Memory

in neural networks (TensorFlow)

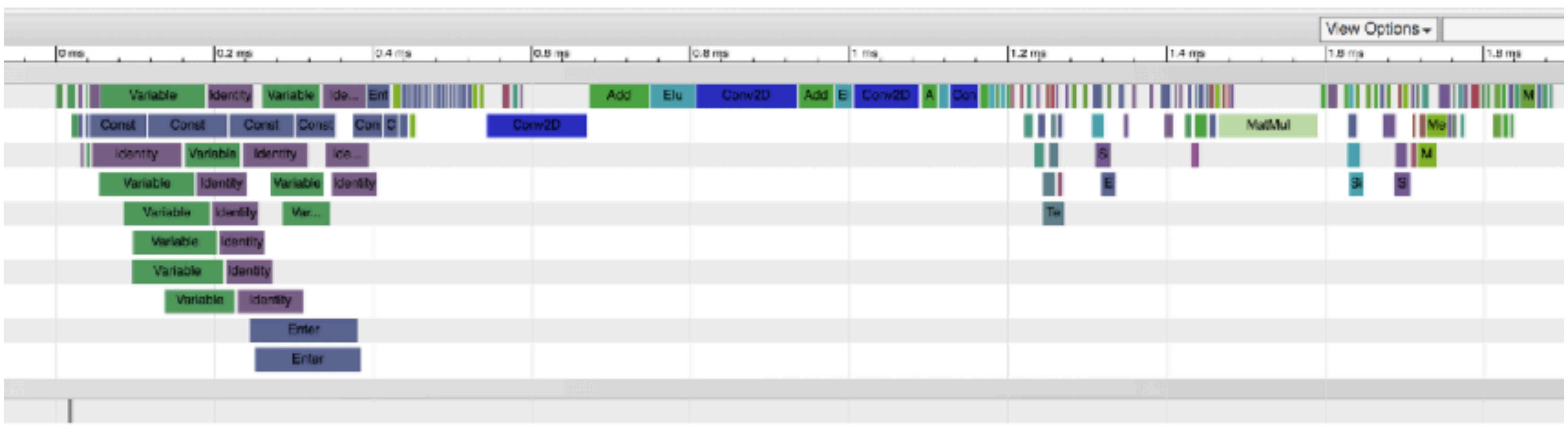
Why memory

- Peak efficiency is achieved for large batches on GPU:
  - 11 T ops/s for 8k-by-8k matrix multiply on TitanX
  - 1.1 T ops/sec for 8k-by-8k matrix multiply on Xeon V3
  - 0.1 G ops/s for 256-by-256 matrix multiply on TitanX/Xeon
  - Small batch runtime dominated by once-per-batch overhead (ie, var reads)
- Money doesn't buy more GPU memory
  - $800 — 1080TI: 12GB
  - $6k — P100: 16GB
  - $??? — V100: 16GB

