ECE 901 Paper Presentation

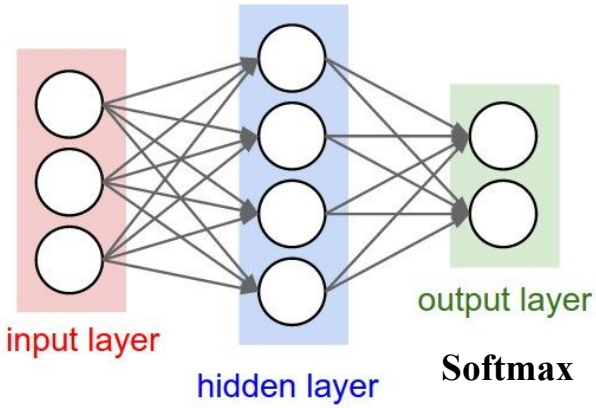
Training Deep Nets with Sublinear Memory Cost

Tianqi Chen, Bing Xu, Chiyuan Zhang and Carlos Guestrin

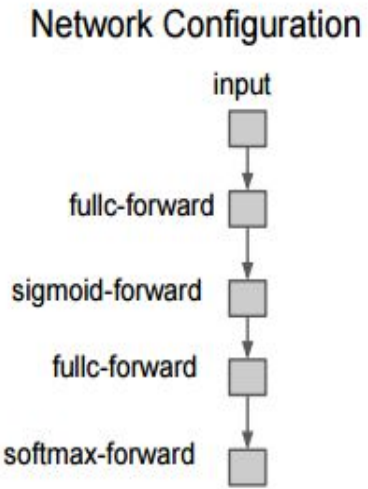
Presented by: Akshay Sood, Sneha Rudra

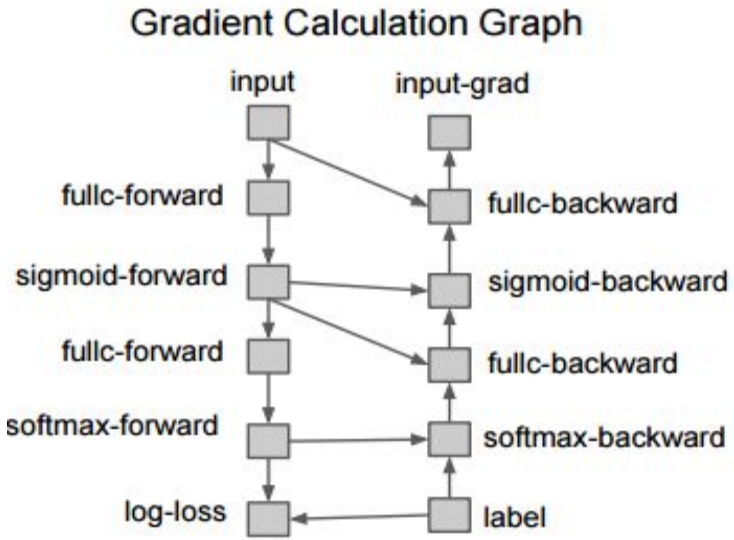
Computation Graph

- A computation graph - used in many frameworks Theano, TensorFlow, MXNet
- Nodes represent operations and edges represent the dependencies between the operations



Source: http://cs231n.github.io/





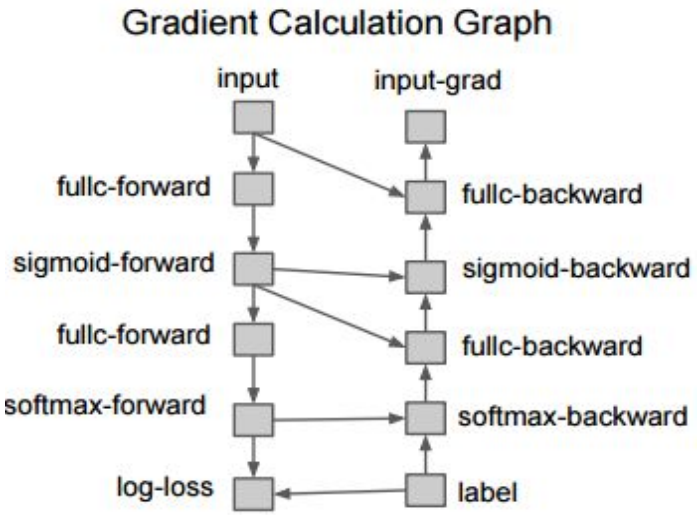
Memory Optimization with Computation Graph

