Stabilizing Gradients for Deep Neural Networks

Inderjit S. Dhillon
UT Austin & Amazon

International Center for Theoretical Sciences (ICTS)
Bangalore, India
Sept 27, 2018

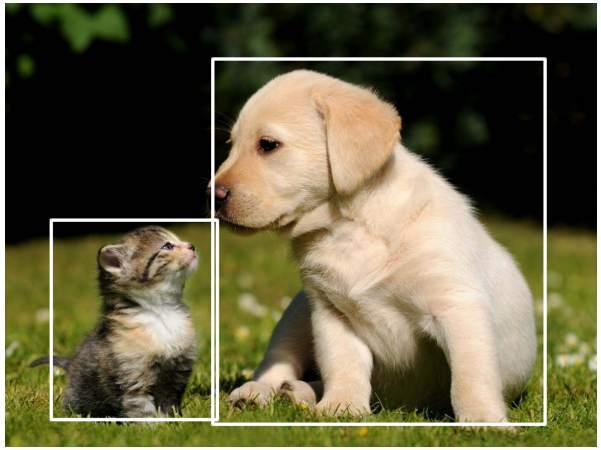
Joint work with Jiong Zhang and Qi Lei (UT Austin)

Outline

Machine Learning 101

- Goal of Supervised Learning: Given training data $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$, predict y for unseen x.
- Form prediction as $y = f(x; \theta)$, where f is parameterized by θ
- From training data, infer the parameters θ using a loss function $L(\mathcal{D}) = \sum_{i=1}^N \ell(f(x_i), y_i)$
- Choice of f and L are crucial components in machine learning — common choices lead to:
  - Linear Regression
  - Logistic Regression
  - Support Vector Machines
  - Kernel Machines
  - Gradient Boosted Decision Trees
  - Neural Networks
  - Deep Learning

Things we can do with Deep Learning



Object Detection

Mastering Go

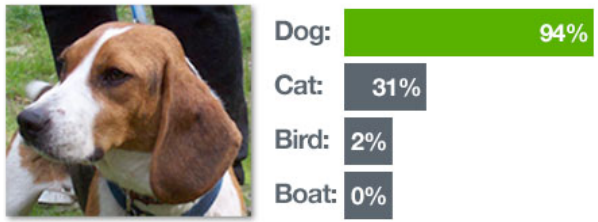


Image classification

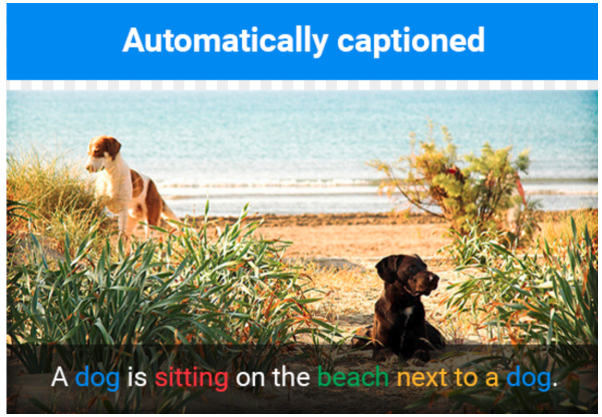
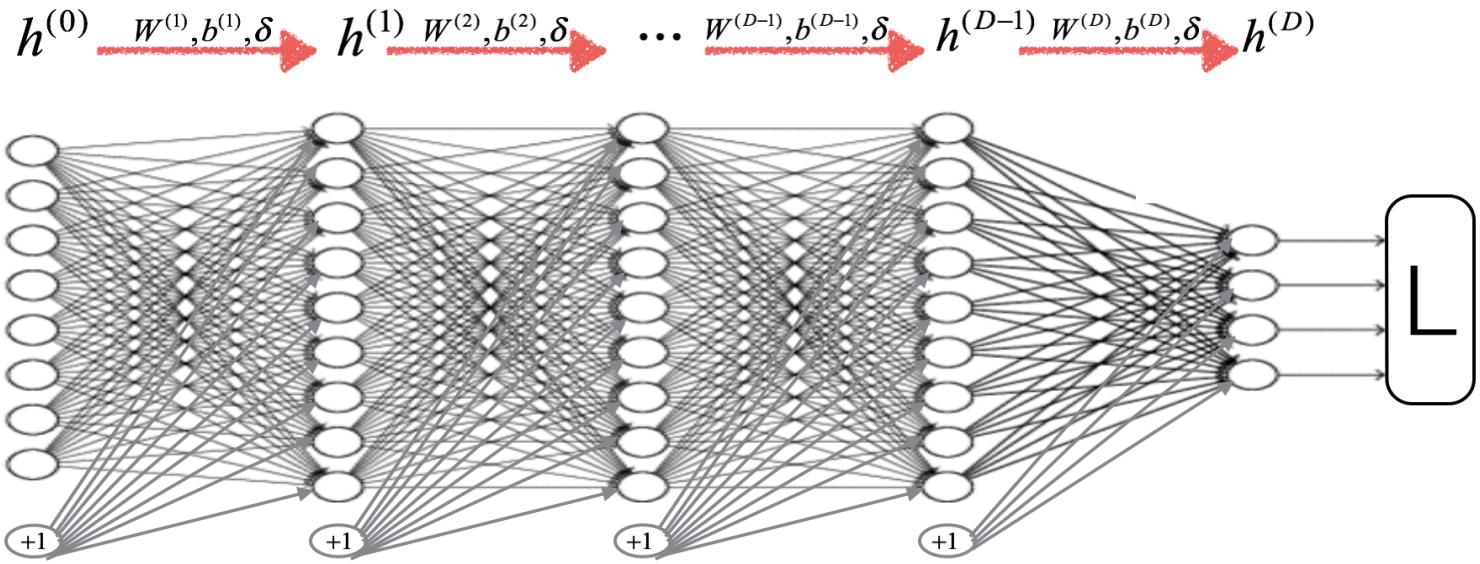


