



**OpenAI**

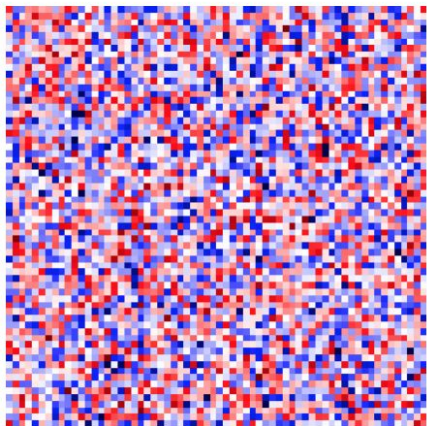
# Small World Network Architectures

NIPS 2017 Workshop

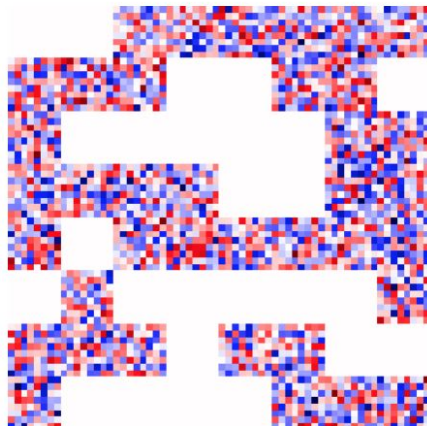
# Small World Networks

- We'd like to explore training models with very wide hidden states.
- More active memory, more information bandwidth, more easily linearly separable states.
- Don't want to pay the quadratically increasing compute cost.
- This means the network starts out sparse vs pruning a dense network.
- Just a few steps through a small world network can approximate fully connected networks, but in a factorized way. This helps improve parameter efficiency.
- We can implement reasonably efficient sparse layers with block-sparsity.
- So far we've just explored static sparsity but learned sparsity is the obvious next step. Think NAS down to the neuron level.

# Block Sparse Matrix Multiplication



Dense weights



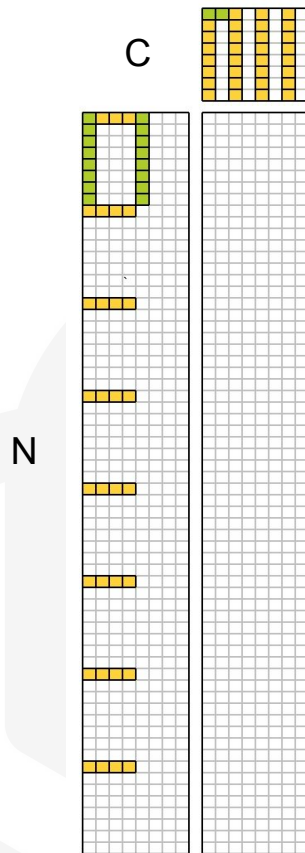
Block-sparse weights

0	0	1	1	1	1	1	1
1	1	1	0	0	1	1	0
1	0	0	0	0	0	1	1
1	1	1	1	0	0	1	1
1	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0
1	1	1	0	1	1	0	0
1	0	0	0	0	1	1	1

Corresponding sparsity pattern

# Kernel Implementation

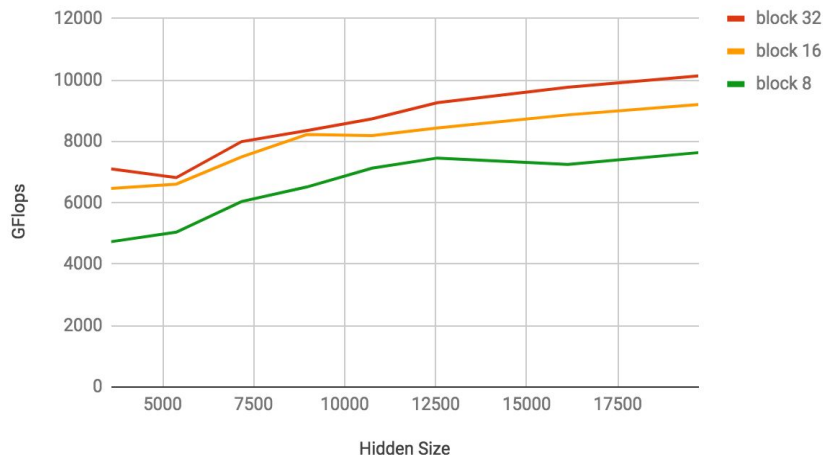
- The sparsity mask is converted into 3 lookup tables specific to each transpose variant (fprop, bprop, update)
- Weights are stored block contiguous (blockid, bsize, bsize).
- Two act/grad feature axis variants are implemented.
- The standard (N,C) layout is implemented efficiently in Maxwell/Pascal assembly with the tile size of 32x32
- Tile sizes of 32 and smaller (in the feature dimension) are implemented in cuda-c using a tile size of Bx64, B in (8,16,32).
- The layout is also swapped to (C,N) for maximally contiguous access. In the bandwidth bound regime, DRAM efficiency is extremely important.
- The weight update operation has the ability to consume lists of activation/gradient pairs to more efficiently reduce the minibatch\*timestep dimension.



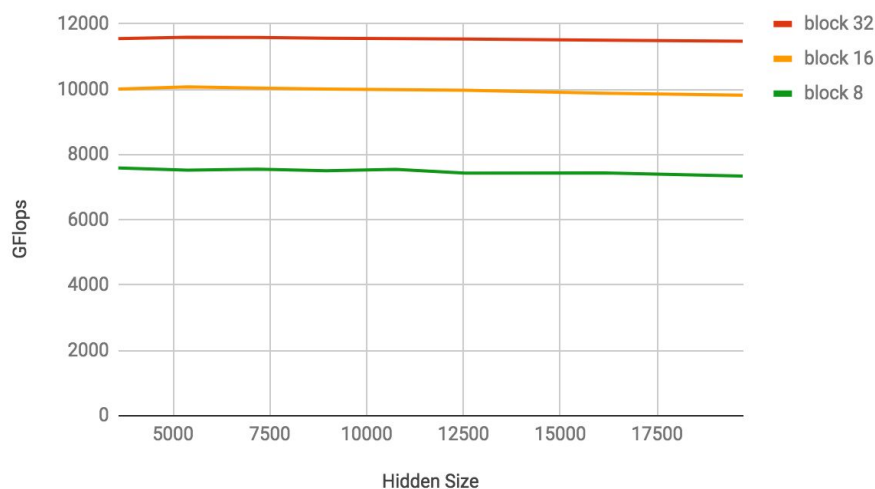
# Kernel Performance

- Here we show Volta performance with a constant number of parameters and varying density from 100% down to 3% (with a corresponding increase in hidden state size)
- Performance increases as you make the network bigger since you get more blocks of work to populate the SMs and occupancy increases
- SRAM based architectures should be able to run these style of networks with little overhead.