https://github.com/yaroslavvb/stuff/blob/master/matmul_benchmark.py

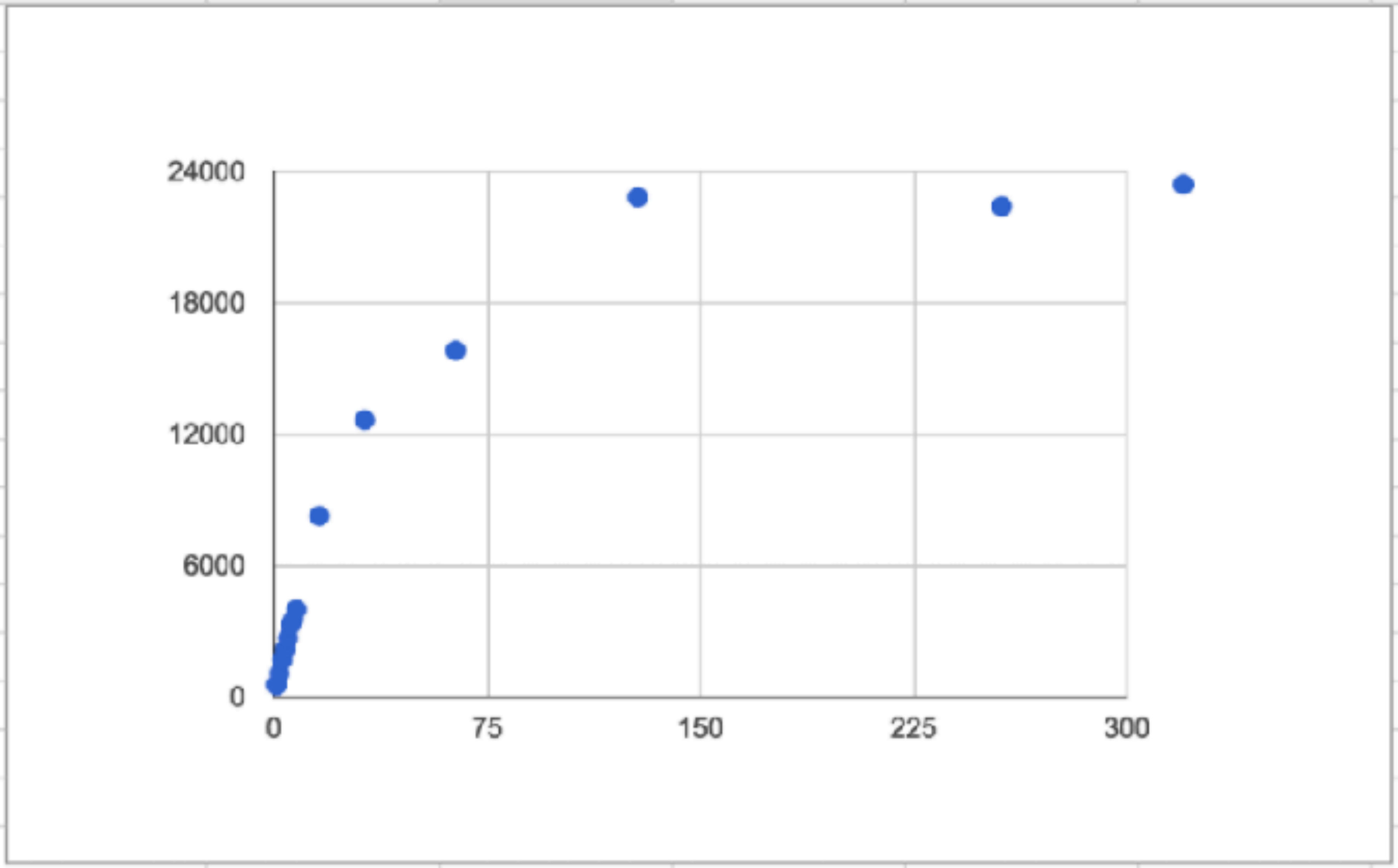
Fprop efficiency

Universe starter agent: 400 fps...too slow



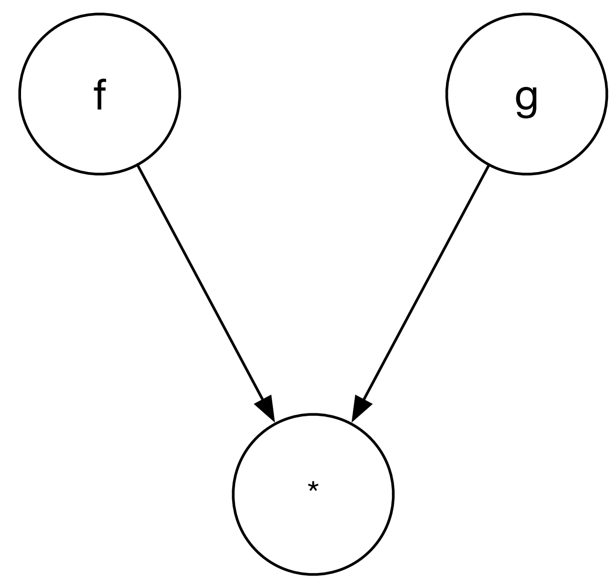
Fprop efficiency

1	565
2	1089
3	1691
4	2196
5	2712
6	3320
7	3569
8	4006
16	8275
32	12664
64	15840
128	22840
256	22415