Image captioning

Feed forward network: Multilayer perceptron(MLP)



- D-layer MLP with activation function δ:

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

- $\Theta = \{W^{(t)}, b^{(t)}\}_{t=1}^{D}$ 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:

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

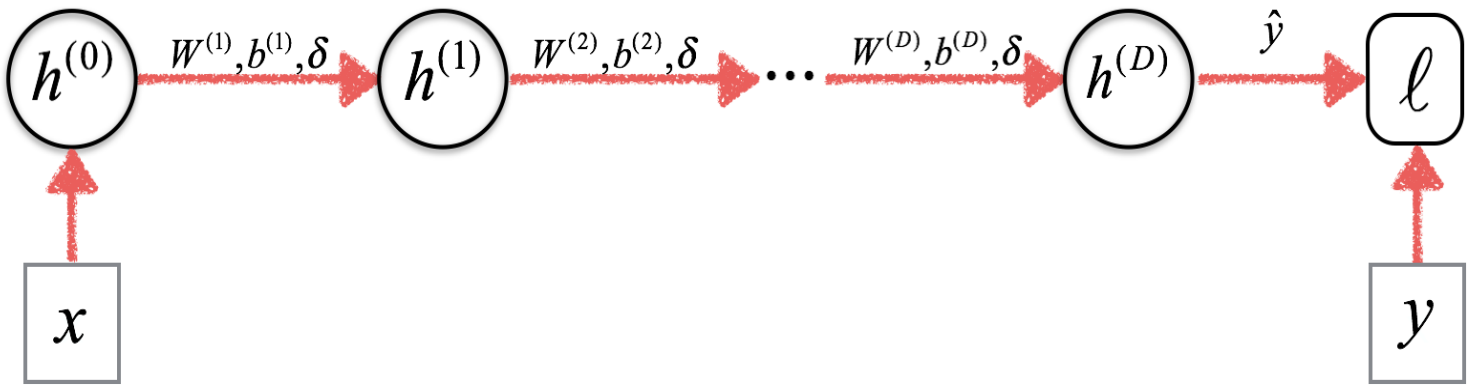
- For regression problems, ℓ could be squared loss:

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

- For classification problems, ℓ could be cross entropy loss:

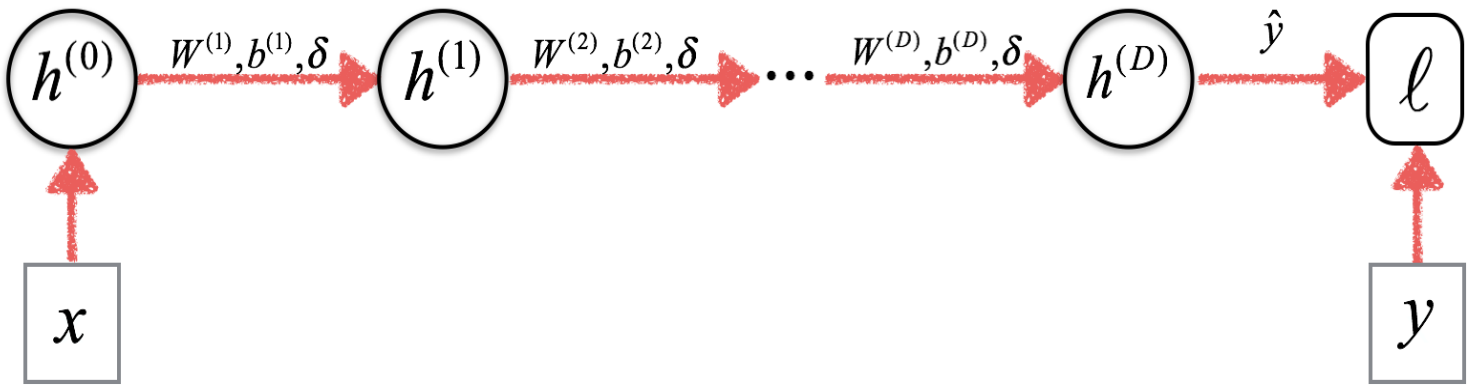
$$\ell(y_i, h^{(D)}) = -\log \left(\frac{\exp(h_{y_i}^{(D)})}{\sum_k \exp(h_k^{(D)})} \right)$$

Loss evaluation: Forward Propagation



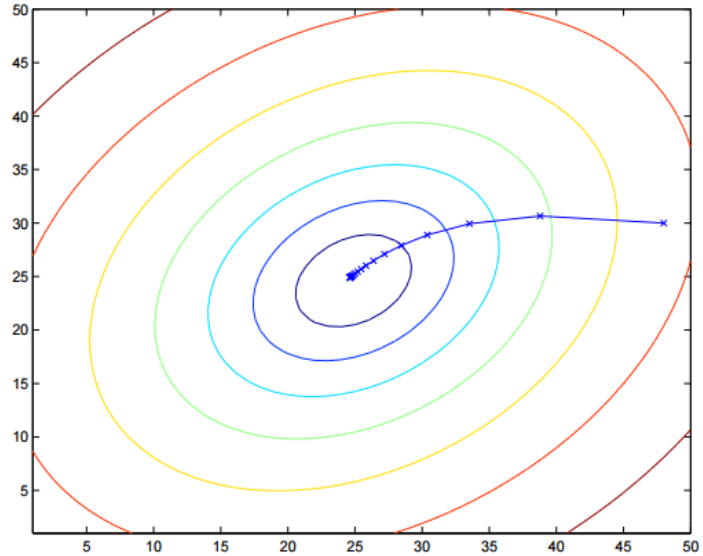
- To evaluate loss ℓ we need to compute $h^{(D)}$. We do this by iteratively evaluating $h^{(t)}$ for $t = 1, 2, \ldots, D$.

