Stabilizing Gradients for Deep Neural Networks

Inderjit S. Dhillon
UT Austin & Amazon

Int”l Conference on Machine Learning and Data Science (ICMLDS)
Noida, India
Dec 15, 2017

Joint work with Jiong Zhang and Qi Lei (UT Austin)
1 Introduction
   ● Deep Learning
   ● Existing Solutions

2 Proposed Solution
   ● Spectral Parameterization
   ● Application to RNN: Spectral RNN
   ● Training algorithms
   ● Extension to General Weight Matrices
   ● Direct conclusions from the gradients

3 Experimental Results
Things we can do with Deep Learning

Object Detection

Mastering Go

Image classification

Image captioning

Dog: 94%
Cat: 31%
Bird: 2%
Boat: 0%

Automatically captioned

A dog is sitting on the beach next to a dog.
Feed forward network: Multilayer perceptron (MLP)

\[ h^{(t)} = \delta(W^{(t)}h^{(t-1)} + b^{(t)}) \]

- \( D \)-layer MLP with activation function \( \delta \):
- \( \Theta = \{W^{(t)}, b^{(t)}\}_{t=1}^{D} \) are model parameters,
- \( h^{(0)} = x \) is the input, and \( h^{(D)} \) is the network’s output.
Feed forward network: Multilayer perceptron (MLP)

- Given dataset \( \{x_i, y_i\}_{i=1}^{N} \), the loss is measured as:

\[
L(X, Y; \Theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, h^{(D)}(x_i; \Theta))
\]

- For regression problems, \( \ell \) could be squared loss:

\[
\ell(y_i, h^{(D)}) = \left\| y_i - h^{(D)} \right\|^2
\]

- For classification problems, \( \ell \) could be cross entropy loss:

\[
\ell(y_i, h^{(D)}) = - \log \left( \frac{\exp(h_{y_i}^{(D)})}{\sum_k \exp(h_k^{(D)})} \right)
\]
To evaluate loss $\ell$ we need to compute $h^{(D)}$. We do this by iteratively evaluating $h^{(t)}$ for $t = 1, 2, \ldots, D$. 
To evaluate loss $\ell$ we need to compute $h^{(D)}$. We do this by iteratively evaluating $h^{(t)}$ for $t = 1, 2, \ldots, D$.

This is called **forward propagation**: the information of input $x$ is propagating forward through the network.
Learning the Parameters: Gradient descent

To minimize loss $L(\Theta) = \sum_{i=1}^{N} \ell(y_i; h^{(D)}(x_i; \Theta))$, we can conduct gradient descent to update parameters $\theta \in \Theta = \{W^{(t)}, b^{(t)}\}_{t=1}^{D}$:

$$\theta \leftarrow \theta - \frac{\eta}{N} \sum_{i \in [N]} \frac{\partial \ell}{\partial \theta} (y_i; h^{(D)}(x_i, \Theta))$$

**Problem:** Even 1 iteration is too expensive when $N$ is huge
Stochastic Gradient Descent (SGD)

SGD to train a neural network:

- A minibatch $B \subset [N]$ is sampled.
- Each parameter $\theta \in \{W^{(t)}, b^{(t)}\}_{t=1}^{D}$ is updated using an estimate of the gradient:

$$
\theta \leftarrow \theta - \frac{\eta}{|B|} \sum_{i \in B} \frac{\partial \ell}{\partial \theta} (y_i; h^{(D)}(x_i, \Theta))
$$

- Step size $\eta$ is usually selected by line search or heuristics like Adam (Kingma et al. 2014).
Recall \( \ell = \ell(y, h^{(D)}) \); \( h^{(t)} = \delta(W^{(t)}h^{(t-1)} + b^{(t)}), \forall t \in [D] \)
Recall $\ell = \ell(y, h^{(D)}); \quad h^{(t)} = \delta(W^{(t)}h^{(t-1)} + b^{(t)}), \quad \forall t \in [D]$

