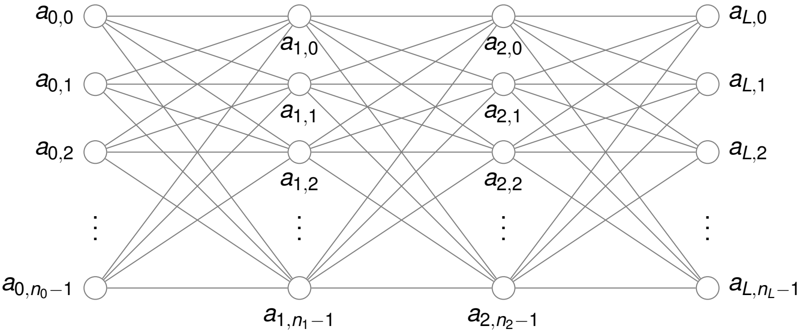
Monte Carlo Methods and Neural Networks

Noah Gamboa and Alexander Keller

Neural Networks

Fully connected layers

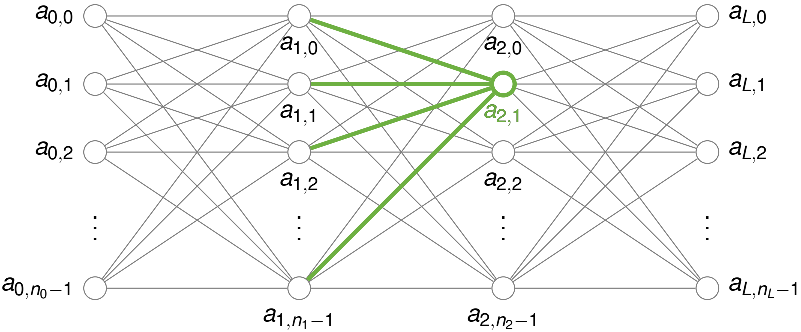
- neurons



Neural Networks

Fully connected layers

- neurons compute $\max\{0, \sum_j w_j a_{i,j}\}$



Monte Carlo Methods all over Neural Networks

Examples

- drop out
- drop connect
- stochastic binarization
- stochastic gradient descent
- fixed pseudo-random matrices for direct feedback alignment
- ...

Monte Carlo Methods all over Neural Networks

Observations

- the brain
  - about 10^{11} nerve cells with to up to 10^{4} connections to others
  - much more energy efficient than a GPU

Monte Carlo Methods all over Neural Networks

Observations

- the brain
  - about 10^{11} nerve cells with to up to 10^4 connections to others
  - much more energy efficient than a GPU

- artificial neural networks
  - rigid layer structure
  - expensive to scale in depth
  - partially trained fully connected

Monte Carlo Methods all over Neural Networks

Observations

- the brain
  - about 10^{11} nerve cells with to up to 10^{4} connections to others
  - much more energy efficient than a GPU
- artificial neural networks
  - rigid layer structure
  - expensive to scale in depth
  - partially trained fully connected
- goal: explore algorithms linear in time and space

Partition instead of Dropout

Partition instead of Dropout

Guaranteeing coverage of neural units

- so far: dropout neuron if threshold $t > \xi$
  - ξ by linear feedback register generator (for example)

Partition instead of Dropout

Guaranteeing coverage of neural units

- so far: dropout neuron if threshold $t > \xi$
  - ξ by linear feedback register generator (for example)

- now: assign neuron to partition $p = \lfloor \xi \cdot P \rfloor$ out of P
  - less random number generator calls
  - all neurons guaranteed to be considered