Loss evaluation: Forward Propagation



- To evaluate loss ℓ 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)

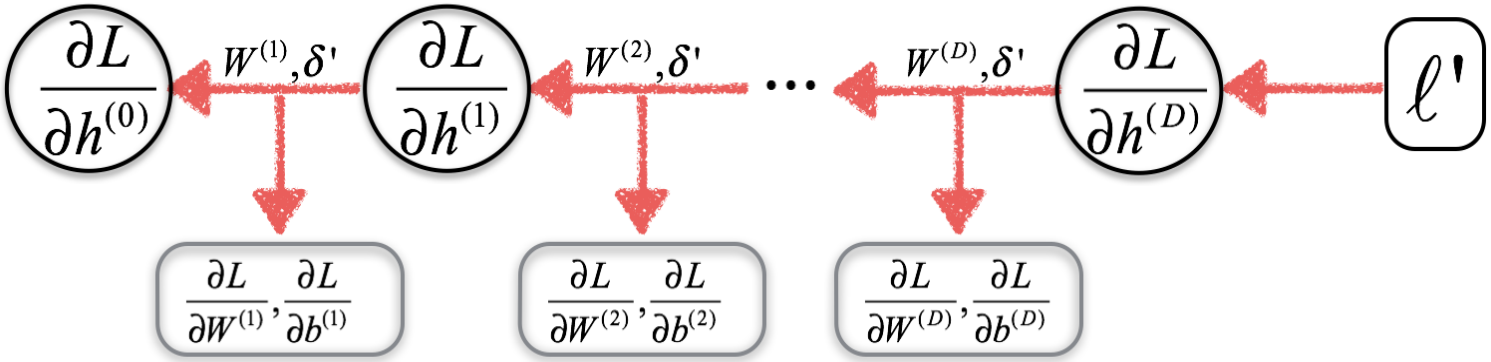
SGD to train a neural network:

- 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:

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

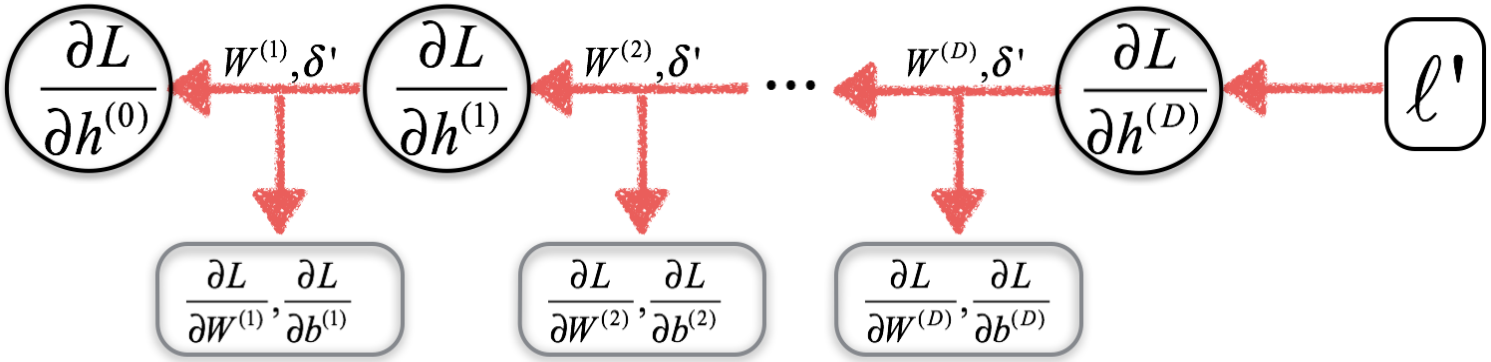
- Step size η is usually selected by line search or heuristics like Adam (Kingma et al. 2014).

Vanishing & Exploding Gradients



- Recall $\ell = \ell(y, h^{(D)})$; $h^{(t)} = \delta(W^{(t)} h^{(t-1)} + b^{(t)}), \forall t \in [D]$

Vanishing & Exploding Gradients



- Recall $\ell = \ell(y, h^{(D)})$; $h^{(t)} = \delta(W^{(t)} h^{(t-1)} + b^{(t)}), \forall t \in [D]$
- By chain rule, derivatives of $(t-1)$-st layer depend on $\frac{\partial \ell}{\partial h^{(t)}}$

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

  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:

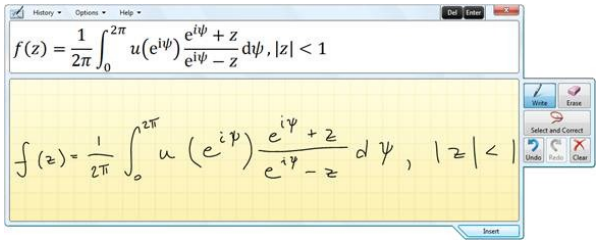
$$\frac{\partial h^{(t)}}{\partial W^{(t)}} = h^{(t-1)} \circ diag(\delta_t'); \frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \left[W^{(t)}\right]^\top diag(\delta_t')$$

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

$$\frac{\partial \ell}{\partial h^{(t)}} \propto \Pi_{i=D}^{t+1} \left[W^{(i)}\right]^\top diag(\delta_i')$$

- Hence, gradients can easily "explode" or "vanish"