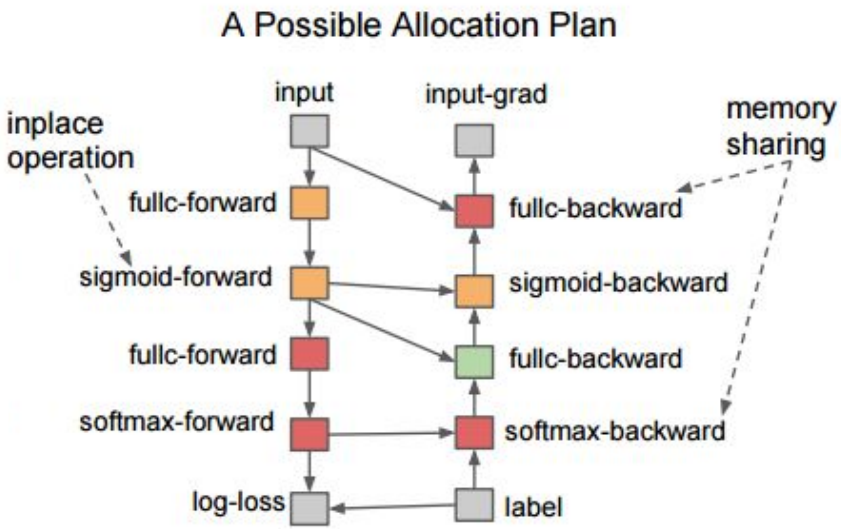
- Deep net training often needs large memory to store the intermediate outputs and gradients
- Each intermediate result corresponds to a node in computation graph
- Two types of memory optimizations can be used
  - ➔ **Inplace operation**: Directly store the output values to memory of a input value
  - ➔ **Memory sharing**: Memory used by intermediate results that are no longer needed can be recycled and used in another node

Memory Optimization with Computation Graph

- First sigmoid transformation carried out using inplace operation, which is then reused by its backward operation. The storage of the softmax gradient is shared with the gradient by the first fully connected layer.