By chain rule, derivatives of $(t - 1)$-st layer depend on $\frac{\partial \ell}{\partial h^{(t)}}$

$$\frac{\partial \ell}{\partial W^{(t)}} = \left[ \frac{\partial h^{(t)}}{\partial W^{(t)}} \right]^{\top} \frac{\partial \ell}{\partial h^{(t)}}; \quad \frac{\partial \ell}{\partial h^{(t)}} = \left[ \frac{\partial h^{(t+1)}}{\partial h^{(t)}} \right]^{\top} \frac{\partial \ell}{\partial h^{(t+1)}}$$

We need to iteratively evaluate $\frac{\partial \ell}{\partial h^{(t)}}$ for $t = D, \ldots, 1$, which is called back propagation.

- Gradient computation requires $h^{(t)}$.
- Thus before each BP, we need to do a FP.
Gradient evaluation: Back propagation

- Recall $h^{(t)} = \delta(W^{(t)} h^{(t-1)} + b^{(t)}), \forall t \in [D]$
- Within each layer, partial derivatives are:

  \[ \frac{\partial h^{(t)}}{\partial W^{(t)}} = h^{(t-1)} \circ \text{diag}(\delta'_t); \quad \frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \left[ W^{(t)} \right]^{\top} \text{diag}(\delta'_t) \]

  Here $\circ$ denotes outer product, $\delta'_t = \delta'(W^{(t)} h^{(t-1)} + b^{(t)})$
- Backpropagation yields:

  \[ \frac{\partial \ell}{\partial h^{(t)}} \propto \prod_{i=D}^{t+1} \left[ W^{(i)} \right]^{\top} \text{diag}(\delta'_i) \]

  Hence, gradients can easily “explode” or “vanish”
Things we can do with Recurrent Neural Networks

Handwriting Recognition

Translation

Question answering

Speech Recognition
Recurrent Neural Networks (RNN)

RNN with activation function $\delta$:

$$h^{(t)} = \delta(W h^{(t-1)} + M x^{(t-1)} + b)$$

$$\hat{y}^{(t)} = Y h^{(t)}$$

Key differences from MLP:
- Parameters $\Theta = \{W, M, b, Y\}$ are shared for each layer
- Input $x_i = \{x_i^{(0)}, x_i^{(1)}, \ldots x_i^{(D)}\}$ is fed sequentially
- Output $\hat{y} = \{\hat{y}^{(0)}, \hat{y}^{(1)}, \ldots \hat{y}^{(D)}\}$ is evaluated at each layer
- Loss is measured as: $L(X, Y; \Theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \hat{y}(x_i; \Theta))$
RNN: forward propagation

Forward propagation:

- Evaluate activations
  \[ h^{(t)} = \delta(W h^{(t-1)} + M x^{(t-1)} + b), \quad t = 1, \ldots, D \] layer by layer

- Evaluate \( \hat{y}^{(t)} = Y h^{(t)}, \quad t = 1, \ldots, D \) layer by layer
RNN: backward propagation

- Similar to MLP, gradients propagate through layers:
  \[
  \frac{\partial h(t)}{\partial W} = h(t) \circ \text{diag}(\delta'_t); \quad \frac{\partial h(t)}{\partial h(t-1)} = W^\top \text{diag}(\delta'_t)
  \]
- Backpropagation:
  \[
  \frac{\partial \ell}{\partial h(t)} \propto \prod_{i=D}^{t+1} W^\top \text{diag}(\delta'_i)
  \]
- Hence, gradients can easily “explode” or “vanish”
- Problem: Long-range dependencies cannot be captured
Sigmoid Activation Function

- Sigmoid function
  \[ \sigma(z) = \frac{1}{1 + e^{-z}} \] has gradient:
  \[ \sigma'(z) = \sigma(z)(1 - \sigma(z)) = \frac{1}{e^z + e^{-z} + 2} \]

- \(|z|\) large \(\Rightarrow\) vanishing gradients

- Solution indicated:
  constrain size of each \(h\)

**Figure:** Sigmoid activation
Activation Functions: Vanishing Gradients

- Saturated activation functions (esp. sigmoid or tanh) ⇒ vanishing gradients
- Absolute value of input is large for saturated activations ⇒ vanishing gradients
- Solution indicated: constrain size of each $h$