Partition instead of Dropout

Guaranteeing coverage of neural units

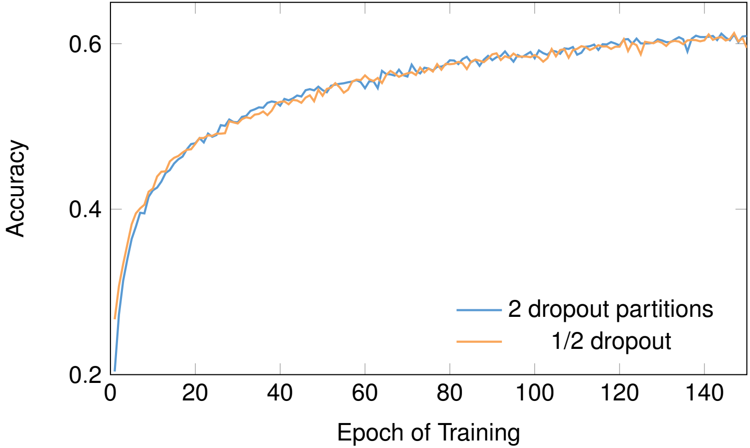
- so far: dropout neuron if threshold $t > \xi$
  - ξ by linear feedback register generator (for example)

- now: assign neuron to partition $p = \lfloor \xi \cdot P \rfloor$ out of P
  - less random number generator calls
  - all neurons guaranteed to be considered

LeNet on MNIST	Average of $t = 1/2$ to $1/9$ dropout	Average of $P = 2$ to 9 partitions
Mean accuracy	0.6062	0.6057
StdDev accuracy	0.0106	0.009

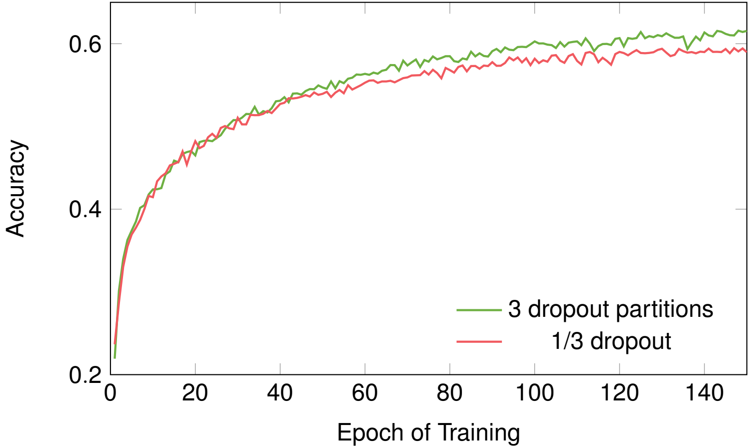
Partition instead of Dropout

Training accuracy with LeNet on MNIST



Partition instead of Dropout

Training accuracy with LeNet on MNIST

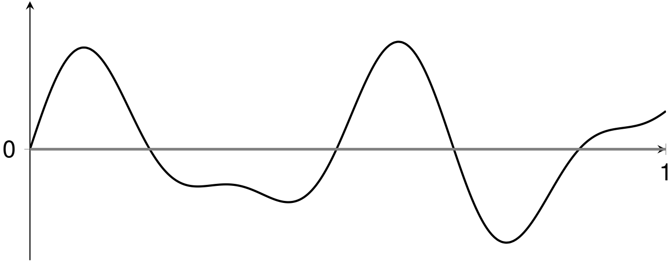


Simulating Discrete Densities

Simulating Discrete Densities

Stochastic evaluation of scalar product

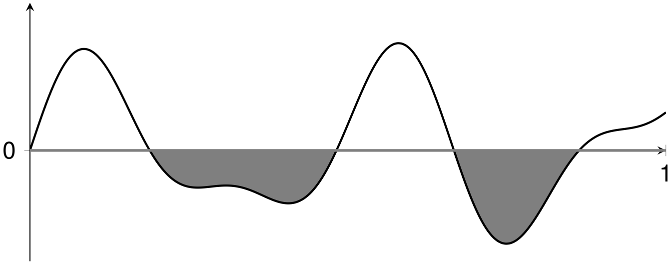
- discrete density approximation of the weights



