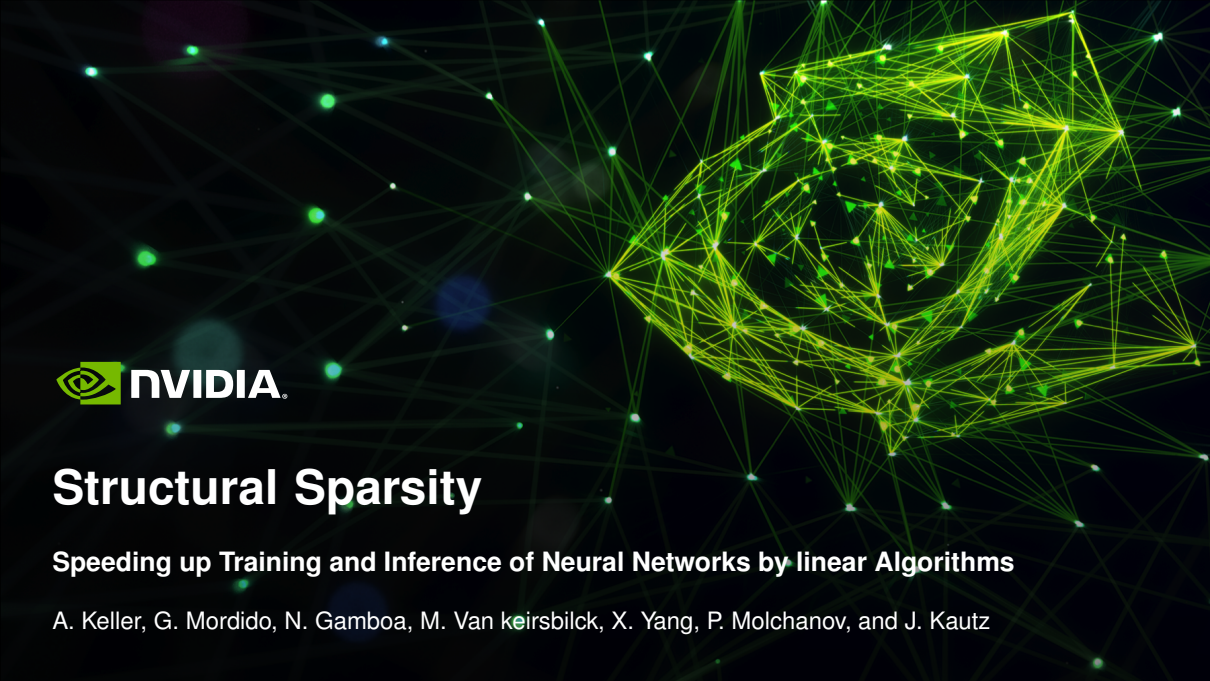


Structural Sparsity

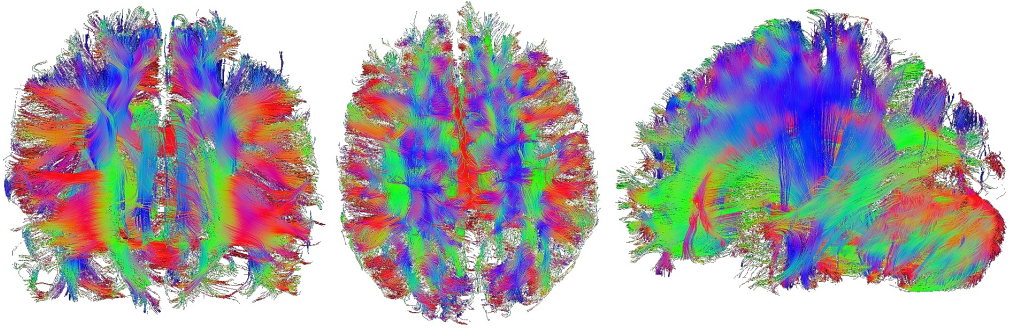
Speeding up Training and Inference of Neural Networks by linear Algorithms

A. Keller, G. Mordido, N. Gamboa, M. Van keirsbilck, X. Yang, P. Molchanov, and J. Kautz

Structural Sparsity

Speeding up training and inference of neural networks by linear algorithms

- the brain cannot be fully connected
  - 10^{11} nerve cells, but only up to 10^4 connections each

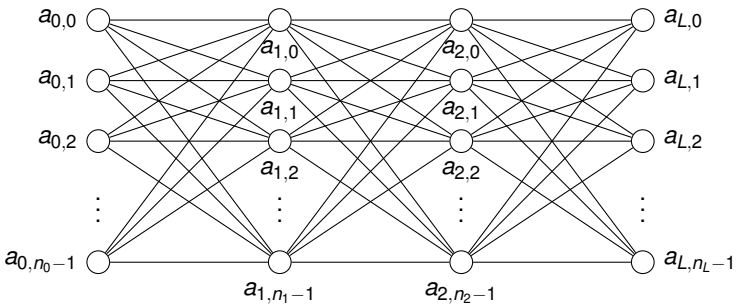


▶ image source Pixabay

Artificial Neural Networks in a Nutshell

Learning high dimensional function approximation

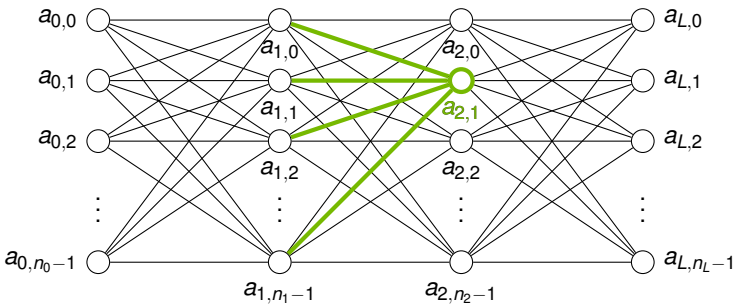
- input layer a_0, $L-1$ fully connected hidden layers, and output layer a_L



Artificial Neural Networks in a Nutshell

Learning high dimensional function approximation

- input layer a_0, $L-1$ fully connected hidden layers, and output layer a_L

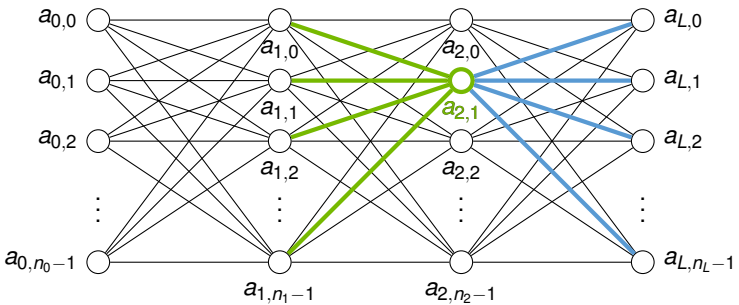


  - n_l rectified linear units (ReLU) $a_{l,i} = \max\{0, \sum w_{l,j,i} \cdot a_{l-1,j}\}$ in layer l

Artificial Neural Networks in a Nutshell

Learning high dimensional function approximation

- input layer a_0, $L-1$ fully connected hidden layers, and output layer a_L

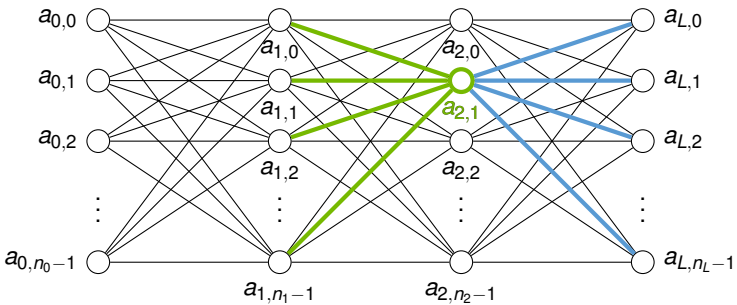


  - n_l rectified linear units (ReLU) $a_{l,i} = \max\{0, \sum w_{l,j,i} \cdot a_{l-1,j}\}$ in layer l
  - backpropagating the error $\delta_{l-1,i} = \sum_{a_{l,j}>0} \delta_{l,j} \cdot w_{l,j,i}$

Artificial Neural Networks in a Nutshell

Learning high dimensional function approximation

- input layer a_0, $L-1$ fully connected hidden layers, and output layer a_L



- n_l rectified linear units (ReLU) $a_{l,i} = \max\{0, \sum w_{l,j,i} \cdot a_{l-1,j}\}$ in layer l
- backpropagating the error $\delta_{l-1,i} = \sum_{a_{l,j}>0} \delta_{l,j} \cdot w_{l,j,i}$, update weights $w'_{l,j,i} = w_{l,j,i} - \lambda \cdot \delta_{l,j} \cdot a_{l-1,i}$ if $a_{l,j} > 0$

Artificial Neural Networks in a Nutshell

Complexity

- number of neural units

$$n = \sum_{l=1}^{L} n_l \qquad \text{where } n_l \text{ is the number of neurons in layer } l$$

Artificial Neural Networks in a Nutshell

Complexity

- number of neural units

$$n = \sum_{l=1}^{L} n_l \qquad \text{where } n_l \text{ is the number of neurons in layer } l$$

- number of weights

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

Artificial Neural Networks in a Nutshell

Complexity

- number of neural units

$$n = \sum_{l=1}^{L} n_l \qquad \text{where } n_l \text{ is the number of neurons in layer } l$$

- number of weights

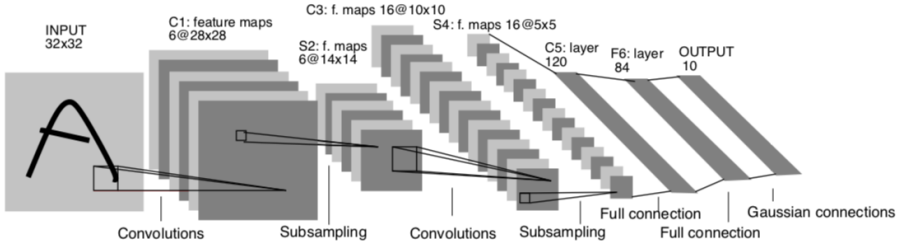
$$n_w = \sum_{l=1}^{L} c \cdot n_l = c \cdot n$$

  - select constant number c of weights per neuron
  - time and space linear in n

Artificial Neural Networks in a Nutshell

Convolutional neural networks

- feature map defined by convolution kernel
  - weight sharing across all neural units of one feature map



► Gradient based learning applied to document recognition

- maximum pooling layer: maximum of tile of neurons in feature map to reduce resolution

Relations to Mathematical Objects

Relations to Mathematical Objects

Maximum pooling layers

- rectified linear unit $\mathrm{ReLU}(x) := \max\{0, x\}$ as a basic non-linearity

Relations to Mathematical Objects

Maximum pooling layers

- rectified linear unit $\mathrm{ReLU}(x) := \max\{0, x\}$ as a basic non-linearity
- for example, leaky ReLU is

$$\mathrm{ReLU}(x) - \alpha \cdot \mathrm{ReLU}(-x)$$

Relations to Mathematical Objects

Maximum pooling layers

- rectified linear unit $\mathrm{ReLU}(x) := \max\{0, x\}$ as a basic non-linearity
- for example, leaky ReLU is

$$\mathrm{ReLU}(x) - \alpha \cdot \mathrm{ReLU}(-x)$$

which for $\alpha = -1$ yields the absolute value

$$|x| = \mathrm{ReLU}(x) + \mathrm{ReLU}(-x)$$

Relations to Mathematical Objects

Maximum pooling layers

- rectified linear unit $\mathrm{ReLU}(x) := \max\{0, x\}$ as a basic non-linearity
- for example, leaky ReLU is

$$\mathrm{ReLU}(x) - \alpha \cdot \mathrm{ReLU}(-x)$$

  which for $\alpha = -1$ yields the absolute value

$$|x| = \mathrm{ReLU}(x) + \mathrm{ReLU}(-x)$$

- hence the maximum of two values is

$$\max\{x, y\} = \frac{x+y}{2} + \left|\frac{x-y}{2}\right| = \frac{1}{2} \cdot (x + y + \mathrm{ReLU}(x-y) + \mathrm{ReLU}(y-x))$$

  which allows one to represent maximum pooling by ReLU functions and introduces skip links

Relations to Mathematical Objects

Layers as iteration

- relation of a residual layer and a differential equation by introduction a step size h

$$a_l = a_{l-1} + h \cdot \quad W_l^{(2)} \max\left\{0, W_l^{(1)} \cdot a_{l-1}\right\}$$

Relations to Mathematical Objects

Layers as iteration

- relation of a residual layer and a differential equation by introduction a step size h

$$a_l = a_{l-1} + h \cdot \quad W_l^{(2)} \max\left\{0, W_l^{(1)} \cdot a_{l-1}\right\} \qquad \text{resembles Euler method}$$

$$\Leftrightarrow \qquad \frac{a_l - a_{l-1}}{h}$$

Relations to Mathematical Objects

Layers as iteration

- relation of a residual layer and a differential equation by introduction a step size h

$$a_l = a_{l-1} + h \cdot \quad W_l^{(2)} \max\left\{0, W_l^{(1)} \cdot a_{l-1}\right\} \qquad \text{resembles Euler method}$$

$$\Leftrightarrow \quad \frac{a_l - a_{l-1}}{h} = \quad W_l^{(2)} \max\left\{0, W_l^{(1)} \cdot a_{l-1}\right\} \qquad \text{which for } h \to 0 \text{ becomes } \dot{a}_l$$

Relations to Mathematical Objects

Layers as iteration

- relation of a residual layer and a differential equation by introduction a step size h

$$a_l = a_{l-1} + h \cdot \quad W_l^{(2)} \max\left\{0, W_l^{(1)} \cdot a_{l-1}\right\} \qquad \text{resembles Euler method}$$

$$\Leftrightarrow \qquad \frac{a_l - a_{l-1}}{h} = \quad W_l^{(2)} \max\left\{0, W_l^{(1)} \cdot a_{l-1}\right\} \qquad \text{which for } h \to 0 \text{ becomes } \dot{a}_l$$

  - select your favorite ordinary differential equation to determine $W_l^{(1)}$ and $W_l^{(2)}$
    - Neural networks motivated by partial differential Equations
  - use your favorite ordinary differential equation solver for both inference and training
    - A radical new neural network design could overcome big challenges in AI

Relations to Mathematical Objects

Learning integral operator kernels

- neural unit with ReLU

$$a_{l,j} := \max\left\{0, \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} a_{l-1,i}\right\}$$

Relations to Mathematical Objects

Learning integral operator kernels

- neural unit with ReLU

$$a_{l,j} := \max\left\{0, \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} a_{l-1,i}\right\} \quad \rightarrow \quad a_{l,j} := \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} \max\{0, a_{l-1,i}\}$$

Relations to Mathematical Objects

Learning integral operator kernels

- neural unit with ReLU

$$a_{l,j} := \max\left\{0, \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} a_{l-1,i}\right\} \quad \rightarrow \quad a_{l,j} := \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} \max\{0, a_{l-1,i}\}$$

written in continuous form relates to high-dimensional integro-approximation

$$a_l(y) := \int_0^1 w_l(x,y) \max\{0, a_{l-1}(x)\} dx$$

Relations to Mathematical Objects

Learning integral operator kernels

- neural unit with ReLU

$$a_{l,j} := \max\left\{0, \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} a_{l-1,i}\right\} \quad \rightarrow \quad a_{l,j} := \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} \max\{0, a_{l-1,i}\}$$

  written in continuous form relates to high-dimensional integro-approximation

$$a_l(y) := \int_0^1 w_l(x,y) \max\{0, a_{l-1}(x)\} dx$$

- recurrent neural network layer in continuous form alludes to integral equation

$$a'_l(y) := \int_0^1 w_l(x,y) \max\{0, a_{l-1}(x)\} dx$$

Relations to Mathematical Objects

Learning integral operator kernels

- neural unit with ReLU

$$a_{l,j} := \max\left\{0, \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} a_{l-1,i}\right\} \quad \rightarrow \quad a_{l,j} := \sum_{i=0}^{n_{l-1}-1} w_{l,j,i} \max\{0, a_{l-1,i}\}$$

  written in continuous form relates to high-dimensional integro-approximation

$$a_l(y) := \int_0^1 w_l(x,y) \max\{0, a_{l-1}(x)\} dx$$

- recurrent neural network layer in continuous form alludes to integral equation

$$a'_l(y) := \int_0^1 w_l(x,y) \max\{0, a_{l-1}(x)\} dx + \int_0^1 w_l^h(x,y) \max\{0, a_l(x)\} dx$$