**Figure: Common activation functions and their derivatives.**
1 Introduction
   - Deep Learning
   - Existing Solutions

2 Proposed Solution
   - Spectral Parameterization
   - Application to RNN: Spectral RNN
   - Training algorithms
   - Extension to General Weight Matrices
   - Direct conclusions from the gradients

3 Experimental Results
Existing Solutions

- Gradient Clipping (Pascanu et al. 2013)

- Initialization with identity/orthogonal matrix (Le, Jaitly & Hinton 2015)
Existing Solutions

- Long Short-Term Memory (LSTM):

\[ z_t = \sigma (W_z \cdot [h_{t-1}, x_t]) \]
\[ r_t = \sigma (W_r \cdot [h_{t-1}, x_t]) \]
\[ \tilde{h}_t = \tanh (W \cdot [r_t \ast h_{t-1}, x_t]) \]
\[ h_t = (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t \]

**Figure:** One node of LSTM. (Colah 2015)
Existing Solutions

- Long Short-Term Memory (LSTM):

Avoids long-term dependency problems by “forget-gate layers”
Existing Solutions

- **Solution:** Keep $W^\top W = I$
  - **uRNN** *(Arjovsky et al., 2016)*
    - $W \in \mathbb{C}^{n \times n}$ is product of reflection, diagonal, and Fourier transform matrices
  - **Full-Capacity uRNN** *(Wisdom et al., 2016)*
  - **unitary RNN** *(Hyland & Ratsch, 2017)*
    - Allow $W \in \mathbb{C}^{n \times n}$ to span the whole unitary group
  - **oRNN** *(orthogonal RNN)* *(Mhammedi et al., 2017)*
    - Allow $W$ to span the whole orthogonal space by using Householder reflectors

- **Problem:** lose expressive power
Existing Solutions

- Solution: encourage orthogonality
  1. factorized RNN (Vorontsov et al. 2017)
     - Parameterize $W = U\Sigma V^\top$, encourage $\Sigma$ to be close to 1, and update $U, V$ by Cayley transform
  2. Parseval networks (Cisse et al. 2017)
     - Regularize with $\|I - W^\top W\|_F^2$

- Problem: high time complexity

- Question: How to solve the gradient vanishing/exploding problem with full expressive power and high efficiency?
Large-Scale LSTMs in Action

Google’s Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat
melvinp,schuster,qvl,krikun,yonghui,zhifengc,nsthorat@google.com

Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, Jeffrey Dean

Abstract

We propose a simple solution to use a single Neural Machine Translation (NMT) model to translate between multiple languages. Our solution requires no changes to the model architecture from a standard NMT system but instead introduces an artificial token at the beginning of the input sentence to specify the required target language. The rest of the model, which includes an encoder, decoder and attention module, remains unchanged and is shared across all languages. Using a shared wordpiece vocabulary, our approach enables Multilingual NMT using a single model without any increase in parameters, which is significantly simpler than previous proposals for Multilingual NMT. On the WMT’14 benchmarks, a single multilingual model achieves comparable performance for English→French and surpasses state-of-the-art results for English→German. Similarly, a single multilingual model surpasses state-of-the-art results for French→English and German→English on WMT’14 and WMT’15 benchmarks, respectively. On production corpora, multilingual models of up to twelve language pairs allow for better translation of many individual pairs. In addition to improving the translation quality of language pairs that the model was trained with, our models can also learn to perform implicit bridging between language pairs never seen explicitly during training, showing that transfer learning and zero-shot translation is possible for neural translation. Finally, we show analyses that hints at a universal interlingua representation in our models and show some interesting examples when mixing languages.
Google Neural Machine Translation Architecture

- Eight-layer bidirectional LSTM with Attention (Johnson et al., 2016)
- For each language pair: 1024 nodes (hidden dimension), 8 LSTM layers with a total of 255M parameters.
- Total training time is on the scale of weeks, on up to 100 GPUs.
1 Introduction
   - Deep Learning
   - Existing Solutions