Simulating Discrete Densities

Stochastic evaluation of scalar product

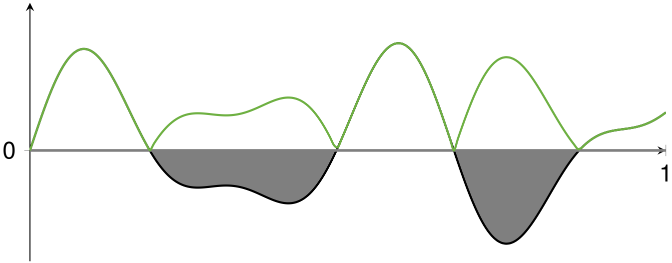
- discrete density approximation of the weights



Simulating Discrete Densities

Stochastic evaluation of scalar product

- discrete density approximation of the weights

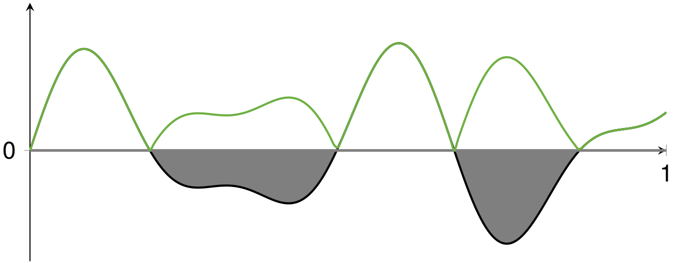


  - remember to flip sign accordingly

Simulating Discrete Densities

Stochastic evaluation of scalar product

- discrete density approximation of the weights



  - remember to flip sign accordingly
  - transform jittered equidistant samples using cumulative distribution function of absolute value of weights

Simulating Discrete Densities

Stochastic evaluation of scalar product

- partition of unit interval by sums $P_k := \sum_{j=1}^{k} |w_j|$ of normalized absolute weights

$$0 = P_0 < P_1 < \cdots < P_m = 1$$

Simulating Discrete Densities

Stochastic evaluation of scalar product

- partition of unit interval by sums $P_k := \sum_{j=1}^{k} |w_j|$ of normalized absolute weights

$$0 = P_0 < P_1 < \cdots < P_m = 1$$

  - using a uniform random variable $\xi \in [0,1)$ we find

    select neuron $i \Leftrightarrow P_{i-1} \leq \xi < P_i$ satisfying $\text{Prob}(\{P_{i-1} \leq \xi < P_i\}) = |w_i|$

Simulating Discrete Densities

Stochastic evaluation of scalar product

- partition of unit interval by sums $P_k := \sum_{j=1}^{k} |w_j|$ of normalized absolute weights

$$0 = P_0 < P_1 < \cdots < P_m = 1$$

  - using a uniform random variable $\xi \in [0,1)$ we find

    select neuron $i \Leftrightarrow P_{i-1} \leq \xi < P_i$ satisfying $\text{Prob}(\{P_{i-1} \leq \xi < P_i\}) = |w_i|$

  - transform jittered equidistant samples using cumulative distribution function of absolute value of weights

Simulating Discrete Densities

Stochastic evaluation of scalar product

- partition of unit interval by sums $P_k := \sum_{j=1}^{k} |w_j|$ of normalized absolute weights

$$0 = P_0 < P_1 < \cdots < P_m = 1$$

  - using a uniform random variable $\xi \in [0,1)$ we find

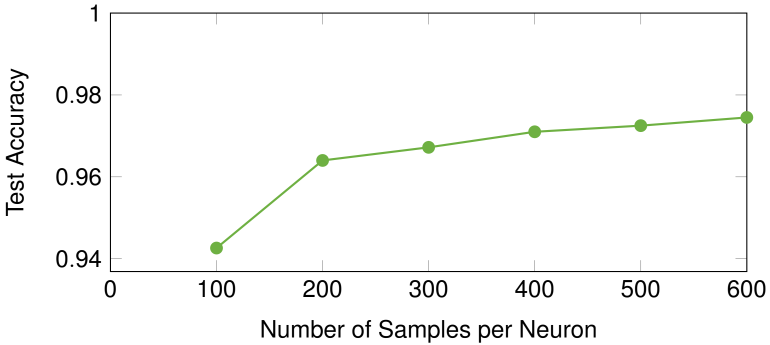
    select neuron $i \Leftrightarrow P_{i-1} \le \xi < P_i$ satisfying $\text{Prob}(\{P_{i-1} \le \xi < P_i\}) = |w_i|$
  - transform jittered equidistant samples using cumulative distribution function of absolute value of weights