Fprop efficiency

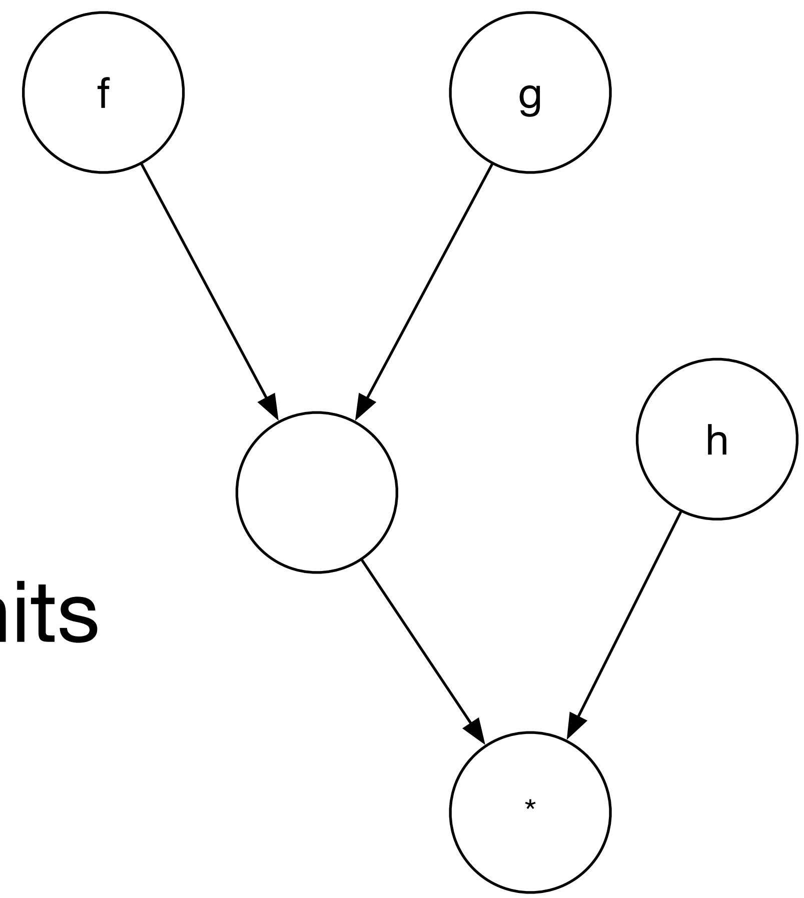
Calculating memory requirements



- f(x) + g(x)
- 2 memory units

Calculating memory requirements

- f(x) + g(x) + h(x)
- either 2 or 3 memory units

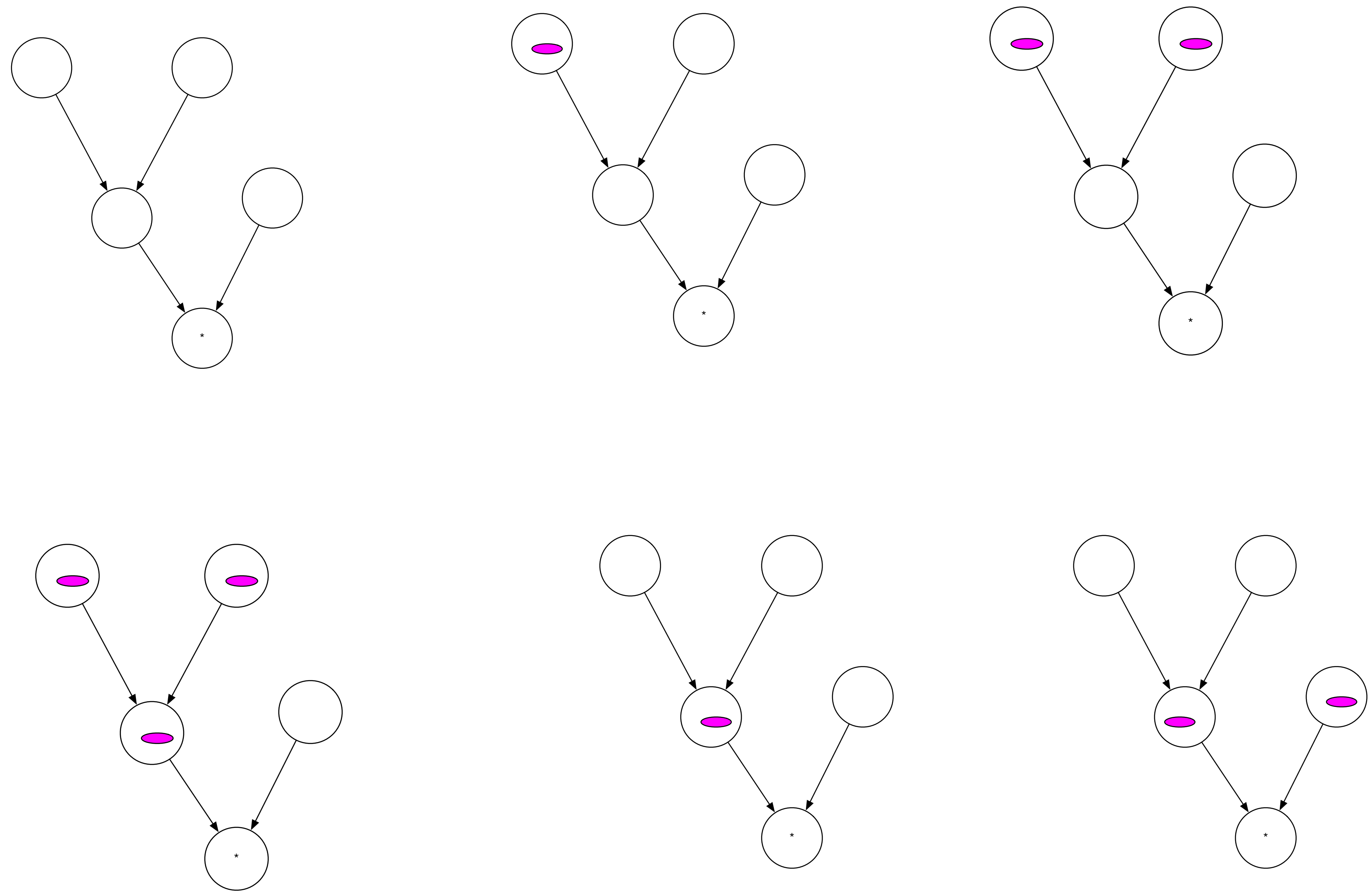


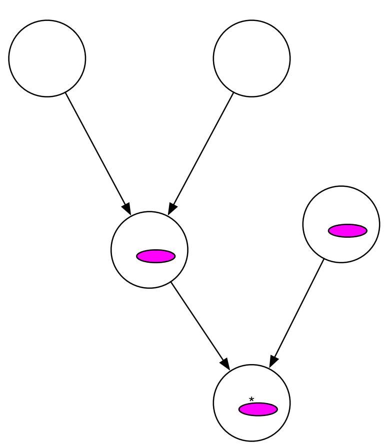
Pebble game

- Rules:

1. can only put pebble on node if all parents have pebbles

2. goal to put pebble on final node



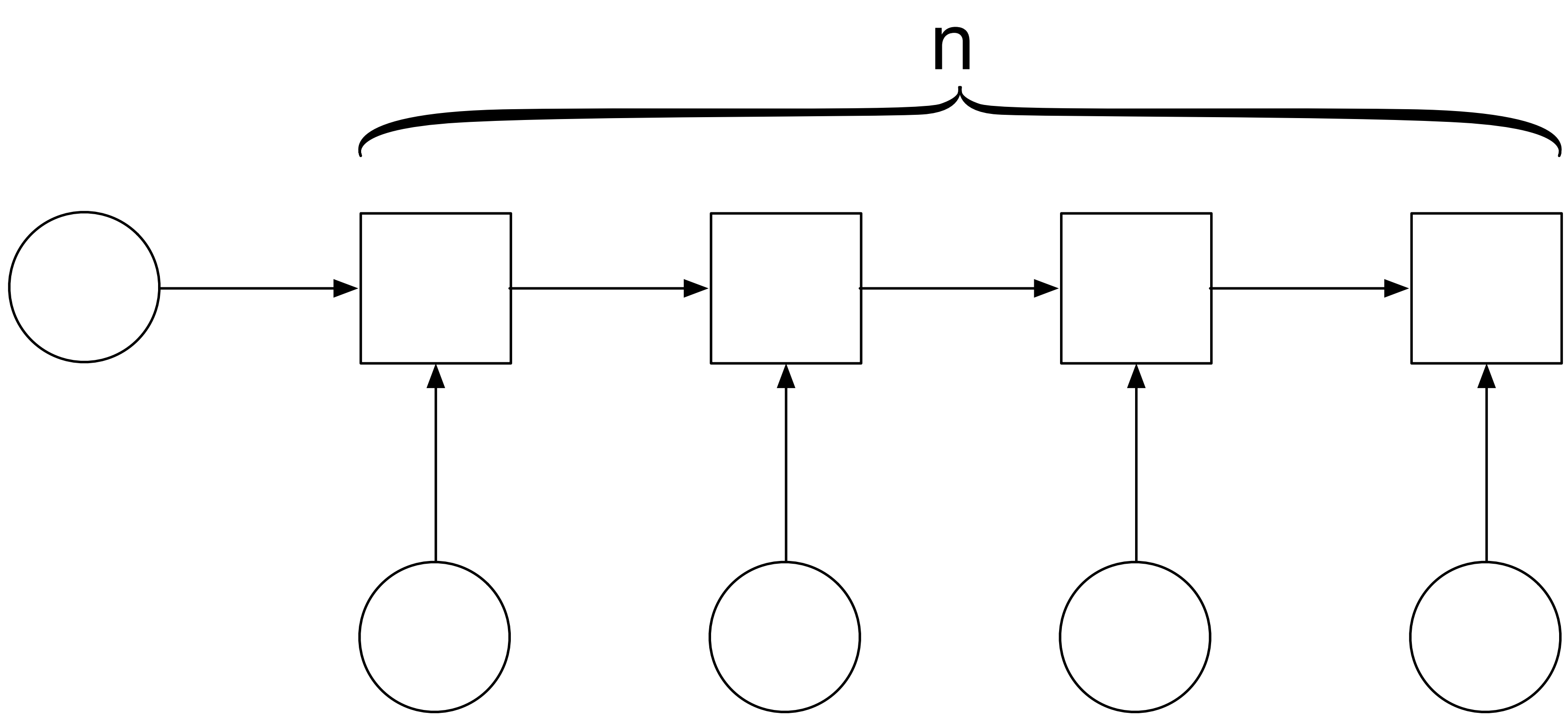


- Rules:

1. can only put pebble on node if all parents have pebbles or no parents

2. goal to put pebble on final node

Example



Best case: 3 units
Worst case: N units
TensorFlow case: ???