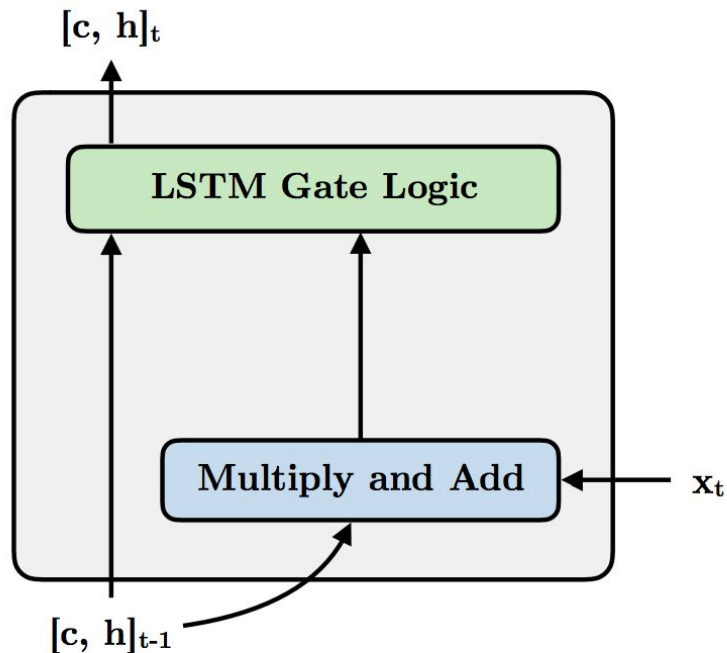
V100 - 12M Params - Forward/Backward Pass



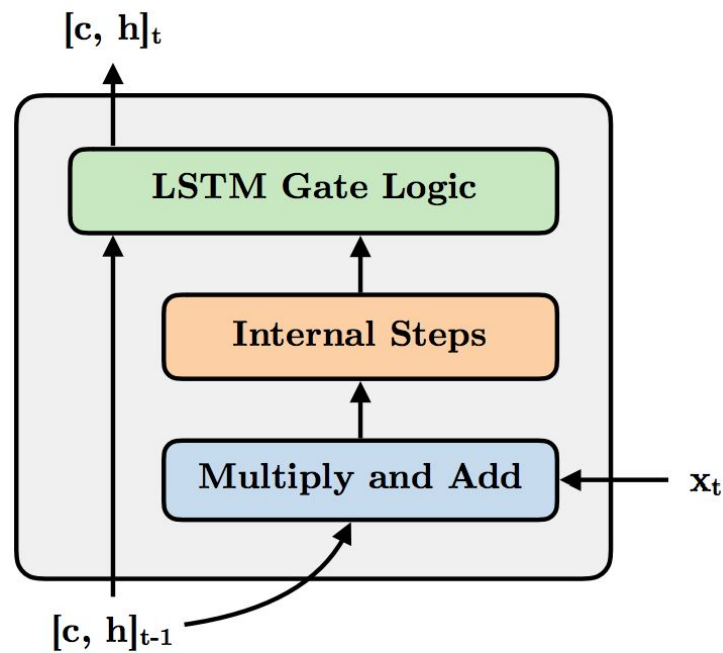
V100 - 12M Params - Weight Update Pass



# mLSTM Architecture with Added Internal Steps



(a) Multiplicative LSTM architecture



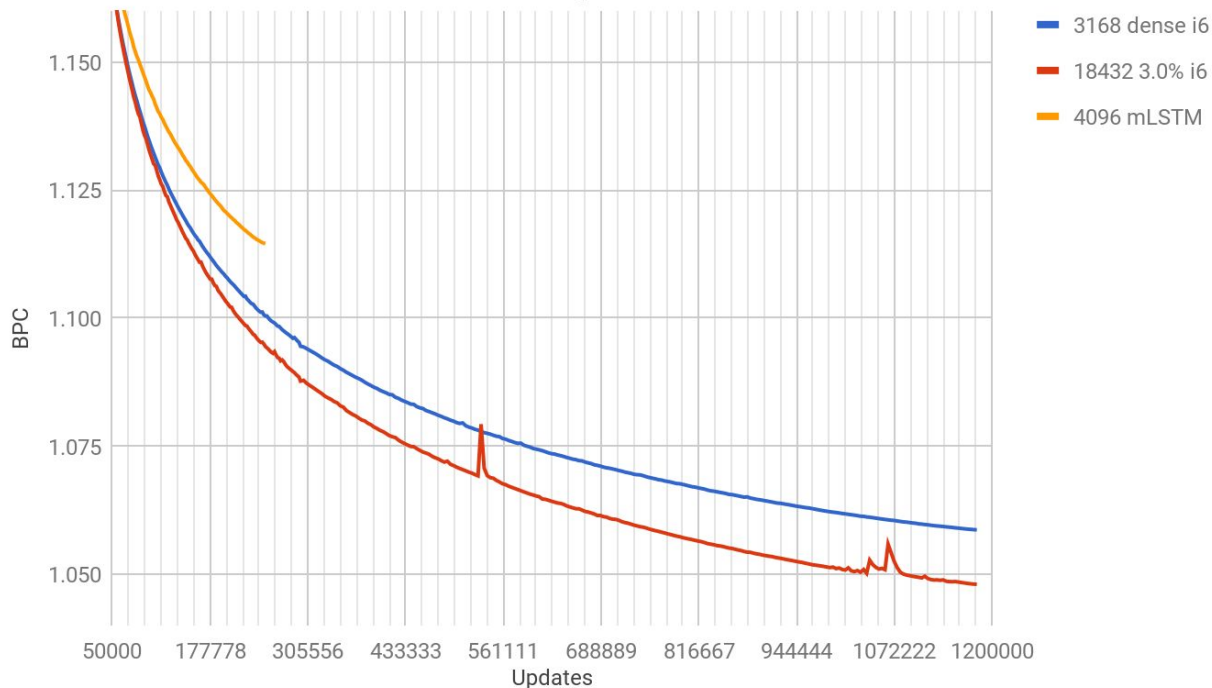
(b) Our LSTM architecture with deep updates

# Training Byte level models with mLSTMs on Amazon Reviews

- We trained 2 new models, one dense and one sparse (with 3% connectivity). Both used 6 internal steps and had the same number of parameters (100M) from the original sentiment neuron paper.
- Both were trained with an aggregate minibatch of 512 over 8 GPUS over 4 epochs of the 37GB Amazon review training set. Training time was about a month.
- The sparse model was trained on a DGX1 due to its better performance on the larger hidden state size (18k sparse vs 3k dense vs 4k original).
- The sparse model had a couple blips in validation accuracy. This was later found to be a result of pushing connectivity below 4% when using the larger blocksize of 32x32. Smaller blocks can train without issues with a greater degree of sparsity.
- The increased performance over the original model was largely due to the introduction of layernorm after every linear operation. Some additional performance was due to the deeper model with additional internal steps.