- in fact derivation of quantization to weights in $\{-1, 0, +1\}$
  - integer weights if a neuron referenced more than once
  - explains why ternary and binary did not work in some articles
  - relation to drop connect and drop out, too

Simulating Discrete Densities

Test accuracy for two layer ReLU feedforward network on MNIST

- able to achieve 97% of accuracy of model by sampling most important 12% of weights!



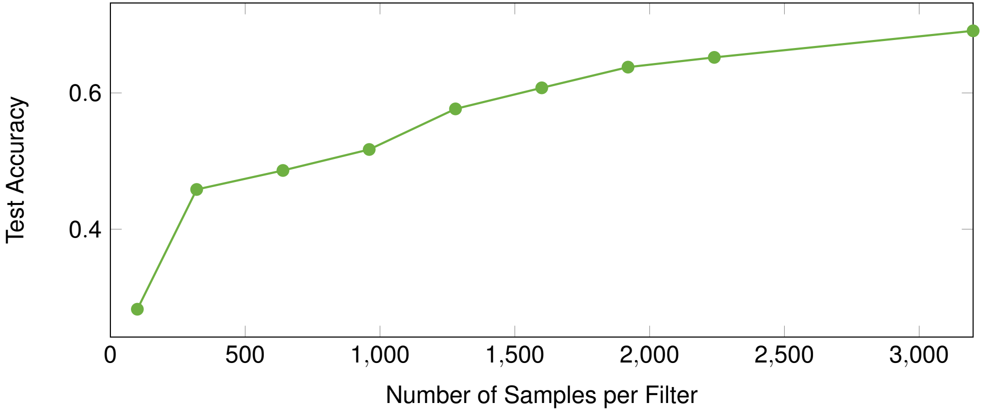
Simulating Discrete Densities

Application to convolutional layers

- sample from distribution of filter (for example, 128x5x5 = 3200)
  - less redundant than fully connected layers
- LeNet Architecture on CIFAR-10, best accuracy is 0.6912
- able to get 88% of accuracy of full model at 50% sampled

