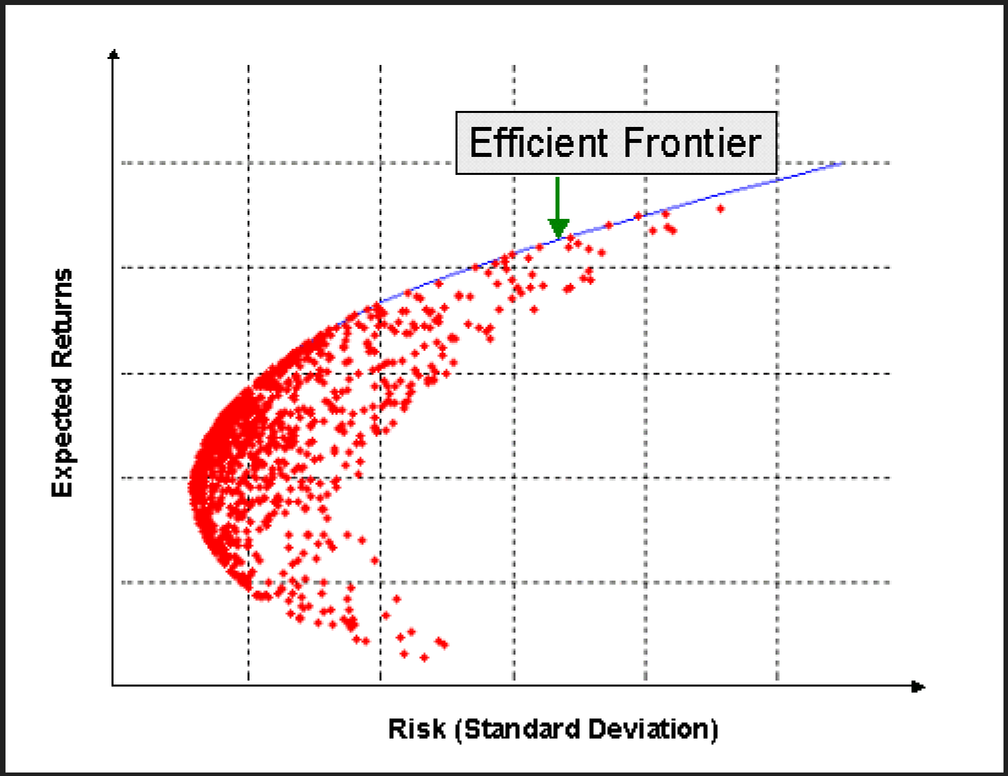
Neural Networks:

compression
and constrained learning

Neural Network Compression

A procedure which reduces the size of a network with an acceptable impact on its test accuracy

The *Efficient Frontier* analogy



Research Questions

- Can we "read" a trained MLP/LSTM to find it out what solution it "represents"?
- What types of solutions can ANNs trained through SGD can and cannot reach?
- Can we estimate the "performance distance" between two ANNs without evaluating on the entire test set?
- How does LSTM training relate to deep MLP training?

Scope Questions

- Classification tasks vs. general.
- How much to invest in "cracking" MLP's before moving to sequence models?

SECTION 1

SOME GENERAL OBSERVATIONS

Typical ANN Layer

- $z = xW + b$
- $a = g(z)$, we'll assume $g = \tanh$
- dims:
  - $z \in \mathbb{R}^{1 \times n}$
  - $a \in \mathbb{R}^{1 \times m}$
  - $W \in \mathbb{R}^{m \times n}$
  - $b \in \mathbb{R}^{1 \times n}$

Typical ANN Layer

- $z = xW + b$
- $a = g(z)$, we'll assume $g = \tanh$
- dims:
  - $z \in \mathbb{R}^{1 \times n}$
  - $a \in \mathbb{R}^{1 \times m}$
  - $W \in \mathbb{R}^{m \times n}$
  - $b \in \mathbb{R}^{1 \times n}$
- Now let's put it in the context of multilayer
  - (l) will denote layer index

Typical ANN layer

- $z^{(l)} = x^{(l)}W^{(l)} + b^{(l)}$
- $a^{(l)} = \tanh(z^{(l)})$

Typical ANN layer

- $z^{(l)} = x^{(l)} W^{(l)} + b^{(l)}$
- $a^{(l)} = \tanh(z^{(l)})$

$$x^{(l+1)} \equiv a^{(l)}$$

$a^{(l)}$ is the input for next layer

What does a connection matrix "do"?