2 Proposed Solution
   - Spectral Parameterization
   - Application to RNN: Spectral RNN
   - Training algorithms
   - Extension to General Weight Matrices
   - Direct conclusions from the gradients

3 Experimental Results
Spectral Parameterization

- An illustration of the parameterization process:

\[ W = U \Sigma V^T \]

A Householder reflector:

Represent \( U \) as:

\[
\begin{pmatrix}
I_{n-2} - 2\frac{u_n u_n^T}{\|u_n\|^2} \\
I_{n-1} - 2\frac{u_{n-1} u_{n-1}^T}{\|u_{n-1}\|^2}
\end{pmatrix} \cdot \begin{pmatrix}
1 \\
I_{n-1} - 2\frac{u_{n-1} u_{n-1}^T}{\|u_{n-1}\|^2}
\end{pmatrix} \cdots \begin{pmatrix}
I_{n-k_1} - 2\frac{u_{k_1} u_{k_1}^T}{\|u_{k_1}\|^2}
\end{pmatrix}
\]
Spectral Parameterization

- Maintain SVD of transition matrix:

\[ W = U\Sigma V^T \]
Spectral Parameterization

- Maintain SVD of transition matrix:

\[ W = U\Sigma V^\top \]

- Parameterize \( U, V \) by products of Householder reflectors:

\[
H_k(u) = \begin{cases} 
(I_n - k \frac{uu^\top}{\|u\|^2}) & , \quad u \neq 0 \\
I_n & , \quad \text{otherwise.}
\end{cases}
\]
Spectral Parameterization

- Maintain SVD of transition matrix:

\[ W = U \Sigma V^\top \]

- Parameterize \( U, V \) by products of Householder reflectors:

\[
\mathcal{H}_k(u) = \begin{cases} 
(I_{n-k} & I_{k-2} uu^\top \|u\|^2) \\
I_n 
\end{cases}, \quad u \neq 0
\]

\[
, \quad \text{otherwise.}
\]

- \( U \leftarrow \prod_{k=n}^{k_1} \mathcal{H}_k(u_k) \)
Spectral Parameterization

- Maintain SVD of transition matrix:
  \[ W = U\Sigma V^\top \]

- Parameterize \( U, V \) by products of Householder reflectors:
  \[
  \mathcal{H}_k(u) = \begin{cases} 
  \left( \begin{array}{c} I_{n-k} \\
  I_{k-2} u u^\top \\
  I_n 
  \end{array} \right), & u \neq 0 \\
  I_n, & \text{otherwise}
  \end{cases}
  \]

- \( U \leftarrow \prod_{k=n}^{k_1} \mathcal{H}_k(u_k) \)
- \( V \leftarrow \prod_{k=n}^{k_2} \mathcal{H}_k(v_k) \)
Proposed parametrization:

\[ M_{k_1,k_2} : (\mathbb{R}^{k_1} \times \ldots \times \mathbb{R}^n) \times (\mathbb{R}^{k_2} \times \ldots \times \mathbb{R}^n) \times (\mathbb{R}^n) \mapsto \mathbb{R}^{n \times n} \]

\[ \left( \{u_i\}_{i=k_1}^n, \{v_i\}_{i=k_2}^n, (\sigma) \mapsto \right. \]

\[ \underbrace{\mathcal{H}_n(u_n)\ldots\mathcal{H}_{k_1}(u_{k_1})}_{U} \underbrace{\text{diag}(\sigma)}_{\Sigma} \underbrace{\mathcal{H}_{k_2}(v_{k_2})\ldots\mathcal{H}_n(v_n)}_{V^\top}. \] (2)

- **Singular values are explicit:**
  \[ M_{k_1,k_2}(\{u_i\}_{i=k_1}^n, \{v_i\}_{i=k_2}^n, \sigma) \] is an \( n \times n \) real matrix with singular values \( \sigma \).
Proposed parametrization:

\[ M_{k_1,k_2} : (\mathbb{R}^{k_1} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^{k_2} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^n) \mapsto \mathbb{R}^{n \times n} \]

\[
\begin{align*}
(\{u_i\}_{i=k_1}^n), (\{v_i\}_{i=k_2}^n), (\sigma) \mapsto \\
\mathcal{H}_n(u_n) \cdots \mathcal{H}_{k_1}(u_{k_1}) \, \text{diag}(\sigma) \, \mathcal{H}_{k_2}(v_{k_2}) \cdots \mathcal{H}_n(v_n). 
\end{align*}
\]  