Simulating Discrete Densities

Test accuracy for LeNet on CIFAR-10



Neural Networks linear in Time and Space

Neural Networks linear in Time and Space

Number n of neural units

- for L fully connected layers

$$n = \sum_{l=1}^{L} n_l$$

where n_l is the number of neurons in layer l

Neural Networks linear in Time and Space

Number n of neural units

- for L fully connected layers

$$n = \sum_{l=1}^{L} n_l$$

where n_l is the number of neurons in layer l

- number of weights

$$n_w = \sum_{l=1}^{L} n_{l-1} \cdot n_l$$

Neural Networks linear in Time and Space

Number n of neural units

- for L fully connected layers

$$n = \sum_{l=1}^{L} n_l$$

where n_l is the number of neurons in layer l

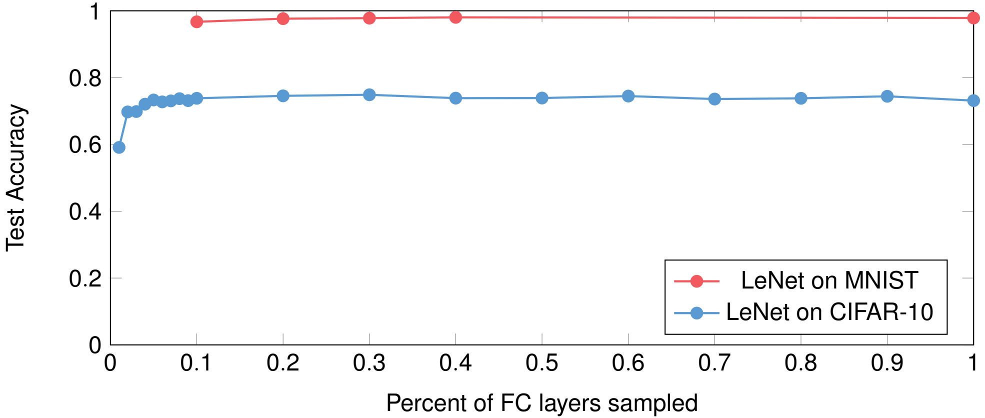
- number of weights

$$n_w = \sum_{l=1}^{L} n_{l-1} \cdot n_l$$

- choose number of weights per neuron such that n proportional to n_w
  - for example, constant number n_w of weights per neuron

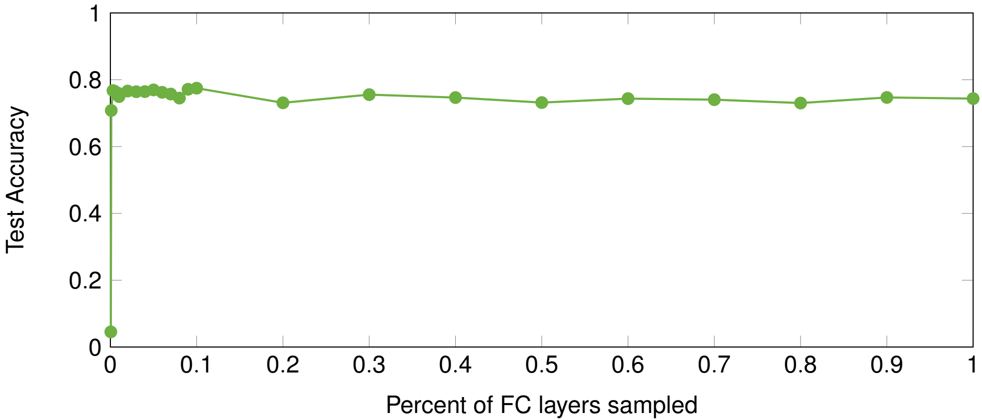
Neural Networks linear in Time and Space

Results



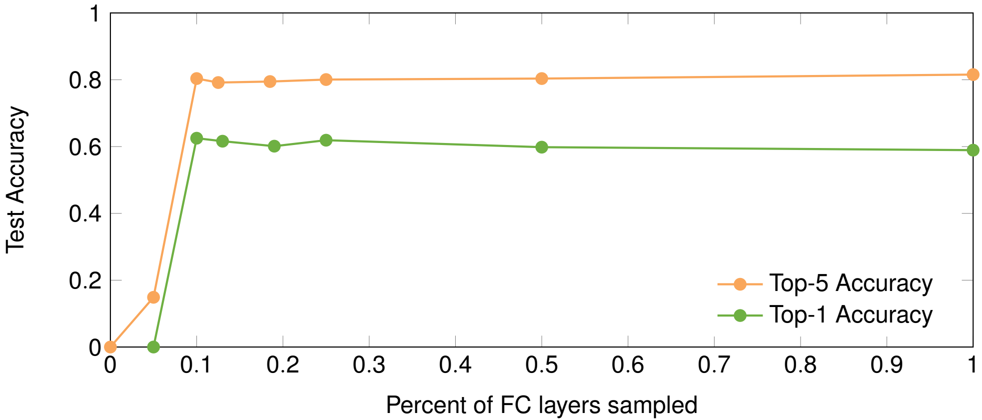
Neural Networks linear in Time and Space