Pebble game

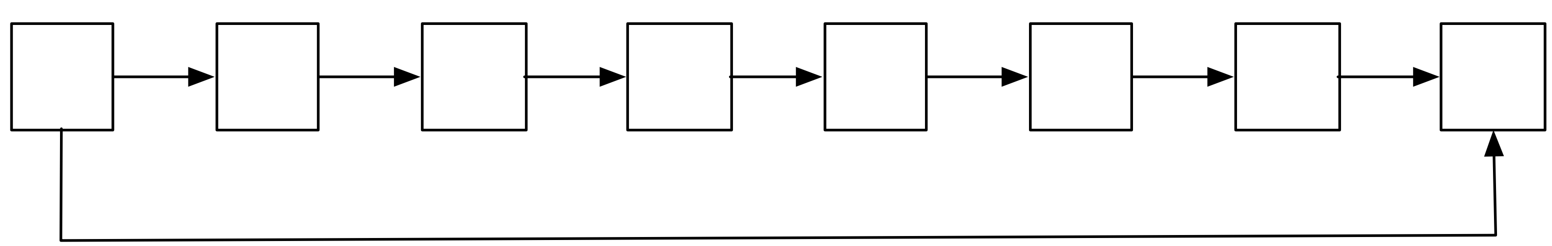
- Number of pebbles needed = peak requirement
- Different schedules produce different requirements
- How to find the optimal schedule?
    - No solution for general computation graphs

“Inapproximability of treewidth, one-shot pebbling, and related layout problems” http://dl.acm.org/citation.cfm?id=2655729

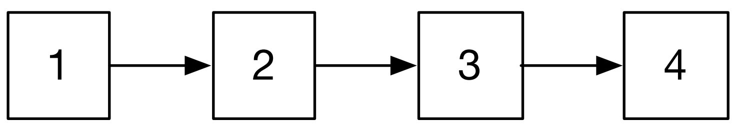
    - But good heuristics exist

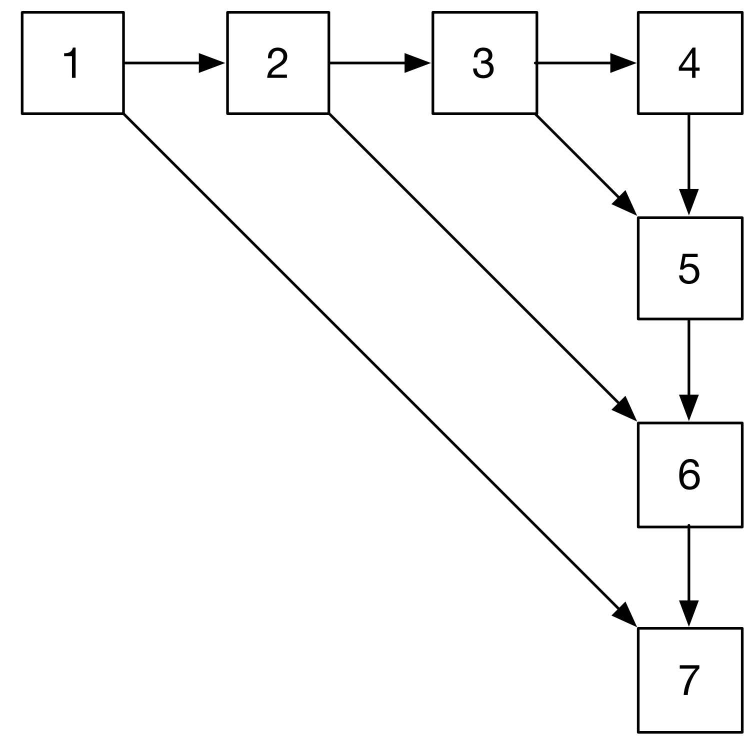
Pebble game

- One-shot pebbling = do not touch nodes already visited = no recompilations
- Multi-shot pebbling = can revisit old nodes = recompilations allowed
- TensorFlow = no recomputations

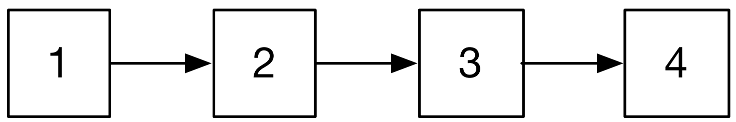


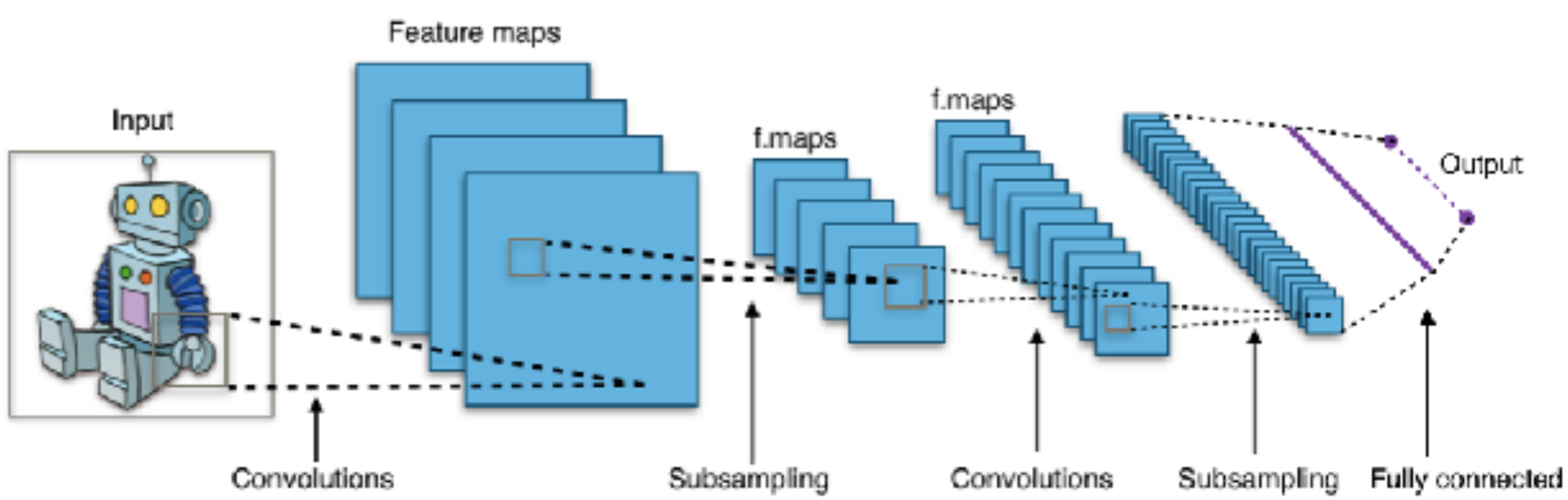
Neural networks





Inference





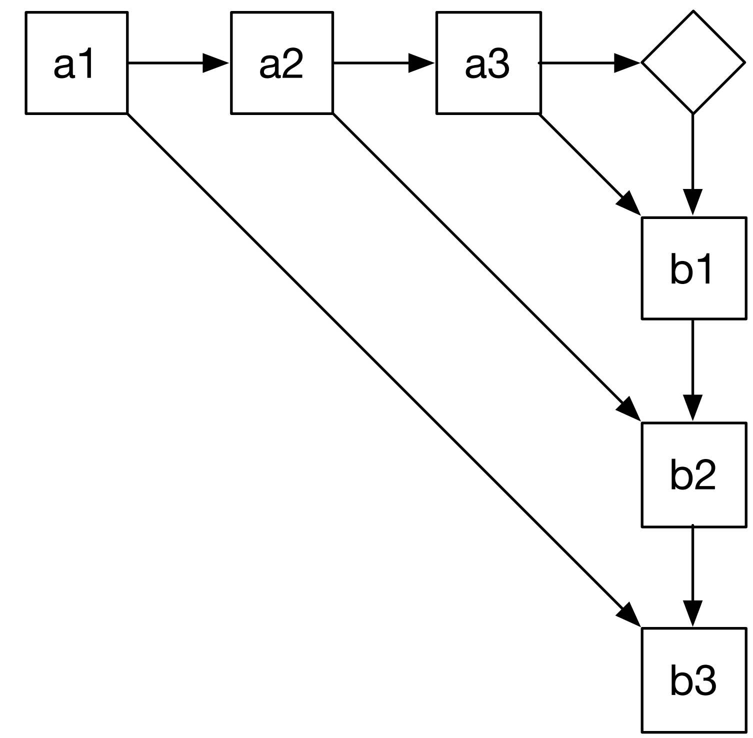
Memory requirement determined by most expensive layer (typically the first fully connected layer)