Things we can do with Recurrent Neural Networks



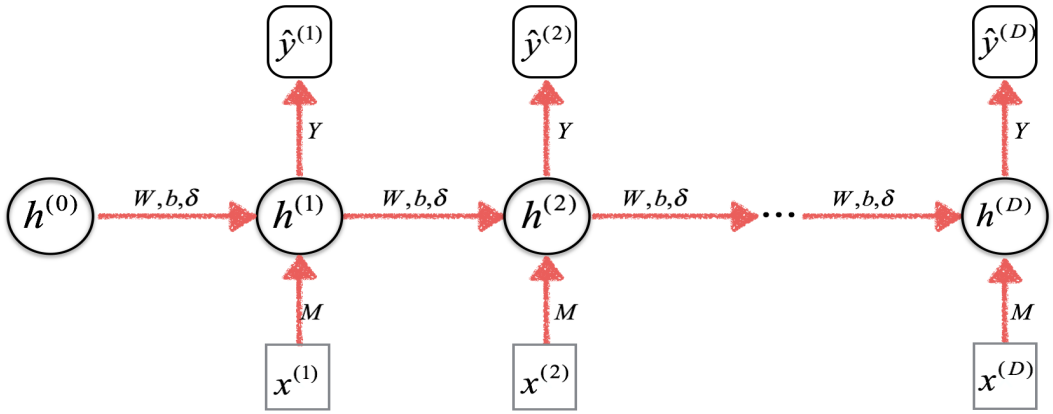
Handwriting Recognition

Translation

Question answering

Speech Recognition

Recurrent Neural Networks (RNN)

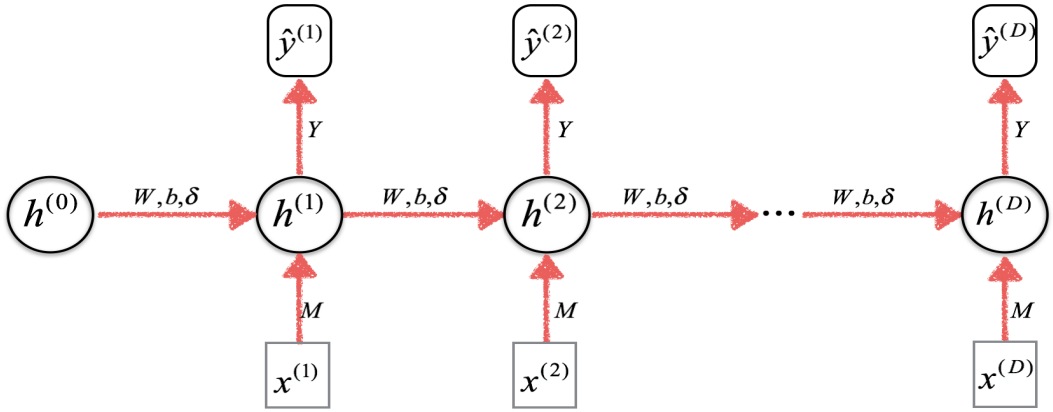


- RNN with activation function δ:

$$h^{(t)} = \delta(Wh^{(t-1)} + Mx^{(t-1)} + b) \tag{1}$$
$$\hat{y}^{(t)} = Yh^{(t)}$$

- 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: $L(X, Y; \Theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \hat{y}(x_i; \Theta))$

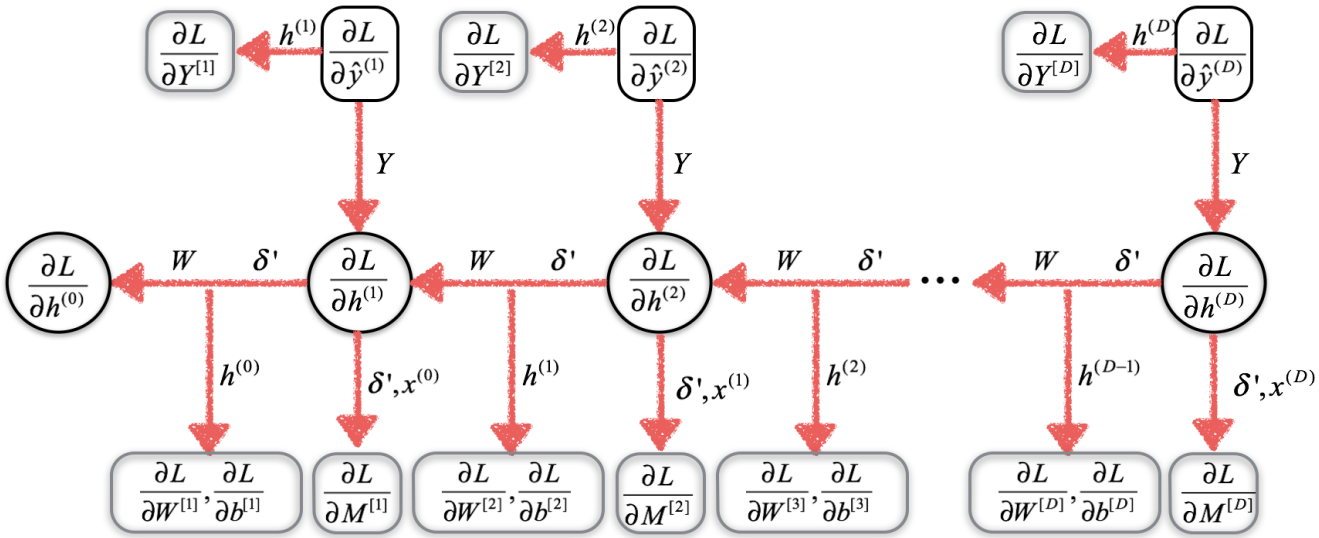
RNN: forward propagation



Forward propagation:

- Evaluate activations
  $h^{(t)} = \delta(Wh^{(t-1)} + Mx^{(t-1)} + b), t = 1, \ldots, D$ layer by layer
- Evaluate $\hat{y}^{(t)} = Yh^{(t)}, t = 1, \ldots, D$ layer by layer

RNN: backward propagation



- Similar to MLP, gradients propagate through layers: $\frac{\partial h^{(t)}}{\partial W} = h^{(t)} \circ diag(\delta_t'); \frac{\partial h^{(t)}}{\partial h^{(t-1)}} = W^\top diag(\delta_t')$
- Backpropagation:

$$\frac{\partial \ell}{\partial h^{(t)}} \propto \Pi_{i=D}^{t+1} W^\top diag(\delta_i')$$

- Hence, gradients can easily "explode" or "vanish"
- Problem: Long-range dependencies cannot be captured

Sigmoid Activation Function

- Sigmoid function $\sigma(z) = 1/(1 + e^{-z})$ 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

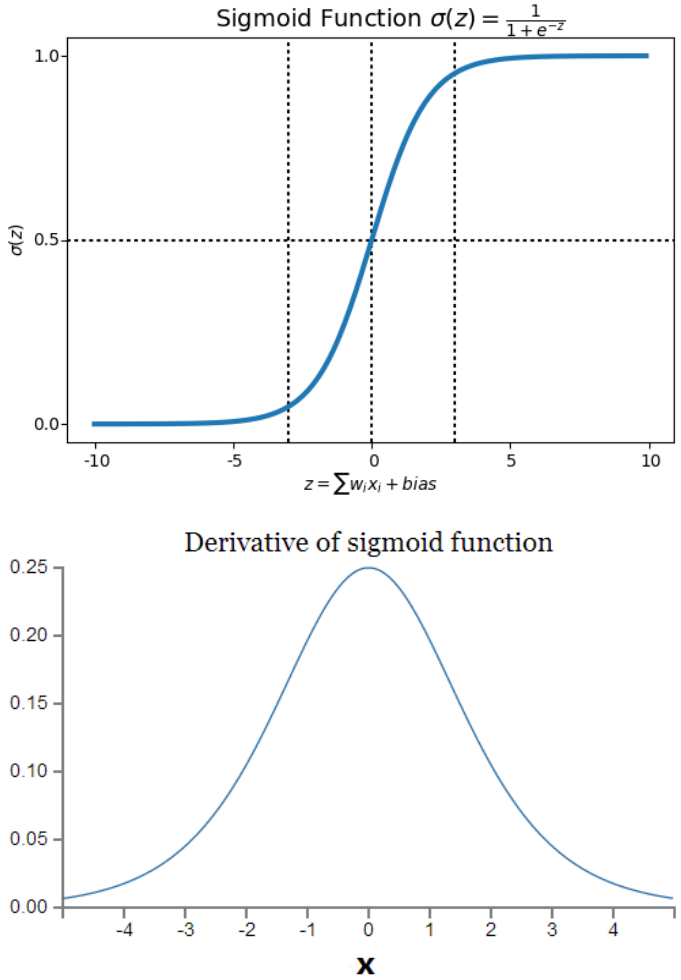


Figure: Sigmoid activation

Activation Functions: Vanishing Gradients

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

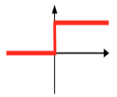
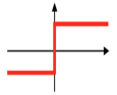
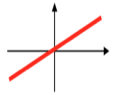
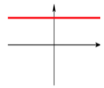
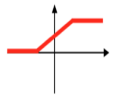
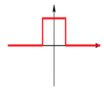
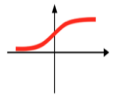
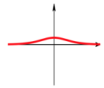
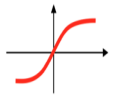
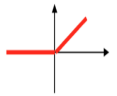
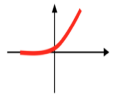
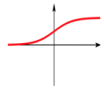
Activation function	Equation	Example	1D Graph	Derivative
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant		
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant		
Linear	$\phi(z) = z$	Adaline, linear regression		
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine		
Logistic (sigmoid)	$\phi(z) = \frac{1}{1+e^{-z}}$	Logistic regression, Multi-layer NN		
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks		
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks		
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks		

Figure: Common activation functions and their derivatives.

Existing Solutions

- Long Short-Term Memory (LSTM):

