Advanced Video Analysis & Imaging (5LSH0), Module 09B

Outline

- Program a Neural Network
  - Define a neural network
  - Auto-differential libraries
    - Dataflow graph.
    - Current auto-differential libraries

- Case studies
  - Image classification
  - Image segmentation
  - Object detection

- Summary

Program a Neural Network

- Define a neural network
  - Define network architecture: operations and layers (e.g. fully-connected? convolution? recurrent?)
  - Define the data I/O: read what data from where?
  - Define a loss function/optimization objective: L2 loss? Softmax?...
    - Define an optimization algorithm: SGD? Momentum SGD? etc.

- Auto-differential Libraries will then take over
  - Dataflow graph.
  - Current auto-differential libraries

- Adapting the architecture to specific needs, change connectivity patterns, attach specialized layers and etc.
- Most of publications in the field, concentrate on designing a new architecture (In case studies section, some well-known architectures for image analysis will be introduced)
Program a Neural Network

- Define a neural network
  - Define network architecture: operations and layers (e.g. fully-connected? convolution? recurrent?)
  - Define the data I/O: read what data from where?
    - Define a loss function/optimization objective: \( L^2 \) loss? Softmax?
    - Define an optimization algorithm: SGD? Momentum SGD? etc.

- Auto-differential Libraries will then take over
  - Dataflow graph.
  - Current auto-differential libraries

---

Define the data I/O: read what data from where?

- Data Input:
  - Using whole input image frame, cropping patches, multi frames (e.g. video)...
  - Data sampling strategy: random, candidate selection, hard mining, ...
  - Using data augmentation: Spatial transformation, spatial distortion, color distortion, fancy PCA, ...

- Data output:
  - What is objective? classification, regression, clustering,...
  - Using single label per image or a labeled bags (multiple-instance learning)
  - Using only unlabeled data (unsupervised learning) or along with labeled data (Semi-supervised learning)
  - Using fine-annotated data along with coarse annotated instances (Weakly supervised)

A supervised learning problem measures the compatibility between a prediction (e.g. the class scores in classification) and the ground truth label

- The data loss takes the form of an average over the data losses for every individual example

\[
L = \frac{1}{N} \sum_{i} L_i
\]

Number of training samples
Program a Neural Network

- Define a loss function/optimization objective.

- Examples of classification loss
  \[ L_i = \frac{1}{N} \sum_i L_i \]
  \[ f = f(x_i; \theta) \]
  \[ L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1) \quad \text{SVM} \]
  \[ L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)^2 \quad \text{Squared hinge loss} \]
  \[ L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_{j} e^{f_j}}\right) \quad \text{Cross entropy (softmax)} \]

- Define a neural network
  - Define network architecture: operations and layers (e.g. fully-connected? convolution? recurrent?)
  - Define the data I/O: read what data from where?
  - Define a loss function/optimization objective: L2 loss? Softmax? ...
  - Define an optimization algorithm: SGD? Momentum SGD? etc.

- Auto-differential Libraries will then take over
  - Dataflow graph.
  - Current auto-differential libraries

Gradient Descent (GD) Variants

- Batch GD
- Stochastic GD (SGD)
- Mini-batch GD
- SGD+Momentum
- Adam
- RMSprop
- Adagrad
- AdaDelta
- AdaMax
- Nadam
- ...

* We will not discuss algorithms that are infeasible to compute in practice for high-dimensional data sets, e.g. second-order methods such as Newton method.

Further reading: [https://ruder.io/optimizing-gradient-descent/](https://ruder.io/optimizing-gradient-descent/)

Problems with Gradient Descent (GD)

- Adjusting proper learning rate can be difficult.
  - If set too small: it leads to painfully slow convergence.
  - If set too large: it can hinder convergence and causes the loss function to fluctuate around the minimum or even diverge.

- Predefined learning rate ($\eta$) schedule.
  - Strategy 1: changing $\eta$ according to the number of running epochs
  - Strategy 2: changing $\eta$ w.r.t the objective function values
  - Problem: unable to adapt to the data characteristics

- Same learning rate applies to all parameter updates.
  - If data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.