(2)

- **Singular values are explicit:**
  \( M_{k_1,k_2}(\{u_i\}_{i=k_1}^n,\{v_i\}_{i=k_2}^n,\sigma) \) is an \( n \times n \) real matrix with singular values \( \sigma \).

- **Full expressivity:** The image of \( M_{1,1} \) is the set of \( n \times n \) real matrices.
Proposed parametrization:

\[ M_{k_1,k_2} : (\mathbb{R}^{k_1} \times \ldots \times \mathbb{R}^n) \times (\mathbb{R}^{k_2} \times \ldots \times \mathbb{R}^n) \times (\mathbb{R}^n) \mapsto \mathbb{R}^{n \times n} \]

\[ \left( \{u_i\}_{i=k_1}^n , \{v_i\}_{i=k_2}^n , (\sigma) \right) \mapsto \mathcal{H}_n(u_n) \ldots \mathcal{H}_{k_1}(u_{k_1}) \underbrace{\text{diag}(\sigma)}_{\Sigma} \mathcal{H}_{k_2}(v_{k_2}) \ldots \mathcal{H}_n(v_n) . \]

1. **Singular values are explicit:**

   \( M_{k_1,k_2}(\{u_i\}_{i=k_1}^n , \{v_i\}_{i=k_2}^n , \sigma) \) is an \( n \times n \) real matrix with singular values \( \sigma \).

2. **Full expressivity:** The image of \( M_{1,1} \) is the set of \( n \times n \) real matrices.

3. **Orthogonal expressivity:** The image of \( M_{k_1,k_2} \) covers the set of \( n \times n \) orthogonal matrices if \( k_1 + k_2 \leq n + 2 \).
\[ \mathcal{M}_{1,1} \] seemingly maps a space of \( n^2 + 2n \) dimensions to a space of \( n^2 \) dimensions.

However, \( \mathcal{H}_k(u_k) \) is invariant to the norm of \( u_k \), so the domain of \( \mathcal{M}_{1,1} \) also has exactly \( n^2 \) dimensions.
Outline

1 Introduction
   - Deep Learning
   - Existing Solutions

2 Proposed Solution
   - Spectral Parameterization
   - Application to RNN: Spectral RNN
   - Training algorithms
   - Extension to General Weight Matrices
   - Direct conclusions from the gradients

3 Experimental Results
In Spectral RNN, we parametrize the transition matrix $W \in \mathbb{R}^{n \times n}$ using $m_1 + m_2$ Householder reflectors.

- Can select $m_1$ and $m_2$ to balance expressive power versus time/space complexity. (Full expressivity if $m_1 = m_2 = n$)
- Can do both forward and backward propagation in $O(n(m_1 + m_2))$ time. (RNN: $O(n^2)$)
- Can explicitly control the singular values. For example, as in (Vorontsov et al. 2017):

$$\sigma_i = 2r(\text{sigmoid}(\hat{\sigma}_i) - 0.5) + \sigma^*, \ i \in [n] \quad (3)$$

$\Rightarrow \sigma_i \in [\sigma^* - r, \sigma^* + r]$. Usually $\sigma^*$ is set to 1 and $r \ll 1$. 