Memory allocation for each node, same color indicates shared memory

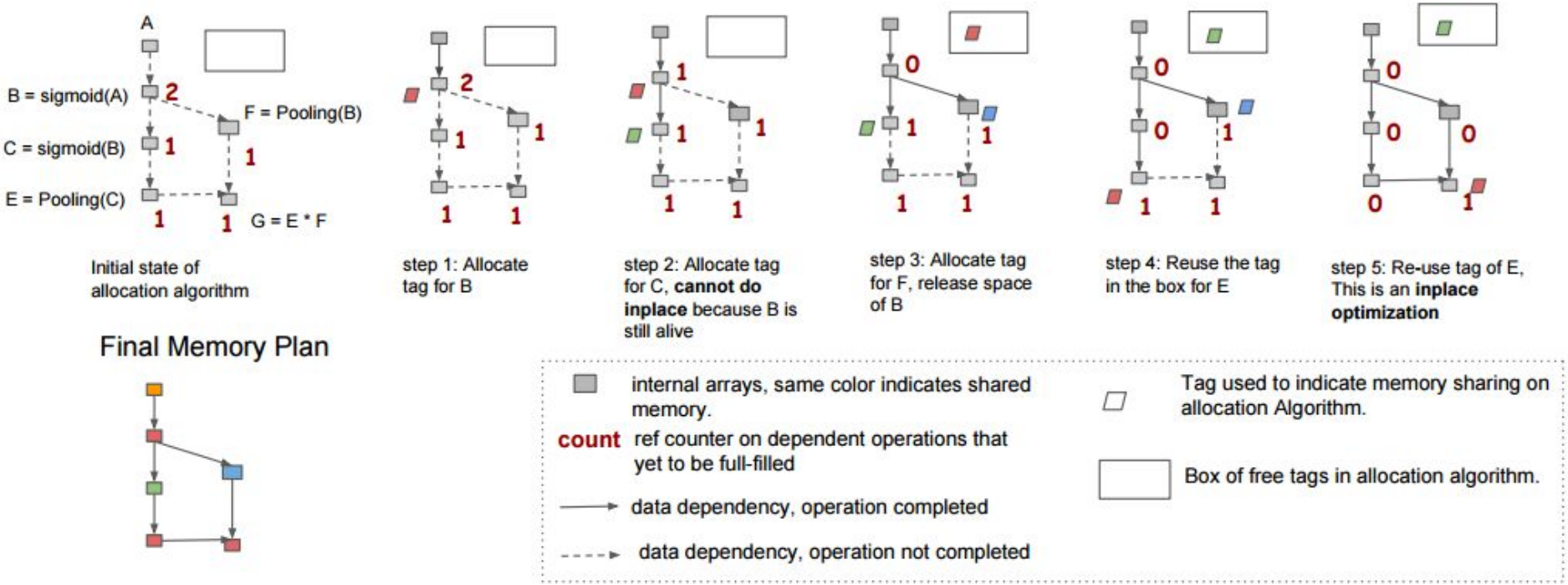




O(n) Memory Allocation Heuristic

- Memory sharing only between nodes whose lifetimes do not overlap
- One option to identify such nodes - run a graph-coloring algorithm on conflicting graph - but this would cost $O(n^2)$ computation time
- Heuristic with O(n) time complexity
  - ➔ Traverse the graph in topological order, and use a counter to indicate the liveness of each record
  - ➔ Inplace operation when no other pending operations that depend on node's input (i.e. when node counter =1)
  - ➔ Memory sharing when a recycled tag is used by another node (i.e. when node counter = 0)

O(n) Memory Allocation Heuristic



Used as a static algorithm, to allocate the memory to each node before the execution to avoid overhead of garbage collection during runtime

Memory-computation tradeoff

- Memory optimization over the computation graph can reduce memory footprint for both training and prediction.

- However, most gradient operators depend on the intermediate results of the forward pass, so we still need O(n) memory for intermediate results to train the network

- Idea: drop some of the intermediate results during the forward pass, and recover them using extra forward computation when needed

Segmentation

- Idea: divide the neural network into segments
  - Only store output of each segment during the forward pass; throw away intermediate results
  - During backprop, redo within-segment computations

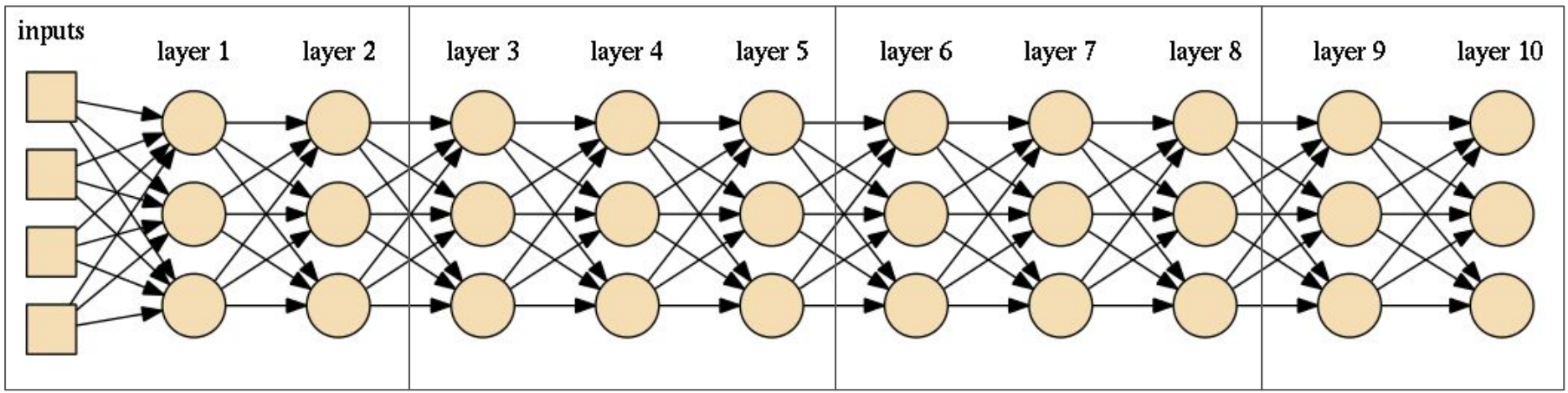


Image source: http://neuron.tuke.sk/jaksa/slides/Jaksa-CITseminar2014/nets/4-3-3-3-3-3-3-3-3-3.png

Repeating intermediate computation

- User specifies a function $m : V \to \mathbb{N}$ ("mirror count") on the nodes of the computation graph, specifying the number of recomputations permitted for each node