Gradient Descent (GD) Variants

- How much data we use to compute the gradient?
  - Batch GD: The entire training dataset
    \[ \theta = \theta - \eta \nabla f(\theta) \]
  - Mini-batch GD: A part of dataset
    \[ \theta = \theta - \eta \nabla f(\theta; x^{(t)}; y^{(t)}) \]
  - Stochastic GD (SGD): Only 1 sample
    \[ \theta = \theta - \eta \nabla f(\theta; x^{(t)}; y^{(t)}) \]

- SGD + Momentum

  Gradient descent without momentum (left), with momentum(right)

  \[ \nu_t = \gamma \nu_{t-1} + \eta \nabla J(\theta) \]
  \[ \theta = \theta - \nu_t \]

  $\nu_t$ is the update vector at time step $t$ and $\gamma$ is momentum term.
Gradient Descent (GD) Variants

- Visualization – Noisy moons data example
  Smoothing effects of momentum-based techniques (which also results in overshooting and correction).

- Visualization – Beale’s function example
  Due to the large initial gradient, velocity-based techniques shoot off and bounce around.

- Visualization – Long valley example
  Effect of scaling-based on gradient information

- Visualization – Saddle point example
  Notice here that SGD and Momentum find it difficult to break the symmetry.
Gradient Descent (GD) Variants

Goal:
• Move quickly in directions with small but consistent gradients.
• Move slowly in directions with big but inconsistent gradients.

Solution:
Adaptively tune learning rate based on observed behavior (e.g. Adam, RMSProp,...)

• **RMSprop**: dividing the learning rate for weight by a running average of the magnitudes of recent gradients for that weight

\[
E[g^2]_t = \gamma . E[g^2]_{t-1} + (1 - \gamma) . g_t^2
\]

\[
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} . g_t
\]

Typical values: \( \gamma = 0.9, \eta = 10^{-3} \)

Program a Neural Network

• Define a neural network
  • Define network architecture: operations and layers (e.g. fully-connected? Convolution? Recurrent?)
  • Define the data I/O: read what data from where?
  • Define a loss function/optimization objective: \( L2 \) loss? Softmax? Ranking Loss?
  • Define an optimization algorithm: SGD? Momentum SGD? etc.

• Auto-differential Libraries will then take over
  • Dataflow graph.
  • Current auto-differential libraries
A Computational Layer in DL

- A layer in a neural network is composed of a few finer computational operations
  - A layer \( l \) has input \( x \) and output \( z \), and transforms \( x \) into \( z \) following:
    \[ y = Wx + b, \quad z = \text{ReLU}(y) \]
  - Denote the transformation of layer \( l \) as \( f_l \), which can be represented as a dataflow graphs: the input \( x \) flow though the layer

\[ x \rightarrow f_l \rightarrow z \]

From Layers to Network

- Training the neural network involves deriving the gradient of its parameters with a backward pass (next slides)

Back-propagation through a NN

- The backward computation proceeds by sequentially executing \( f_1, f_2, f_3, \ldots, f_L \)

\[ f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow \ldots \rightarrow f_L \]

- Denote the backward pass through a layer \( l \) as \( b_l \)
  - \( b_l \) derives the gradients of the input \( x(\text{d}x) \), given the gradient of \( z \) as \( \text{d}z \), as well as the gradients of the parameters \( W, b \)
  - \( \text{d}x \) will be the backward input of its previous layer \( l - 1 \)
  - Backward pass can be thought as a backward dataflow where the gradient flow through the layer

\[ dx \rightarrow dz \rightarrow b_l \]
A Layer as a Dataflow Graph

• Give the forward computation flow, gradients can be computed by auto differentiation
  • Automatically derive the backward gradient flow graph from the forward dataflow graph

A Network as a Dataflow Graph

• Gradients can be computed by auto differentiation
  • Automatically derive the gradient flow graph from the forward dataflow graph