Training

$$c = f(g(h(x)))$$

$$\frac{dc}{dx} = f'(\underbrace{g(h(x))}_{\text{a3}})g'(\underbrace{h(x)}_{\text{a2}})h'(\underbrace{x}_{\text{a1}})$$

$$\frac{dc}{dx} = \underbrace{\underbrace{\underbrace{f'(g(h(x)))}_{\text{b1}}g'(h(x))}_{\text{b2}}h'(x)}_{\text{b3}}$$



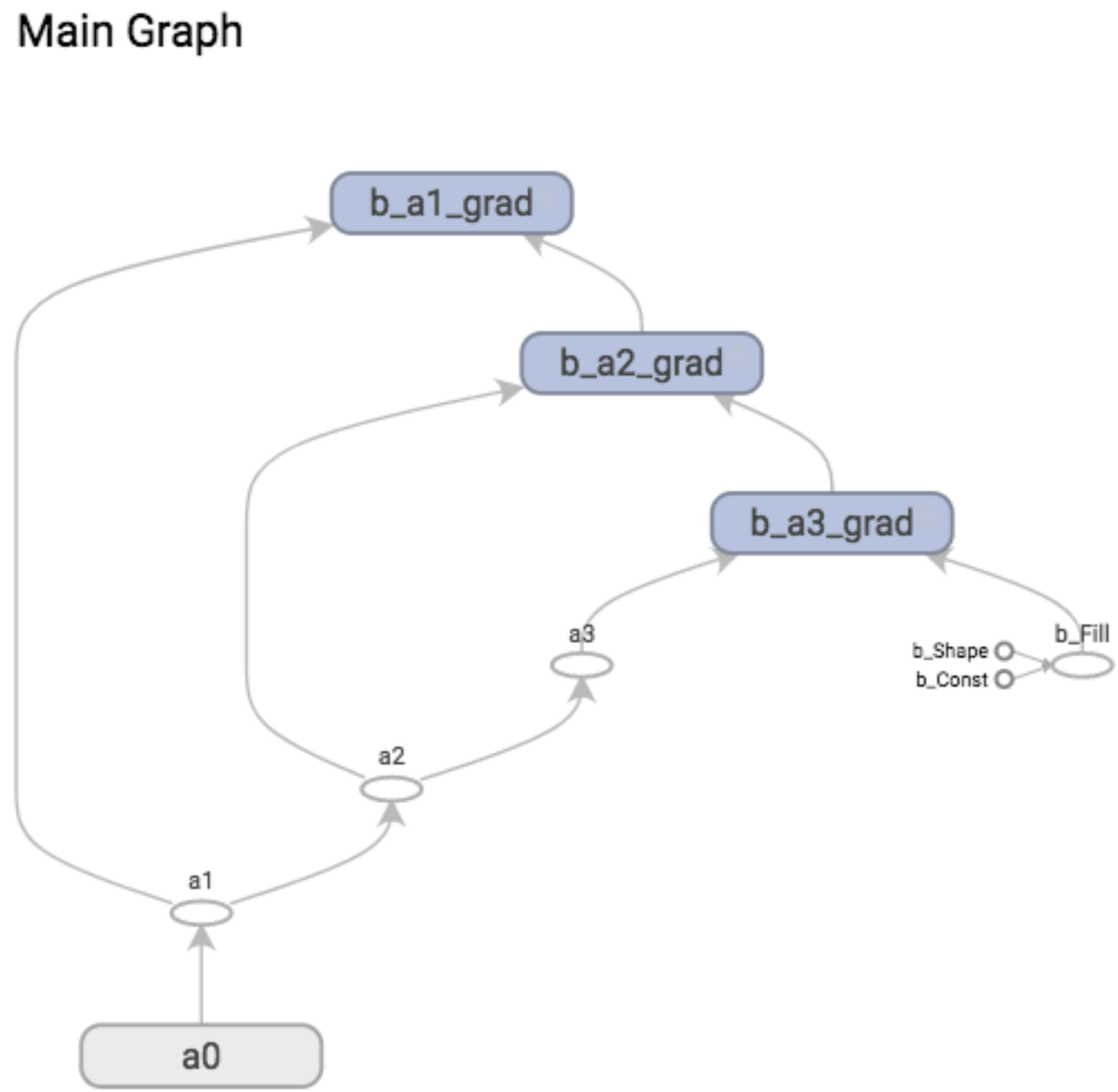
Training

3]:
```
tf.reset_default_graph()
node_mbs = 1
length = 4

dtype = np.float32
n = node_mbs * 250000
a0_ = tf.ones((n,), dtype=dtype)
a0 = tf.Variable(a0_, name="a0")
a = a0
for i in range(1, length):
    name = "a"+str(i)
    a = tf.tanh(a, name=name)

grad = tf.gradients([a], [a0])[0]
sess = create_session()
```