# Training Byte level models with mLSTMs on Amazon Reviews

Amazon Sentiment Models - 100M params



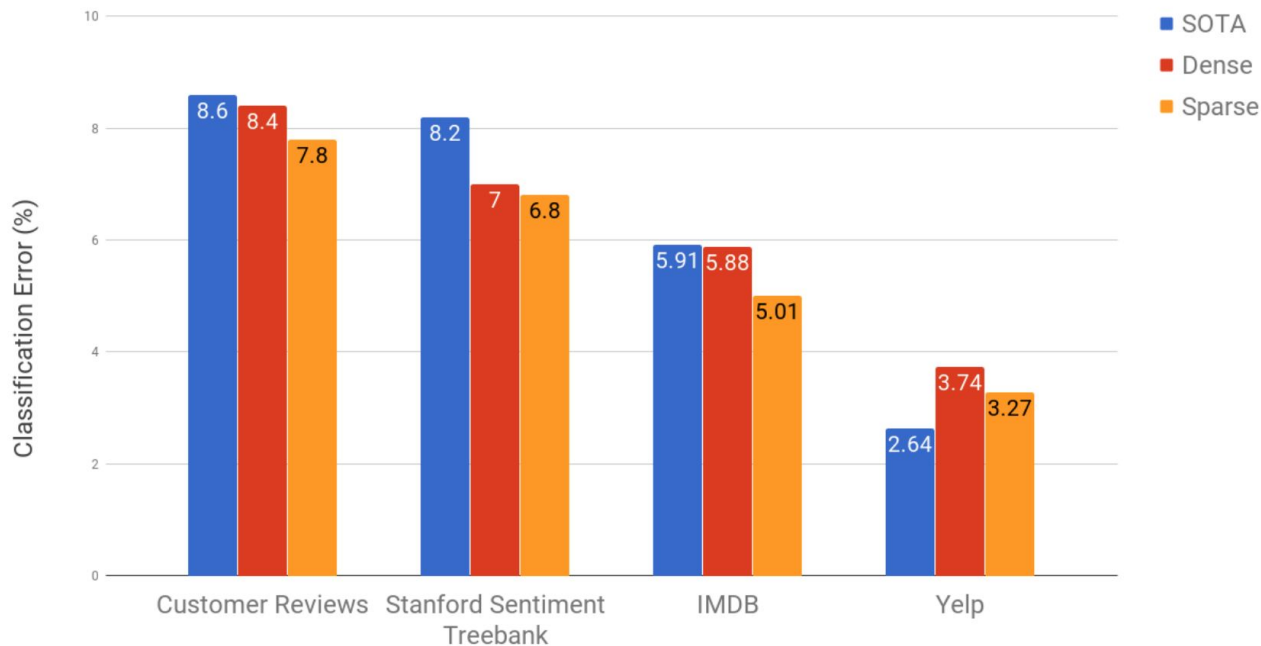


- One of the hopes of being able to train wide sparse networks was that the additional active memory would allow the model to keep track of longer time dependencies
- From the samples like the example below, it clearly can ... but so can the dense model trained with the same amount of data.
- It seems having a big enough model, trained with enough data so it can learn more abstract/higher order features is more important for this property.

## >1k character sample with highly coherent text maintained throughout:

For the money these are great **headphones**. I can wear them for several hours with no problems. The sound quality is excellent for earbuds so it's no difference from other standard headphones or speakers. I have them connected to my cell phone (Android) and listen to Sony Walkman MP3 player while I run. The sound and volume are crystal clear with or without the buds. I much prefer the memory form ear buds than any of the ear bud style they are out of production. The piece that goes over your head isn't very comfortable and I have only ever worn the during my run. When I'm not using them they fit very nicely and it is very nice to get ready at a good chance at reaching your desired song without being too close to your ears. The cord is long enough so you don't need any other extra wires for them to go in the space between the Xbox Game Cube and **audio jacks in your mp3 player**. Maybe this is no big deal for older gear but I can still do my music with them while I mow the lawn with **both ears** good. I would definitely recommend these for purchase.

# Sentiment Analysis Results

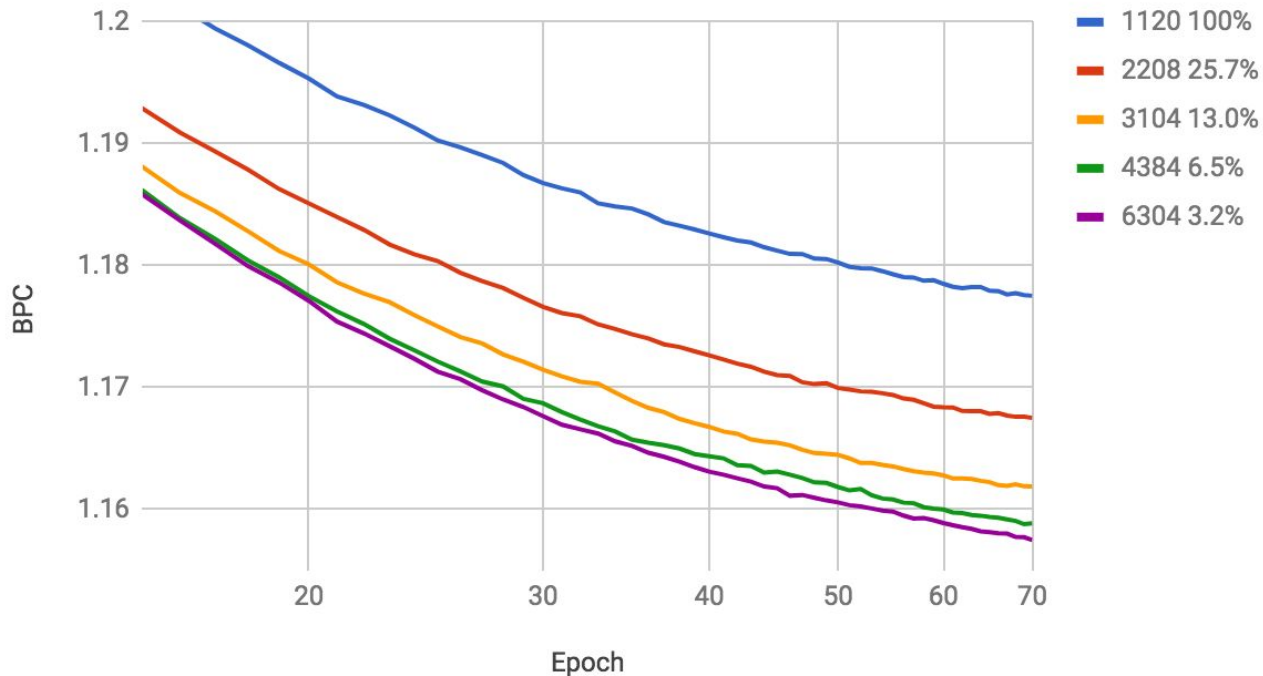


While the dense and sparse models don't differ much in sample quality, there is a big difference in sentiment classification. The wider state affords a much higher capacity linear classifier on top. The big improvement was with document level sentiment analysis (IMDB).