Test accuracy for AlexNet on CIFAR-10



Neural Networks linear in Time and Space

Test accuracy for AlexNet on ILSVRC12



Neural Networks linear in Time and Space

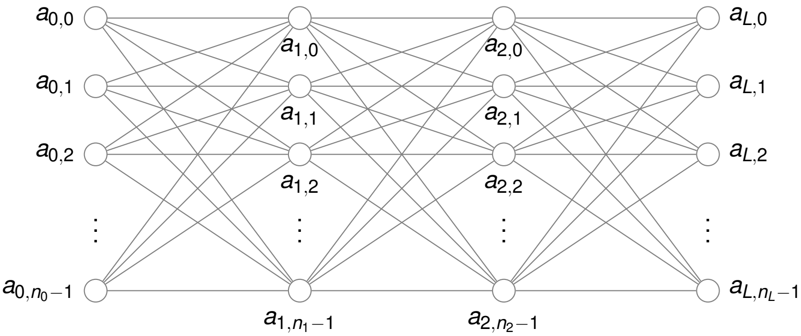
Sampling paths through networks

- complexity bounded by number of paths times depth
- strong indication of relation to Markov chains
- importance sampling by weights

Neural Networks linear in Time and Space

Sampling paths through networks

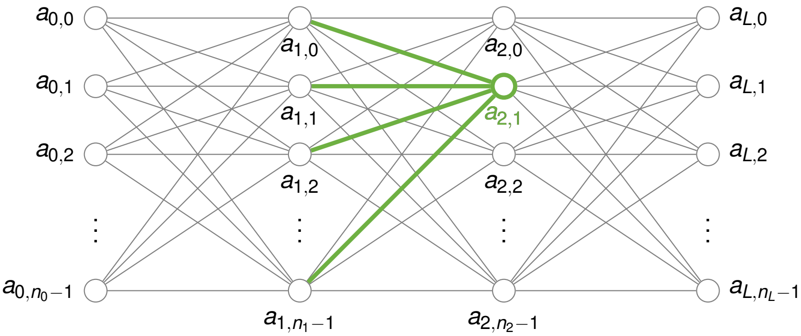
- sparse from scratch



Neural Networks linear in Time and Space

Sampling paths through networks

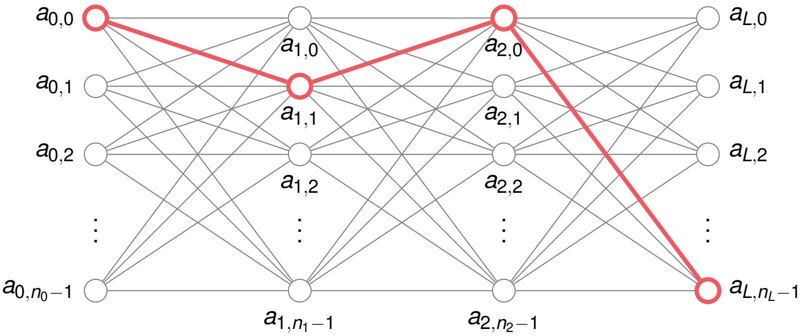
- sparse from scratch



Neural Networks linear in Time and Space

Sampling paths through networks

- sparse from scratch

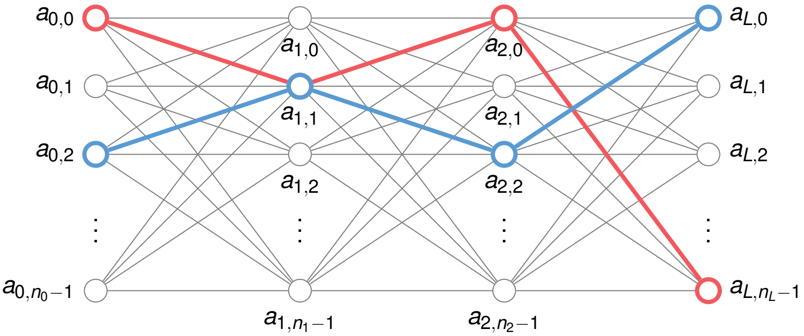


  - guaranteed connectivity

Neural Networks linear in Time and Space

Sampling paths through networks

- sparse from scratch

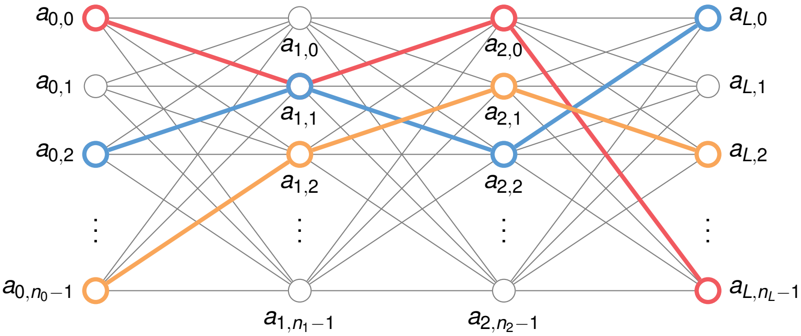


  - guaranteed connectivity

Neural Networks linear in Time and Space

Sampling paths through networks

- sparse from scratch

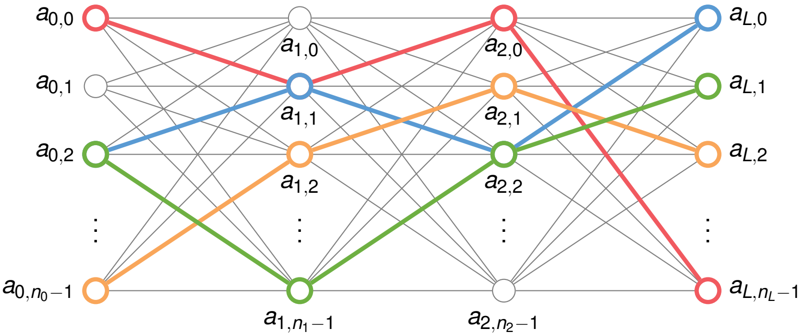