4]:
```
show_graph(ungroup_gradients=True)
```



https://github.com/yaroslavvb/stuff/blob/master/node-merge.ipynb

TensorFlow memory

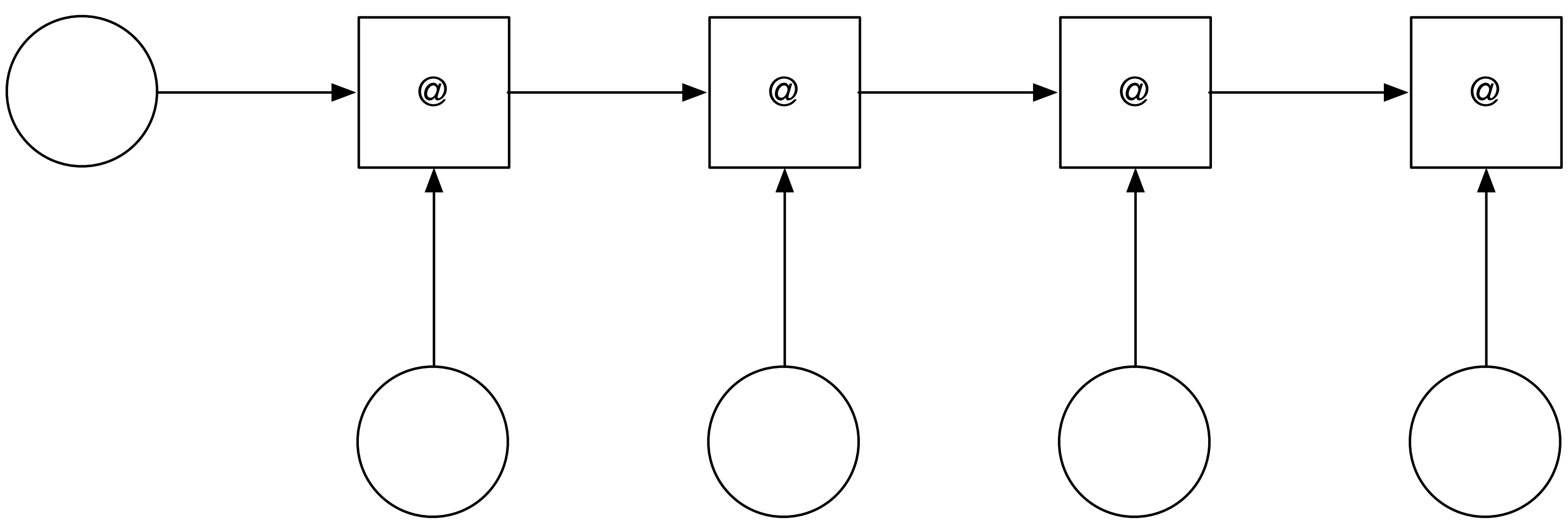
- Which order does it pick? (look in executor.cc)

TensorFlow memory

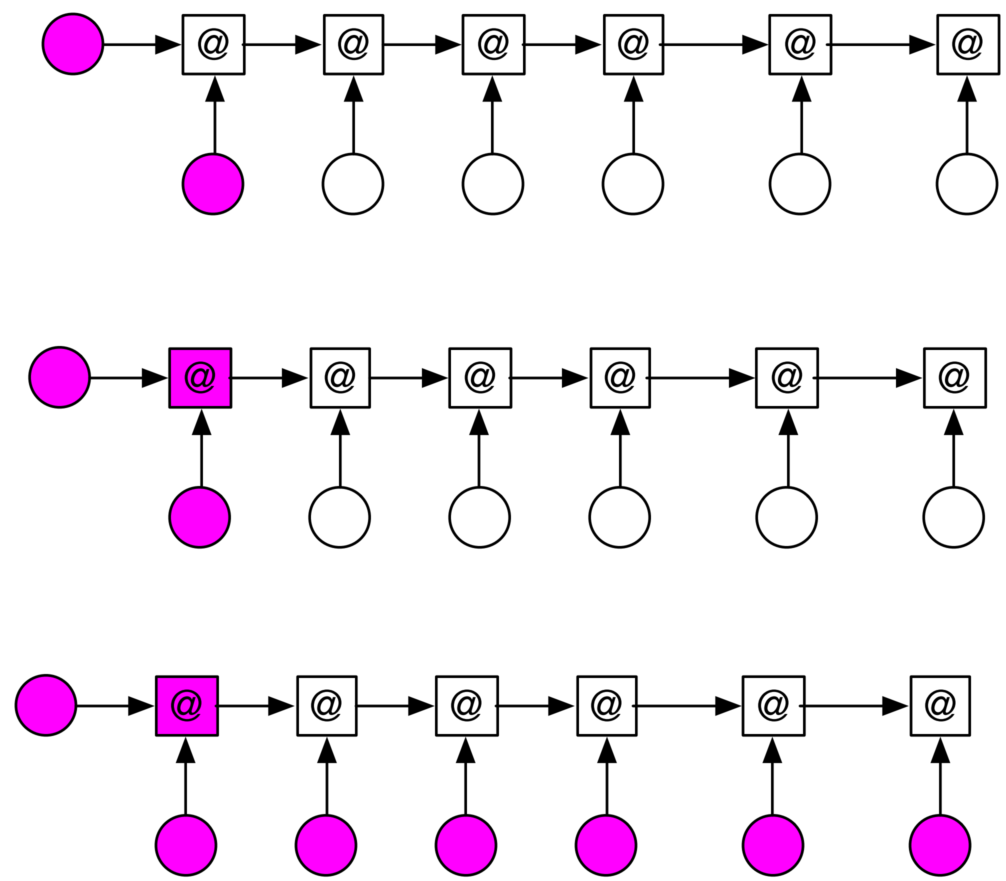
```
result = tf.random_uniform((size, size))

for i in range(n):

    result = result @ tf.random_uniform((size, size))
```



TensorFlow memory



8k matmul takes 100ms
8k-by-8k random_uniform takes 4ms

How to monitor memory

- TensorFlow manages it’s own memory, so nvidia-smi is useless

1. parse LOG_MEMORY allocation/deallocation messages (https://github.com/yaroslavvb/memory_util)

2. Extract it from Timeline

3. Write custom TensorFlow op that queries allocator on demand

https://github.com/yaroslavvb/memory_probe_ops

memory_util example

- https://github.com/yaroslavvb/notebooks/blob/master/mnist-memory.ipynb

- https://github.com/yaroslavvb/memory_util

timeline

run_metadata = tf.RunMetadata()
run_options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
sess.run(model.train_op, options=run_options, run_metadata=run_metadata)

```
node_name: "a02_add"
all_start_micros: 1505768360742529
op_start_rel_micros: 32
op_end_rel_micros: 80
all_end_rel_micros: 137
memory {
  allocator_name: "GPU_0_bfc"
  allocator_bytes_in_use: 171016448
}
output {
  tensor_description {
    dtype: DT_FLOAT
    shape {
      dim {
        size: 250000
      }
    }
    allocation_description {
      requested_bytes: 1000000
      allocated_bytes: 1000192
      allocator_name: "GPU_0_bfc"
      allocation_id: 3735
      ptr: 1108455317760
    }
  }
}
timeline_label: "a02_add = Add(a01_add, a02_tanh)"
scheduled_micros: 1505768360742492
memory_stats {
}
```