  - weights w^h establish recurrence, e.g. for processing sequences of data

Relations to Mathematical Objects

Monte Carlo methods and computer graphics

- structural equivalence of integral equations and reinforcement learning
- learning integro-approximation from noisy/sampled data

Relations to Mathematical Objects

Monte Carlo methods and computer graphics

- structural equivalence of integral equations and reinforcement learning
- learning integro-approximation from noisy/sampled data
- examples of random sampling
  - pseudo-random initialization
  - training by stochastic gradient descent
  - regularization by drop-out and drop-connect
  - random binarization
  - sampling by generative adversarial networks
  - fixed pseudo-random matrices for direct feedback alignment

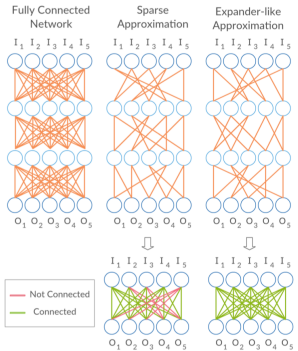
- ▶ Learning light transport the reinforced way
- ▶ Machine learning and integral equations
- ▶ Noise2Noise: Learning image restoration without clean data

Representing Artificial Neural Networks by Paths

Representing Artificial Neural Networks by Paths

Speeding up training and inference of neural networks by linear algorithms

- connections based on expander graphs

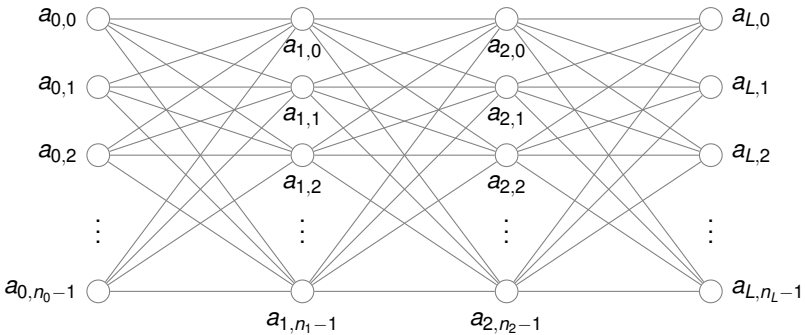


- Deep Expander Networks: Efficient Deep Networks from Graph Theory

Representing Artificial Neural Networks by Paths

Speeding up training and inference of neural networks by linear algorithms

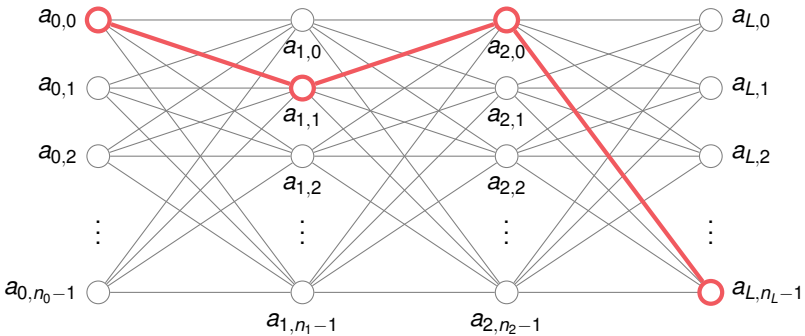
- guaranteed connectivity



Representing Artificial Neural Networks by Paths

Speeding up training and inference of neural networks by linear algorithms

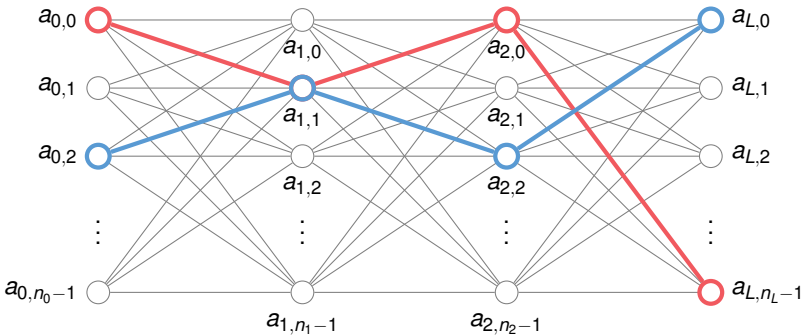
- guaranteed connectivity



Representing Artificial Neural Networks by Paths

Speeding up training and inference of neural networks by linear algorithms

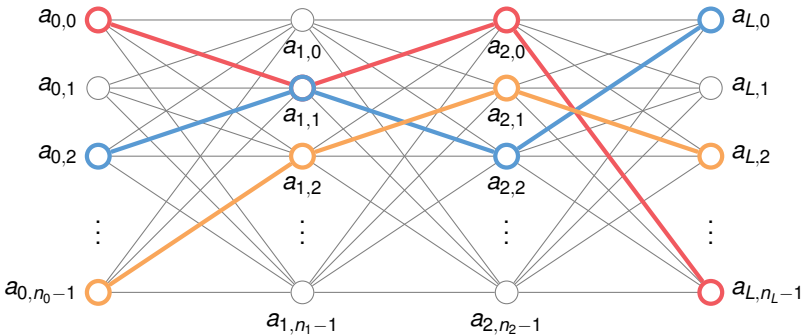
- guaranteed connectivity



Representing Artificial Neural Networks by Paths

Speeding up training and inference of neural networks by linear algorithms

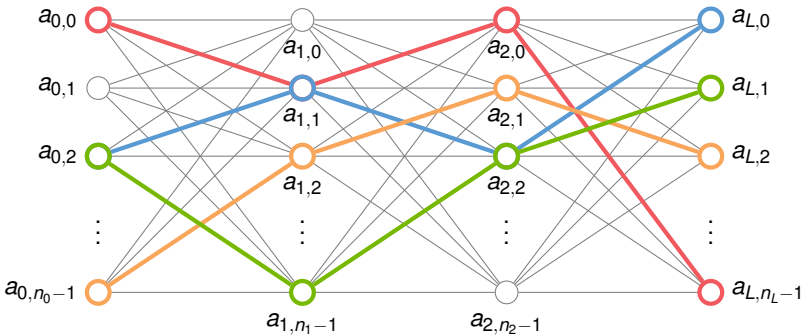
- guaranteed connectivity



Representing Artificial Neural Networks by Paths

Speeding up training and inference of neural networks by linear algorithms

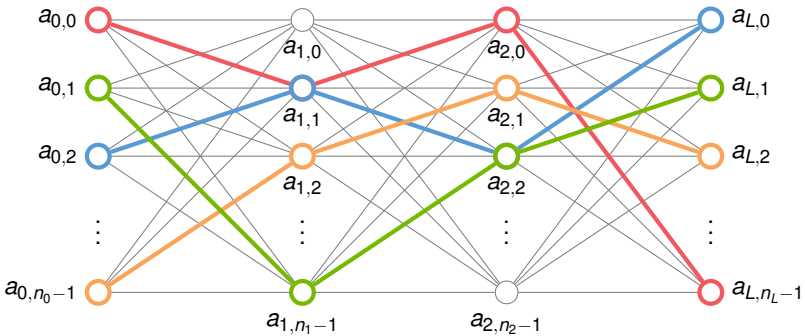
- guaranteed connectivity



Representing Artificial Neural Networks by Paths

Speeding up training and inference of neural networks by linear algorithms

- guaranteed connectivity and coverage



  - complexity linear number of paths and depth

Representing Artificial Neural Networks by Paths

Initialization: One weight per edge

```
LayerOffset[0] = NeuronsPerLayer[0];

for (int l = 1; l < Layers; ++l)
{
   for (int i = 0; i < Paths; ++i)
      Weight[l][i] = InitialWeight; // deterministic instead of random

  LayerOffset[l] = LayerOffset[l - 1] + NeuronsPerLayer[l];
}

a = new float[LayerOffset[Layers - 1]];
error = new float[LayerOffset[Layers - 1]];
```

Representing Artificial Neural Networks by Paths

Initialization: Generating paths

```
int Offset = 0;

for (int l = 0; l < Layers; ++l)
{
   for (int i = 0; i < Paths; ++i)
      Path[l][i] = Offset + (int) (drand48() * NeuronsPerLayer[l]);

   Offset += NeuronsPerLayer[l];
}
```

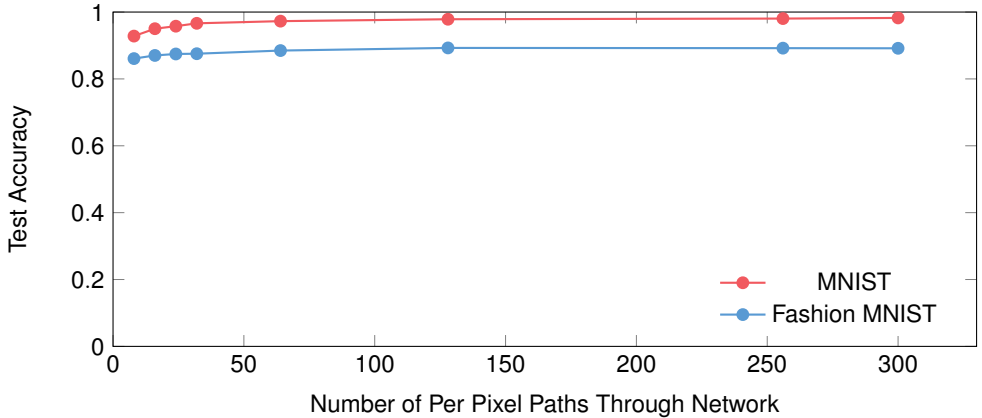
Representing Artificial Neural Networks by Paths

Inference: Sparse network, sequentially streamed weights

```
for (int i = 0; i < LayerOffset[0]; ++i)
   a[i] = input[i];

for (int i = LayerOffset[0]; i < LayerOffset[Layers - 1]; ++i)
   a[i] = 0.0f; // alternatively copy bias[i] values to a[i]

for (int l = 1; l < Layers; ++l)
   for (int i = 0; i < Paths; ++i)
     if(a[Path[l - 1][i]] > 0.0f) // implicit ReLU
        a[Path[l][i]] += Weight[l][i] * a[Path[l - 1][i]];
```

Representing Artificial Neural Networks by Paths

Results: 4 layer feedforward network (784/300/300/10) trained sparse from scratch

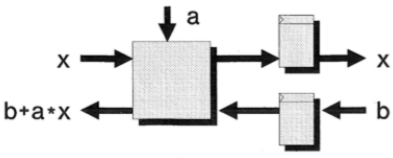


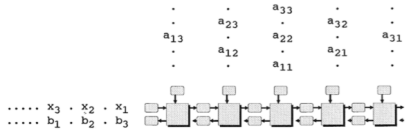
Parallel Computer Architecture

Parallel Computer Architecture

A blast from the past

- systolic arrays

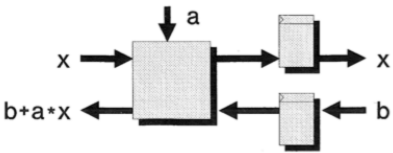


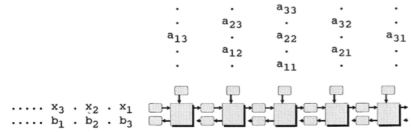


Parallel Computer Architecture

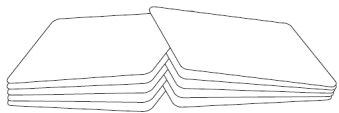
A blast from the past

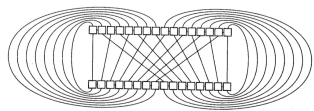
- systolic arrays





- perfect shuffle networks
  - matrix transposition, fast Fourier transform, evaluating polynomials, sorting, optical neural networks

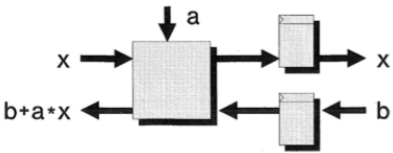


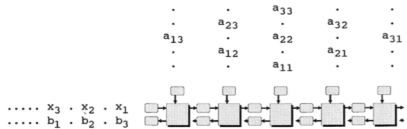


Parallel Computer Architecture

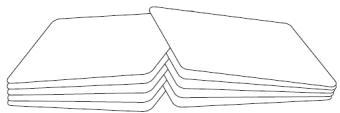
A blast from the past

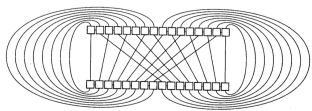
- systolic arrays in Google's tensor processing units (TPU)





- perfect shuffle networks
  - matrix transposition, fast Fourier transform, evaluating polynomials, sorting, optical neural networks

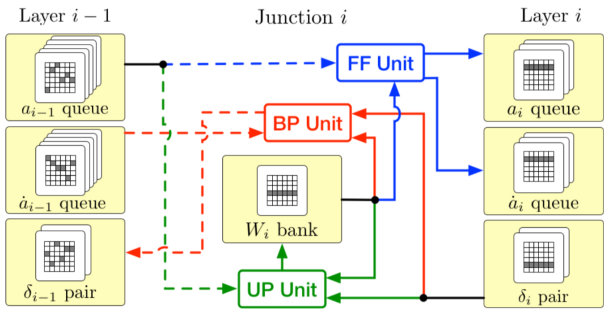




Parallel Computer Architecture

Permutations instead of perfect shuffle

- interleaver design



► Pre-Defined Sparse Neural Networks with Hardware Acceleration

Parallel Computer Architecture

Permutations instead of perfect shuffle

- interleaver design

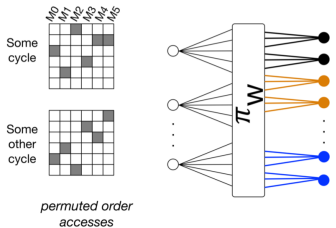
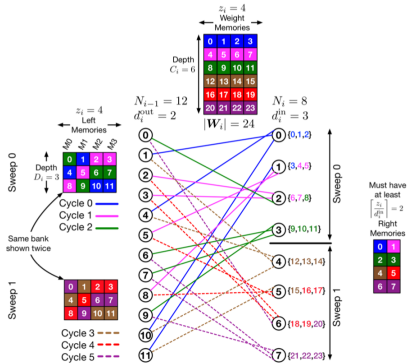


Fig. 2. Reading $z = 6$ weights corresponding to 2 right neurons in each cycle. When traced back through π_W, this requires reading 6 left activation memories in permuted order.

Parallel Computer Architecture

Permutations instead of perfect shuffle

- interleaver design



Parallel Computer Architecture

Permutations instead of perfect shuffle

- interleaver design

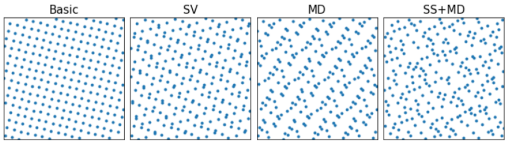


Fig. 4. Various π_W patterns using parameters $p = 64$, $fo = 4$ and $z = 16$. Interleaver size $= p \times fo = 256$.