- Rotate → Stretch&reflect → Rotate

- Columns as Features

What does a connection matrix "do"?

- Rotate → Stretch&reflect → Rotate
  - The entire matrix is an **affine tranformation** in hyperspace
  - Result may be in a lower- or higher- dimension space
  - This approach corresponds to the matrix's *Singular Value Decomposition.*
- Columns as Features

What does a connection matrix "do"?

Columns as *Features*

What does a connection matrix "do"?

Columns as *Features*

- The features are non-linearly combined layer by layer...
  - ... or in the same layer (two layers are enough)

What does a connection matrix "do"?

Columns as *Features*

- The features are non-linearly combined layer by layer...
  - ... or in the same layer (two layers are enough)
- Before last layer: Convert each *feature* into a *score*
  - each **column** represents the scores of a class
  - The predicted class is decided by highest score

What does a connection matrix "do"?

Columns as *Features*

- The features are non-linearly combined layer by layer...
  - ... or in the same layer (two layers are enough)
- Before last layer: Convert each *feature* into a *score*
  - each **column** represents the scores of a class
  - The predicted class is decided by highest score
- Probabilstic interpretation of scores?

What does a connection matrix "do"?

Columns as *Features*

- The features are non-linearly combined layer by layer...
  - ... or in the same layer (two layers are enough)
- Before last layer: Convert each *feature* into a *score*
  - each **column** represents the scores of a class
  - The predicted class is decided by highest score
- Probabilstic interpretation of scores?
- I doubt it

What does a connection matrix "do"?

Columns as *Features*

- The features are non-linearly combined layer by layer...
  - ... or in the same layer (two layers are enough)
- Before last layer: Convert each *feature* into a *score*
  - each **column** represents the scores of a class
  - The predicted class is decided by highest score
- Probabilstic interpretation of scores?
- I doubt it
- During training, scores only go up!
  - (we'll see why)

What does a *nonlinearity* "do"?

What does a *nonlinearity* "do"?

- "Squash" together values (beyond a threshold)

What does a *nonlinearity* "do"?

- "Squash" together values (beyond a threshold)
- Reduces information..

What does a *nonlinearity* "do"?

- "Squash" together values (beyond a threshold)
- Reduces information..
  - but makes things more interesting

What does a *nonlinearity* "do"?

- "Squash" together values (beyond a threshold)
- Reduces information..
  - but makes things more interesting
- Example:

What does a *nonlinearity* "do"?

- "Squash" together values (beyond a threshold)
- Reduces information..
  - but makes things more interesting
- Example:
  - $z = 4x_1 + 2x_2 + 3x_3$
    - changes in values of $x_{(\cdot)}$ have the same effect regardless of z

What does a *nonlinearity* "do"?

- "Squash" together values (beyond a threshold)
- Reduces information..
  - but makes things more interesting
- Example:
  - $z = 4x_1 + 2x_2 + 3x_3$
    - changes in values of $x_{(\cdot)}$ have the same effect regardless of z
  - $a = \tanh(z + 0.5)$
    - for {z | z > 1.3 ∨ z < 2.3}, small changes in $x_{(\cdot)}$ have almost no effect on a
    - a now has a *qualitative* rather than *quantitative* interpretation

What does a nonlinearity "do"? (2)

What does a nonlinearity "do"? (2)

- *Saturated* activations correspond to *decision planes*

What does a nonlinearity "do"? (2)

- *Saturated* activations correspond to *decision planes*
- Layer by layer (or in the same layer), planes combine into (non-convex) *regions*

What does a nonlinearity "do"? (2)

- *Saturated* activations correspond to *decision planes*
- Layer by layer (or in the same layer), planes combine into (non-convex) *regions*
- Therefore, (saturated) activations in intermediate layers are *region indicators*

What does a nonlinearity "do"? (2)

- *Saturated* activations correspond to *decision planes*
- Layer by layer (or in the same layer), planes combine into (non-convex) *regions*
- Therefore, (saturated) activations in intermediate layers are *region indicators*
- These regions
  - have soft boundaries
  - and may overlap.