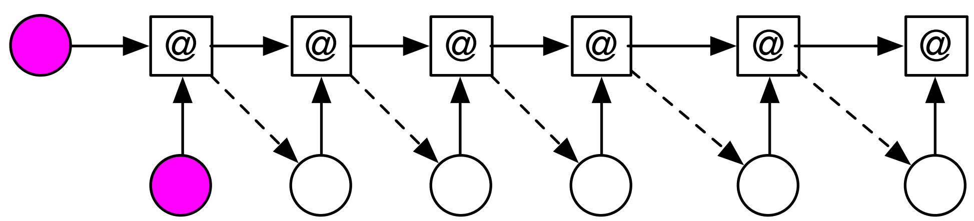
https://github.com/yaroslavvb/stuff/blob/master/memory%20tracking.ipynb

Improving memory usage

- Pick better execution order
- Forget/recompute intermediate Tensors
- Use TensorFlow functions
- Offload to main memory

Better execution order

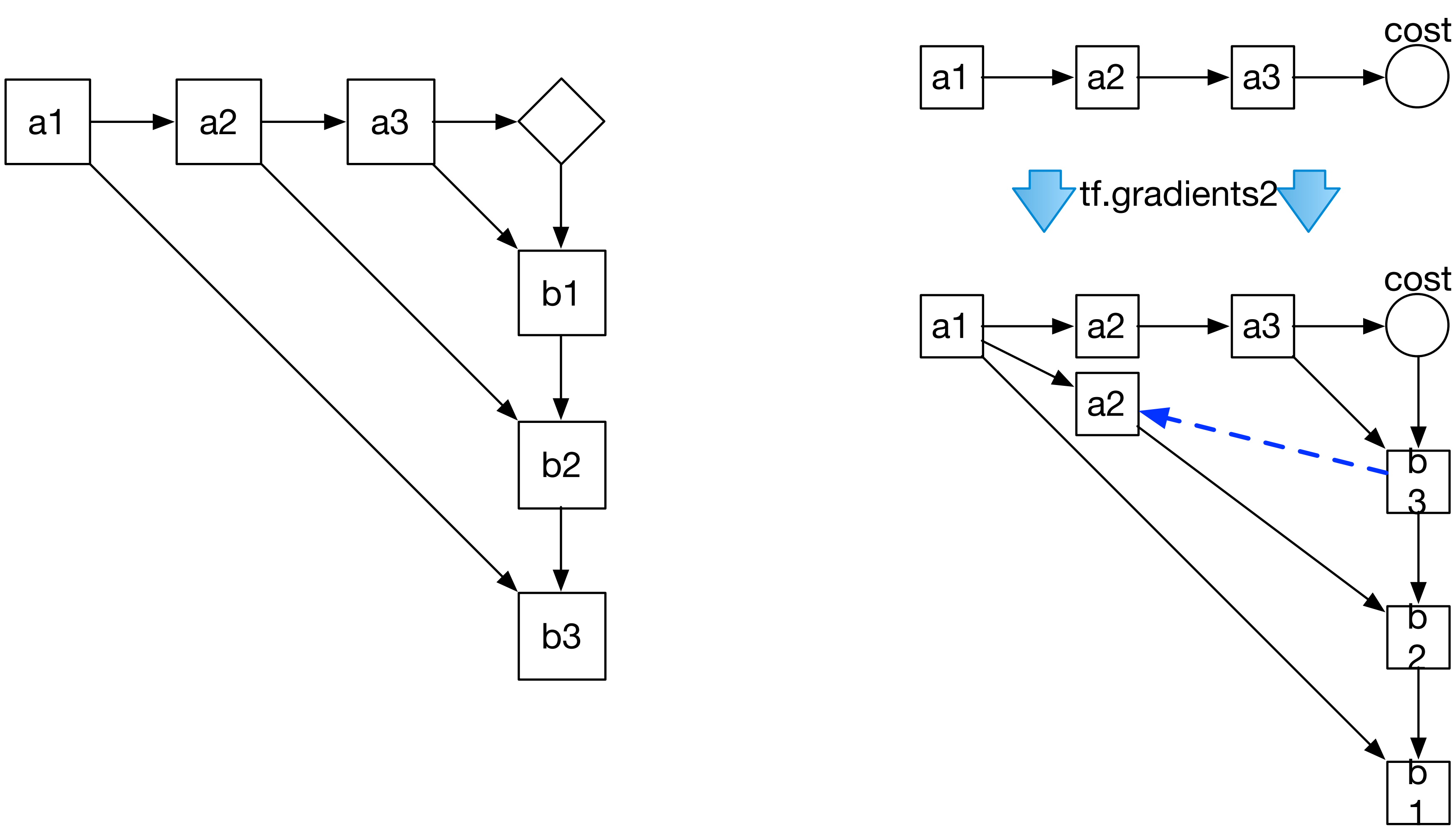
Add enough control dependencies so that execution order is deterministic.



Pick execution order where nodes that are needed later, are also computed later

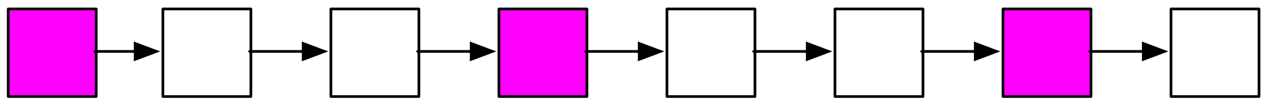
https://github.com/yaroslavvb/stuff/tree/master/linearize