  - guaranteed connectivity

Neural Networks linear in Time and Space

Sampling paths through networks

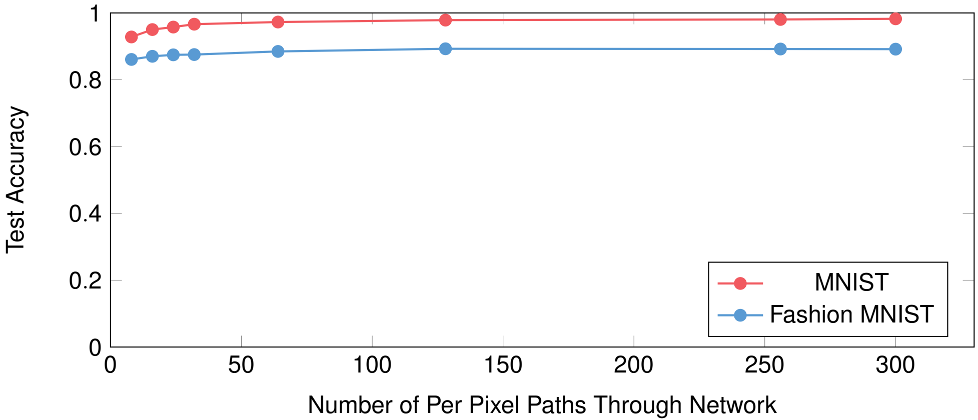
- sparse from scratch



  - guaranteed connectivity

Neural Networks linear in Time and Space

Test accuracy for 4 layer feedforward network (784/300/300/10)



Monte Carlo Methods and Neural Networks

Monte Carlo Methods and Neural Networks

Summary

- dropout partitions reduce variance
  - using much less random numbers

Monte Carlo Methods and Neural Networks

Summary

- dropout partitions reduce variance
  - using much less random numbers

- simulating discrete densities explains $\{-1, 0, 1\}$ and integer weights
  - compression and quantization without retraining

Monte Carlo Methods and Neural Networks

Summary

- dropout partitions reduce variance
  - using much less random numbers

- simulating discrete densities explains $\{-1, 0, 1\}$ and integer weights
  - compression and quantization without retraining

- neural networks with linear complexity for both inference and training
  - sparse from scratch
  - sampling paths through neural networks instead of drop connect and drop out