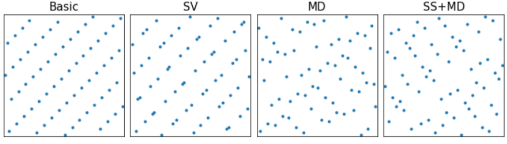


Fig. 5. Corresponding π_A patterns for Fig. 4. Interleaver size $= p = 64$.

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

$$x_i^{(l)} = \left(\frac{1}{2} \cdots \frac{1}{2^m}\right) \cdot \underbrace{C_l \cdot \begin{pmatrix} a_0(i) \\ \vdots \\ a_{m-1}(i) \end{pmatrix}}_{\text{multiplication in } F_2} \in [0,1)$$

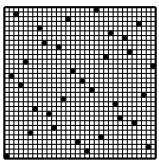
  - generator matrix C_l determined by l-th primitive polynomial

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



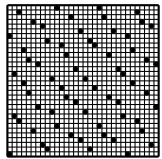
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



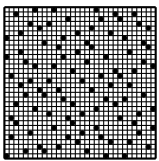
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



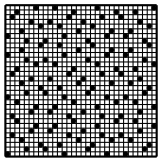
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



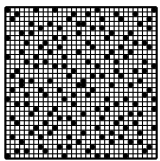
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



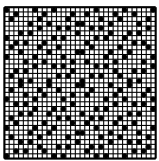
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



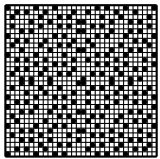
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



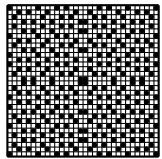
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



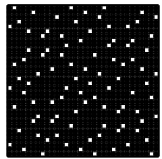
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



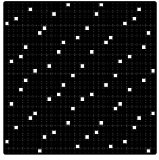
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



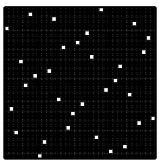
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



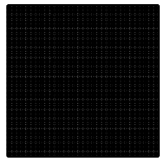
  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

```
uint x = 0;
for (uint k = 0; i; i >>= 1, ++k)
    if (i & 1)
        x ^= C[k]; // SIMD addition of column k of C_l
```



  - bit swizzling similar to mapping addresses to caches

Deterministic Permutations

Sobol' low discrepancy sequence

- each component is a $(0,1)$-sequence in base 2
  - contiguous blocks of length of powers of 2 are permutations

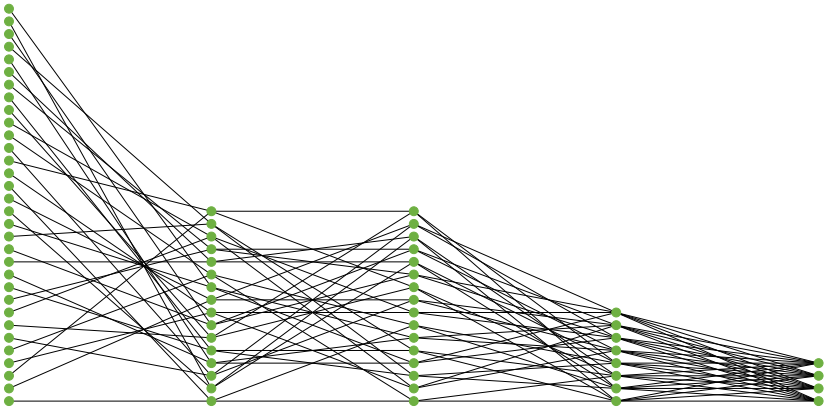
```
int Offset = 0;
for (int l = 0; l < Layers; ++l)
{
    for (int i = 0; i < Paths; ++i)
        Path[l][i] = Offset + (int) (Sobol(i, l) * NeuronsPerLayer[l]);

    Offset += NeuronsPerLayer[l];
}
```

  - `Sobol(i, l)` as hardware address generator

Examples of Sparse-from-Scratch Artificial Neural Networks

Deterministic connections by the Sobol' sequence

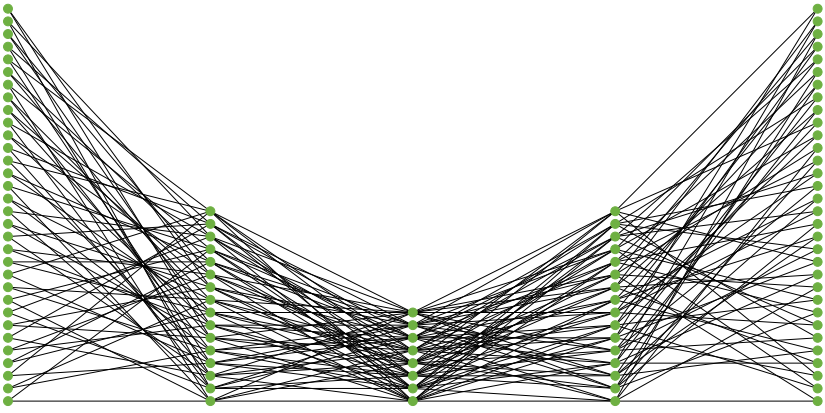
- sparse classifier



Examples of Sparse-from-Scratch Artificial Neural Networks

Deterministic connections by the Sobol' sequence

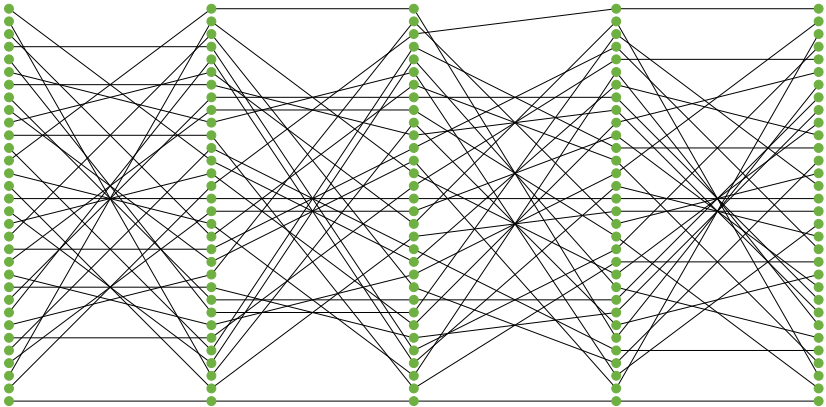
- sparse autoencoder



Examples of Sparse-from-Scratch Artificial Neural Networks

Deterministic connections by the Sobol' sequence

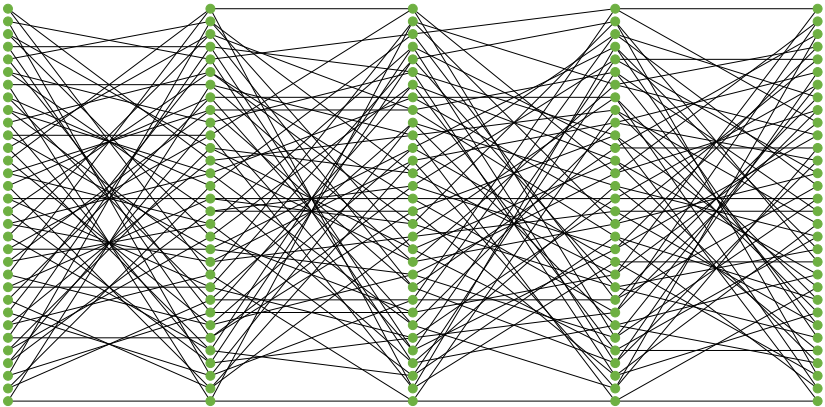
- sparse layers



Examples of Sparse-from-Scratch Artificial Neural Networks

Deterministic connections by the Sobol' sequence

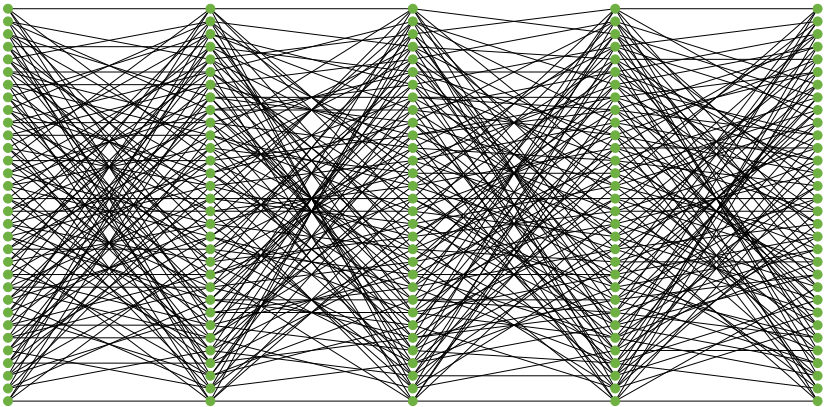
- sparse layers



Examples of Sparse-from-Scratch Artificial Neural Networks

Deterministic connections by the Sobol' sequence

- sparse layers



Structural Sparsity

Paths generated by the Sobol' sequence

- deterministic sparse connections by collision-free and progressive low discrepancy permutations
  - even as alternative to dropout and dropconnect
  - also works with weight sharing for convolutional layers
- deterministic initialization
- streaming weights

Structural Sparsity

Paths generated by the Sobol' sequence

- deterministic sparse connections by collision-free and progressive low discrepancy permutations
  - even as alternative to dropout and dropconnect
  - also works with weight sharing for convolutional layers
- deterministic initialization
- streaming weights
- linear complexity in both inference and backpropagation
  - lends itself to hardware architecture

Instant Quantization and Online Sparsity

Instant Quantization and Online Sparsity

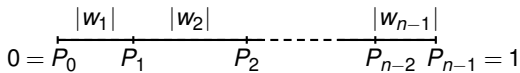
Ternarization by simulating discrete probability densities

- without loss of generality, assume $\sum_{k=0}^{n-1} |w_k| = \|w\|_1 = 1$
- define partition of unit interval by $P_m := \sum_{k=1}^{m} |w_k|$

Instant Quantization and Online Sparsity

Ternarization by simulating discrete probability densities

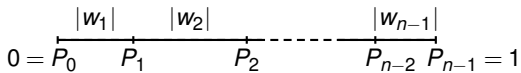
- without loss of generality, assume $\sum_{k=0}^{n-1} |w_k| = \|w\|_1 = 1$
- define partition of unit interval by $P_m := \sum_{k=1}^{m} |w_k|$



Instant Quantization and Online Sparsity

Ternarization by simulating discrete probability densities

- without loss of generality, assume $\sum_{k=0}^{n-1} |w_k| = \|w\|_1 = 1$
- define partition of unit interval by $P_m := \sum_{k=1}^{m} |w_k|$



- then, given N uniformly distributed samples $x_i \in [0,1)$

$$\sum_{k=0}^{n-1} w_k a_k \approx \frac{1}{N} \sum_{i=0}^{N-1} \underbrace{\operatorname{sign}(w_{j_i})}_{\in\{-1,0,1\}} \cdot a_{j_i},$$

where the index $j_i \in \{0,\dots,n-1\}$ is uniquely determined by $P_{j_i-1} \le x_i < P_{j_i}$

Instant Quantization and Online Sparsity

Network normalization

- given a positive factor $f \in \mathbb{R}^+$ and

$$\max\left\{0, \sum_{k=0}^{n_{l-1}-1} w_{l,k,i} a_{l-1,k} - b_{l,i}\right\} = f \cdot \max\left\{0, \frac{\sum_{k=0}^{n_{l-1}-1} w_{l,k,i} a_{l-1,k} - b_{l,i}}{f}\right\}$$

Instant Quantization and Online Sparsity

Network normalization

- given a positive factor $f \in \mathbb{R}^+$ and

$$\max\left\{0, \sum_{k=0}^{n_{l-1}-1} w_{l,k,i} a_{l-1,k} - b_{l,i}\right\} = f \cdot \max\left\{0, \frac{\sum_{k=0}^{n_{l-1}-1} w_{l,k,i} a_{l-1,k} - b_{l,i}}{f}\right\}$$

- neural units may be normalized by selecting the linear factor

$$f = \|w_{l,i}\|_1 = \sum_{k=0}^{n_{l-1}-1} |w_{l,k,i}|$$

Instant Quantization and Online Sparsity

Network normalization

- given a positive factor $f \in \mathbb{R}^+$ and

$$\max\left\{0, \sum_{k=0}^{n_{l-1}-1} w_{l,k,i} a_{l-1,k} - b_{l,i}\right\} = f \cdot \max\left\{0, \frac{\sum_{k=0}^{n_{l-1}-1} w_{l,k,i} a_{l-1,k} - b_{l,i}}{f}\right\}$$

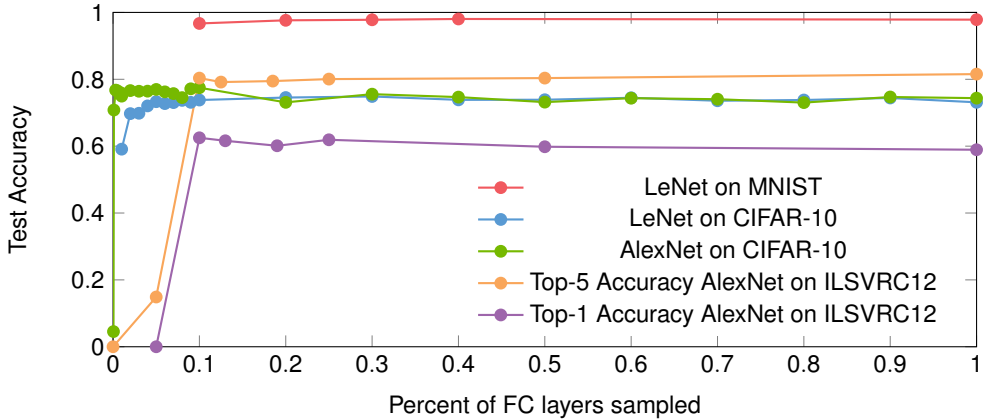
- neural units may be normalized by selecting the linear factor

$$f = \|w_{l,i}\|_1 = \sum_{k=0}^{n_{l-1}-1} |w_{l,k,i}|$$

- networks may be normalized
  - feed forward these factors layer by layer
  - store the resulting scaling factor for each output
  - possible, whenever positive linear factor property is available

Instant Quantization and Online Sparsity

Sampling paths backwards from the outputs by importance sampling the weights



Instant Quantization and Online Sparsity

Sampling paths backwards from the outputs by importance sampling the weights

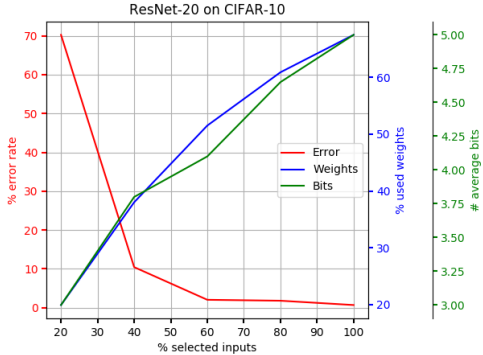
- quantizing the convolutional and dense weights without retraining

Error on CIFAR-10					
model	full res.	Δ TTQ (2-bit)	full res.	Δ ours	weights
ResNet-20	8.23%	0.64%	6.25%	0.67% (5.0 bits)	67.51%
ResNet-32	7.67%	-0.04%	5.80%	0.54% (4.85 bits)	68.24%
ResNet-44	7.18%	-0.16%	5.56%	0.52% (4.93 bits)	68.69%
ResNet-56	6.80%	-0.36%	5.57%	0.57% (4.78 bits)	68.92%

Instant Quantization and Online Sparsity

Sampling paths backwards from the outputs by importance sampling the weights

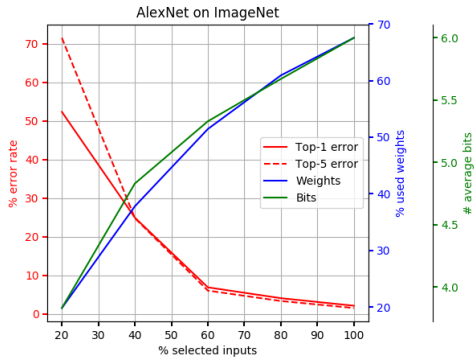
- average number of bits depending on number of samples