Gradient Descent via Back-propagation

• The computational workflow of deep learning
  • Forward, which we usually also call inference: forward dataflow
  • Backward, which derives the gradients: backward gradient flow
  • Apply/update gradients and repeat

Mathematically

\[ \theta^{(t)} = \theta^{(t-1)} + \varepsilon \cdot \nabla_L(\theta^{(t-1)}, D^{(t)}) \]

Model parameters Forward Data

Auto-differential Libraries

• Auto-differential Library automatically derives the gradients following the back-propagation rule.
• A lot of auto-differentiation libraries have been developed:
• So-called Deep Learning toolkits
Deep Learning Toolkits

- They are adopted differently in different domains
- For example

![Caffe](image1)
![TensorFlow](image2)
![CNTK](image3)

![PyTorch](image4)
![DyNet](image5)
![Chainer](image6)

Imperative: immediate evaluation
Symbolic: write symbols to assemble the networks first, evaluate later

Deep Learning Toolkits

- They are also designed differently
  - Symbolic vs. imperative programming

<table>
<thead>
<tr>
<th>Symbolic</th>
<th>Imperative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good: easy to optimize (e.g., distributed, batching, parallelization) for developers</td>
<td>Good: more flexible: write one line, evaluate one line</td>
</tr>
<tr>
<td>Bad: the way of programming might be counter-intuitive</td>
<td>Bad: less efficient</td>
</tr>
<tr>
<td>Hard to debug for user programs</td>
<td>More difficult to optimize</td>
</tr>
<tr>
<td>Less flexible: you need to write symbols before actually doing anything</td>
<td></td>
</tr>
</tbody>
</table>

A = Variable('A')
B = Variable('B')
C = B * A
D = C + Constant(1)
# compiles the function
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)

import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
Deep Learning Toolkits

- They are also designed differently
  - dataflow graphs vs. layer-by-layer construction

Good and Bad of Dataflow Graphs

- Dataflow graphs seem to be a dominated choice for representing deep learning models
  - What's good for dataflow graphs
    • Good for static workflows: define once, run for arbitrary batches/data
    • Programming convenience: easy to program once you get used to it.
    • Easy to parallelize/batching for a fixed graph
    • Easy to optimize: a lot of off-the-shelf optimization techniques for graph
  - What's bad for dataflow graphs
    • Not good for dynamic workflows: need to define a graph for every training sample
    • Hard to program dynamic neural networks: how can you define dynamic graphs using a language for static graphs? (e.g. LSTM, tree-LSTM).
    • Not easy for debugging.
    • Difficult to parallelize/batching across multiple graphs: every graph is different, no natural batching.

Case Studies

* Image Classification
  - AlexNet (2012)
  - VGGNet (2013)
  - GoogleNet (2014)
  - Residual Net (2015)

* Image Segmentation
  - Fully Convolutional Networks (2013)
  - SegNet (2015)

* Object Detection
  - RCNN/ Fast, Faster RCNN (2014, 2015)

* Recommended to read the papers for details.
* Image Classification – example I
  
  - **AlexNet** (2012)
  
  - Data augmented with image crops of 224 x 224 and horizontal flips.
  - RGB pixel intensities altered with principal components. Refer paper for more details.
  - The authors trained on two GPUs hence splitting the outputs of the convolutions (also known as grouped convolutions) in the diagram above.

* Image Classification – example I (continue)
  
  - ReLUs are applied after each convolution and full connection (except the one before softmax).
  - They also added Local Response Normalization (not covered in the course) before Max Pooling 1 and 2.
  - Local Response Normalization had minimal performance gains and are rarely (if not never) used in practice. Refer the paper to read more.
  - Top-5 error rate on test set 15.3% vs 26.2% previous best.

* Image Classification – example II
  
  - **VGGNet** (2013) – Very Deep Convolutional Networks for Large Scale Image Recognition
  
  - Two variants - VGG16 and VGG19.
  - Used only 3 x 3 kernels. Stacking multiple 3 x 3 kernels can result in larger receptive field while using less number of parameters.
  - For example stacking two 3 x 3 kernels can result in an effective receptive field of 5 x 5.
  - Two 3 x 3 kernels result in 18 parameters where as a single 5 x 5 kernel results in 25.

