Lesson 09
Convolutional Neural Network - Advanced Techniques

Ing. Marek Hrúz, Ph.D.

Katedra Kybernetiky
Fakulta aplikovaných věd
Západočeská univerzita v Plzni
Nesterov Momentum

- This method is already well established and recommended to use
- The idea is – first go to the point where we should end up (the momentum gradient direction $v^t$)
- Then correct the estimate by computing the gradient in the “look-ahead” point

$$\omega^{t+1} = \omega^t + v^t - \epsilon \cdot \nabla L (\omega^t + v^t)$$

(1)

- where $v^t = \alpha \cdot v^{t-1} - \beta \cdot \epsilon \cdot \omega^{t-1} - \epsilon \cdot \nabla L (\omega^{t-1})$ is the momentum
- For comparison with classical momentum:

$$\omega^{t+1} = \omega^t + v^{t+1}$$

(2)
Nesterov Momentum vs. Momentum

Momentum update

- momentum step
- gradient step
- actual step
Nesterov Momentum vs. Momentum

Nesterov momentum update

- momentum step
- actual step
- “lookahead” gradient step (bit different than original)
Per-parameter adaptive learning rate methods

- There are drawbacks of static learning rates
- The learning rate is also global - bad
- There is a way to estimate the learning rate from the gradient estimate
- The gradient is the direction and rate of the largest growth of a function
- But we apply the gradient in the **domain** of the function
Gradient visualization

\[ f(x) = x^2 \]

\[ f'(-0.6) = -1.2 \]
Gradient application
Gradient visualization

$f(x) = 0.4x^2$

$f'(-0.6) = -0.48$
Gradient application
Per-parameter adaptive learning rate methods

- A steep function has a large gradient - but steepness (intuitively) means that we are close to local extreme - the step should be small
- A shallow function has a small gradient - but shallowness (intuitively) means that we are far from extreme - the step should be large
- We can make use of this - we apply an adaptive learning rate
Adagrad

- We introduce an parameter of cumulative squared gradient magnitudes:

\[ \sigma_i = \sigma_i + \|\nabla J (\omega_i^t) \|^2 \]

\[ \omega_i^{t+1} = \omega_i^t - \frac{\epsilon}{\sqrt{\sigma_i + \xi}} \nabla J (\omega_i^t) \]

- where \( \xi \) is a small constant, \( \omega_i^t \) is a vector of weights of \( i^{th} \) neuron

- The method is aggressive and updates of gradients go to zero (since \( \sigma_i \) always grows)
Adadelta & RMSProp

- Adadelta is the reaction to weak point of Adagrad (dying updates)
- The always growing magnitude of gradient history is replaced by a running average

\[ \sigma_i = \gamma \sigma_i + (1 - \gamma) \| \nabla J (\omega^t_i) \|^2 \]  \hspace{1cm} (5)

- The learning rate is also approximated (we want it to have the same hypothetical units as gradient) as a running average of parameter updates

\[ \epsilon^{t+1} = \gamma \epsilon^t + (1 - \gamma) \Delta \omega^t_i \]  \hspace{1cm} (6)

- Then we can write the update as

\[ \omega^{t+1}_i = \omega^t_i - \frac{\sqrt{\epsilon^t + \xi}}{\sqrt{\sigma_i + \xi}} \nabla J (\omega^t_i) \]  \hspace{1cm} (7)

- Independently discovered RMSProp neglects the learning rate approximation
Adam

- Adaptive Moment Estimation (Adam) additionally approximates the running average of non-squared gradients (first moment)

\[ m_i = \beta_1 m_i + (1 - \beta_1) \nabla J(\omega^t_i) \]  

(8)

- The running average of squared gradient magnitudes is kept (second moment)

\[ v_i = \beta_2 v_i + (1 - \beta_2) \| \nabla J(\omega^t_i) \|^2 \]  

(9)

- The update becomes:

\[ \omega_{i+1} = \omega_i - \frac{\epsilon}{\sqrt{v_i + \xi}} m_i \]  

(10)
Object Detecting Networks - time-lapse

- **DPM**
  - FPS: 0.5
  - mAP: 34.3

- **R-CNN**
  - FPS: -
  - mAP: 58.5

- **Fast R-CNN**
  - FPS: 0.5
  - mAP: 70

- **Faster R-CNN**
  - FPS: 7
  - mAP: 73.2

- **YOLO**
  - FPS: 45
  - mAP: 63.4

- **SSD**
  - FPS: 58
  - mAP: 72.1

Time:
- Nov 2013
- Apr 2015
- June 2015
- Dec 2015

Results on test sample Pascal VOC 2007. Training on trainval sets 2007+2012
Region based CNN

- R-CNN
- Propose regions - selective search, that is merging of super pixels (2 sec per image)
- Extracts features from warped proposals
- Train per class SVM on CNN features
Region Proposal Network

- Faster R-CNN
- Two modules that share parameters:
  - Region proposal model
  - Classification module
- Region proposal - a small CNN is applied to the last layer of a pre-trained CNN (VGG-16; $14 \times 14 \times 512$)
  - The small CNN is a convolutional layer (with $n \times n$ kernels) followed by a fully connected layer (512 dim)
  - After that there are two FC layers - $2k$ neurons for objectiveness and $4k$ neurons for location
Region Proposal Network

