partial C}{\partial w_k}\right)^2 = (1-\beta_2)\sum_{i=0}^{k}\beta_2^{k-i}\left(\frac{\partial C}{\partial w_k}\right)^2,\\
w_{k+1} &= w_k - \eta\frac{v_{k+1}}{\sqrt{c_{k+1}}}\\
&= w_k - \eta\frac{\beta_1 v_k + (1-\beta_1)\frac{\partial C}{\partial w_k}}{\sqrt{\beta_2 c_k + (1-\beta_2)\left(\frac{\partial C}{\partial w_k}\right)^2}}.
\end{aligned}
$$

Typically, $\beta_1 \approx 0.9$, $\beta_2 \approx 0.999$.

Comparison



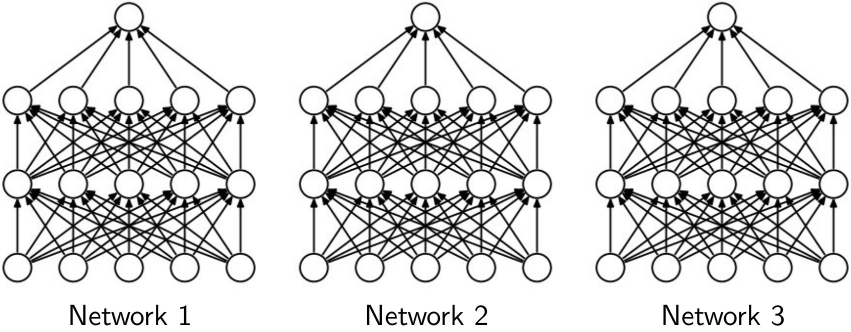
100
80
60
40
20
0
1
2
3
Accuracy (%)
0
20
40
60
80
100
Epoch
1 RMSProp Adam
2 AdaGrad
3 Momentum/
Nesterov's Momentum
SGD

References

- Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. (2013). *"On the importance of initialization and momentum in deep learning."* ICML (3), 28, 1139-1147.
- Duchi, J., Hazan, E., & Singer, Y. (2011). *"Adaptive subgradient methods for online learning and stochastic optimization."* Journal of Machine Learning Research, 12(Jul), 2121-2159.
- T. Tieleman & G. E. Hinton. Lecture 6.5-rmsprop: *"Divide the gradient by running average of its recent magnitude."* , 2012
- Kingma, D., & Ba, J. (2014). *"Adam: A method for stochastic optimization."* arXiv preprint arXiv:1412.6980.
- *"CS231n Winter 2016: Lecture 6: Neural Networks Part 3/Intro to ConvNets"*

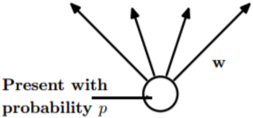
Outline

Ensemble



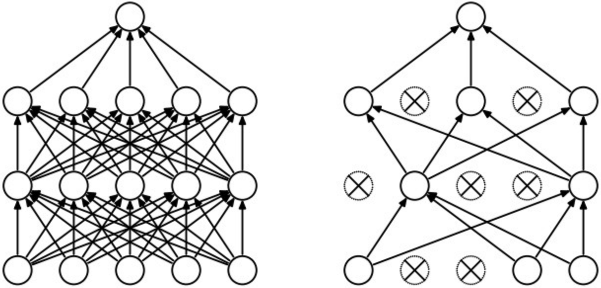
Train each network with different weights an hyperparameter initialisations.
Compute probability of each class for each data.
Taking their average improves accuracy by about 2% (empirically).

Dropout



(a) At training time

(b) At test time


(a) Standard Neural Net

(b) After applying dropout.

Dropout

Method	Testing Error
mlpconv + Fully Connected	11.59%
mlpconv + Fully Connected + Dropout	10.88%
mlpconv + Global Average Pooling	10.41%

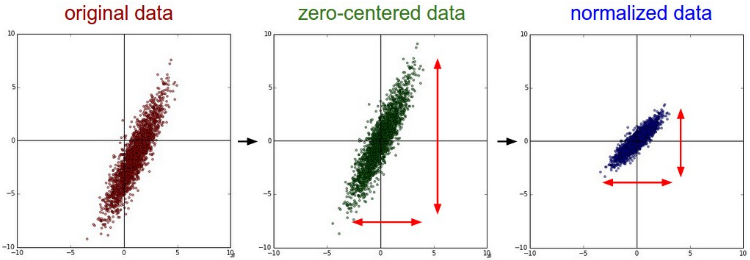
References

Srivastava, Nitish, et al. *"Dropout: a simple way to prevent neural networks from overfitting."* Journal of Machine Learning Research 15.1 (2014): 1929-1958.