* Image Classification – example II (continue)
  
  - VGG16 – 16 layers deep, VGG19 – 19 layers deep.
  - Stacked multiple 3 x 3 convolution layers together to make deeper network.
  - Each convolution layer and fully connected layer (except one before softmax) followed by ReLU.
  - For architecture details and hyper parameter settings refer paper.
  - Top-5 error rate 6.8%
### Image Classification – example III

- **GoogleNet (2014)** – Going Deeper with Convolutions
  - Introduced inception module. (left: naive version, right: with dimensionality reduction)
  - Naive version – concatenation of outputs of 1 x 1, 3 x 3, 5 x 5 convolutions and max pooling.
  - Inception with Dimensionality reduction – 1 x 1 convolutions with lower number outputs before 3 x 3 and 5 x 5 convolution.

- Before outputs are concatenated, they are padded so as to maintain same size (width and height).
- Inception modules stacked over one another – 22 layers deep.
- Average pooling before last fully connected layer.
- Dropout of 0.7 after average pooling.
- For architecture details and clearer view of GoogleNet, refer the paper.
- Top – 5 error rate 6.67% on test set.
- Other variants - Inception V2, V3, V4

### Image Classification – example IV

  - Introduced Residual modules, building blocks of ResNet.
  - Degradation - In very deep networks, both training and test error increases as network gets deeper (not overfitting or underfitting).
  - Solution - add identity mapping to the output.
  - No extra parameters needed.
  - 34 layers plain and residual network on the right.

- Residual block. (left: residual block, right: residual block with bottleneck)
  - Defined as $x_{l+1} = x_l + f(x_l)$ where $x_{l+1}$ is the $l + 1$th layer, $x_l$ is the $l$th layer and $f(x_l)$ is the applied transformation.
  - Easier to propagate gradients through shorter connections.
  - Residual Block with bottleneck used in architectures with 50+ layers.
  - Note: Batch Normalization applied after each convolution.
**Image Classification**
- We were able to train a 1202-layer ResNet without degradation problems but overfits.
- Top-5 error rate of 4.49% (single model ResNet - 152).
- Later ablation studies showed that ResNet is an ensemble of shallower networks.
- Residual networks do not use all paths equally for propagating gradients.
- Effective paths much shorter w.r.t. network depth.
- Other interesting reads:
  - Wide Residual Networks
  - Identity mapping in Deep Residual Networks
  - Residual Networks of Residual Networks
  - Aggregated Residual Transformations for Deep Neural Networks
  - Densely Connected Convolutional Networks (Best paper CVPR 2017)
  - Attention ResNet for Image classification
  - Squeeze and excitation Networks (ILSVRC 2017 winner)

**Image Segmentation - example I**
- Fully Convolutional Network (2013)
- Simple up-sampling has coarser results.
- Added skip connections – sum (element-wise) pooling layers after 1 x 1 convolution with transposed convolution layer outputs.
- Adding skip connection results in finer outputs
- FCN variants – FCN - 32s, FCN - 16s, FCN – 8s
- The number in the FCN variant denotes the stride taken in the last transposed convolution layer, FCN – 8s has a stride of 8 in the last convolution layer.
- Mean IoU – 62.2% on Pascal VOC 2012.
- Fully connected layers replaced with convolution layers.
- Image up-sampled using transposed convolution (also known as fractionally strided convolution or sometimes as deconvolution).
- Stacking few transposed convolution layers with ReLU can learn non-linear up-sampling.
- The backbone architecture can replaced with VGGNet, GoogleNet, ResNet etc. and then stack transposed convolution.
**Image Segmentation - example II**

- SegNet (2015)

- Encoder – Decoder structure without any fully connected layers.
- The encoder is not down-sampled vigorously due to loss of spatial resolution.
- Boundary and small object information difficult to learn in street view images.
- Pooling indices stored from encoder and restored at decoder side.
- Storing pooling indices captures boundary information before down-sampling.