Outline

1 Introduction
   - Deep Learning
   - Existing Solutions

2 Proposed Solution
   - Spectral Parameterization
   - Application to RNN: Spectral RNN
   - Training algorithms
   - Extension to General Weight Matrices
   - Direct conclusions from the gradients

3 Experimental Results
Forward propagation

- Only aspect different from regular RNN in forward propagation is computation of $Wh^{(t-1)}$:

$$Wh^{(t)} = \mathcal{H}_n(u_n) \cdots \mathcal{H}_{n-m_1+1}(u_{n-m_1+1}) diag(\sigma) \mathcal{H}_{n-m_2+1}(v_{n-m_2+1}) \cdots \mathcal{H}_n(v_n) h^{(t)}$$

- Can be done efficiently through $m_1 + m_2$ inner products and vector additions. For each reflector:

$$\mathcal{H}_k(u_k) h = \left( I_n - \frac{2 \hat{u}_k \hat{u}_k^\top}{\hat{u}_k^\top \hat{u}_k} \right) h = h - 2 \frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k} \hat{u}_k$$
Backward propagation

- Let $L(\{u_i\}, \{v_i\}, \sigma, M, Y, b)$ be the loss or objective function, $C^{(t)} = Wh^{(t)}$, $\hat{\Sigma} = \text{diag}(\hat{\sigma})$. Given $\frac{\partial L}{\partial C^{(t)}}$, we define:

$$
\frac{\partial L}{\partial u_k^{(t)}} := \left[ \frac{\partial C^{(t)}}{\partial u_k^{(t)}} \right]^\top, \quad \frac{\partial L}{\partial v_k^{(t)}} := \left[ \frac{\partial C^{(t)}}{\partial v_k^{(t)}} \right]^\top;
$$

$$
\frac{\partial L}{\partial \Sigma^{(t)}} := \left[ \frac{\partial C^{(t)}}{\partial \Sigma^{(t)}} \right]^\top, \quad \frac{\partial L}{\hat{\Sigma}^{(t)}} := \left[ \frac{\partial \Sigma^{(t)}}{\partial \hat{\Sigma}^{(t)}} \right]^\top;
$$

$$
\frac{\partial L}{\partial h^{(t-1)}} := \left[ \frac{\partial C^{(t)}}{\partial h^{(t-1)}} \right]^\top.
$$

- Back propagation for Spectral RNN requires $\frac{\partial C^{(t)}}{\partial u_k^{(t)}}$, $\frac{\partial C^{(t)}}{\partial v_k^{(t)}}$, $\frac{\partial C^{(t)}}{\partial \Sigma^{(t)}}$ and $\frac{\partial C^{(t)}}{\partial h^{(t-1)}}$. 
Partial gradients can be computed iteratively ($\hat{h} := \mathcal{H}_k(u_k)h$ and $g := \frac{\partial L}{\partial \hat{h}}$):

\[
\frac{\partial L}{\partial h} = \left[ \frac{\partial \hat{h}}{\partial h} \right]^\top \frac{\partial L}{\partial \hat{h}} = \left( I_n - \frac{2\hat{u}_k\hat{u}_k^\top}{\hat{u}_k^\top \hat{u}_k} \right) g = g - 2\frac{\hat{u}_k^\top g}{\hat{u}_k^\top \hat{u}_k} \hat{u}_k
\]

\[
\frac{\partial L}{\partial \hat{u}_k} = \left[ \frac{\partial \hat{h}}{\partial \hat{u}_k} \right]^\top \frac{\partial L}{\partial \hat{h}} = -2 \left( \frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k} I_n + \frac{1}{\hat{u}_k^\top \hat{u}_k} h\hat{u}_k^\top - 2\frac{\hat{u}_k^\top h}{(\hat{u}_k^\top \hat{u}_k)^2} \hat{u}_k^\top \hat{u}_k \right) g
\]

\[
= -2\frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k} g - 2\frac{\hat{u}_k^\top g}{\hat{u}_k^\top \hat{u}_k} h + 4\frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k} \hat{u}_k^\top g \hat{u}_k
\]

Thus backward propagation can also be done in $O((m_1 + m_2)n)$ time.
1 Introduction
- Deep Learning
- Existing Solutions

2 Proposed Solution
- Spectral Parameterization
- Application to RNN: Spectral RNN
- Training algorithms
- Extension to General Weight Matrices
- Direct conclusions from the gradients