How do the nonlinearities affect training?

After reaching an *approximate* fit, futher epochs are expected to "harden" region boundaries because:

- Random walk is not symmetric: larger $|z|$ implies smaller stepsize.
- *neg-log-softmax* error term is always positive - class scores "race" to infinity

SECTION 2

SURVEY OF SELECTED PAPERS

Main Takeaway from reading so far

There are many approaches, and all of them work exceptionally well!

Imposed Constraint	Interpretation	Benefit
Low bit depth	Coarser search-grid	up to 100x faster
"Fix together" arbitrary connection elements	Impose correlations between columns	4x reduction in number of parameters
Matrix separated into 2 low-rank matrices	Columns constrained to a subspace	?x compression

Binarized networks

(Courbariaux *et al.* 2016)

Concept: Develop an MLP with all connections weights restricted to +1 and -1

Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1

Matthieu Courbariaux*[1] MATTHIEU.COURBARIAUX@GMAIL.COM
Itay Hubara*[2] ITAYHUBARA@GMAIL.COM
Daniel Soudry[3] DANIEL.SOUDRY@GMAIL.COM
Ran El-Yaniv[2] RANI@CS.TECHNION.AC.IL
Yoshua Bengio[1,4] YOSHUA.UMONTREAL@GMAIL.COM

[1]Université de Montréal
[2]Technion - Israel Institute of Technology
[3]Columbia University
[4]CIFAR Senior Fellow
*Indicates equal contribution. Ordering determined by coin flip.

Binarized networks

(Courbariaux *et al.* 2016)

Concept: Develop an MLP with all connections weights restricted to +1 and -1

Overview

Binarized networks

(Courbariaux *et al.* 2016)

Concept: Develop an MLP with all connections weights restricted to +1 and -1

Overview

- Successfully trained a "BNN" (binarized neural network) on an image classification task

Binarized networks

(Courbariaux *et al.* 2016)

Concept: Develop an MLP with all connections weights restricted to +1 and -1

Overview

- Successfully trained a "BNN" (binarized neural network) on an image classification task
- Reached same performance as reference network

Binarized networks

(Courbariaux *et al.* 2016)

Concept: Develop an MLP with all connections weights restricted to +1 and -1

Overview

- Successfully trained a "BNN" (binarized neural network) on an image classification task
- Reached same performance as reference network
- with just a modest increase in number of nodes per layer

Binarized networks (2)

Method's benefits

Binarized networks (2)

Method's benefits

- Demonstrated **7x** time reduction (through custom CUDA kernel)

Binarized networks (2)

Method's benefits

- Demonstrated **7x** time reduction (through custom CUDA kernel)
- computations should reduce by ~$6*10^2$:
  - Multiply two 32-bit floating-point numbers: ~**600 ops**
  - Multiply two 1-bit numbers: **1 op** (XNOR gate)

Binarization - Limitations

Binarization - Limitations

- Custom hardware and compiler optimizations are required.

Binarization - Limitations

- Custom hardware and compiler optimizations are required.
- Performance outside image classification: not tested

Binarization - Limitations

- Custom hardware and compiler optimizations are required.
- Performance outside image classification: not tested
- Cumbersome 'hybrid' compuational model still required:

Binarization - Limitations

- Custom hardware and compiler optimizations are required.
- Performance outside image classification: not tested
- Cumbersome 'hybrid' compuational model still required:
  - Inputs: floating point

Binarization - Limitations

- Custom hardware and compiler optimizations are required.
- Performance outside image classification: not tested
- Cumbersome 'hybrid' compuational model still required:
  - Inputs: floating point
  - Intermediate layers: binary