# Additional Experiments

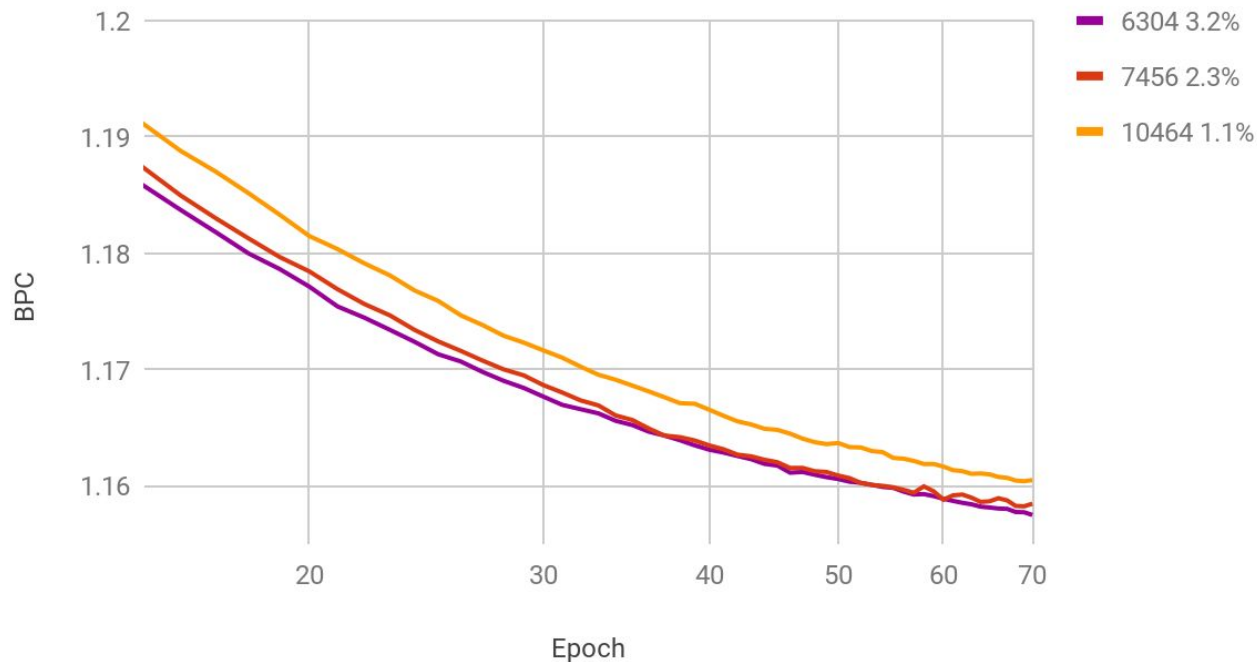
# Varying Sparsity from Dense down to 3%

Sparsity: 11M Params, 5 iSteps, 8 Blocksize

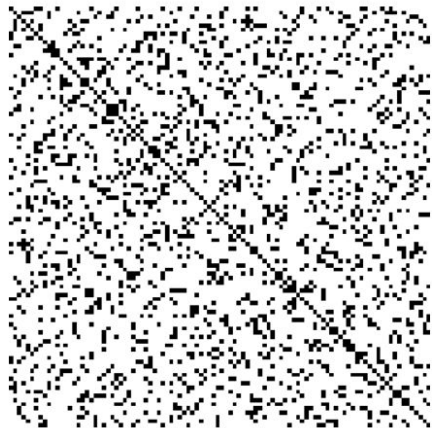


# Varying Sparsity from 3% down to 1%

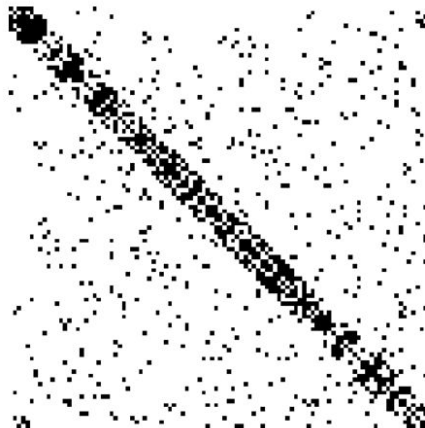
Sparsity: 11M Params, 5 iSteps, 8 Blocksize



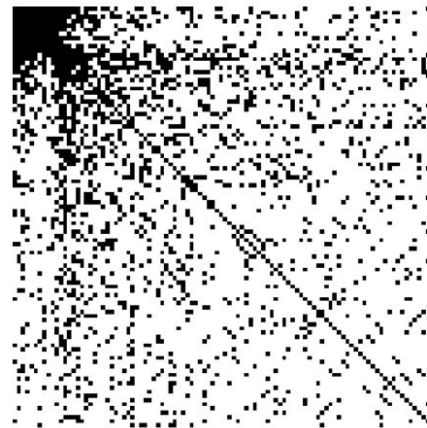
# Looking at different Small World Architectures