Instant Quantization and Online Sparsity

Sampling paths backwards from the outputs by importance sampling the weights

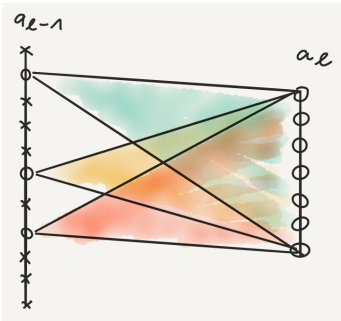
- average number of bits depending on number of samples



Instant Quantization and Online Sparsity

Sparse Inference on feature layers

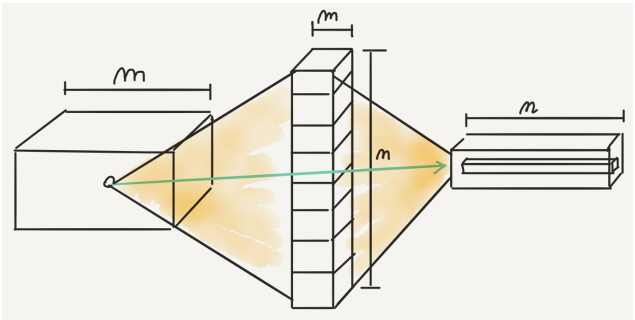
- sampling according to activations during inference
  - resembles a spiking network
  - resembles maximum pooling without loss of resolution



Instant Quantization and Online Sparsity

Sparse Inference on feature layers

- sampling according to activations during inference
  - resembles a spiking network
  - resembles maximum pooling without loss of resolution



Instant Quantization and Online Sparsity

Sparse Inference to address convolutional layers

- VGG16 with 2 dense layers on CIFAR-10 achieved 93.59% test accuracy
- subsampled with 20% activations in convolutional layers, and 5% activations in dense layers

sparse from	avg. used $a_{l,i}$ / layer	avg. $a_{l,i} \neq 0$ / layer	accuracy
1st Conv	46.18%	15.73%	28.93%
2nd Conv	48.02%	15.45%	90.53%
#F = 128	48.31%	15.41%	91.62%
#F = 256	47.06%	15.37%	92.55%
#F = 512	41.55%	15.89%	92.83%

Structural Sparsity

Speeding up training and inference of neural networks by linear algorithms

- representing neural networks by paths generated by low discrepancy sequences
  - alludes to hardware implementation
- instant quantization of trained networks
  - without retraining
- online sparsity
  - to subsample with convolutional layer

Structural Sparsity

Linear Algorithms for Recurrent Neural Networks (RNNs)

1. Lessons from Convolutional Neural Networks
2. Recurrent Neural Networks (RNNs)
3. Structurally Sparse RNNs
4. Experiments

Lessons from Convolutional Neural Networks (CNNs)

Structural sparsity

- maps to hardware much better than unstructured sparsity
- strong regularization by sparsity and weight sharing
- constraining the network structure can enhance learning

Lessons from Convolutional Neural Networks (CNNs)

Structural sparsity

- maps to hardware much better than unstructured sparsity
- strong regularization by sparsity and weight sharing
- constraining the network structure can enhance learning
- AlexNet architectural innovations: ► VGG ► Inception ► ResNet

Lessons from Convolutional Neural Networks (CNNs)

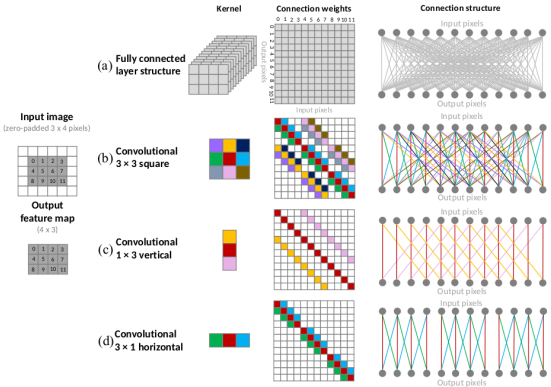
Structural sparsity

- maps to hardware much better than unstructured sparsity
- strong regularization by sparsity and weight sharing
- constraining the network structure can enhance learning
- AlexNet architectural innovations: ► VGG ► Inception ► ResNet
  - incremental updates, preserve 'state' through layers (► DenseNet)
  - feature compression with 1x1 convolutions ('bottleneck', ► SqueezeNet)
  - grouped convolutions (► ShuffleNet)
  - depthwise separable convolutions (► MobileNet)

Lessons from Convolutional Neural Networks (CNNs)

Structural sparsity

- convolutional layers are sparsely connected feedforward layers

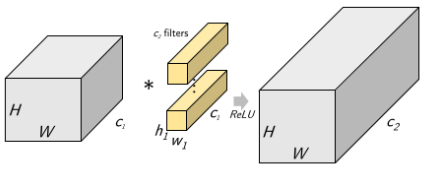


- Ioannau 2017, 'Structural priors in DNNs'

Lessons from Convolutional Neural Networks (CNNs)

Structural sparsity: Grouped convolutions

- split neurons in G independent groups

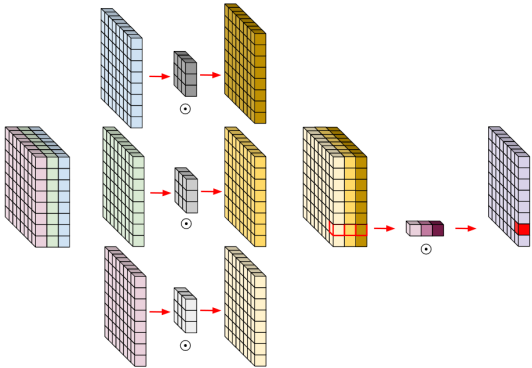


▸ Ioannau 2017, 'Structural priors in DNNs'

Lessons from Convolutional Neural Networks (CNNs)

Structural sparsity: (Depthwise) separable convolutions

- separate transformations over spatial and channel dimensions



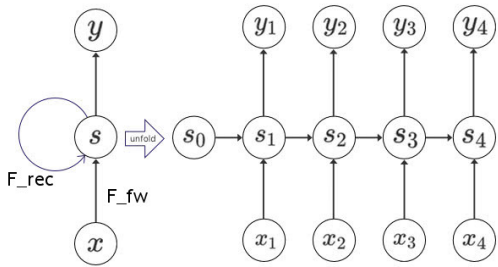
Bendersky

Recurrent Neural Networks (RNNs)

Principle

- process a sequence x_t of inputs, keeping a 'memory' or 'state' s_t

$$s_t = a[f_{fw}(x_t) + f_{rec}(s_{t-1})]$$



- the feedforward transformations f_{fw} can be sparsified as in CNN/FNN
- the recurrent transformation f_{rec} requires more care

Recurrent Neural Networks (RNNs)

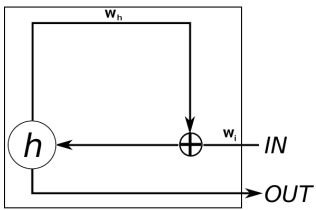
History

- unrolling (Minsky et al., 1969)
- backpropagation through time (Rumelhart et al., 1985)
- Time Delay NNs (Waibel,1989)
- FIR/IIR neurons (Tsoi and Back, 1994) (see ConvSeq2Seq networks ▶ WaveNet, 2016)
- Simple RNNs (Elman, 1990)
- controlling hidden state using gates: LSTM , (Hochreiter et al., 1997)

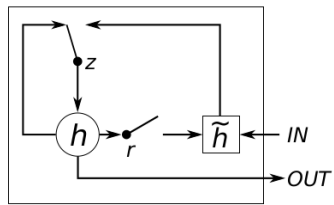
Recurrent Neural Networks (RNNs)

History: Gated RNNs

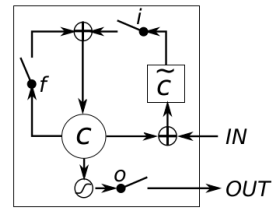
- simplifications
  - GRU (Cho et al. 2014)
  - QRNN (Bradbury et al. 2017) and ▶ SRU, (Lei et al., 2017)
  - li-GRU (Ravanelli et al., 2018)



RNN cell



GRU cell (Cho 2014)



LSTM cell (Hochreiter 1997)

Recurrent Neural Networks (RNNs)

Difficult to train: Overfitting and stability

- gradient clipping, gating, initialization, regularization
  - dropout for RNNs (Zaremba et al., 2014)
  - identity initialization for self-connections (Le et al., 2015)
  - Unitary RNNs (Arjovsky et al., 2015)
  - variety of regularization techniques (Merity et al.,2017)

Structurally Sparse RNNs

Overview

- sparsity $\Rightarrow$ lower cost, same capacity
- large sparse $>$ small dense networks. ▶ Kalchbrenner, 2018, ▶ Narang, 2017, ▶ Gray, 2017
- pruning
  - ▶ Han et al., 2015 (CNNs), ▶ ISS LSTM (Wen et al., 2017), ▶ Narang et al., 2017
  - complicate training, may require tuning
  - often orthogonal to architecture improvements ▶ SqueezeNet
- structured sparsity is much better suited for hardware acceleration

Structurally Sparse RNNs

Example

- focus on sparsifying recurrent weights: $s_t = a[f_{fw}(x_t) + f_{rec}(s_{t-1})]$
  - $f_{fw}(x_t)$ is timestep-independent $\Rightarrow$ parallellization possible
  - $f_{rec}(s_{t-1})$ is sequential!
- Example: Language Modeling
- DiagRNN: $> 10x$ speedup compared to full RNN

Model	Train time/batch	Infer time/batch	Model size
PyTorch, RNN	80 ms	16.5 ms	47.5 MB
PyTorch diagRNN	63 ms	13.7 ms	25.1 MB
PyTorch, cuDNN RNN	43 ms	8.2 ms	47.5 MB
PyTorch, CUDA diagRNN	23 ms	0.9 ms	25.1 MB

Structurally Sparse RNNs

Overview

- analogy
  - full CNN $\rightarrow$ grouped CNN $\rightarrow$ separable CNN
  - full RNN $\rightarrow$ grouped RNN $\rightarrow$ diagonal RNN
- examples
  - ▶ Diagonal LSTM (Subakan et al., 2017)
  - ▶ Block-sparse LSTMs (Gray et al., 2017)

  Grouped LSTMs: ▶ Kuchaiev et al. 2017, ▶ Beletti et al. 2018, ▶ Demeester et al. 2018

Structurally Sparse RNNs

An old idea

- diagonal RNN is just fully connected ANN with 1 additional weight per neuron
  - this weight is essential, as it enables comparison with past states, i.e. memory

Structurally Sparse RNNs

An old idea

- diagonal RNN is just fully connected ANN with 1 additional weight per neuron
  - this weight is essential, as it enables comparison with past states, i.e. memory

- simple RNNs (Elman, 1990)
- diagonal RNN (Ku et al., 1992)
- ring-structured RNNs (Young et al., 1993)
- block-diagonal RNN (Sivakumar et al., 1999)

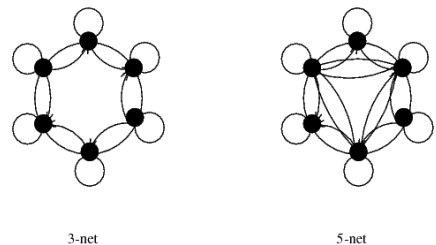


Figure 2: The Connection in the Hidden Layer of a 3-net and a 5-net with Six Hidden Units

Figure: Young et al., 1993

Structurally Sparse RNNs

Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons

- recurrent connections across two timesteps

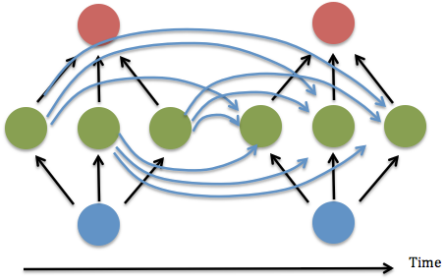


Figure: Full RNN

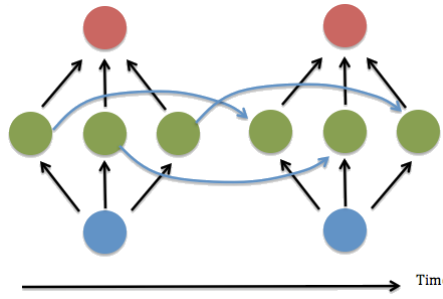


Figure: Diagonal RNN

▶ source

Structurally Sparse RNNs

Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons

- unstructured sparse RNN: challenging to accelerate
  - $N_h \cdot N_h \cdot (1 - p)$ recurrent weights
- grouped RNN: G independent neuron groups
  - $\frac{N_h \cdot N_h}{G}$ recurrent weights

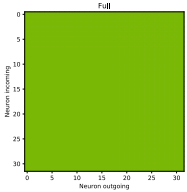
Structurally Sparse RNNs

Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons

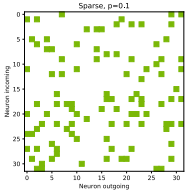
- unstructured sparse RNN: challenging to accelerate
  - $N_h \cdot N_h \cdot (1-p)$ recurrent weights
- grouped RNN: G independent neuron groups
  - $\frac{N_h \cdot N_h}{G}$ recurrent weights
- band diagonal RNN: connect only to $C-1$ neighboring neurons
  - $N_h \cdot C$ recurrent weights
- diagonal RNN: every neuron independent
  - N_h recurrent weights

Structurally Sparse RNNs

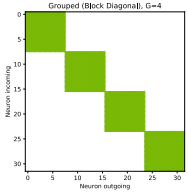
Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons



full RNN, N_h=32



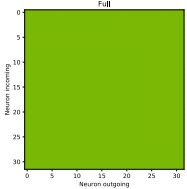
sparse RNN, $p = 0.1$



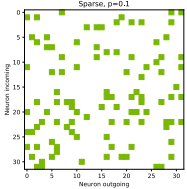
grouped RNN, $G = 4$

Structurally Sparse RNNs

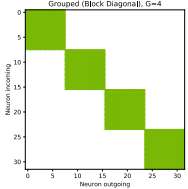
Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons



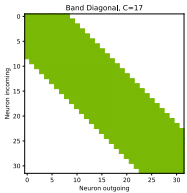
full RNN, N_h=32



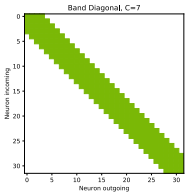
sparse RNN, $p = 0.1$



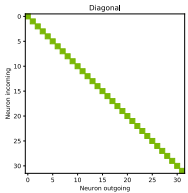
grouped RNN, $G = 4$



band RNN, $C = 17$



band RNN, $C = 7$

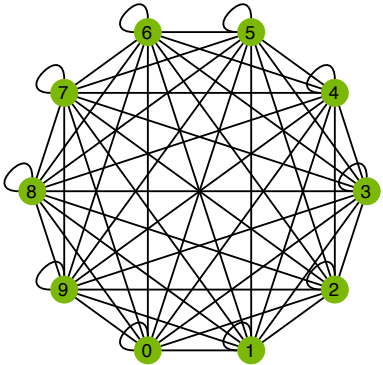


diag RNN, $C = 1$

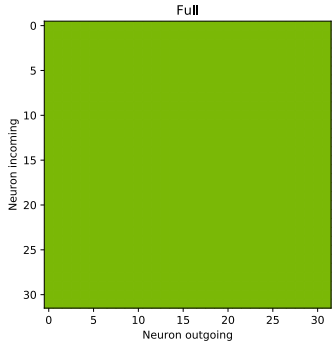
Structurally Sparse RNNs

Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons

- full RNN: connections to all other neurons including self-connection



connectivity

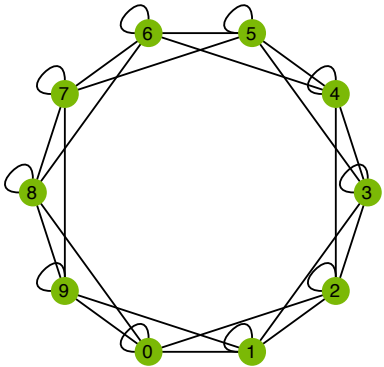


W_{hh}

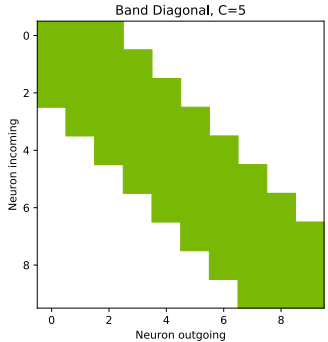
Structurally Sparse RNNs

Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons

- band RNN: connections to next 2 neurons on left and right plus self-connection $\Rightarrow C = 5$



connectivity

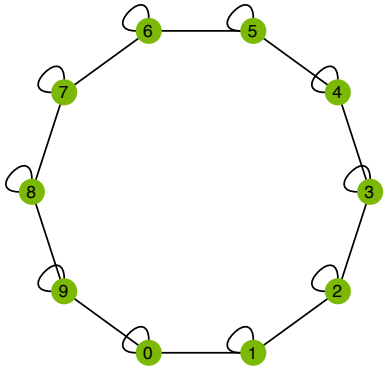


W_{hh}

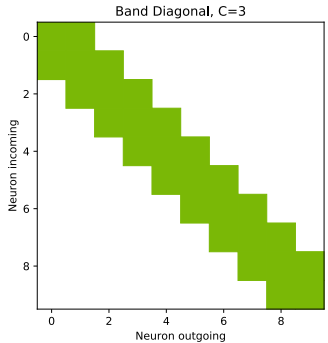
Structurally Sparse RNNs

Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons

- band RNN: connections next neuron on left and right plus self-connection $\Rightarrow C = 3$



connectivity

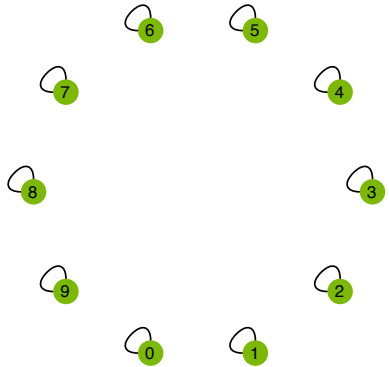


W_{hh}

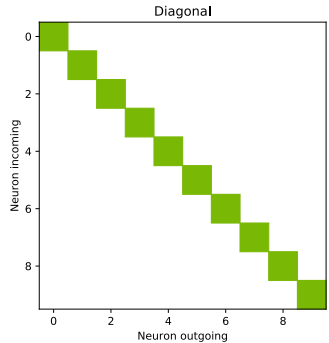
Structurally Sparse RNNs

Hidden-to-Hidden Weight Matrix W_{hh} for N_h hidden neurons

- diagonal RNN: only self-connections $\Rightarrow C = 1$



connectivity



W_{hh}

Structurally Sparse RNNs

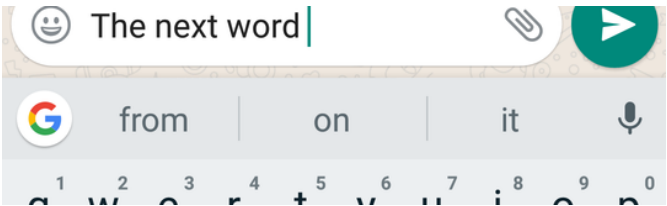
Parallelization Opportunity

- in diagonal RNNs, each neuron is independent $\Rightarrow$ we can parallelize!
  - Quasi-RNN (Bradbury et al., 2017)
  - Simple Recurrent Unit (Lei et al., 2017)
  - Independently RNN (Wang et al., 2018)
  - Parallelizing Linear RNNs with parallel scan algorithm (Martin and Cundy., 2018)
- up to 10x speedup and weight reduction as compared to full LSTM, at minimal cost

Results

PTB language modeling

- Language Modeling $\Rightarrow$ estimate word probability based on seen words
- $P(w_t|w_{t-k},...,w_{t-1})$



▶ source

Results

PTB language modeling

- input: word embedding (400 features)

Model $L = 1, N_h = 400$	Test perplexity (lower is better)	# RNN weights total and recurrent	
LSTM	85.66	1.28×10^6	6.40×10^5
Group LSTM, G=4	85.73	6.42×10^5	1.60×10^3
Diag LSTM, C=1	85.53	6.42×10^5	$\mathbf{1.60} \times 10^3$
Band LSTM, C=9	85.75	6.54×10^5	1.43×10^4

- structurally sparse networks match or improve full LSTM, at lower cost

Results

PTB language modeling

- larger, 3-layer networks

Model $L = 3, N_h = 1150$	Test perplexity (lower is better)	# RNN weights total and recurrent	
LSTM	65.09	2.02×10^7	1.12×10^7
Group LSTM, G=4	63.81	1.17×10^7	2.80×10^6
Diag LSTM, C=1	65.02	8.98×10^6	$\mathbf{1.08} \times 10^4$
Band LSTM, C=23	**63.70**	9.21×10^6	2.43×10^5

- structurally sparse networks match or improve full LSTM, at lower cost

Results

PTB language modeling

- increasing the width of the band

Model $L = 3, N_h = 1150$	Test perplexity (lower is better)	# RNN weights total and recurrent	
LSTM	65.09	2.02×10^7	1.12×10^7
Group LSTM, G=4	63.81	1.17×10^7	2.80×10^6
Diag LSTM, C=1	65.02	8.98×10^6	$\mathbf{1.08} \times 10^4$
Band LSTM, C=23	**63.70**	9.21×10^6	2.43×10^5
Band LSTM, C=105	63.85	1.01×10^7	1.11×10^6
Band LSTM, C=460	64.06	1.30×10^7	4.05×10^6

Results

Phoneme classification/Speech Recognition

- task: classify audio frames

Results

Phoneme classification on TIMIT

- input: audio data sequence (39 MFCC features/ 20ms)

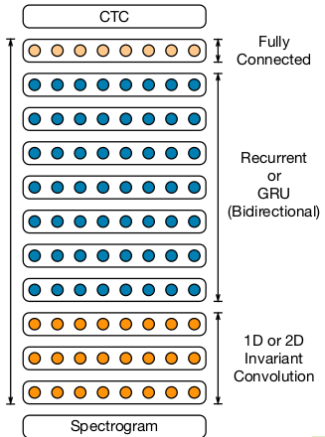
Model $L = 1, N_h = 512$	Test accuracy (higher is better)	# RNN weights total and recurrent	
Full LSTM, 1x512	78.3	1.13×10^6	1.05×10^6
Diag LSTM, 1x512	77.6	8.19×10^4	2.05×10^3
Band LSTM, 1x512, C=103	**78.8**	2.81×10^5	2.11×10^5

- DiagLSTM is 0.7% worse than full LSTM, with 500x less recurrent weights
- BandLSTM is 0.5% better than full LSTM, with 5x less recurrent weights

Results

Speech recognition on VCTK

- task: audio to text
- Deep Speech 2 architecture (▶ Amodei 2015, 'Deep Speech 2')
  - input: audio spectrogram [time, frequency]
  - 2 convolutional layers
  - N_{rec} recurrent layers (with batch normalization)
  - CTC loss function
- no beam search/language model etc.



Results

Speech recognition on VCTK

Model	Test CER (lower is better)	# RNN weights total and recurrent	
Full GRU 5x768	8.57	3.82×10^7	1.77×10^7
Diag GRU 5x768, C=1	9.69	2.06×10^7	4.61×10^4
Band GRU 5x768, C=103	9.87	2.39×10^7	3.41×10^6
Diag GRU 5x1100, C=1	**8.21**	3.81×10^7	$\mathbf{6.60}\times10^4$
Band GRU 5x1000, C=201	**8.08**	3.80×10^7	5.76×10^6

- DiagGRU is 1.12% worse than full GRU, but has $\approx 400x$ less recurrent weights
- larger sparse networks perform better and can be accelerated easily

Summary and Takeaway

- structured sparsity works in RNNs as well as CNNs
- structured sparsity is powerful
  - better performance/cost trade-off
  - straightforward mapping to hardware
  - accelerate training and inference
- diagonal RNNs well suited for acceleration
  - # recurrent weights is linear in # neurons
  - sequential part: only elementwise multiplications
  - parallelize over neurons

PreRNN and BandRNN for Video Understanding

Xiaodong Yang

with Pavlo Molchanov, Matthijs Van Keirsbilck, Alex Keller, Jan Kautz

Sequential Learning Problems

machine translation

handwriting modeling

polyphonic music modeling

speech recognition

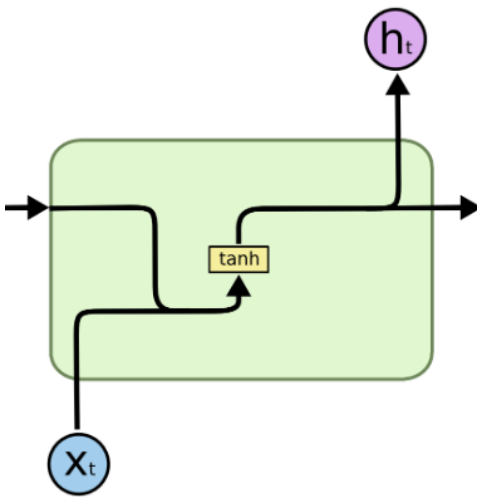
intelligent video analytics

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

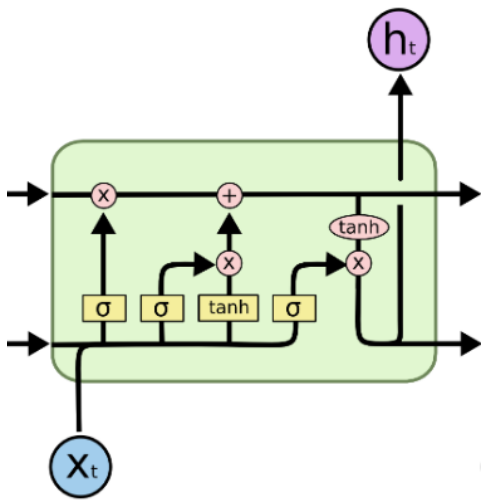
Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

language modeling

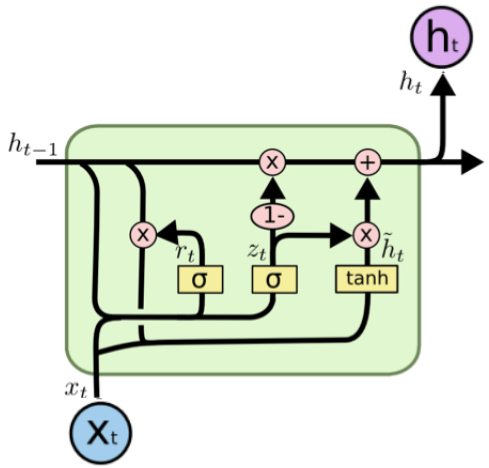
RNNs in Sequential Learning



Vanilla RNN (VRNN)



Long Short-Term Memory (LSTM)

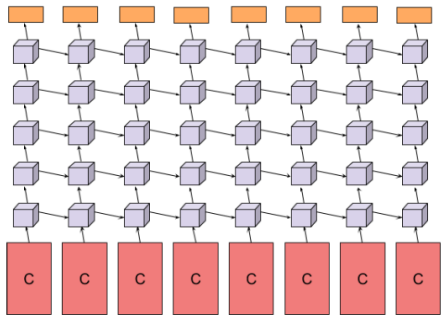


Gated Recurrent Unit (GRU)

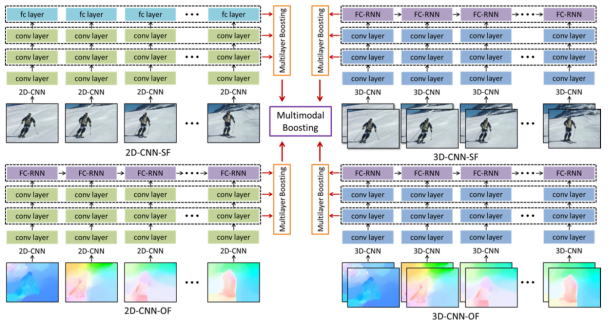
Image courtesy of Christopher Olah.

RNNs in Video Understanding

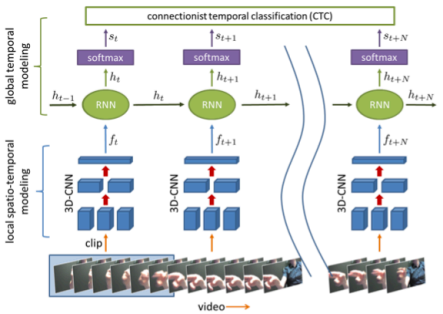
Examples



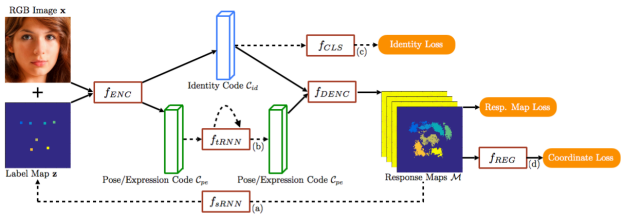
Ng et al. CVPR 2015



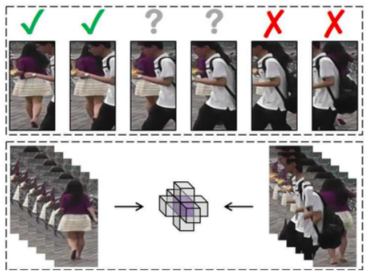
Yang et al. ACMMM 2016



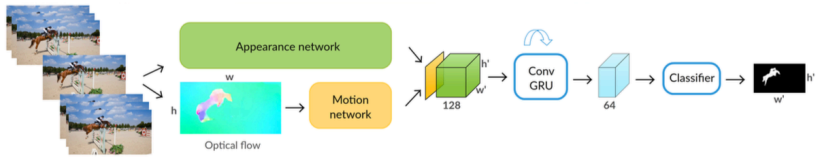
Molchanov et al. CVPR 2016



Peng et al. ECCV 2016



Zhou et al. CVPR 2017



Tokmakov et al. ICCV 2017

RNNs in Video Understanding

Distinct Properties of Videos

- Processing unit in a more structured format such as image or snippet
  - CNNs serve as backbone networks
  - Pre-trained on large-scale image or video datasets
  - How to construct RNNs to better leverage the pre-trained CNNs
- Large redundancy and diverse temporal dependencies on different applications
  - Such as facial alignment, hand gesture recognition, activity recognition
  - Poorly understood which recurrent structure or which gating mechanism best suits

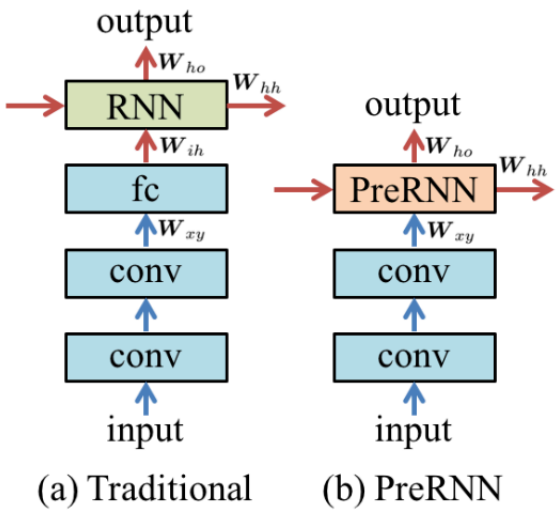
PreRNN+BandRNN for Video Understanding

Overview

- PreRNN: make pre-trained CNNs recurrent by transforming pre-trained convolutional or fully connected layers into recurrent layers
- PreRNN-SIH: simplify input-to-hidden states and reduce recurrent parameters
- BandRNN: sparsify hidden-to-hidden weights and further reduce recurrent parameters