Binarization - Limitations

- Custom hardware and compiler optimizations are required.
- Performance outside image classification: not tested
- Cumbersome 'hybrid' compuational model still required:
  - Inputs: floating point
  - Intermediate layers: binary
  - Class scores: integers

Binarization - Analysis

- *Any* arbitrary vector in hyperspace can be represented as:
  - A length $l \in \mathbb{R}^+$, and angles $\phi_1, \phi_2, \ldots \phi_{d-1}$
  - $\phi_i \in (0, 2\pi)$

Binarization - Analysis

- *Any* arbitrary vector in hyperspace can be represented as:
  - A length $l \in \mathbb{R}^+$, and angles $\phi_1, \phi_2, \ldots \phi_{d-1}$
  - $\phi_i \in (0, 2\pi)$
- ANN matrix columns $\iff$ arbitrary vectors

Binarization - Analysis

- *Any* arbitrary vector in hyperspace can be represented as:
  - A length $l \in \mathbb{R}^+$, and angles $\phi_1, \phi_2, \ldots \phi_{d-1}$
  - $\phi_i \in (0, 2\pi)$
- ANN matrix columns $\Longleftrightarrow$ arbitrary vectors
- BNN matrix columns $\Longleftrightarrow$ Constrained vectors:
  - $l = \sqrt{d}$
  - $\hat{\phi}_i \cdot \hat{\phi}_j = \cos\theta_{ij} \in \pm(1 - \frac{2k}{d})$ where $k = 1, \ldots, d/2$

Binarization - Analysis

- *Any* arbitrary vector in hyperspace can be represented as:
  - A length $l \in \mathbb{R}^+$, and angles $\phi_1, \phi_2, \ldots \phi_{d-1}$
  - $\phi_i \in (0, 2\pi)$
- ANN matrix columns $\iff$ arbitrary vectors
- BNN matrix columns $\iff$ Constrained vectors:
  - $l = \sqrt{d}$
  - $\hat{\phi}_i \cdot \hat{\phi}_j = \cos\theta_{ij} \in \pm(1 - \frac{2k}{d})$ where $k = 1, \ldots, d/2$

Yet they succeed!

Binarization - Conclusions

- Length is not necessary to represent "states"
  - All activations are saturatd
- Good solutions do not require "infinite" resolution in input space.
- Note that XNOR gates span the complete functional space.

Repeating elements in a connection matrix

Repeating elements in a connection matrix

Compressing Neural Networks with the Hashing Trick

Wenlin Chen[*] WENLINCHEN@WUSTL.EDU
James T. Wilson[*] J.WILSON@WUSTL.EDU
Stephen Tyree[*†] STYREE@NVIDIA.COM
Kilian Q. Weinberger[*] KILIAN@WUSTL.EDU
Yixin Chen[*] CHEN@CSE.WUSTL.EDU

[*] Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO, USA
[†] NVIDIA, Santa Clara, CA, USA

Repeating elements in a connection matrix

Compressing Neural Networks with the Hashing Trick

Wenlin Chen* WENLINCHEN@WUSTL.EDU
James T. Wilson* J.WILSON@WUSTL.EDU
Stephen Tyree*† STYREE@NVIDIA.COM
Kilian Q. Weinberger* KILIAN@WUSTL.EDU
Yixin Chen* CHEN@CSE.WUSTL.EDU
* Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO, USA
† NVIDIA, Santa Clara, CA, USA

Concept: Save memory and multiplications, by arbitrarily constraining different entries to the same value