Outline

Outline

Problem



- Internal Covariate Shift
  - Consider layer a^l,

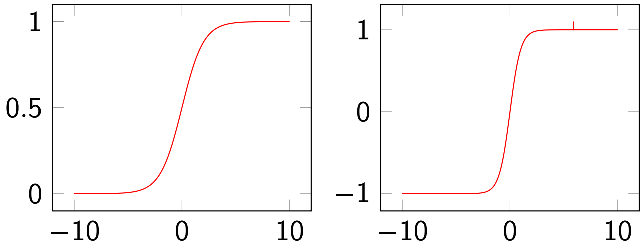
$$a^l = g\left(W^l \cdot a^{l-1} + b^l\right)$$

$$W^l_{k+1} = W^l_k - \eta \frac{\partial C}{\partial W^l_k}$$

  - Distribution of the a^{l-1} changes every epoch, difficult to learn
  - When there is fixed distribution in a^{l-1}, W^l does not have to readjust.

Outline

Batch Normalization



- Batch Normalisation
  - Whitening z^l before committing $a^l = g(z^l)$.
  - $a^l = g(\widehat{z^l})$ where $\widehat{z^l} = norm(z^l)$.
  - $y^l = \gamma^l \widehat{z^l} + \beta^l$
    to prevent $\sigma\left(\widehat{z^l}\right)$ or $\tanh\left(\widehat{z^l}\right)$ being stuck at the linear region
  - Extra terms to learn in each batch norm layer γ^l, β^l.

Backpropagation:

$$\frac{\partial C}{\partial \hat{z}_i^l} = \frac{\partial C}{\partial y_i^l}\gamma^l$$

$$\frac{\partial C}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial C}{\partial \hat{z}_i^l} \cdot (z_i^l - \mu_B) \cdot \frac{-1}{2}(\sigma_B^2)^{-\frac{3}{2}}$$

$$\frac{\partial C}{\partial \mu_B} = \left(\sum_{i=1}^{m} \frac{\partial C}{\partial \hat{z}_i^l} \cdot \frac{-1}{\sqrt{\sigma_B^2}}\right) + \frac{\partial C}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^{m} 2(\mu_B - z_i^l)}{m}$$

$$\implies \frac{\partial C}{\partial y_i^l} = \left[\frac{\partial C}{\partial \hat{z}_i^l} \cdot \frac{-1}{\sqrt{\sigma_B^2}}\right] + \left[\frac{\partial C}{\partial \sigma_B^2} \cdot \frac{2(z_i^l - \mu_B)}{m}\right] + \left[\frac{\partial C}{\partial \mu_B} \cdot \frac{1}{m}\right]$$

Batch Normalization

Backpropagation:

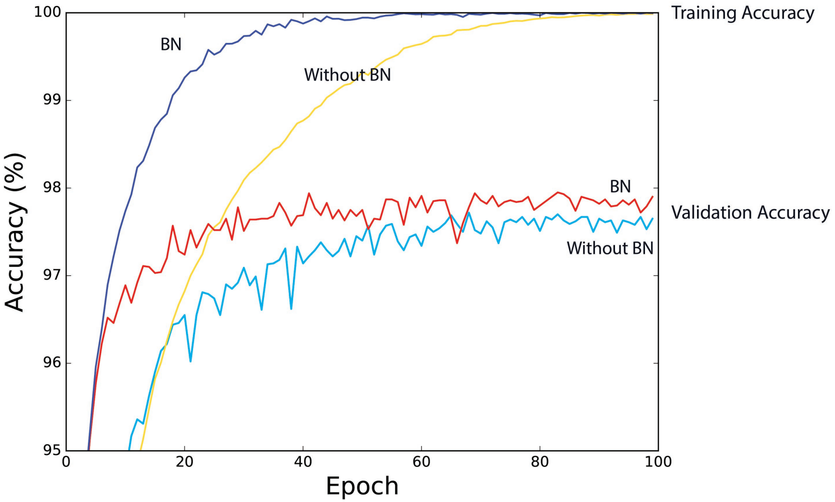
$$\frac{\partial C}{\partial \gamma^l} = \sum_{i=1}^{m} \frac{\partial C}{\partial y_i^l} \widehat{z}_i^l$$

$$\frac{\partial C}{\partial \beta^l} = \sum_{i=1}^{m} \frac{\partial C}{\partial y_i^l}$$

$$\frac{\partial C}{\partial w^l} = \frac{\partial C}{\partial z_i^l} a^{l-1}$$

Ioffe, S., & Szegedy, C. (2015). *"Batch normalization: Accelerating deep network training by reducing internal covariate shift."* arXiv preprint arXiv:1502.03167.

Batch Normalization



Batch Normalization

- Benefits
  - Larger learning rate η can be used.
  - Accelerate η decay due to greater training speed.
  - Acts as a regulariser.
    - Can reduce Dropout strength.
    - Can reduce λ in L1 or L2 weights penalty.
  - Reduces saturation.

Thank You