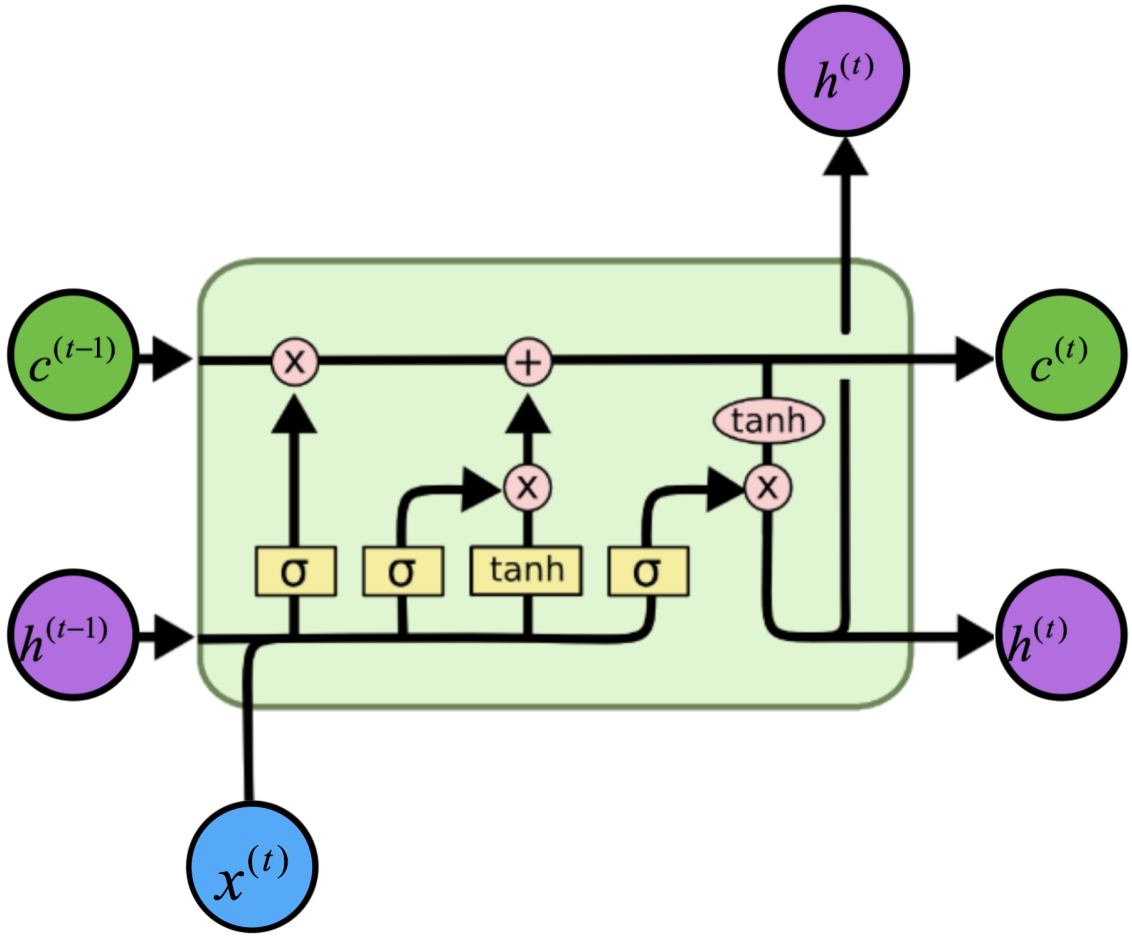
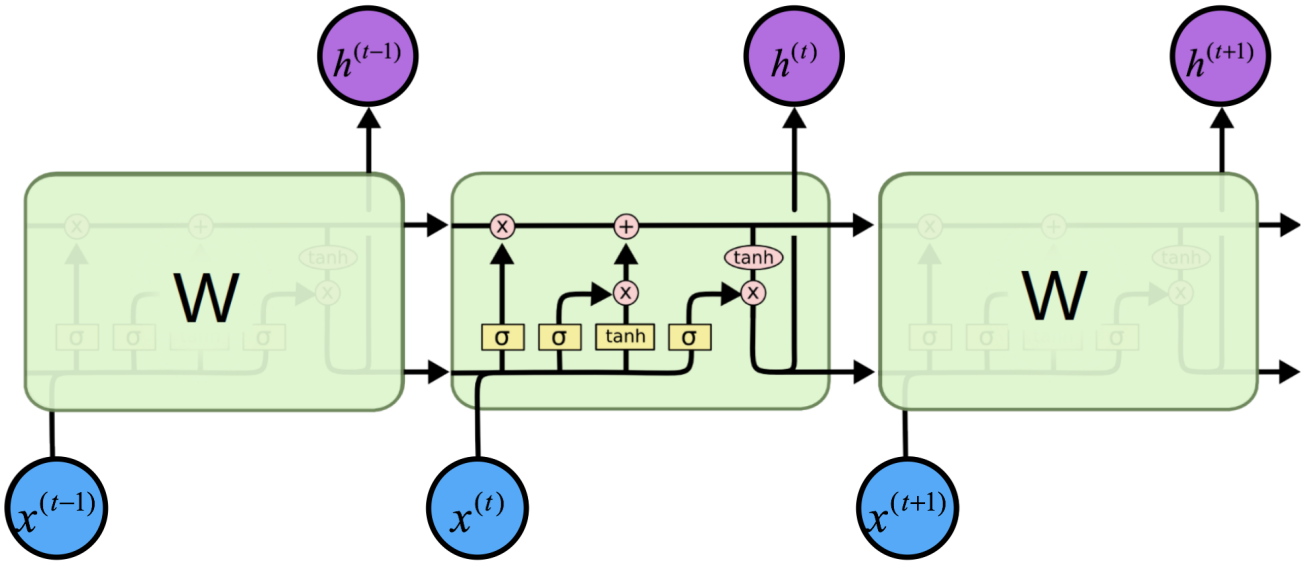


Figure: One node of LSTM. (Colah 2015)

Existing Solutions

- Long Short-Term Memory (LSTM):



Avoids long-term dependency problems by "forget-gate layers"