Rewire the graph for recompilation



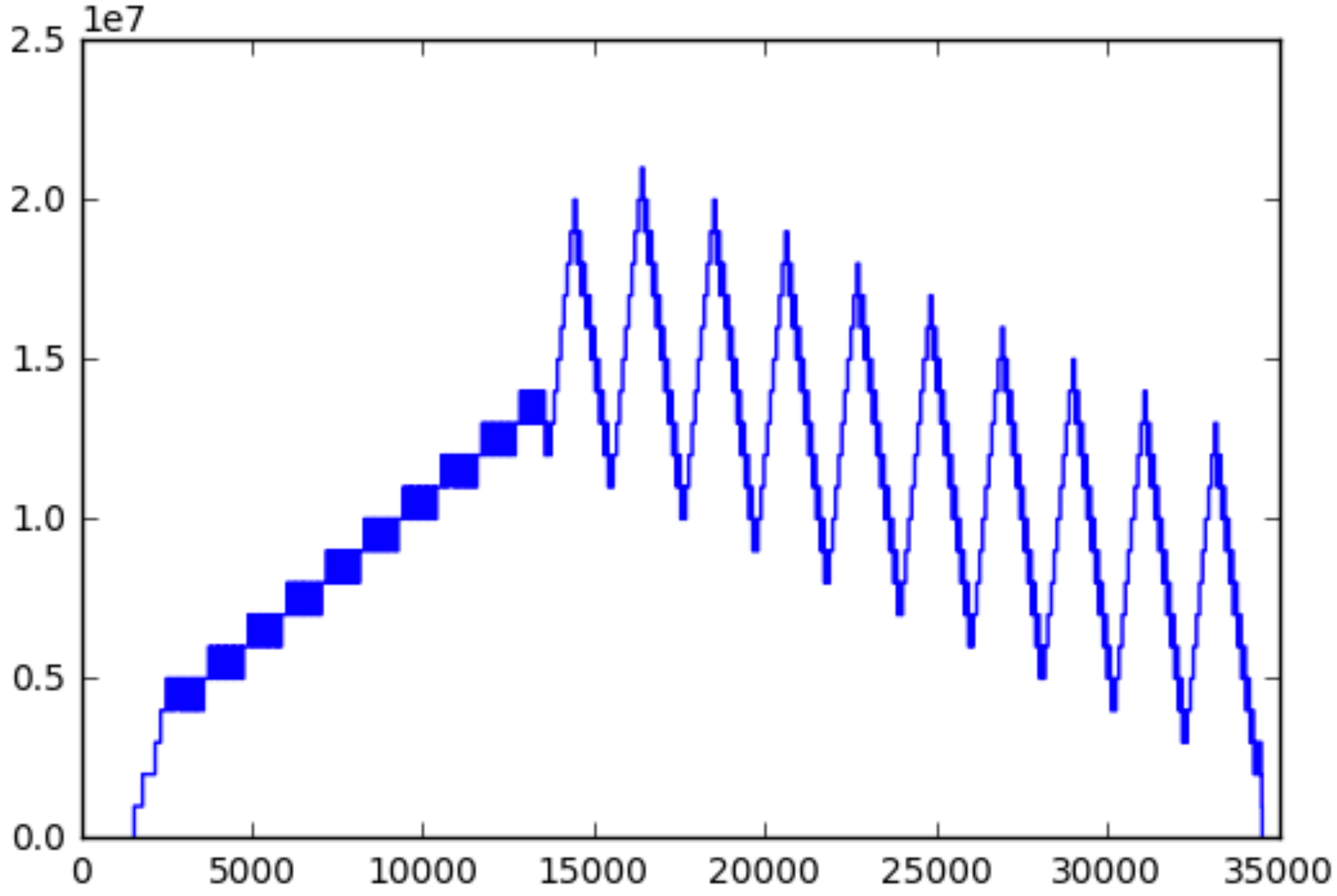
https://github.com/yaroslavvb/stuff/blob/master/simple_rewiring.ipynb

sqrt(n) saving



Training Deep Nets with Sublinear Memory Cost

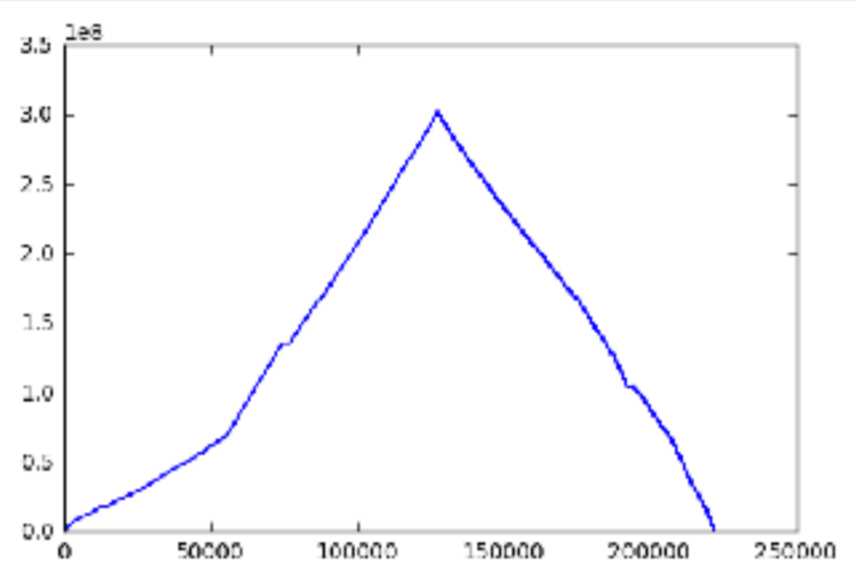
Tianqi Chen, Bing Xu, Chiyuan Zhang, Carlos Guestrin

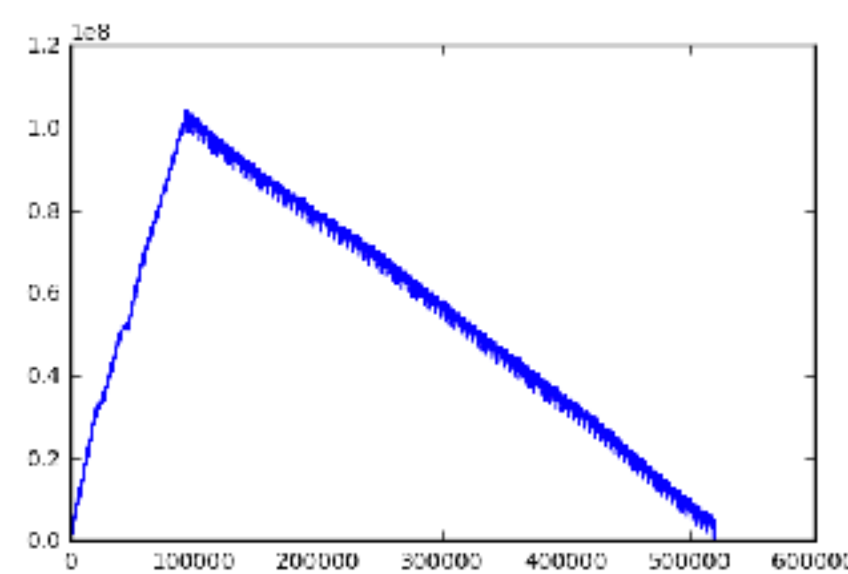


Functions to recompute intermediate results

```
@function.Defun(tf.float32, func_name="tanh3")
def tanh3(a):
      return tf.tanh(tf.tanh(tf.tanh(a)))
```

intermediate results are forgotten
similar to graph rewiring, but requires modifying model construction code
technique used by Google Translation and LM models





https://github.com/yaroslavvb/stuff/blob/master/saving%20memory%20by%20using%20functions.ipynb

Offload to main RAM

- Instead of forgetting/recomputing, save to main memory (rewriting graph, using swap_memory=True, or grappler)
- makes sense for O(n^3) ops (conv2d, matmul)
- doesn't make sense for O(n^2) ops (everything else)
- 7x faster to recompute tf.mul on GPU than load from memory (10x faster for tf.concat)
- https://github.com/yaroslavvb/stuff/blob/master/gpu-memory-transfer.ipynb