- 2k scores
- 4k coordinates

cls layer
reg layer

256-d

intermediate layer

sliding window
conv feature map

k anchor boxes
Region Proposal Network

- $k$ is the number of anchor boxes
- The anchors have different sizes and aspect ratios
- In original paper they use 3 sizes and 3 ratios
- The outputs of the $2k$ and $4k$ FC layers are relative to the position and size of the appropriate anchor box
Region Proposal Network - learning

- The targets are deduced from the ground truth data (annotated regions of objects)
- Each anchor that has Intersection over Union (IoU) greater than 0.7 is considered a positive sample
- The objectiveness of such anchor (in $2k$ FC layer) is set to positive
- The regression of the anchor (position and size) is computed from the ground truth region
- Negative anchors are such that have IoU lower than 0.3
- The loss is defined as:

$$L\left(\{p_i\}, \{t_i\}\right) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p^*_i) + \lambda \frac{1}{N_{reg}} \sum_i p^*_i L_{reg}(t_i, t^*_i)$$

(11)

- $L_{cls}$ is a 2 dim softmax, $L_{reg}$ is a smoothed $L^1$-norm
Region Recognition - Detection Network

- Uses Fast R-CNN
- The weights are shared - the same features are used for proposal and detection
- Each proposed region is recognized - but watch out! the regions have different sizes
- That is why you need to use the ROI max-pooling
- This is quite hard to implement and is deprecated by Single Shot Multi-Box Detector
- ... thus, need not to learn (YEAH!)
Region Recognition - Detection Network

Region Proposal Network

proposals

RoI pooling

classifier

feature maps

conv layers

image
Single Shot Multi-Box Detector (SSD)
Single Shot Multi-Box Detector (SSD)
Single Shot Multi-Box Detector (SSD)

Input Image

300x300x3

VGG up to conv4_3

38x38x512

VGG up to fc7

Normalization

Detector & classifier 1

Detector & classifier 2

19x19x1024

Conv layers

10x10x512

Conv layers

5x5x256

Conv layers

3x3x256

Avg pooling

1x1x256

Detector & classifier 3

Detector & classifier 4

Detector & classifier 5

Detector & classifier 6
Single Shot Multi-Box Detector (SSD)

Input Image

300x300x3

VGG up to conv4_3

38x38x512

VGG up to fc7

19x19x1024

Conv layers

10x10x512

Conv layers

5x5x256

Conv layers

3x3x256

Avg pooling

1x1x256

Normalization

Detector & classifier 1

Detector & classifier 2

Detector & classifier 3

Detector & classifier 4

Detector & classifier 5

Detector & classifier 6

Fast Non-Maximum Suppression (Fast NMS)

Final detections
Detection and Classification

Diagram:
- Default box generator
- Conv 3x3 (localization)
- Conv 3x3 (confidence)
- 75 default boxes
- 5x5x12
- 5x5x63
- Detector & classifier

Mathematical expressions:
- \[ P(\text{car}) = 0.6 \]
- \[ P(\text{car}) = 0.7 \]
Default Boxes

location of center of anchors

Input Image

anchors

feature maps
Default Boxes

Input Image

- 300 x 300
- 168 x 168
- 168/\sqrt{2}
- 168*\sqrt{2}

Grid
- 5 x 5
Default Boxes

Input Image

300

168

168/√2

168√2

168∗√2

168/√2

5

5

5
SSD - all together

Default boxes

Localization

Confidence

Input Image (300 x 300)
SSD - all together

Default boxes

Localization

Confidence

Input Image (300 x 300)
Training of SSD

- Each ground truth box is matched to the default box (anchor) with the best Jaccard overlap (IoU).
- Each GT box thus has only one matching default box.
- This is extended by adding default boxes with at least 0.5 Jaccard overlap.
- From these matches the deltas of $x$, $y$, width, and height are computed and also the classifier gets its label.
Training Objective

\[
L(x, c, l, g) = \frac{1}{N} (L_{\text{conf}}(x, c) + \alpha L_{\text{loc}}(x, l, g)) \tag{12}
\]

- \(x\) is the predicted class, \(c\) is the GT class, \(l\) is the regressed deltas of default boxes, \(g\) is the GT deltas
- \(N\) is the number of matched default boxes
- \(L_{\text{loc}}\) is a smooth \(L^1\) loss, \(L_{\text{conf}}\) is softmax

\[
L_{\text{loc}} = \sum_i \text{smooth}_{L1}(l_i - g_i) \tag{13}
\]

\[
\text{smooth}_{L1}(x) = \begin{cases} 
0.5x^2 & \text{if } |x| < 1 \\
|x| - 0.5 & \text{otherwise}
\end{cases} \tag{14}
\]
Hard negative mining

- After matching a lot of default boxes will be negatives (background class)
- This gives an imbalanced training set with lots of default boxes
- The negatives are ranked according the confidence
- Only a portion of the negatives if chosen (3:1) for the gradient update

Data augmentation:
- Use the entire original image
- Sample the original image so that the patches have IoU with the object at least 0.1, 0.3, 0.5, 0.7, or 0.9
- Randomly sample the image into patches