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

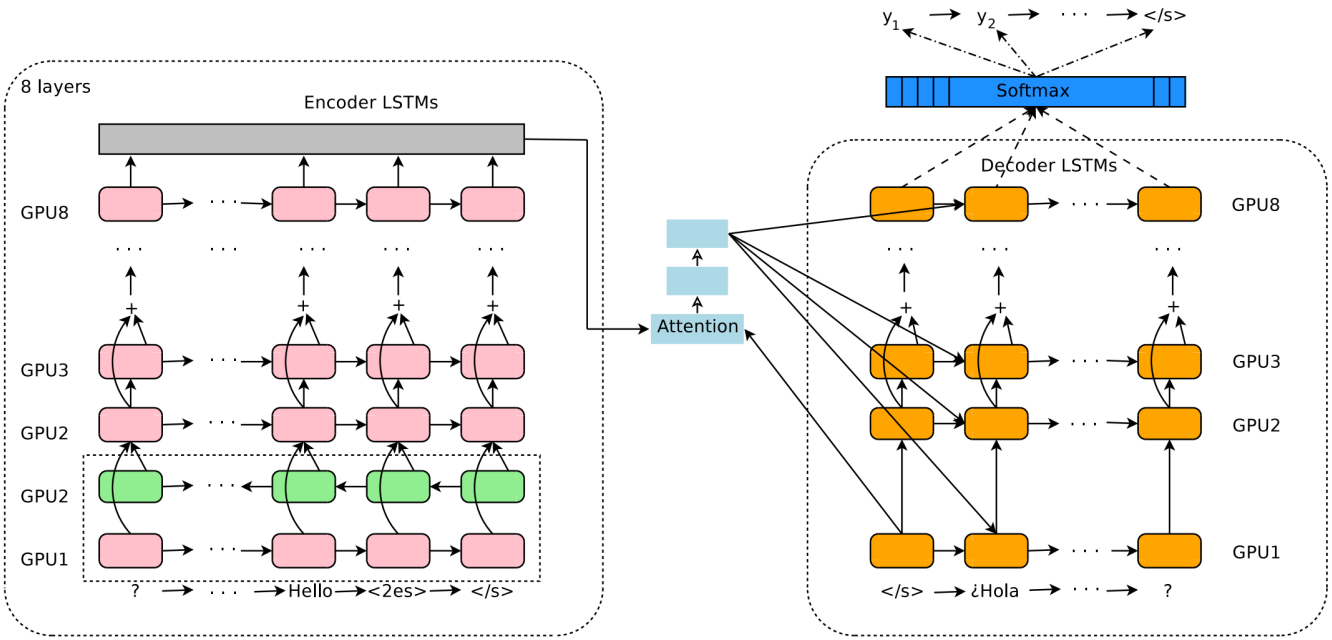
Fernanda Viégas, Martin Wattenberg, Greg Corrado,
Macduff Hughes, Jeffrey Dean

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

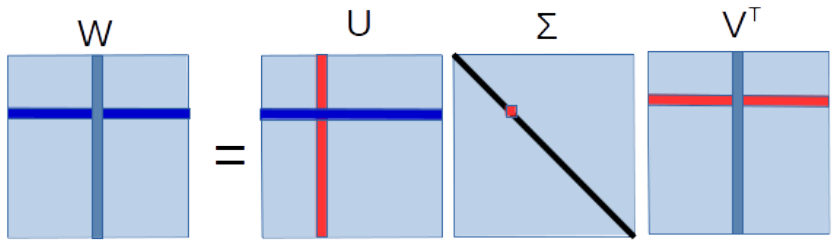
- Eight-layer bidirectional LSTM with Attention (Johnson et. al., 2016)
- For each language pair: 1024 nodes (hidden dimension), 8 LSTM layers with a total of 255M parameters.
- Total training time is on the scale of weeks, on up to 100 GPUs.



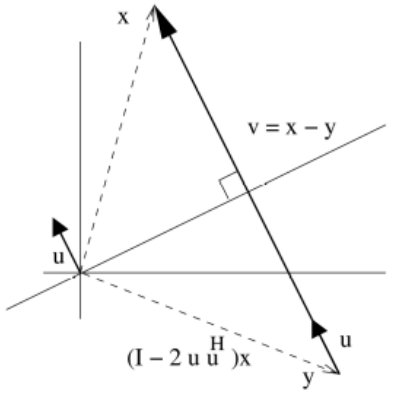
Outline

Spectral Parameterization

- An illustration of the parameterization process:



A Householder reflector:



Represent U as:

$$\left(I_n - 2\frac{u_n u_n^\top}{\|u_n\|^2} \right) \cdot \begin{pmatrix} 1 & \\ & I_{n-1} - 2\frac{u_{n-1} u_{n-1}^\top}{\|u_{n-1}\|^2} \end{pmatrix} \cdots \begin{pmatrix} I_{n-k_1} & \\ & I_{k_1} - 2\frac{u_{k_1} u_{k_1}^\top}{\|u_{k_1}\|^2} \end{pmatrix}$$

Spectral Parameterization

- Maintain SVD of transition matrix:

$$W = U\Sigma V^\top$$

Spectral Parameterization

- Maintain SVD of transition matrix:

$$W = U\Sigma V^\top$$

- Parameterize U, V by products of Householder reflectors:

$$\mathcal{H}_k(u) = \begin{cases} \begin{pmatrix} I_{n-k} & \\ & I_k - 2\frac{uu^\top}{\|u\|^2} \end{pmatrix} & , \quad u \neq \mathbf{0} \\ I_n & , \quad \text{otherwise.} \end{cases}$$

Spectral Parameterization

- Maintain SVD of transition matrix:

$$W = U\Sigma V^\top$$

- Parameterize U, V by products of Householder reflectors:

$$\mathcal{H}_k(u) = \begin{cases} \begin{pmatrix} I_{n-k} & \\ & I_k - 2\frac{uu^\top}{\|u\|^2} \end{pmatrix} & , \quad u \neq \mathbf{0} \\ I_n & , \quad \text{otherwise.} \end{cases}$$

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

Spectral Parameterization

- Maintain SVD of transition matrix:

$$W = U\Sigma V^\top$$

- Parameterize U, V by products of Householder reflectors:

$$\mathcal{H}_k(u) = \begin{cases} \begin{pmatrix} I_{n-k} & \\ & I_k - 2\frac{uu^\top}{\|u\|^2} \end{pmatrix} & , \quad u \neq \mathbf{0} \\ I_n & , \quad \text{otherwise.} \end{cases}$$