Random



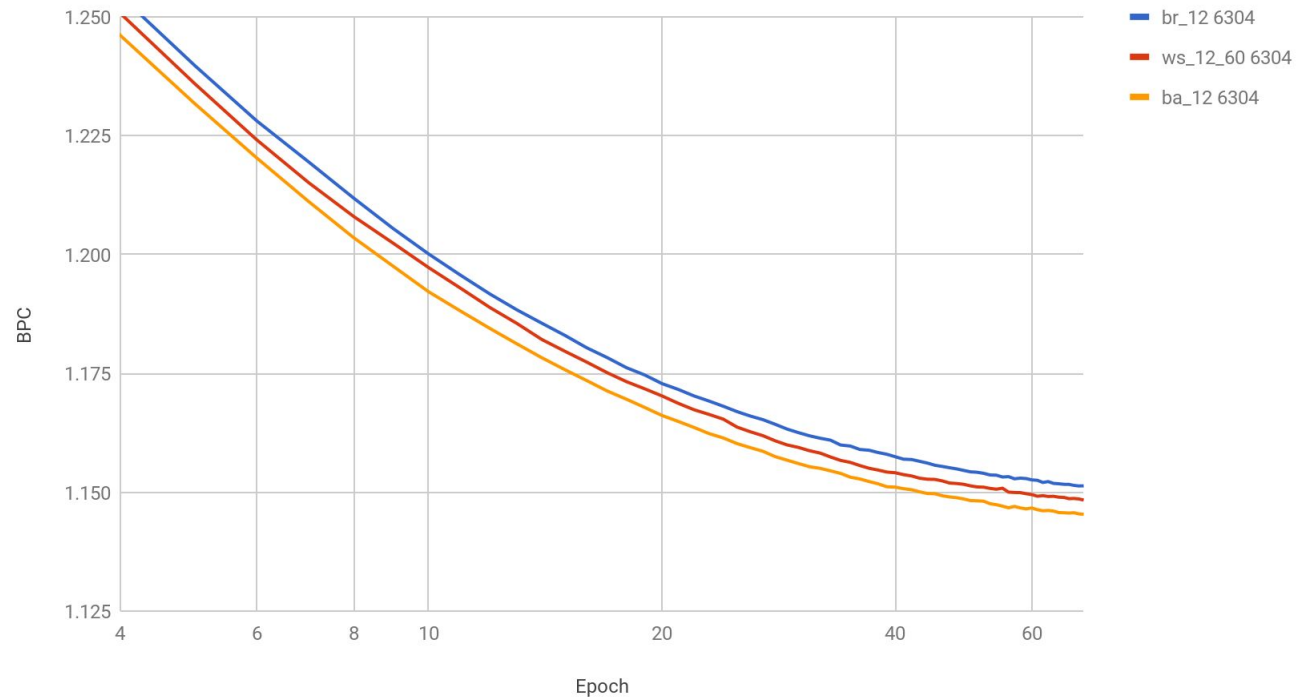
1-Dimensional Watts-Strogatz



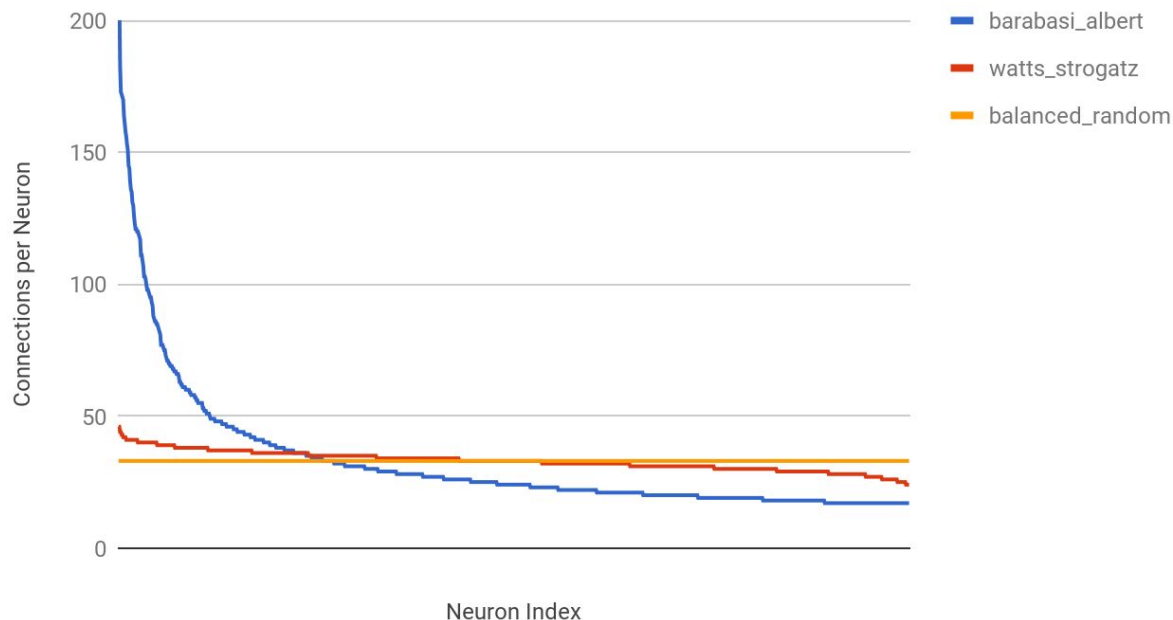
Barabasi-Albert

- We always include the diagonal blocks so that weights can be simply initialized with identity.
- When paired with layer norm this initialization performs best.

## Barabasi Albert vs Watts Strogatz vs Balanced Random



## Connections per Neuron

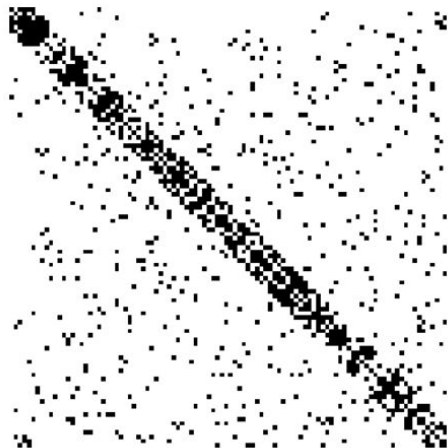


Barabasi-Albert generally follows a power law in terms of connections per neuron. Language models likely prefer some densely connected neurons. It would be interesting to probe what might be represented in these regions. Barabasi-Albert also has an advantage in terms of speed of mixing from internal step to step.

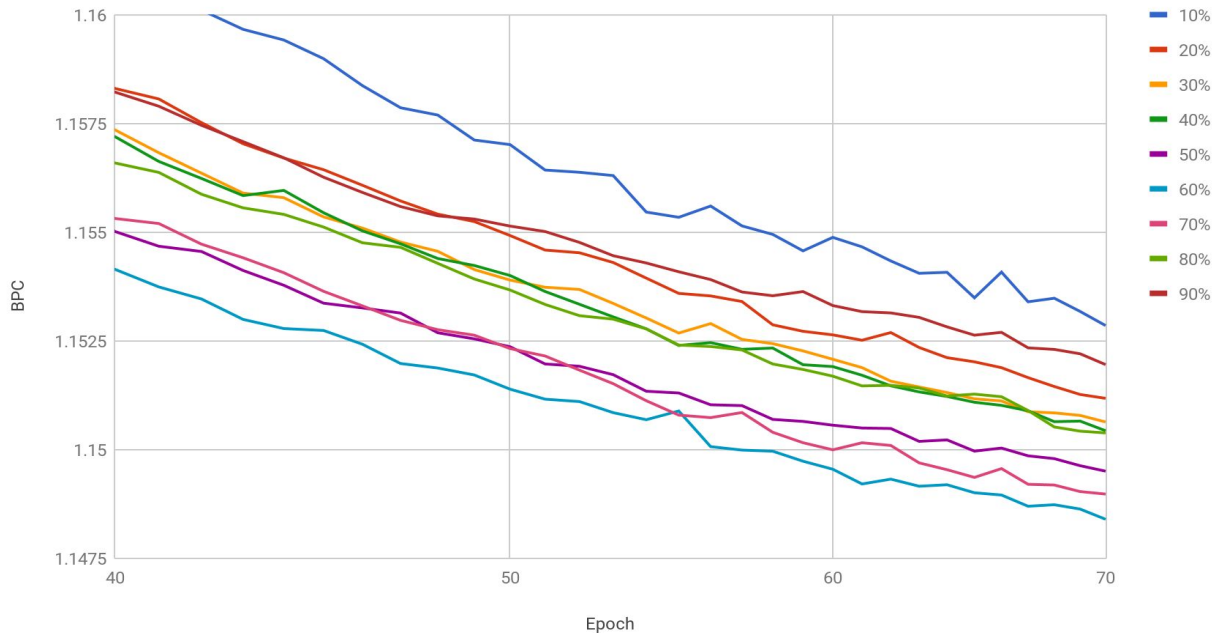


- Looking at varying the probability of connections being knocked off the diagonal band and turned into longer range random connection.
- Around 50-70% seems to work best. This is good news for hardware as having a decent amount of local structure makes this easier to optimize.