PreRNN for Video Understanding

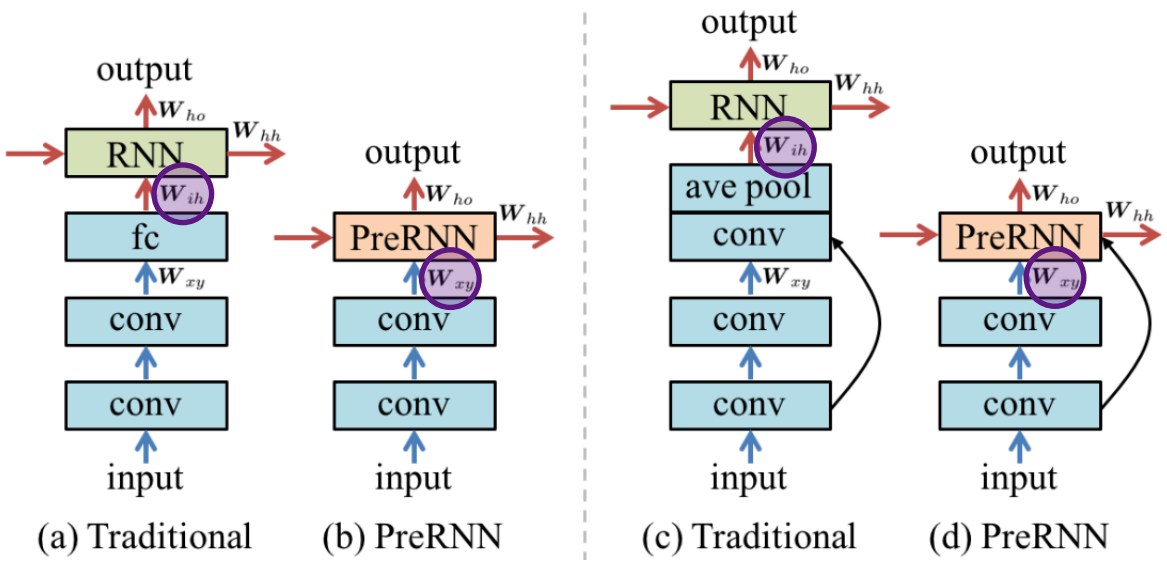
Overview



A schematic overview of the traditional RNN and the proposed PreRNN.

PreRNN for Video Understanding

Overview



A schematic overview of the traditional RNN and the proposed PreRNN.

Traditional RNNs

Notation

- VRNN

$$\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})$$

- LSTM

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{W}_{ii}\boldsymbol{y}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{W}_{if}\boldsymbol{y}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{W}_{io}\boldsymbol{y}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \qquad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_{ic}\boldsymbol{y}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \qquad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

- GRU

$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{W}_{ir}\boldsymbol{y}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{W}_{iz}\boldsymbol{y}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})) \qquad \boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t$$

$\mathcal{H}$ activation function

Traditional RNNs

Input-to-Hidden State

- VRNN

$$\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})$$

- LSTM

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{W}_{ii}\boldsymbol{y}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{W}_{if}\boldsymbol{y}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{W}_{io}\boldsymbol{y}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \qquad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_{ic}\boldsymbol{y}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \qquad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

- GRU

$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{W}_{ir}\boldsymbol{y}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{W}_{iz}\boldsymbol{y}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})) \qquad \boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t$$

$\mathcal{H}$ activation function

Traditional RNNs

Hidden-to-Hidden State

- VRNN

$$\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})$$

- LSTM

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{W}_{ii}\boldsymbol{y}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{W}_{if}\boldsymbol{y}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{W}_{io}\boldsymbol{y}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \qquad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_{ic}\boldsymbol{y}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \qquad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

- GRU

$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{W}_{ir}\boldsymbol{y}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{W}_{iz}\boldsymbol{y}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})) \qquad \boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t$$

$\mathcal{H}$ activation function

PreRNN

Transformation for VRNN

- A feedforward layer in CNNs

$$\boldsymbol{y} = \mathcal{H}(\boldsymbol{W}_{xy} \circ \boldsymbol{x})$$

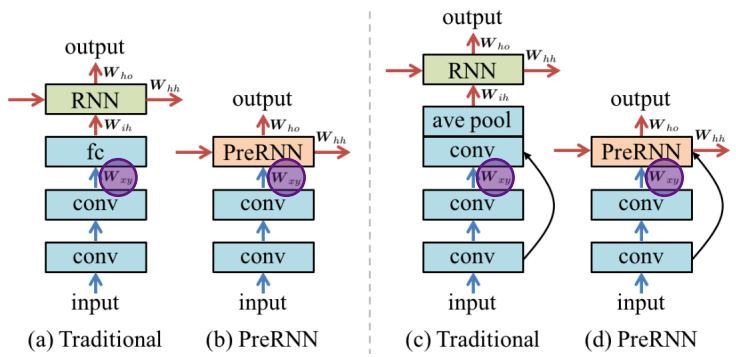
- PreVRNN

$$\boldsymbol{y}_t = \begin{cases} \mathcal{H}(\boldsymbol{W}_{xy}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{y}_{t-1}) & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{H}(\mathcal{P}(\mathcal{B}(\boldsymbol{W}_{xy} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) + \boldsymbol{W}_{hh}\boldsymbol{y}_{t-1}) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

$\mathcal{H}$ activation function

$\mathcal{B}$ batch normalization

$\mathcal{P}$ pooling



A schematic overview of the traditional RNN and the proposed PreRNN.

PreRNN

Transformation for LSTM

- A feedforward layer in CNNs

$$\boldsymbol{y} = \mathcal{H}(\boldsymbol{W}_{xy} \circ \boldsymbol{x})$$

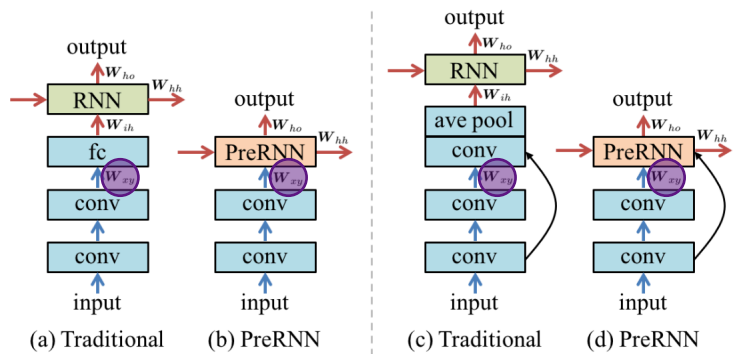
- Gate-dependent input-to-hidden state

$$\boldsymbol{u}_t(g) = \begin{cases} \boldsymbol{W}_{ig}^{p} \boldsymbol{x}_t & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{P}(\mathcal{B}(\boldsymbol{W}_{ig}^{p} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

- PreLSTM

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{u}_t(i) + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{u}_t(f) + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$

$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{u}_t(o) + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \qquad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{u}_t(c) + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$



A schematic overview of the traditional RNN and the proposed PreRNN.

$\mathcal{H}$ activation function

$\mathcal{B}$ batch normalization

$\mathcal{P}$ pooling

PreRNN

Transformation for GRU

- A feedforward layer in CNNs

$$\boldsymbol{y} = \mathcal{H}(\boldsymbol{W}_{xy} \circ \boldsymbol{x})$$

- Gate-dependent input-to-hidden state

$$\boldsymbol{u}_t(g) = \begin{cases} \boldsymbol{W}_{ig}^{p} \boldsymbol{x}_t & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{P}(\mathcal{B}(\boldsymbol{W}_{ig}^{p} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

- PreGRU

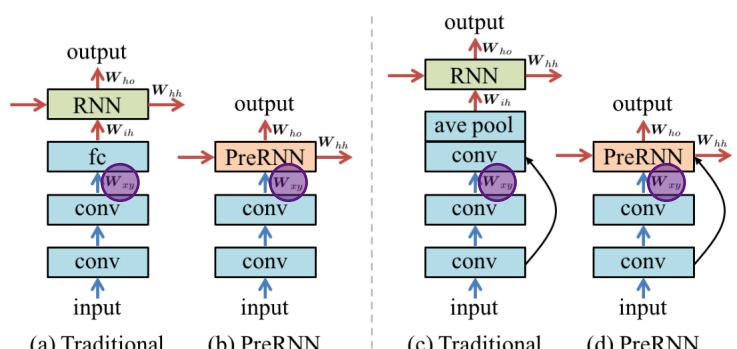
$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{u}_t(r) + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{u}_t(z) + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{u}_t(h) + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}))$$

$\mathcal{H}$ activation function

$\mathcal{B}$ batch normalization

$\mathcal{P}$ pooling



A schematic overview of the traditional RNN and the proposed PreRNN.

PreRNN

Comparison to Traditional RNN

- VRNN => PreVRNN

$$\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})$$

$$\boldsymbol{y}_t = \begin{cases} \mathcal{H}(\boldsymbol{W}_{xy}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{y}_{t-1}) & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{H}(\mathcal{P}(\mathcal{B}(\boldsymbol{W}_{xy} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) + \boldsymbol{W}_{hh}\boldsymbol{y}_{t-1}) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

- LSTM => PreLSTM

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{W}_{ii}\boldsymbol{y}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \quad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{W}_{if}\boldsymbol{y}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{W}_{io}\boldsymbol{y}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \quad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_{ic}\boldsymbol{y}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{u}_t(i) + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \quad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{u}_t(f) + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{u}_t(o) + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \quad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{u}_t(c) + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$

- GRU => PreGRU

$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{W}_{ir}\boldsymbol{y}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \quad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{W}_{iz}\boldsymbol{y}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{ih}\boldsymbol{y}_t + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}))$$

$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{u}_t(r) + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \quad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{u}_t(z) + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{u}_t(h) + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}))$$

$\mathcal{H}$ activation function
$\mathcal{B}$ batch normalization
$\mathcal{P}$ pooling

PreRNN-SIH

Transformation for LSTM

- A feedforward layer in CNNs

$$\boldsymbol{y} = \mathcal{H}(\boldsymbol{W}_{xy} \circ \boldsymbol{x})$$

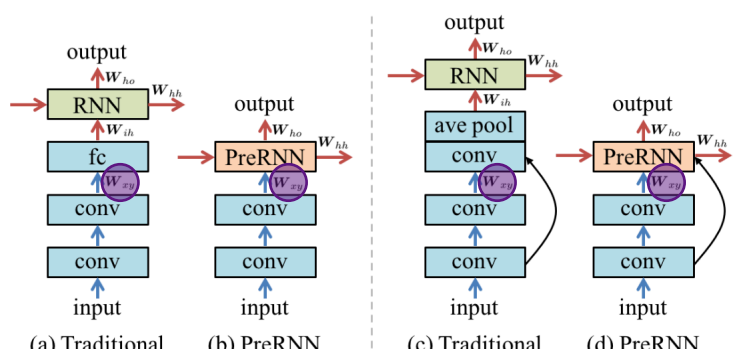
- Single input-to-hidden (SIH) state

$$\boldsymbol{v}_t = \begin{cases} \boldsymbol{W}_{xy}\boldsymbol{x}_t & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{P}(\mathcal{B}(\boldsymbol{W}_{xy} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

- PreLSTM-SIH

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$

$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \qquad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{v}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$



A schematic overview of the traditional RNN and the proposed PreRNN.

$\mathcal{H}$ activation function

$\mathcal{B}$ batch normalization

$\mathcal{P}$ pooling

PreRNN-SIH

Transformation for GRU

- A feedforward layer in CNNs

$$\boldsymbol{y} = \mathcal{H}(\boldsymbol{W}_{xy} \circ \boldsymbol{x})$$

- Single input-to-hidden (SIH) state

$$\boldsymbol{v}_t = \begin{cases} \boldsymbol{W}_{xy}\boldsymbol{x}_t & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{P}(\mathcal{B}(\boldsymbol{W}_{xy} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

- PreGRU-SIH

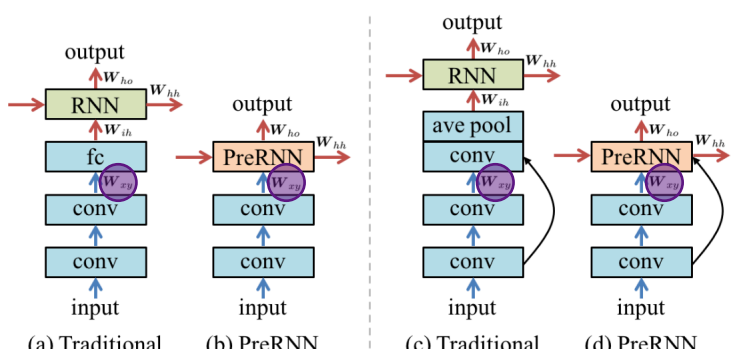
$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \qquad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{v}_t + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}))$$

$\mathcal{H}$ activation function

$\mathcal{B}$ batch normalization

$\mathcal{P}$ pooling



A schematic overview of the traditional RNN and the proposed PreRNN.

PreRNN-SIH

Comparison to PreRNN

- (Gate-dependent => Single) input-to-hidden state

$$\boldsymbol{u}_t(g) = \begin{cases} \boldsymbol{W}_{ig}^{p}\boldsymbol{x}_t & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{P}(\mathcal{B}(\boldsymbol{W}_{ig}^{p} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

$$\boldsymbol{v}_t = \begin{cases} \boldsymbol{W}_{xy}\boldsymbol{x}_t & \text{a } \texttt{fc} \text{ layer} \\ \mathcal{P}(\mathcal{B}(\boldsymbol{W}_{xy} * \boldsymbol{x}_t) + \boldsymbol{\gamma}_t) & \text{a } \texttt{conv} \text{ layer} \end{cases}$$

- PreLSTM => PreLSTM-SIH

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{u}_t(i) + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \quad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{u}_t(f) + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{u}_t(o) + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \quad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{u}_t(c) + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$

$$\boldsymbol{i}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}) \quad \boldsymbol{f}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{o}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}) \quad \tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{v}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1})$$

- PreGRU => PreGRU-SIH

$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{u}_t(r) + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \quad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{u}_t(z) + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{u}_t(h) + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}))$$

$$\boldsymbol{r}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hr}\boldsymbol{h}_{t-1}) \quad \boldsymbol{z}_t = \text{sigm}(\boldsymbol{v}_t + \boldsymbol{W}_{hz}\boldsymbol{h}_{t-1})$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{v}_t + \boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}))$$

$\mathcal{H}$ activation function
$\mathcal{B}$ batch normalization
$\mathcal{P}$ pooling

Applications

Diversity

Applications	Sequences	CNNs	Datasets	Objectives
Sequential Face Alignment	Color	VGG16 [38]	300VW [7]	ℓ_2
Hand Gesture Recognition	Color & Depth	C3D [43]	NVGesture [28]	CTC [15]
Action Recognition	Color & Flow	ResNet50 [20]	UCF101 [39]	NLL

Summary of the diverse experiments in terms of applications, video types, pre-trained backbone CNNs, benchmark datasets, and objective functions.