3 Experimental Results
Extension to Non-square matrices

- For any real matrix $W \in \mathbb{R}^{m \times n}$ (assume $m < n$) with reduced SVD:
  \[
  W = U(\Sigma|0)(V_L|V_R)^	op = U\Sigma V_L^	op
  \]
  where $U \in \mathbb{R}^{m \times m}$, $\Sigma \in \text{diag}(\mathbb{R}^m)$, $V_L \in \mathbb{R}^{n \times m}$.
  - There exist $u_n, \ldots, u_{k_1}$ and $v_n, \ldots, v_{k_2}$ s.t.
    \[
    U = \mathcal{H}_m^m(u_m) \cdots \mathcal{H}_{k_1}^m(u_{k_1}), \quad V = \mathcal{H}_n^n(v_n) \cdots \mathcal{H}_{k_2}^n(v_{k_2}).
    \]
  - SVD parameterization for any matrix:
    \[
    \mathcal{M}_{k_1,k_2}^{m,n} : (\mathbb{R}^{k_1} \times \ldots \times \mathbb{R}^m) \times (\mathbb{R}^{k_2} \times \ldots \times \mathbb{R}^n) \times (\mathbb{R}^{\min(m,n)}) \mapsto \mathbb{R}^{m \times n}
    \]
    \[
    \left(\{u_i\}_{i=k_1}^m, \{v_i\}_{i=k_2}^n, (\sigma) \mapsto \right)
    \left(\mathcal{H}_m^m(u_m) \cdots \mathcal{H}_{k_1}^m(u_{k_1})\right) (\Sigma) \left(\mathcal{H}_{k_2}^n(v_{k_2}) \cdots \mathcal{H}_n^n(v_n)\right).
    \]
  - Can be applied to any Deep MLP (offers an alternative to Residual Networks)
Outline

1 Introduction
   - Deep Learning
   - Existing Solutions

2 Proposed Solution
   - Spectral Parameterization
   - Application to RNN: Spectral RNN
   - Training algorithms
   - Extension to General Weight Matrices
   - Direct conclusions from the gradients

3 Experimental Results
Direct conclusions from the gradients

For Spectral RNN, recall Gradient for activations is:

\[
\frac{\partial h(t)}{\partial h(k)} = \prod_{t \geq i \geq k} \frac{\partial h(i)}{\partial h(i-1)} = \prod_{t \geq i \geq k} W^\top \text{diag}(\delta'(h_{i-1}))
\]

- Solves the exploding gradient problem:

\[
\|W\|_2 \leq 1 + \epsilon \implies \left\| \frac{\partial h(t)}{\partial h(0)} \right\| \leq (1 + \epsilon)^t
\]

- Mitigates the vanishing gradient problem:

\[
\sigma_{\text{min}}(W) \geq 1 - \epsilon \implies \sigma_{\text{min}} \left( \frac{\partial h(t)}{\partial h(0)} \right) \geq \min |\delta'|^t (1 - \epsilon)^t
\]
Spectral MLP

- Generalization of MLP is bounded by its spectral Lipschitz constant $L$ (Bartlett et al. 2017)
  - Spectral MLP guarantees $L \leq (1 + \epsilon)^t$, if we control singular values s.t. $\|W\|_2 \leq 1 + \epsilon$
- Weight matrices are Parseval tight frames $\implies$ robustness in predictions (Cisse et al. 2017)
  - Spectral MLP guarantees near orthogonal weight matrix, namely tight frames
The Addition Task

Each input data includes two sequences

- top sequence: values sampled uniformly from $[0, 1]$
- bottom sequence: binary sequence with two 1’s and the rest are 0
- output: the dot product between the two sequences

Figure: The Addition Task from (Le et. al 2015).
Addition Task: Results

Figure: RNNs on addition task with $L$ layers & $n_h$ hidden dimension.
Multi-label Datasets

- Multiple labels with each data point.
- Each label $y$ is $M$-dimensional sparse binary vector.
- Data sets used are described in http://manikvarma.org/downloads/XC/XMLRepository.html