# Watts Strogattz

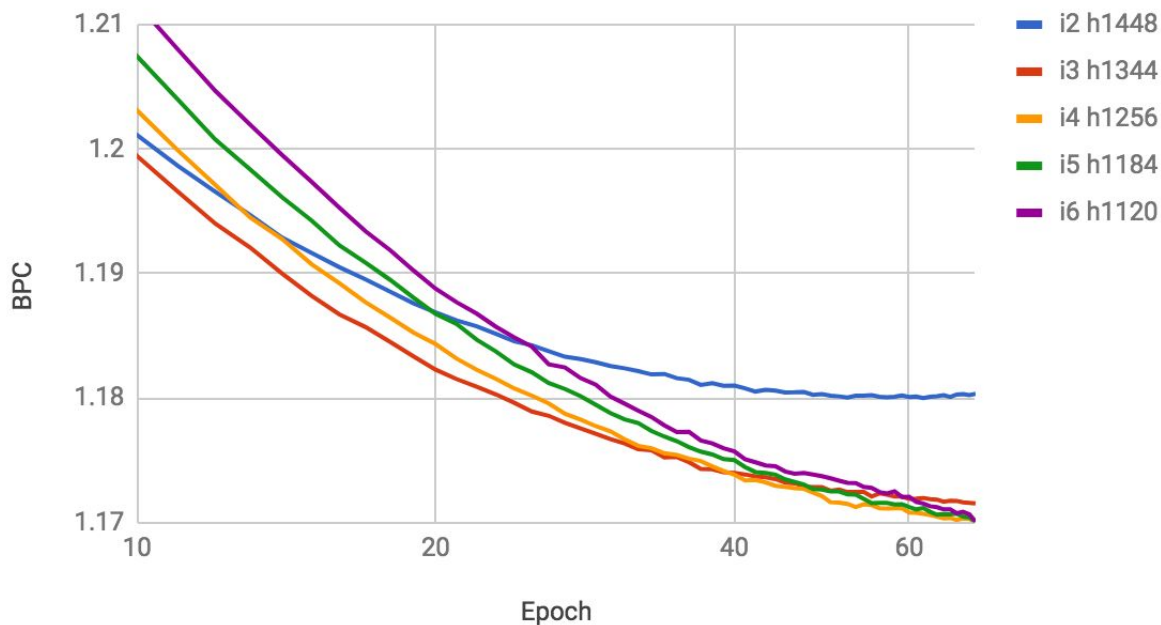


## Watts Strogatz - Comparing Probability of Long Range Connection

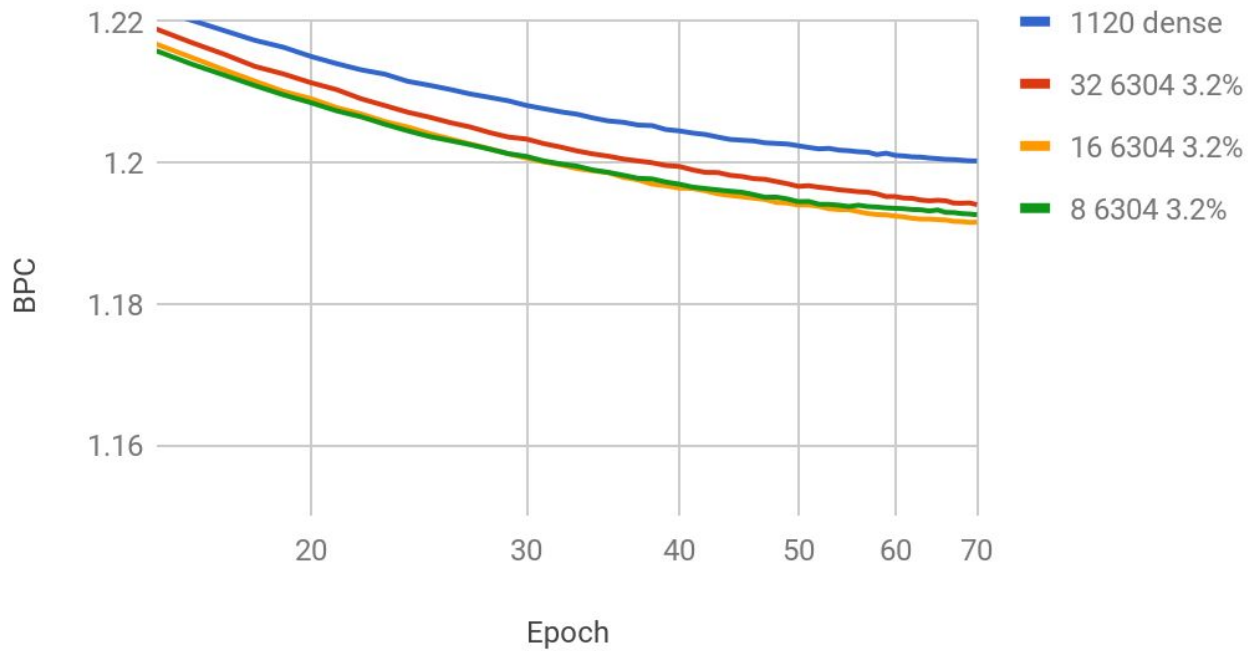


# Looking at dense networks and varying the number of internal steps

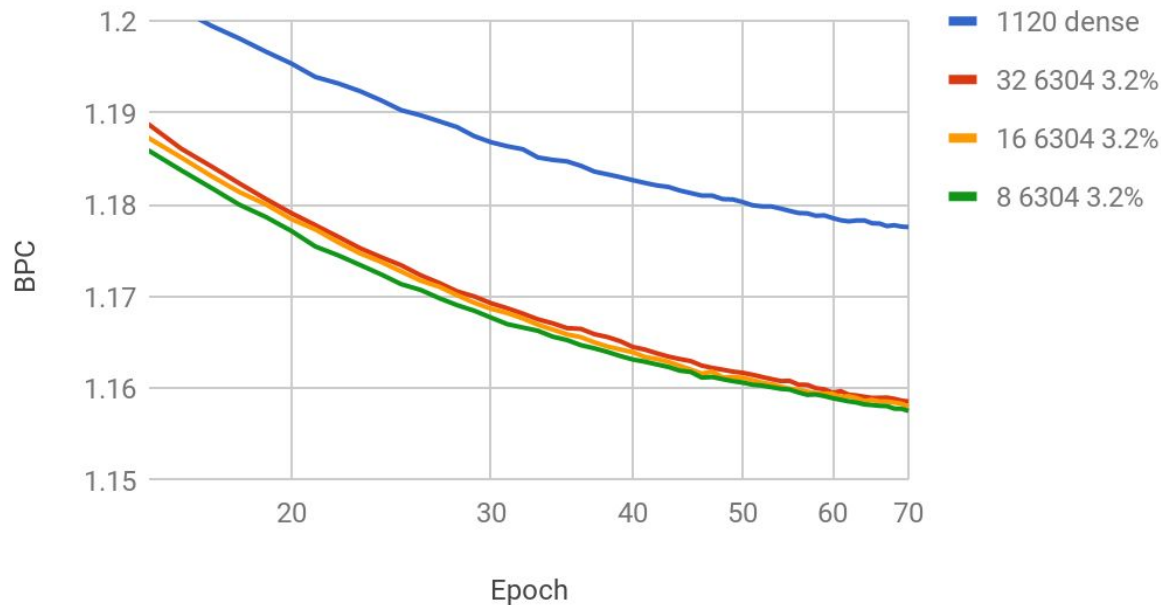
Dense iSteps comarison (12.5M Params)



## Blocksize Comparison: 3% Connected - 3 iSteps



## Blocksize Comparison: 3% Connected - 5 iSteps



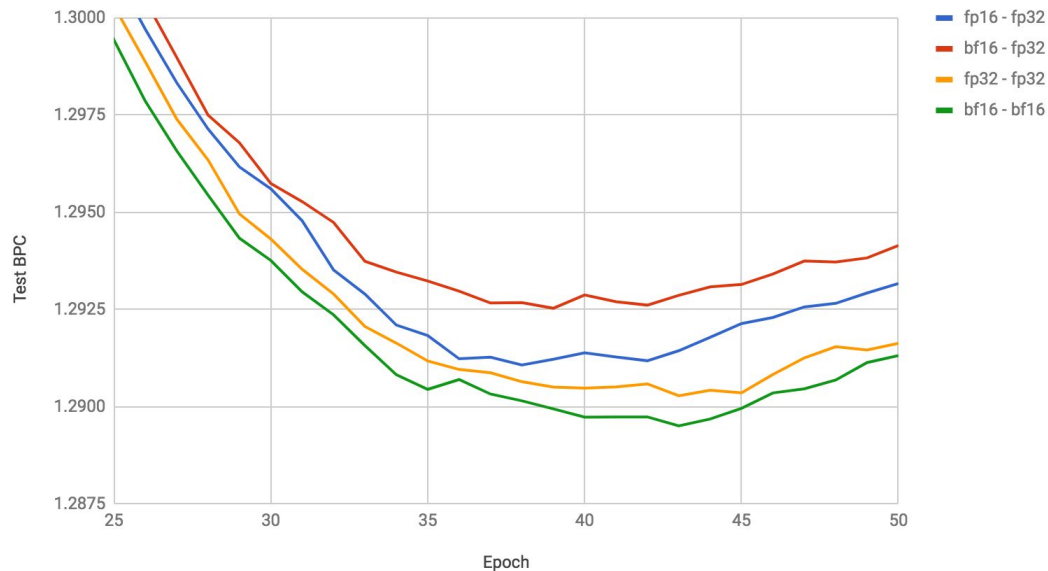
- Gap between dense and sparse widens as you increase the number of internal steps
- Surprisingly, the difference between block sizes is very small.
- This difference mostly goes away with less sparse models.

# Low Precision Training

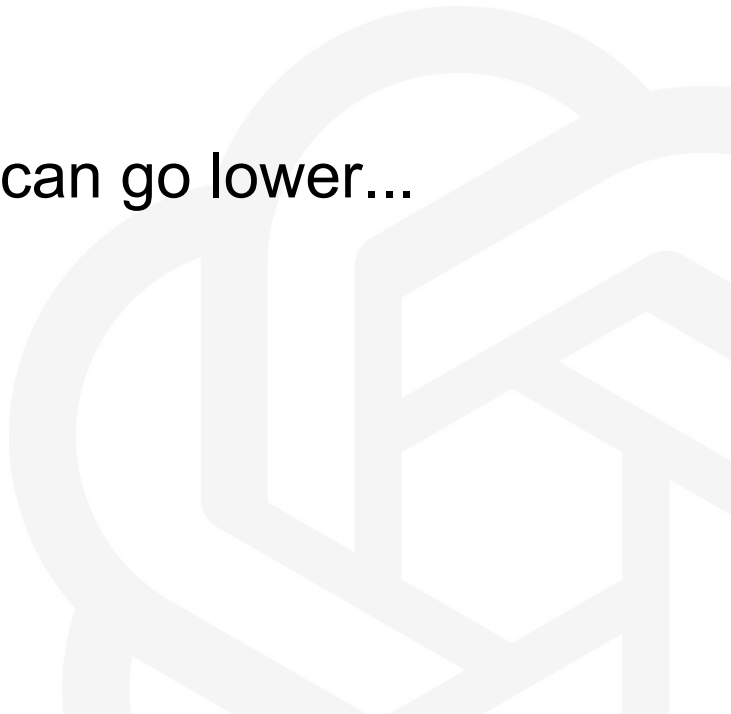
- Sparse Networks are generally bandwidth bound on today's GPUs (or any DRAM based architecture)
- Tensorcore implementations would only marginally increase performance
- Therefore, we'd like to reduce precision as much as possible to maximize bandwidth
- `tf.bfloat16` or truncated float performs just as well as IEEE float16 without the hassle of having to scale costs or worry about NaNs or Infs. Basically a drop in replacement for float32.
- Full numerical support for `tf.bfloat16` in tensorflow or other frameworks should be pretty easy to implement.

Here we compare high precision with mixed precision (low precision forward and high precision backwards) with just end to end low precision. Fp16-fp16 was omitted as cost scaling techniques weren't used here. The differences in final BPC values are tiny here but it shows that 7 bits of mantissa for the activations, weights and gradient works just fine.