Repeating elements in a connection matrix

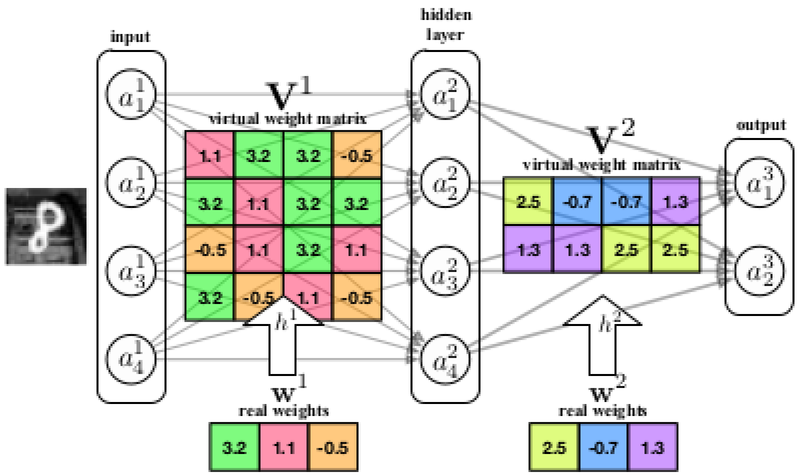


Figure 1. An illustration of a neural network with random weight sharing under compression factor $\frac{1}{4}$. The $16+9=24$ virtual weights are compressed into 6 real weights. The colors represent matrix elements that share the same weight value.

Repeating elements in a connection matrix

- Decide on $K^{(l)}$, free parameters per layer, $K^{(l)} \ll M^{(l)} \times N^{(l)}$
- Create a hash function $h : [M] \times [N] \to [K]$
- Set $V_{ij} = w_{h(i,j)}$
  - V_{ij} is the (virtual) connection matrix
  - $w_{(\cdot)}$ is a vector of K parameters

Repeating elements - Analysis

Repeating elements - Analysis

- Simple and fairly generic (CNNs, RNNs, ...)

Repeating elements - Analysis

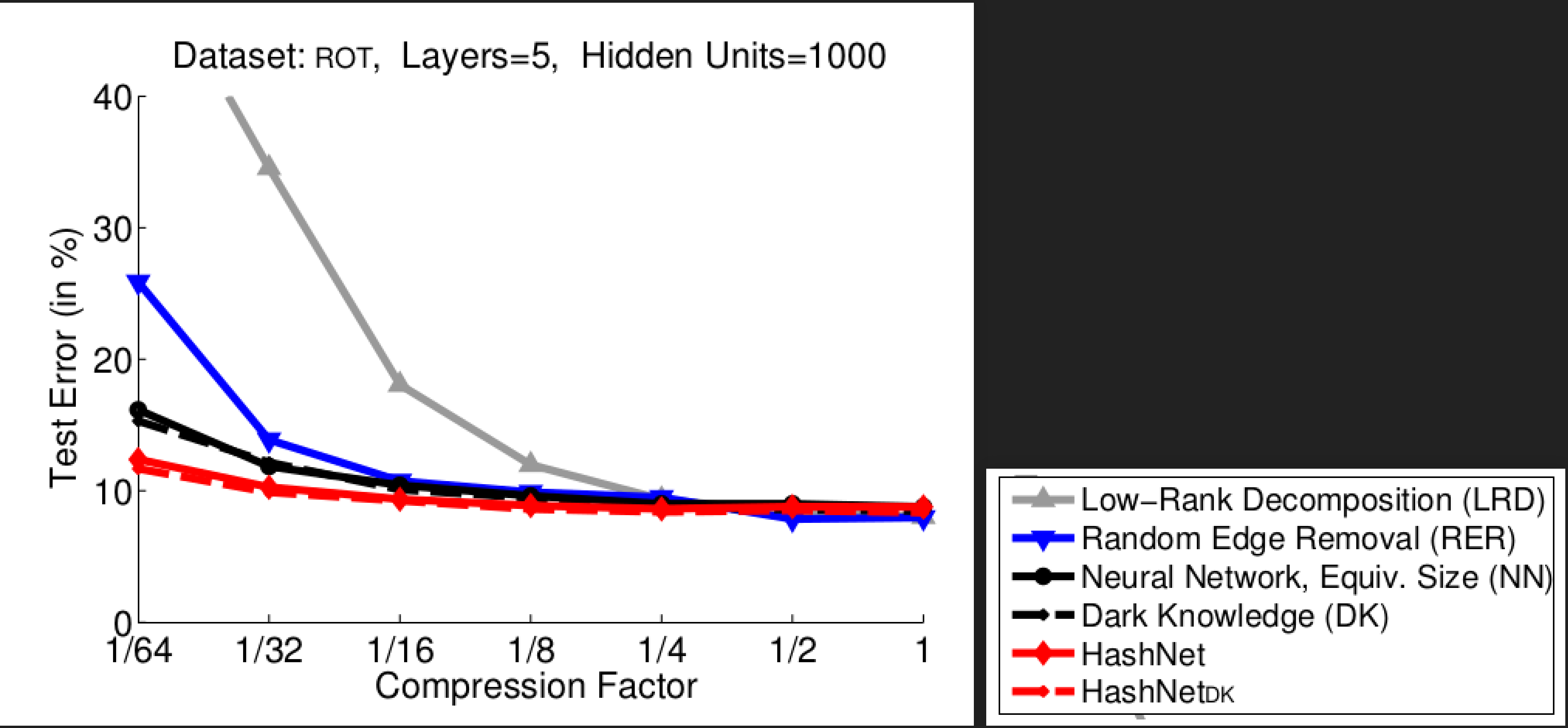
- Simple and fairly generic (CNNs, RNNs, ...)
- Adjustable compression factor which exceeds binarization
  - 1:64 $\Longleftrightarrow$ ½ bit per entry!