Applications

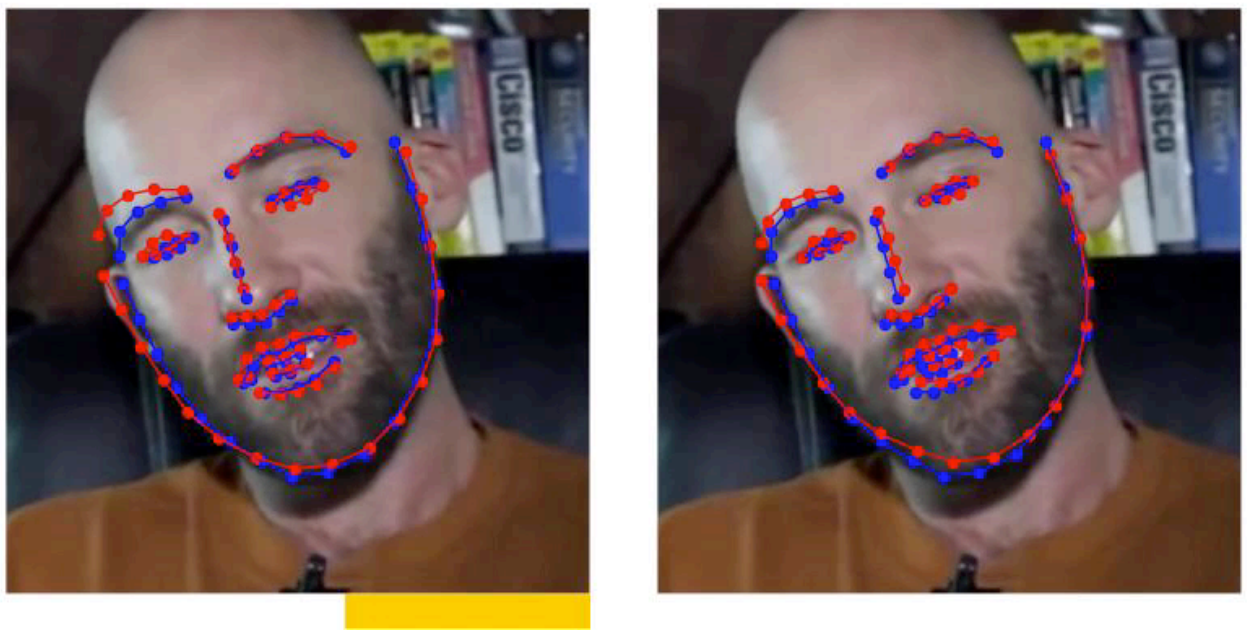
Face Alignment

Applications	Sequences	CNNs	Datasets	Objectives
Sequential Face Alignment	Color	VGG16 [38]	300VW [7]	ℓ_2
Hand Gesture Recognition	Color & Depth	C3D [43]	NVGesture [28]	CTC [15]
Action Recognition	Color & Flow	ResNet50 [20]	UCF101 [39]	NLL

Summary of the diverse experiments in terms of applications, video types, pre-trained backbone CNNs, benchmark datasets, and objective functions.

Applications

Face Alignment



Examples of detected facial landmarks on the 300VW dataset by traditional GRU (left) and PreGRU (right).

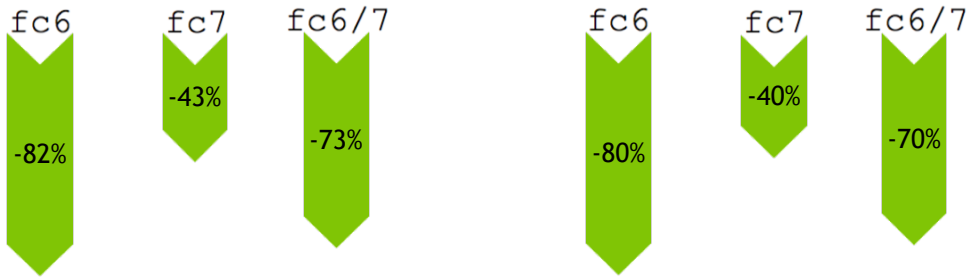
Blue dots: ground truth
Red dots: detected landmarks
Green bar: PreGRU with larger error
Yellow bar: traditional GRU with larger error
Bar length: error scale

Applications

Face Alignment

	Traditional		PreRNN			PreRNN-SIH		
	1 layer	2 layers	fc6	fc7	fc6/7	fc6	fc7	fc6/7
VRNN	0.704	0.716	0.757	0.742	**0.763**	-	-	-
LSTM	0.718	0.671	**0.769**	0.754	0.746	0.743	0.746	0.719
GRU	0.722	0.698	**0.772**	0.755	0.761	0.768	0.748	0.762

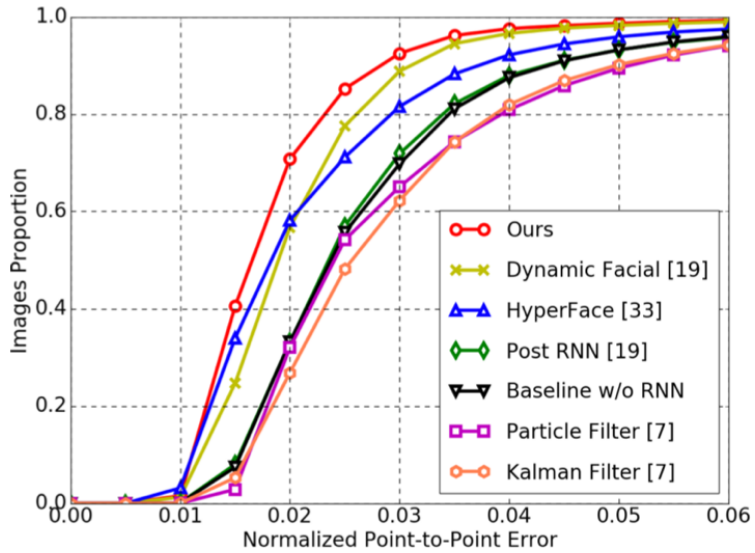
AUC of traditional RNNs and our proposed PreRNN(-SIH) on 300VW.



PreLSTM-SIH vs. LSTM

PreGRU-SIH vs. GRU

Ratios of reduced recurrent parameters by PreRNN-SIH.



Comparison of our approach with the state-of-the-art methods.

Applications

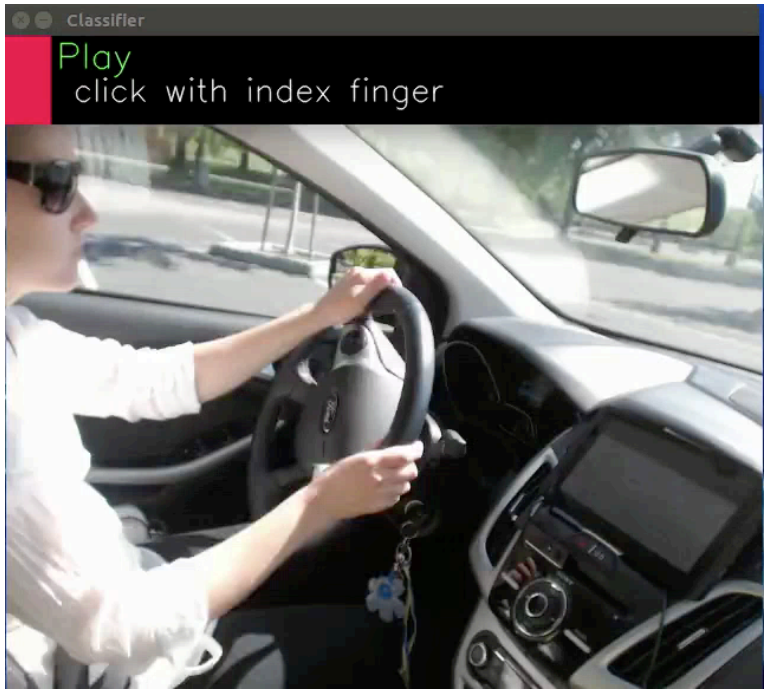
Hand Gesture Recognition

Applications	Sequences	CNNs	Datasets	Objectives
Sequential Face Alignment	Color	VGG16 [38]	300VW [7]	ℓ_2
Hand Gesture Recognition	Color & Depth	C3D [43]	NVGesture [28]	CTC [15]
Action Recognition	Color & Flow	ResNet50 [20]	UCF101 [39]	NLL

Summary of the diverse experiments in terms of applications, video types, pre-trained backbone CNNs, benchmark datasets, and objective functions.

Applications

Hand Gesture Recognition



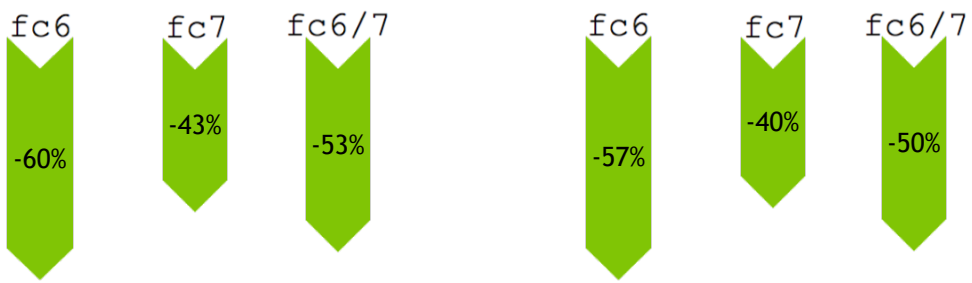
PreVRNN based hand gesture recognition system for in-car media player control.

Applications

Hand Gesture Recognition

	Traditional		PreRNN			PreRNN-SIH		
	1 layer	2 layers	fc6	fc7	fc6/7	fc6	fc7	fc6/7
VRNN	83.3%	80.8%	81.9%	82.9%	**84.4%**	-	-	-
LSTM	81.3%	81.3%	81.7%	81.9%	82.7%	80.0%	81.7%	**84.2%**
GRU	81.9%	82.5%	82.1%	81.0%	83.1%	**84.4%**	79.8%	83.8%

Classification accuracy of traditional RNNs and our proposed PreRNN(-SIH) on NVGesture.



PreLSTM-SIH vs. LSTM

PreGRU-SIH vs. GRU

Ratios of reduced recurrent parameters by PreRNN-SIH.

Method	Modality	Accuracy
C3D [43]	Color	69.3%
R3DCNN [28]	Color	74.1%
Ours	Color	**76.5%**
SNV [49]	Depth	70.7%
C3D [43]	Depth	78.8%
R3DCNN [28]	Depth	80.3%
Ours	Depth	**84.4%**
Two-Stream [37]	Color + Flow	65.6%
iDT [45]	Color + Flow	73.4%
R3DCNN [28]	Five Modalities	83.8%
Baseline (w/o RNN)	Color + Depth	81.0%
Ours	Color + Depth	**85.0%**

Comparison of our approach with the state-of-the-art methods.

Applications

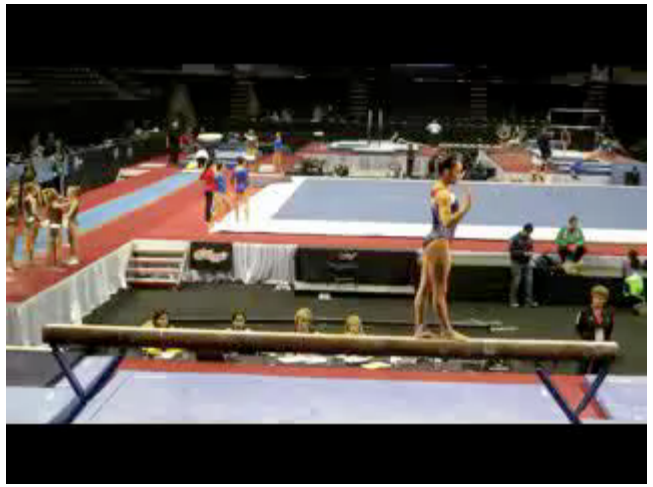
Action Recognition

Applications	Sequences	CNNs	Datasets	Objectives
Sequential Face Alignment	Color	VGG16 [38]	300VW [7]	ℓ_2
Hand Gesture Recognition	Color & Depth	C3D [43]	NVGesture [28]	CTC [15]
Action Recognition	Color & Flow	ResNet50 [20]	UCF101 [39]	NLL

Summary of the diverse experiments in terms of applications, video types, pre-trained backbone CNNs, benchmark datasets, and objective functions.

Applications

Action Recognition



PreGRU => balance beam
Traditional GRU => floor gymnastics

PreGRU => skijet
Traditional GRU => kayaking

Examples of misclassified videos by traditional GRU, but corrected by PreGRU.

Applications

Action Recognition

	Traditional			PreRNN			PreRNN-SIH		
	Color	Flow	Comb	Color	Flow	Comb	Color	Flow	Comb
VRNN	82.9%	83.6%	91.6%	83.8%	84.6%	**92.7%**	-	-	-
LSTM	83.4%	84.0%	92.5%	85.3%	84.8%	93.2%	85.0%	84.6%	**93.5%**
GRU	83.6%	83.8%	92.2%	84.3%	85.2%	**93.7%**	84.9%	84.7%	93.3%

Classification accuracy of traditional RNNs and our proposed PreRNN(-SIH) on UCF101.

PreLSTM-SIH vs. LSTM

PreGRU-SIH vs. GRU

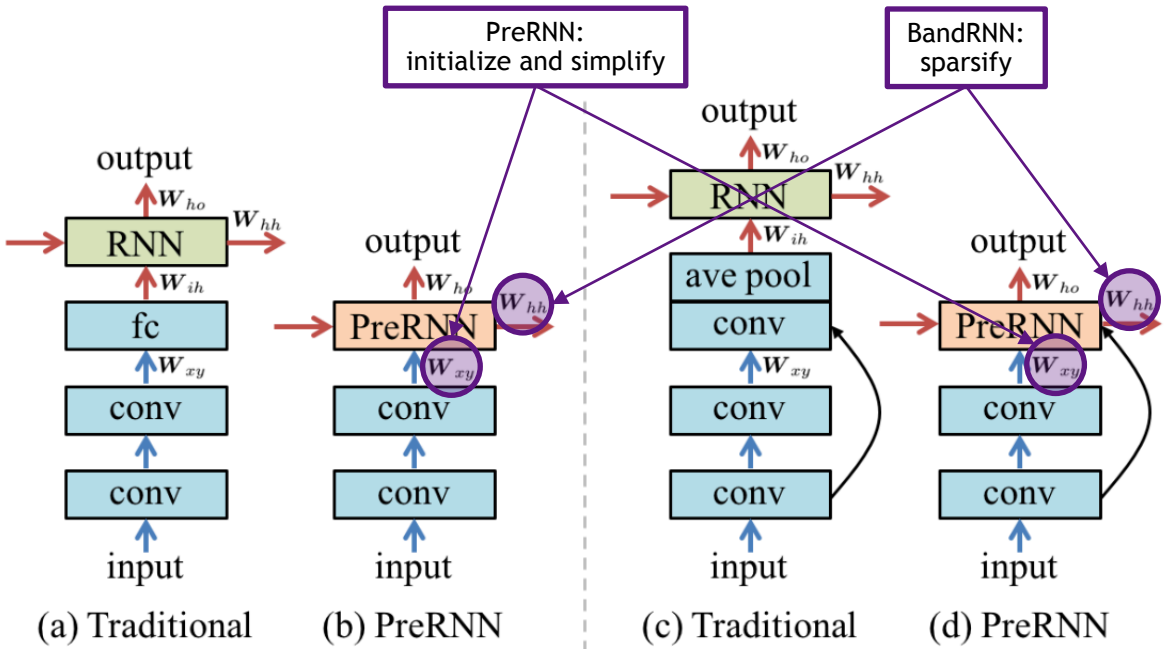
Ratios of reduced recurrent parameters by PreRNN-SIH.

Method	Accuracy
Dynamic Image Nets [3]	76.9%
Long-Term Recurrent ConvNet [10]	82.9%
Composite LSTM Model [40]	84.3%
C3D [43]	85.2%
iDT [45]	86.4%
Two-Stream ConvNet [37]	88.0%
Multilayer Multimodal Fusion [48]	91.6%
Long-Term ConvNets [44]	91.7%
Two-Stream Fusion [14]	92.5%
Spatiotemporal ResNets [12]	93.4%
Inflated 3D ConvNets [4]	93.4%
Temporal Segment Networks [46]	**94.2%**
Spatiotemporal Multiplier Nets [13]	**94.2%**
Baseline (w/o RNN)	91.7%
Ours	**94.3%**

Comparison of our approach with the state-of-the-art methods.