Text8 - 90M Params - 16 bit Comparison

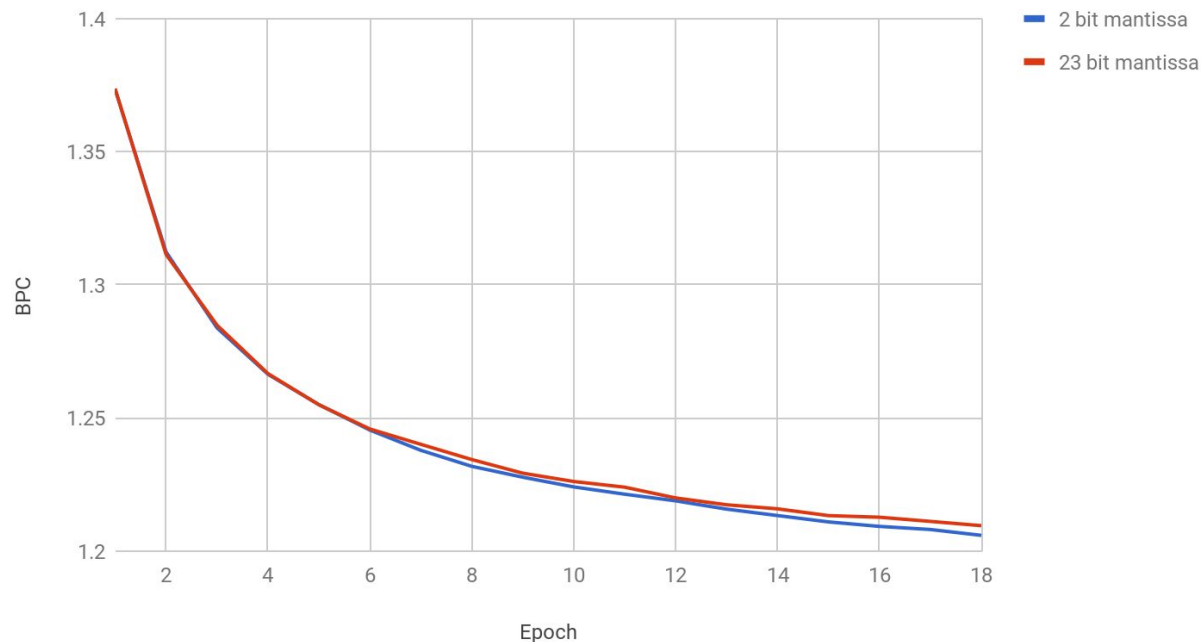


16 bit formats are nice, but we can go lower...



- Apparently, only 2 bits of mantissa in your activation and param gradient memory format are needed (provided your local accumulations are in high precision)
- More experiments are needed in different kinds of networks.

mLSTM Low Precision Gradients



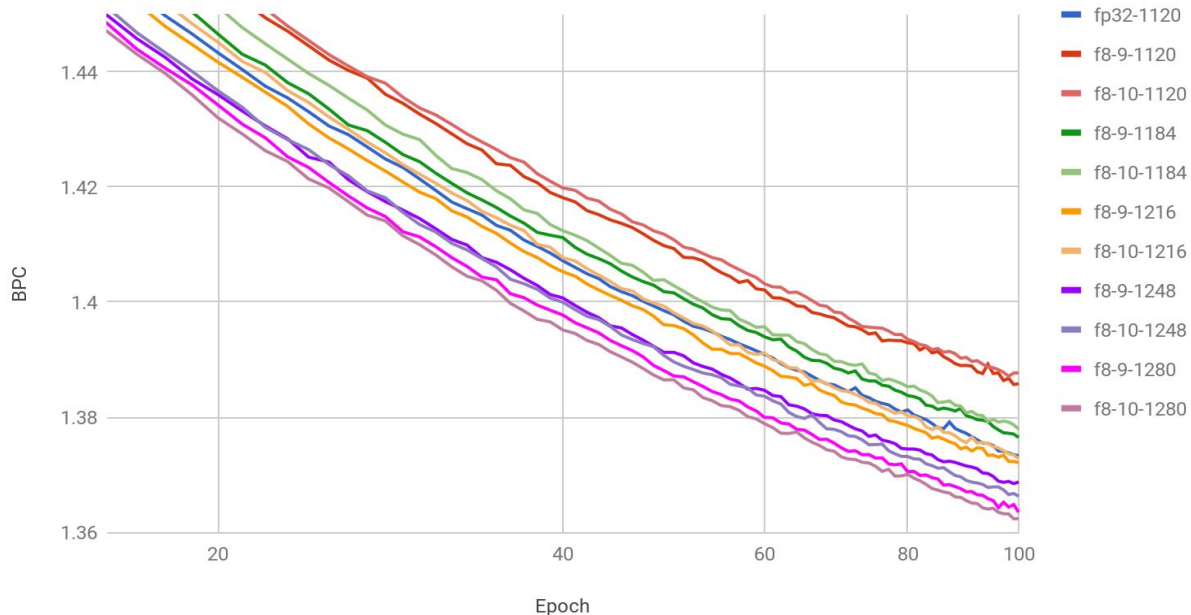


# Training a Network in FP8

- Only needing 2 bits of mantissa in the gradients means that a 1s-5e-2m fp8 should work just fine, since we know from fp16 that five bits of exponent are enough (provided the exponent bias covers the needed range)
- Gradients can use a static exponent bias range of  $\sim 2^1 - 2^{31}$
- No need to represent NaNs or Infs
- For weights and activations we can use 1s-4e-3m or 1s-3e-4m depending on the dynamic range of the network. The goal should be to maximize the bit utilization of the 8 bit format.
- The number of bits used in the forward pass for activations and weights is a big factor in the network capacity. By reducing the precision here we can make up for the capacity loss by just widening (or deepening) the network some.
- For very large networks that are over-parameterized and less parameter efficient, reducing precision can actually help increase accuracy (without adjusting size)
- Accumulations still need to be in higher precision but accumulation error is proportional to  $\log_2$  of the number of reductions. Reductions can be grouped or made more tree-like to greatly reduce the number accumulation bits required.
- Higher precision copies of the params are still needed for the weight updates.
- I've only done some limited testing with fp8 on LSTMs so far. More validation on other networks is needed.

- By increasing hidden state size by 8.5% we can match fp32 accuracy.
- As you increase the size of the network you need to bump up the activation exponent bias range to account for increased variance from deeper reductions (from a max of  $2^9$  to  $2^{10}$  here).
- Something like block floating point could manage the exponent bias automatically

LSTM Training in FP8 - Varying Hidden 1120 through 1280



# Convolution

- Currently released kernels are old and I'm almost done with replacements
- Now using direct convolution for much better performance with smaller dimensions (N, C, K)
- dense, block/grouped and depthwise separable
- pooling + norms (batchnorm, layernorm, weightnorm)
- CNHW layout for using with the CN bsmm kernels
- Might make another pass with CNHWc layout
- For separable models most params now contained in the 1x1 layers, so sparsify them. Convolution is already pretty sparse in the spatial dim.

# The End

Code: <https://github.com/openai/blocksparse>

Blog and Paper: <https://blog.openai.com/block-sparse-gpu-kernels>