- **Trivial case:** m = 0 for all nodes: original computation graph, no intermediate result dropped

- **Common case:** m(v) = 1 for nodes within each segment, m(v) = 0 for output node of each segment: recompute within-segment nodes exactly once

Example

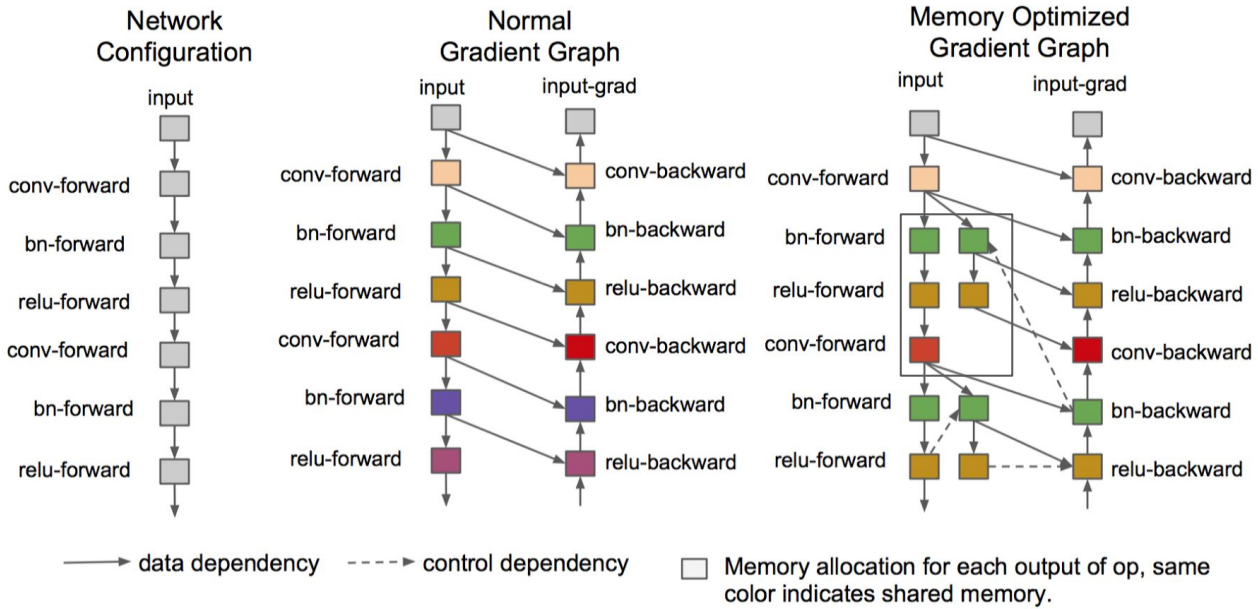
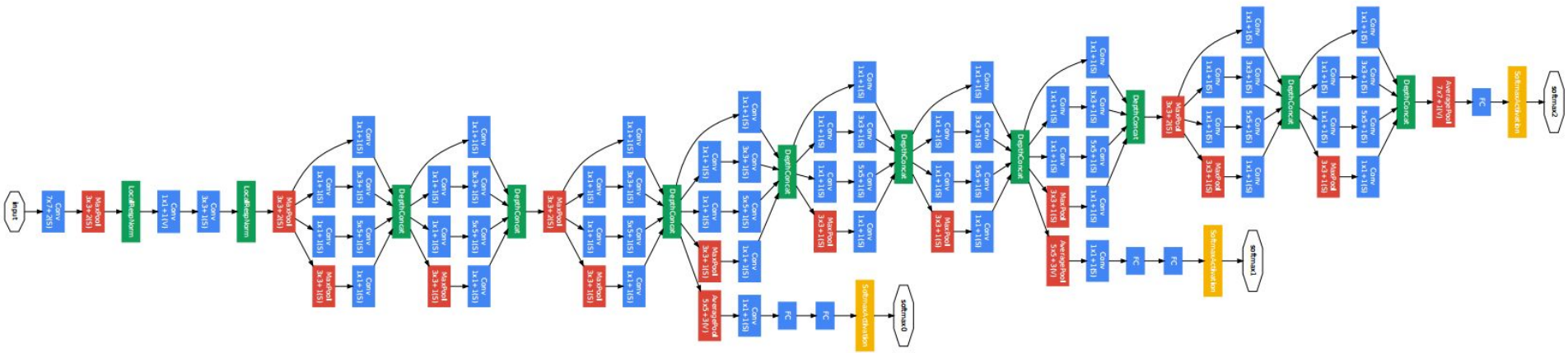


Figure 3: Memory optimized gradient graph generation example. The forward path is *mirrored* to represent the re-computation happened at gradient calculation. User specifies the mirror factor to control whether a result should be dropped or kept.