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Feature dimension</th>
<th>Label dimension</th>
<th>$N_{train}$</th>
<th>$N_{test}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mediamill</td>
<td>120</td>
<td>101</td>
<td>30993</td>
<td>12914</td>
</tr>
<tr>
<td>Bibtex</td>
<td>1836</td>
<td>159</td>
<td>4880</td>
<td>2515</td>
</tr>
<tr>
<td>Delicious</td>
<td>500</td>
<td>983</td>
<td>12920</td>
<td>3185</td>
</tr>
<tr>
<td>Wiki10</td>
<td>101938</td>
<td>30938</td>
<td>14146</td>
<td>6616</td>
</tr>
</tbody>
</table>
Multi-label learning: doubly stochastic

- Binary cross entropy used as loss:
  \[ l(h, y) = \sum_{i=1}^{M} y_i \log(h_i) + (1 - y_i) \log(1 - h_i) \]

- For large label space, taking full derivative of loss is expensive.

- Doubly stochastic w.r.t. examples and label dimensions:
  - In each iteration randomly sample a minibatch \( B \) from dataset
  - For each label \( y \in \mathbb{R}^M \), randomly sample \( k \) dimensions
  - Only take derivatives w.r.t. those dimensions.
  - Reduces complexity from \( O(Mn_h) \) to \( O(kn_h) \)
Multi-label learning: Results with MLPs

**Figure:** Top one accuracy of MLP models with $L$ layers and 128 hidden dimension. Dropout rate of 0.1 is used.
Multi-label learning: Results with MLPs

Figure: MLP models on Delicious/Wiki10 with $L$ layers and 128 hidden dimension. Dropout rate of 0.1 is used.
MNIST classification from a Sequence of Pixels

- MNIST dataset: training data of size 60,000 & testing data of size 10,000.
- Data preprocessing: each $28 \times 28$ image pixels $\xrightarrow{\text{flatten}}$ a vector as input of RNN models for each layer

<table>
<thead>
<tr>
<th>Models</th>
<th>Hidden dimension</th>
<th># parameters</th>
<th>Test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectral RNN (ours)</td>
<td>$256 (m_1, m_2 = 16)$</td>
<td>$\approx 13k$</td>
<td>97.6</td>
</tr>
<tr>
<td>oRNN (Mhammedi et al., 2017)</td>
<td>$256 (m = 16)$</td>
<td>$\approx 11k$</td>
<td>97.2</td>
</tr>
<tr>
<td>RNN (Vorontsov et al., 2017)</td>
<td>128</td>
<td>$\approx 35k$</td>
<td>94.1</td>
</tr>
<tr>
<td>uRNN (Arjovsky et al., 2016)</td>
<td>512</td>
<td>$\approx 16k$</td>
<td>95.1</td>
</tr>
<tr>
<td>RC uRNN (Wisdom et al., 2016)</td>
<td>512</td>
<td>$\approx 16k$</td>
<td>97.5</td>
</tr>
<tr>
<td>FC uRNN (Wisdom et al., 2016)</td>
<td>116</td>
<td>$\approx 16k$</td>
<td>92.8</td>
</tr>
<tr>
<td>factorized RNN (Vorontsov et al., 2017)</td>
<td>128</td>
<td>$\approx 32k$</td>
<td>94.6</td>
</tr>
<tr>
<td>LSTM (Vorontsov et al., 2017)</td>
<td>128</td>
<td>$\approx 64k$</td>
<td>97.3</td>
</tr>
</tbody>
</table>
MNIST classification from a Sequence of Pixels

Figure: RNN models on MNIST
MNIST classification from a Sequence of Pixels

- Data processing: flattened each image pixels to a vector of length 784 as input vector of MLP
- Network architecture: 40 layers with hidden dimension 32 or 30 to 100 layers.

Figure: MLP models on MNIST with $L$ layers $n_h$ hidden dimension
Conclusions

Efficient spectral parameterization of weight matrices in deep networks that:

- allows explicit control over its singular values to eliminate/reduce the exploding/vanishing gradient problem
- no loss of expressive power
- similar time complexity as vanilla RNN
- seems to have better generalization and is easier to train

Promising direction, but lots of work to be done....
References


References