Repeating elements - Analysis

- Simple and fairly generic (CNNs, RNNs, ...)
- Adjustable compression factor which exceeds binarization
  - 1:64 $\Longleftrightarrow$ ½ bit per entry!
- outperforms other methods(?)

Repeating elements - Analysis

- Simple and fairly generic (CNNs, RNNs, ...)
- Adjustable compression factor which exceeds binarization
  - 1:64 ⟺ ½ bit per entry!
- outperforms other methods(?)



Connection hasing vs. feature hashing

For z_i (the layer outputs pre-nonlinearity):

$$z_i = \sum_{j=1}^{m} V_{ij} a_j$$

Equivalently $z_i = \mathbf{w}^T \phi_i(\mathbf{a})$ Where

$$[\phi_i(\mathbf{a})]_k = \sum_{j:h(i,j)=k} a_j$$

Which means that each z_i depends on a sum of an arbitrary subset of the previous layer's activations $a_1, \ldots, a_m$

Factorization

Predicting Parameters in Deep Learning

Misha Denil[1] **Babak Shakibi**[2] **Laurent Dinh**[3]
Marc'Aurelio Ranzato[4] **Nando de Freitas**[1,2]
[1]University of Oxford, United Kingdom
[2]University of British Columbia, Canada
[3]Université de Montréal, Canada
[4]Facebook Inc., USA
{misha.denil,nando.de.freitas}@cs.ox.ac.uk
laurent.dinh@umontreal.ca
ranzato@fb.com

- Concept: "Generate" $W \in \mathbb{R}^{m \times n}$ from UV, $U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{k \times n}$
- Number of parameters drops from mn to $(m+n)k$

Factorization

- Not all authors agree on the effectiveness:
  - U and V "cannot be trained together"
  - U as a "feature bank"
  - Predetermined by network designer
  - or pretrained
- In image processing, "smooth" U's work well.

This approach performed worst in the benchmark conducted by the *Hashing Trick* authors.

Batch Normalization

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Concept: Speed up training by deliberately eliminating the scale and bias of inputs to a layer

Batch Normalization

Concept (cont.):

- Speed up training by deliberately eliminating the scale and bias of inputs to a layer
- Replace the implicit scale and bias of the input population with explicit, learnable scale and bias

Accomplished:

- Achieved faster training and surpassed state-of-the-art performance in image processing
- Is this a "Weaker" form of factorization?

SUMMARY

- Approaches "overlap"
- No theoretical framework
- One successful application is not enough to understand approach
- Smaller networks which "disguise themselves" as larger ones
- Looks like everyone is trying to fool SGD...

NEXT STEPS

Further Reading

- Sequence models - training and compression
- Architecture evolution
- Training on "soft" scores
- SGD analysis

NEXT STEPS

Reasearch

- Implement ordinary vs. hashed MLP and look into training process
- Design a problem with a known solution and check the solutions reached by SGD

Ideas to further develop

- Decision-tree-like splits
- Correlations and linear dependence between activations

THANK YOU!