Example

- Easy application: drop results of low cost operations, keep results that are expensive to compute

- For instance, in a Conv-BatchNorm-Activation CNN pipeline, we can keep the result of convolution but drop the result of batch normalization, activation function and pooling



GoogLeNet

Image source: https://indico.io/blog/wp-content/uploads/2015/08/googlenet.png

Memory complexity

- Assume we divide an n-node network into k segments. Then the memory complexity using a within-segment mirror factor of 1 is given by:

$$\text{cost-total} = \max_{i=1,\ldots,k} \text{cost-of-segment}(i) + O(k) = O\left(\frac{n}{k}\right) + O(k)$$

- Choosing k = √n, cost-total = O(√n), i.e. sublinear in the size of the network, requiring an additional forward pass during training

Recursive formulation

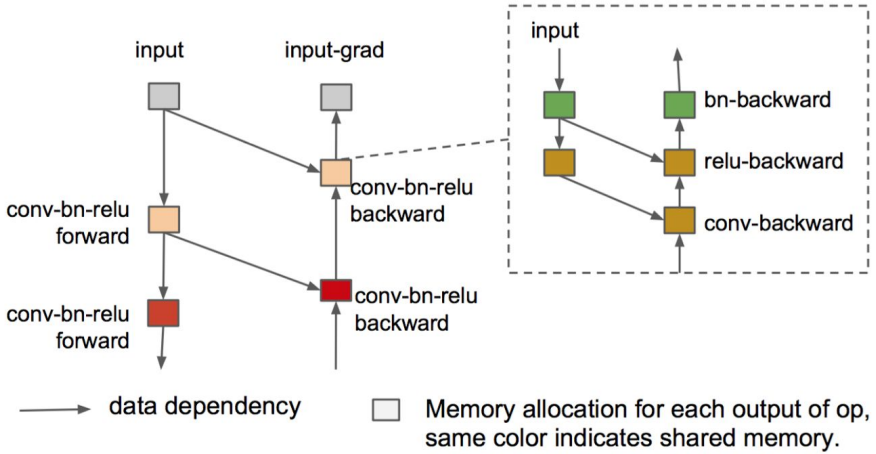


Figure 4: Recursion view of the memory optimized allocations. The segment can be viewed as a single operator that combines all the operators within the segment. Inside each operator, a sub-graph as executed to calculate the gradient.

Recursive formulation

- Let g(n) be memory cost to do forward and backward pass on an n-layer network. Assume we store k intermediate results in the graph and apply strategy recursively

