Lecture 9: CNN Optimization

Bohyung Han
Computer Vision Lab.
bhhan@postech.ac.kr

Complexity of CNNs

CNN Optimization

• Motivation: huge computational costs
  ▪ Both time and space complexity are very large.
  ▪ They are proportional to the number of parameters and layers, and the size of feature maps.

• Source of complexity
  ▪ Convolutional layers
    • Slow due to many redundant operations (convolutions)
    • Easy to be parallelized
    • Filters are small in general but feature map sizes are large.
  ▪ Fully connected layers
    • Fast since the operation can be implemented by matrix-vector multiplication
    • Large memory requirement: a large number of parameters
Operations in Convolutional Neural Networks

- Convolutional layers

\[ F \ast X = (VH) \ast X = V \ast (H \ast X). \]

Low Rank Approximation

- Operations in CNNs
  - Convolutional layers: linear filtering with 3D tensors
  - Fully connected layers: simple matrix-vector multiplication
- CNN parameter approximation
  - Operations in both layers involve parameter matrices, which can be approximated by products of low-rank matrices.
  - Use of approximate matrices incur small differences in output layers.

Filter Bank Approximation

- Removing redundancy across filter banks
  - Reconstructing learned filters using a set of linearly separable filters
  - Reconfiguring learned filters using 1D and 2D filters

Filter Bank Approximation

- Objective
  - Filter reconstruction: optimizing filter itself
  - Data reconstruction: optimizing feature responses
  - Joint reconstruction: considering both filters and responses

2.5x speed-up with no loss in accuracy, 4.5x speed-up with <1% drop in accuracy

Non-Linear Filter Approximation

- Linear approximation
  - Output of a layer is approximated: $y_i = Wx_i$
  - Low-rank assumption of the output: $y_i = MWx_i$, where $\text{rank}(M) \leq d'$
  - Output $y$ is assumed to be on a low-dimensional manifold.

  $$\min_M \sum_i \| (y_i - \bar{y}) - M (y_i - \bar{y}) \|_2^2 \quad \text{such that} \quad \text{rank}(M) \leq d'$$

- Non-linear approximation
  - Approximation of ReLU together

  $$\min_{M,b} \sum_i \| r(y_i) - r(My_i + b) \|_2^2 \quad \text{such that} \quad \text{rank}(M) \leq d'$$

  - Relaxation

  $$\min_{M,b,z} \sum_i \| r(y_i) - r(z_i) \|_2^2 + \lambda \| z_i - (My_i + b) \|_2^2 \quad \text{such that} \quad \text{rank}(M) \leq d'$$

Non-Linear Filter Approximation

- Multiple layer approximation
  - Layer-by-layer approximation: prone to accumulate error
  - Asymmetric reconstruction with noisy input $\hat{x}_i$

  $$\min_M \sum_i \| r(Wx_i) - r(MW\hat{x}_i + b) \|_2^2 \quad \text{such that} \quad \text{rank}(M) \leq d'$$

Network Quantization

- Fixed point approximation
  - Quantizing both weights and activations
  - Identifying optimal fixed point bit-width allocation across layers
  - Data-driven bit-width and step size estimation: relying on Gaussian distribution assumption
  - Considering trade-off between overflow and quantization error

- Results
  - More than 20% reduction in the model size without any loss in accuracy on CIFAR-10
Network Quantization

- Quantized CNN
  - Speed-up the computation
  - Reduce the storage and memory overhead of CNN models
  - Quantize convolutional filters and weight matrices of FC layers
  - Minimize the estimation error of each layer’s response
  - 4-6× speed-up and 15-20× compression with 1% point loss of accuracy

- Main idea
  - Product quantization: initially proposed for approximate nearest neighbor search
  - Quantize subvectors independently and generate a large number of quantized vectors using Cartesian product of subvector quantizations
  - [Jegou2011TPAMI]


Network Quantization

- Results


Network Quantization

- Product quantization of weight matrix


Binary Networks

- Two versions
  - Binary-Weight-Networks: binary-valued filters
  - XNOR-Networks: binary filters and inputs for convolutions

Binary Networks

- Binary-Weight-Networks
  - Goal: \( I \ast W \approx (I \oplus B) \alpha \), where \( \oplus \) is convolution without multiplications
  - Objective function
    \[
    \arg\min_{b, \alpha} J(b, \alpha) \equiv \| w - \alpha b \|^2
    \]
  - Optimization for \( B \)
    \[
    J(b, \alpha) = \alpha^2 b^T b - 2\alpha w^T b + w^T w = -2\alpha w^T b + (\text{constant})
    \]
    \[
    b^* = \arg\max_B w^T b \quad \text{such that} \quad b \in \{+1, -1\}^n
    \]
  - Optimization for \( \alpha \)
    \[
    \frac{\partial}{\partial \alpha} J(B, \alpha) = 2ab^T b - 2w^T b = 0
    \]
    \[
    \alpha^* = \frac{w^T b}{b^T b} = \frac{w^T \text{sign}(w)}{n} = \frac{1}{n} \| w \|_1
    \]


---

Training Binary-Weight-Networks

- Binarize the weights during the forward pass and backward propagation
- Weight updates in floating points to handle tiny changes effectively

Algorithm 1 Training an L-layers CNN with binary weights:

1. Initialize the network with random weights.
2. For each iteration:
   1. Forward pass: compute the output \( \hat{y} \)
   2. Backward pass: compute the gradients
3. Update the weights: \( W_{new} = \text{sign}(W_{old}) \)

Pruning Low Magnitude Weights

- Simple approach
  - Pruning unimportant connections (with near zero weights)
  - Training network, pruning weights, retraining network (repeat)

**Channel Pruning**

- **Filter pruning**
  - Motivation: reducing computational cost significantly
  - By identifying filters having a small effect on the output accuracy
- **Main idea**
  - Prune the channels corresponding to the filters with smallest magnitudes!
  - This idea is correlated to but better than activation-based pruning.
  - Filter pruning in the lower layers incurs filter updates in the upper ones.

  ![Channel Pruning Diagram](image)


**Eliminating Redundant Convolutions**

- **Motivation and main idea**
  - Speeds up the bottleneck convolutional layers by skipping their evaluation in some of the spatial positions
  - Inspired by the loop perforation technique from source code optimization
  - Interpolates missing activations using nearest neighbors
  - Accelerates 2-4x in AlexNet and VGG
- **Perforation mask**
  - Marks positions for exact convolutions
  - Uniform: selects mask randomly and generates clusters (not desirable)
  - Grid:
  - Pooling structure: computes exact convolutions that are included in more pooling windows
  - Impact: estimates the impact of perforation of each position on the CNN loss function, and then removes the least important positions

  ![Perforation Mask Diagram](image)


**MobileNets**

- **Depthwise separable convolution**
  - Factorizing standard convolution
    - Depthwise convolution: applies a single filter to each input channel.
    - Pointwise convolution: applies a 1×1 convolution to combine the outputs of the depthwise convolution.
  - Drastically reducing computation and model size

  ![Depthwise Separable Convolution](image)


**MobileNets**

- **Two simple global hyperparameters**
  - Width multiplier: thinner model
  - Resolution multiplier: reduced representation
  - Controlling trade off between latency and accuracy

  ![MobileNets Diagram](image)