**Image Segmentation - example II (continue)**

- SegNet results on Camvid dataset – 11 classes from street view images
- SegNet mean IoU – 60.1% on Camvid dataset.

**Object Detection - example I**

- RCNN (Regions with CNN features)

- Object detection - three stages
  - Object localized through region proposal method (selective search, edge boxes etc.)
  - Features extracted using CNN (either from fully connected layer or convolution layer from deeper part of the network)
  - Training linear SVMs on CNN features.

**Other segmentation architectures:**

- Learning Deconvolution Network for Semantic Segmentation
- DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs
- Conditional Random Fields as Recurrent Neural Networks
- The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation
- Pyramid Scene Parsing Network
- Stacked Deconvolutional Network for Semantic Segmentation
* **Object Detection - example I (continue)**
  
  − **RCNN (Regions with CNN features)**
  
  − Region proposal
    
    • Authors use selective search algorithm
    
    • Generates 2000 region proposals during testing
  
  − **CNN feature extraction**
    
    • Pretrained with ImageNet
    
    • Input for CNN warped to fixed size.
    
    • Train on new data for classification
  
  − **Classification using Linear SVM**
    
    • Better results with deeper layers.
    
    • Trade off with computation time, feature vector can be selected from early layers

* **Object Detection – other examples**

  − **Fast RCNN and Faster RCNN**

  − **Framework of Fast RCNN (above).**
    
    • Expensive part of RCNN is the region proposal part.
    
    • Fast and Faster RCNN tries to alleviate this problem.
    
    • Faster RCNN uses a Region Proposal Network (RPN) to propose regions.

* **Object Detection - other examples (continue)**

  − **Fast RCNN**
    
    • Object proposal performed on last convolutional layer.
    
    • Each object proposal converted to a fixed using Region of Interest (RoI)
      Pooling
    
    • For example, proposal has size of 10 x 15 pixels and the desired output needed is of 5 x 5 pixels. The pooling window takes a size of 2 x 3.
    
    • It is branched to two outputs. One to a softmax classifier and other two a
      bounding box regressor.
    
    • The network is jointly trained and the objective function is combined with the classification and regression losses

* **Object Detection - other examples (continue)**

  − **Faster RCNN** proposes regions using a
    
    Region Proposal Network (RPN) on the last
    
    convolution layer.
    
    − The RPN combined with Fast RCNN results in the Faster RCNN framework.
    
    − The convolution feature map is shared with the RPN which reduces the computational complexity
    
    − RPN is a fully convolutional network that is slid over a n x n window of the last
      convolution layer feature map.
    
    − The RoI pooling after RPN produces a fixed size output.
**Object Detection - other examples (continue)**

- At each position, N anchor boxes with 3 scales and 3 aspect ratios.
- Anchor boxes are translation invariant.
- The fixed size output is connected to two branches for classification and regression.
- The RPN and the network is trained end-to-end.
- For details how the loss functions are defined, refer the paper.

<table>
<thead>
<tr>
<th></th>
<th>RCNN</th>
<th>Fast RCNN</th>
<th>Faster RCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Average Precision</td>
<td>66.0%</td>
<td>66.9%</td>
<td>66.9%</td>
</tr>
<tr>
<td>Test time per image</td>
<td>50 sec.</td>
<td>2 sec.</td>
<td>0.2 sec.</td>
</tr>
</tbody>
</table>

**Object Detection**

- Other object detection works:
  - SSD: Single shot Multi-Box Detector
  - R-FCN: Object Detection via Region-based Fully Convolutional Networks
  - Mask R-CNN
  - Focal Loss for Dense Object Detection

**Generalization Problem of Neural Networks**

**Pitfall of Neural Networks**

Examples of image misclassification after applying a small distortion for AlexNet (all incorrectly predicted as *ostrich!*)
* Deep learning (DL) has a modular design. This makes possible presenting the network as dataflow graph.

* Auto differentiable libraries facilitates the development of DL.

* Advances in image analysis (classification, segmentation, localization,...) speeded up fast by DL.

* Reliability and stability of DL models are still a controversial issue.