$$g(n) = k + g(\frac{n}{k+1}) \implies g(n) = k \log_{k+1}(n)$$

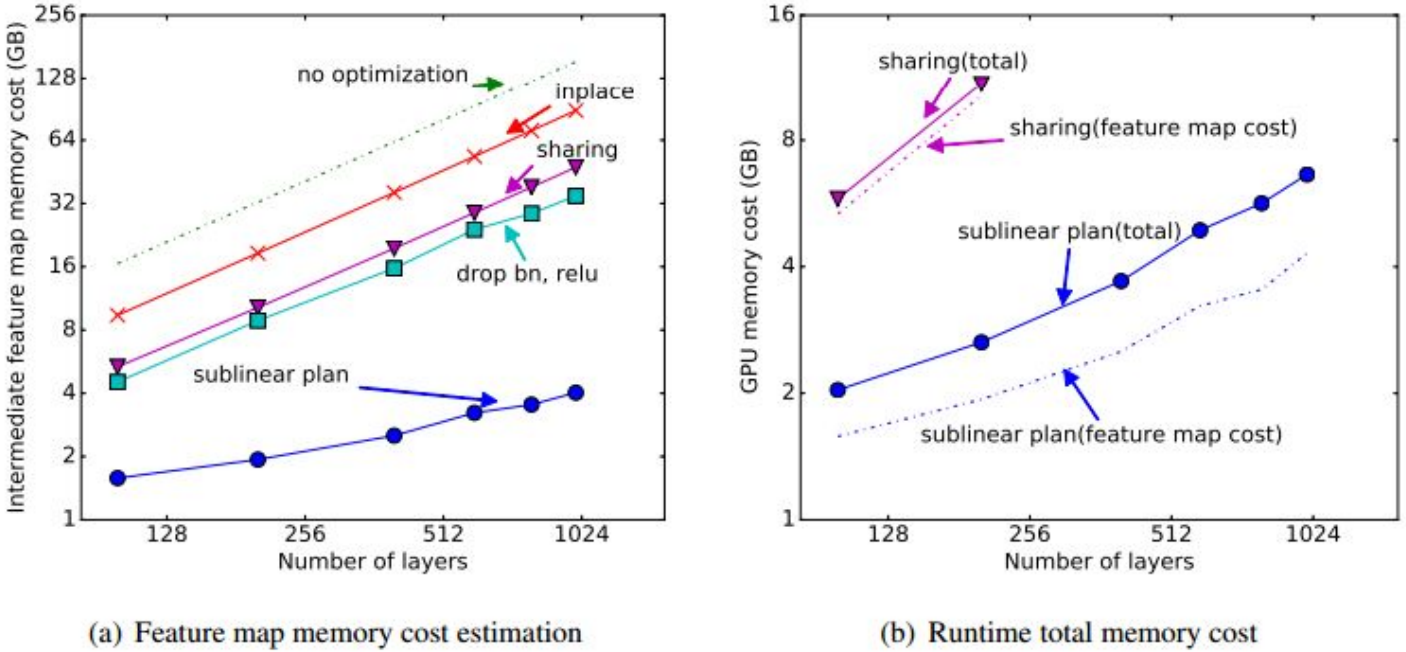
- Special case: k = 1 gives us g(n) = $\log_2(n)$, with the forward pass cost O(n$\log_2(n)$).

Experiments

- Memory allocation algorithms compared:
  - ➔ No optimization - allocate memory to each node in the graph
  - ➔ Inplace
  - ➔ All system optimizations - Inplace + Sharing
  - ➔ Drop batch normalization and relu + All system optimizations (only for convolutional net)
  - ➔ Sublinear plan + All system optimizations (trade computation for memory)

Experiment 1 - Deep Convolutional Network

- CNN deep residual network architecture (ResNet) for image classification



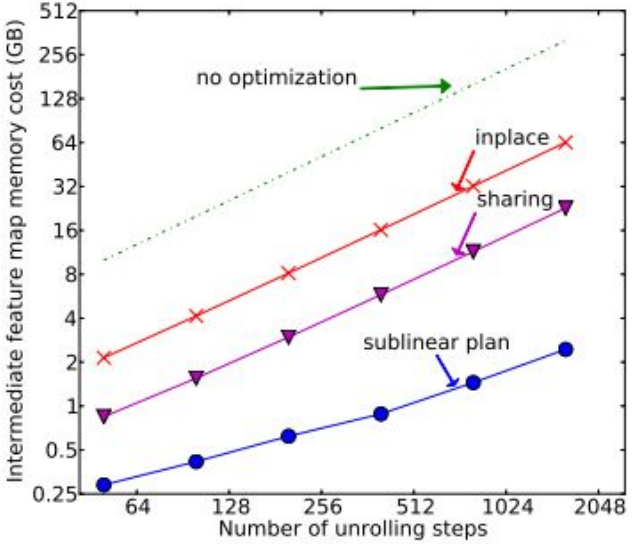
(a) Feature map memory cost estimation

(b) Runtime total memory cost

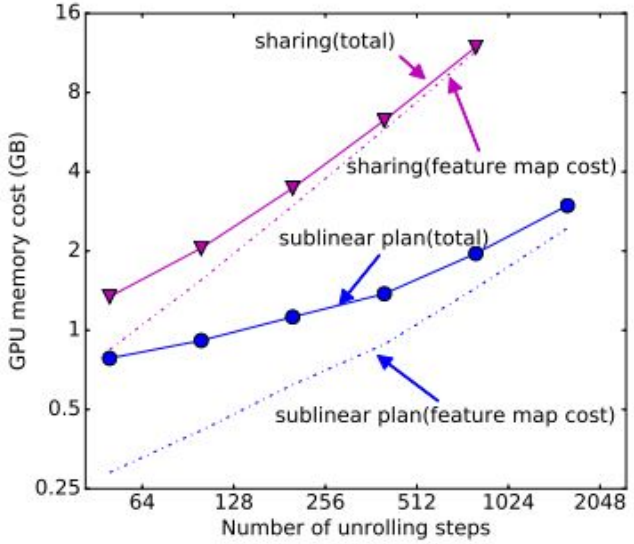
- System optimizations help reduce memory cost by factor of two to three (but memory cost still linear with respect to # of layers, only possible to train a 200 layer ResNet with the best GPU)
- Sublinear Plan - 1000 layer ResNet using < 7GB of GPU memory