PreRNN+BandRNN

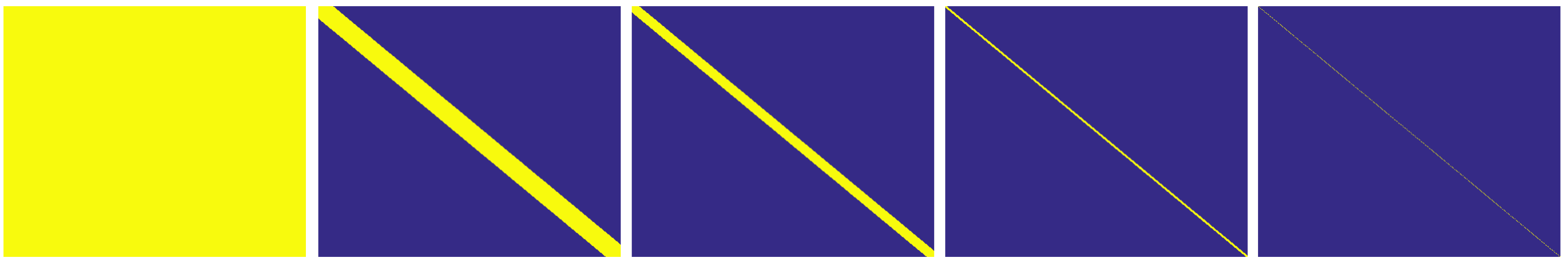
Overview



A schematic overview of the traditional RNN and the proposed PreRNN+BandRNN.

BandRNN

Sparsify Hidden-to-Hidden Weight Matrix



full (2048x2048) | 201 (10%) | 101 (5%) | 21 (1%) | diag (1)

PreRNN+BandRNN

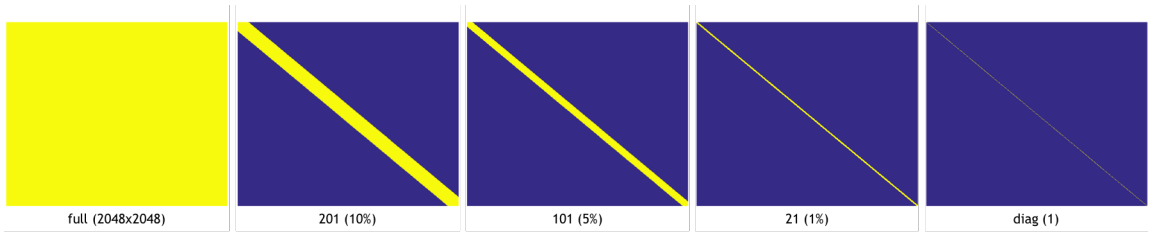
Action Recognition

PreVRNN	Sparsity (H2H)
91.9%	diag
92.3%	1%
92.0%	5%
92.2%	10%
92.7%	full
Traditional VRNN: 91.6%	
Baseline (w/o RNN): 91.2%	

PreLSTM	PreLSTM-SIH	Sparsity (H2H)
92.7%	92.6%	diag
92.8%	93.0%	1%
92.9%	92.7%	5%
93.2%	92.9%	10%
93.2%	93.5%	full
Traditional LSTM: 92.5%		
Baseline (w/o RNN): 91.2%		

PreGRU	PreGRU-SIH	Sparsity (H2H)
92.8%	92.5%	diag
92.8%	92.5%	1%
92.8%	92.3%	5%
92.9%	92.6%	10%
93.7%	93.3%	full
Traditional GRU: 92.2%		
Baseline (w/o RNN): 91.2%		

Classification accuracy of PreRNN(-SIH) with various sparsity of hidden-to-hidden weight matrices on UCF101.

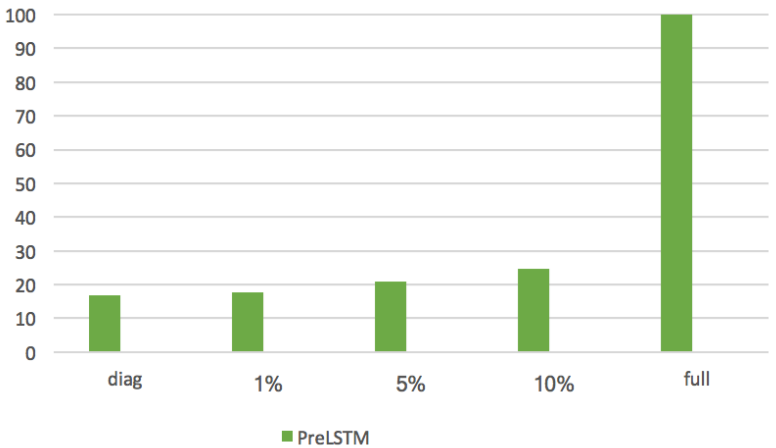


PreRNN+BandRNN

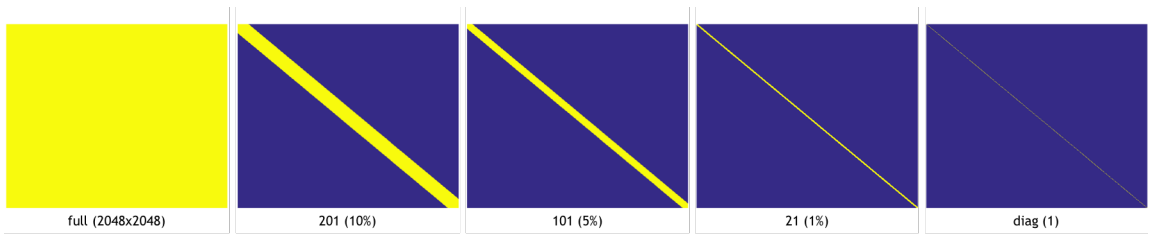
Action Recognition

PreLSTM	PreLSTM-SIH	Sparsity (H2H)
92.7%	92.6%	diag
92.8%	93.0%	1%
92.9%	92.7%	5%
93.2%	92.9%	10%
93.2%	93.5%	full
Traditional LSTM: 92.5%		
Baseline (w/o RNN): 91.2%		

Classification accuracy of PreLSTM(-SIH) with various sparsity of hidden-to-hidden weight matrices on UCF101.



Ratios (%) of recurrent parameters of PreLSTM(-SIH) to traditional LSTM with various sparsity of hidden-to-hidden weight matrices.

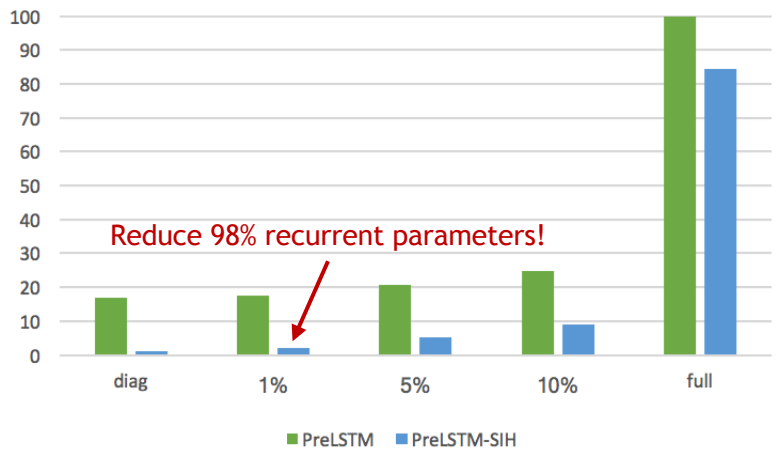


PreRNN+BandRNN

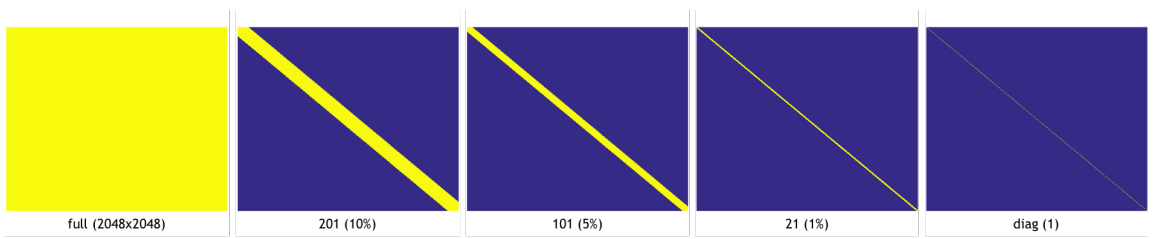
Action Recognition

PreLSTM	PreLSTM-SIH	Sparsity (H2H)
92.7%	92.6%	diag
92.8%	93.0%	1%
92.9%	92.7%	5%
93.2%	92.9%	10%
93.2%	93.5%	full
Traditional LSTM: 92.5%		
Baseline (w/o RNN): 91.2%		

Classification accuracy of PreLSTM(-SIH) with various sparsity of hidden-to-hidden weight matrices on UCF101.



Ratios (%) of recurrent parameters of PreLSTM(-SIH) to traditional LSTM with various sparsity of hidden-to-hidden weight matrices.



PreRNN+BandRNN for Video Understanding

Summary

- PreRNN: better leverage strong generalization of pre-trained CNNs
- PreRNN-SIH: simplify input-to-hidden states and largely reduce input-to-hidden recurrent parameters
- BandRNN: sparsify hidden-to-hidden weight matrices and further significantly reduce hidden-to-hidden recurrent parameters
- PreRNN+BandRNN: simple and effective, produce better or comparable results to traditional RNNs, while only introduce super lightweight recurrent parameters

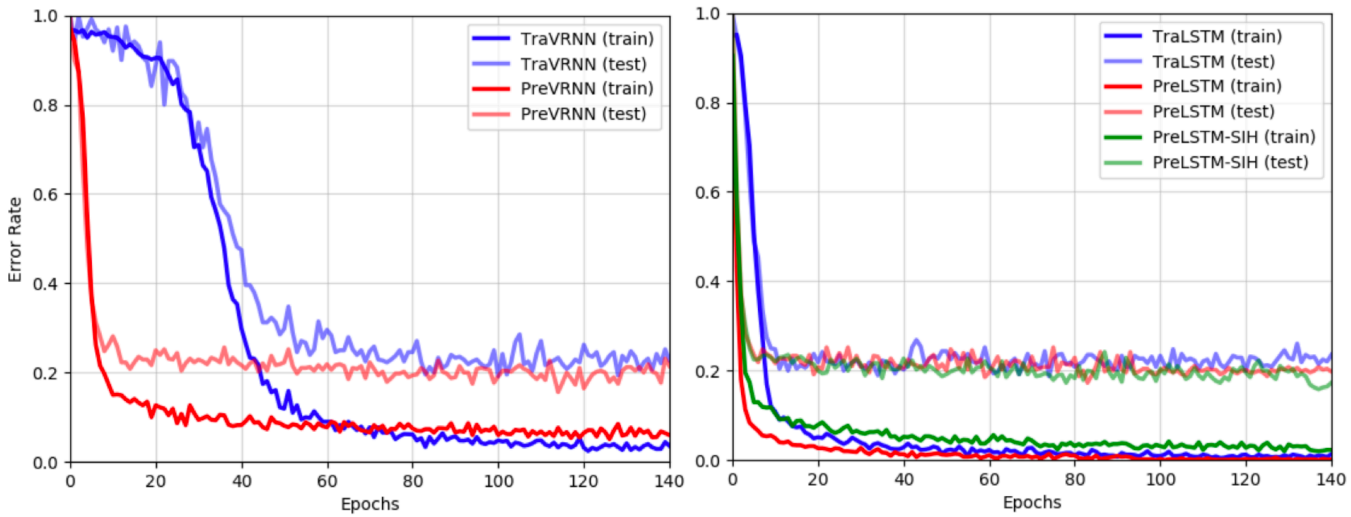
Majority of this work can be found at:

X. Yang, P. Molchanov, J. Kautz. **Making Convolutional Networks Recurrent for Visual Sequence Learning**. CVPR, 2018.


NVIDIA

Convergence

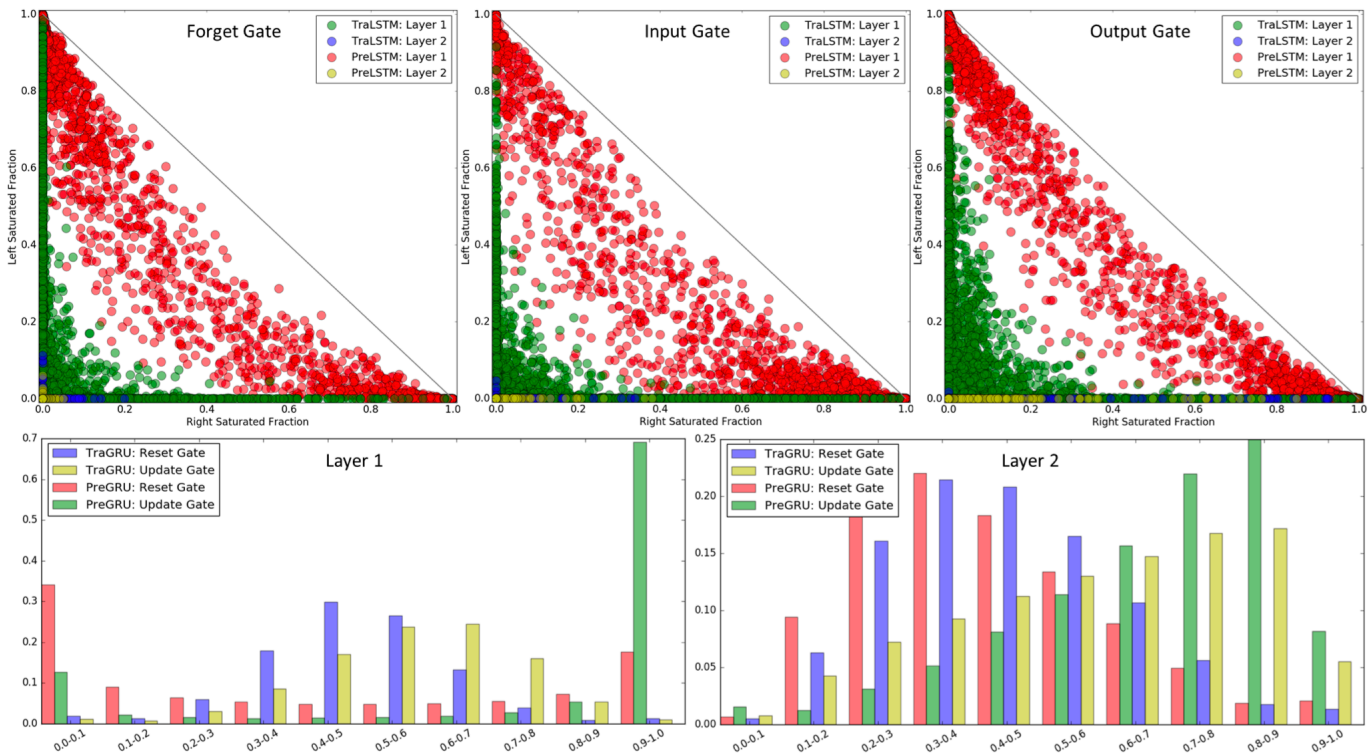
PreRNN Converges Faster



Comparison of the training processes between the traditional RNN and our proposed PreRNN(-SIH) for VRNN (left) and LSTM (right).

Understanding RNNs

Internal Mechanism of Traditional RNN and PreRNN



Examples of the gate activation distribution for LSTM and GRU. Top: saturation plots of the fraction of times that each gate unit is left or right saturated for LSTM. Bottom: activation histograms over 10 bins for GRU.