Experiment 2 - LSTM for Long Sequences

- LSTM under a long sequence unrolling setting - typical setting for speech recognition. Sublinear plan gives more than 4x reduction over the optimized memory plan



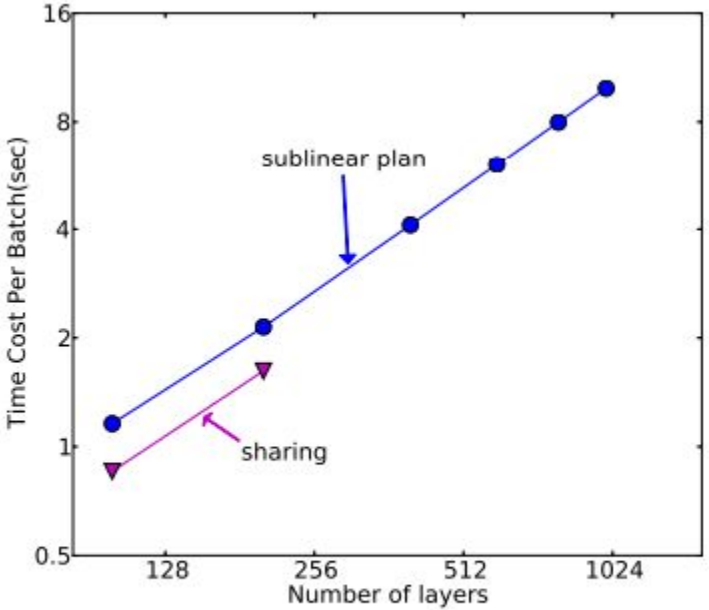
(a) Feature map memory cost estimation



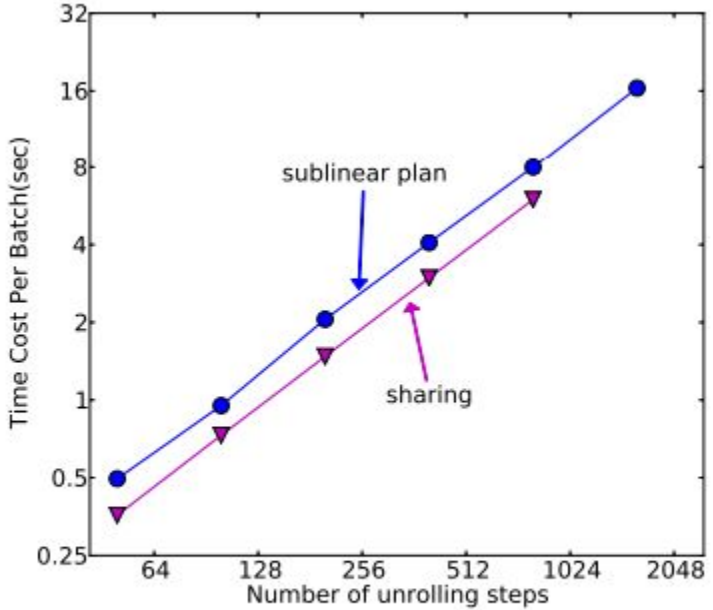
(b) Runtime total memory cost

Impact on Training Speed

- Runtime cost is benchmarked on a single Titan X GPU
- Sublinear allocation strategy costs 30% additional runtime compared to the normal strategy (double forward cost in gradient calculation)



(a) ResNet



(b) LSTM

Conclusion

- Systematic approach to reduce the memory consumption of the intermediate feature map in deep net training
- Computation graph liveness analysis used to enable memory sharing
- Possible to trade computation with memory
- Combination of techniques can help train n layer deep neural network with only $O(\sqrt{n})$ memory cost with one extra forward computation cost per mini-batch.