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

Spectral Parameterization

Proposed parametrization:

$$\begin{aligned}\mathcal{M}_{k_1,k_2} :&(\mathbb{R}^{k_1} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^{k_2} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^n) \mapsto \mathbb{R}^{n \times n} \\ &\left(\{u_i\}_{i=k_1}^n\right), \left(\{v_i\}_{i=k_2}^n\right), (\sigma) \mapsto \\ &\underbrace{\mathcal{H}_n(u_n)...\mathcal{H}_{k_1}(u_{k_1})}_{U} \underbrace{diag(\sigma)}_{\Sigma} \underbrace{\mathcal{H}_{k_2}(v_{k_2})...\mathcal{H}_n(v_n)}_{V^\top}. \end{aligned} \tag{2}$$

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

Spectral Parameterization

Proposed parametrization:

$$\begin{aligned}\mathcal{M}_{k_1,k_2} :&(\mathbb{R}^{k_1} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^{k_2} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^n) \mapsto \mathbb{R}^{n \times n} \\ &\left(\{u_i\}_{i=k_1}^n\right), \left(\{v_i\}_{i=k_2}^n\right), (\sigma) \mapsto \\ &\underbrace{\mathcal{H}_n(u_n)...\mathcal{H}_{k_1}(u_{k_1})}_{U} \underbrace{diag(\sigma)}_{\Sigma} \underbrace{\mathcal{H}_{k_2}(v_{k_2})...\mathcal{H}_n(v_n)}_{V^\top}. \end{aligned} \tag{2}$$

- **Singular values are explicit**: $\mathcal{M}_{k_1,k_2}(\{u_i\}_{i=k_1}^n, \{v_i\}_{i=k_2}^n, \sigma)$ is an $n \times n$ real matrix with singular values σ.
- **Full expressivity**: The image of $\mathcal{M}_{1,1}$ is the set of $n \times n$ real matrices.

Spectral Parameterization

Proposed parametrization:

$$\begin{aligned}\mathcal{M}_{k_1,k_2} :&(\mathbb{R}^{k_1} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^{k_2} \times ... \times \mathbb{R}^n) \times (\mathbb{R}^n) \mapsto \mathbb{R}^{n \times n} \\ &\left(\{u_i\}_{i=k_1}^n\right), \left(\{v_i\}_{i=k_2}^n\right), (\sigma) \mapsto \\ &\underbrace{\mathcal{H}_n(u_n)...\mathcal{H}_{k_1}(u_{k_1})}_{U} \underbrace{diag(\sigma)}_{\Sigma} \underbrace{\mathcal{H}_{k_2}(v_{k_2})...\mathcal{H}_n(v_n)}_{V^\top}. \end{aligned} \tag{2}$$

- **Singular values are explicit**: $\mathcal{M}_{k_1,k_2}(\{u_i\}_{i=k_1}^n, \{v_i\}_{i=k_2}^n, \sigma)$ is an $n \times n$ real matrix with singular values σ.
- **Full expressivity**: The image of $\mathcal{M}_{1,1}$ is the set of $n \times n$ real matrices.
- **Orthogonal expressivity**: The image of $\mathcal{M}_{k_1,k_2}$ covers the set of $n \times n$ orthogonal matrices if $k_1 + k_2 \leq n + 2$.

Outline

Spectral RNN: RNN with SVD parameterization

- 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):

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

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

Outline

Direct conclusions from the gradients

For Spectral RNN, recall Gradient for activations is:

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

- 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:

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

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 $\Longrightarrow$ robustness in predictions (Cisse et al. 2017)
  - Spectral MLP guarantees near orthogonal weight matrix, namely tight frames

Outline

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

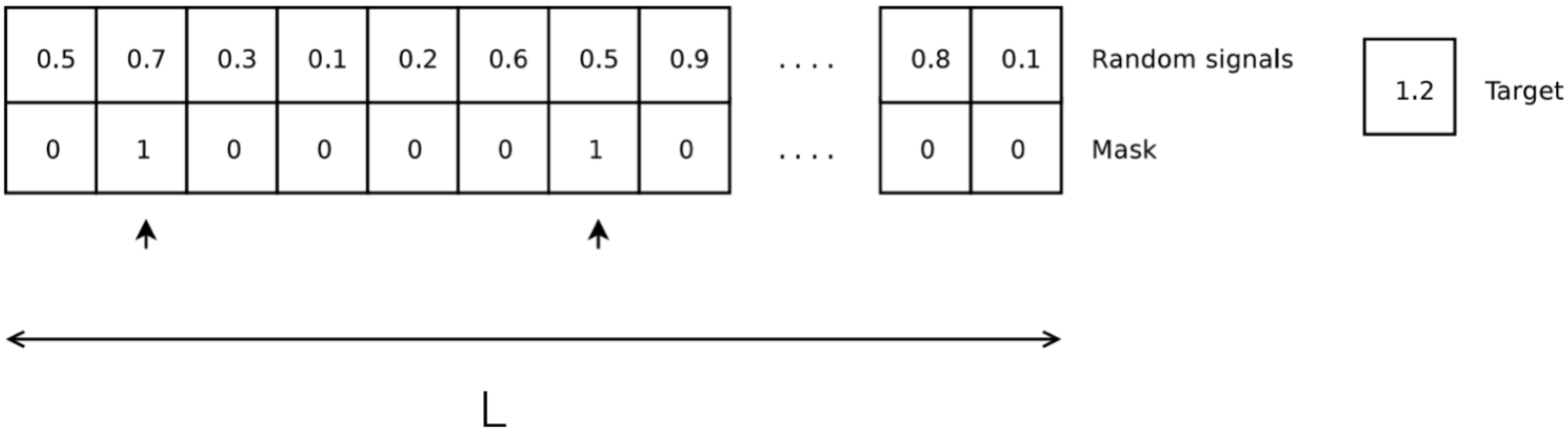


Figure: The Addition Task from (Le et. al 2015).

Addition Task: Results

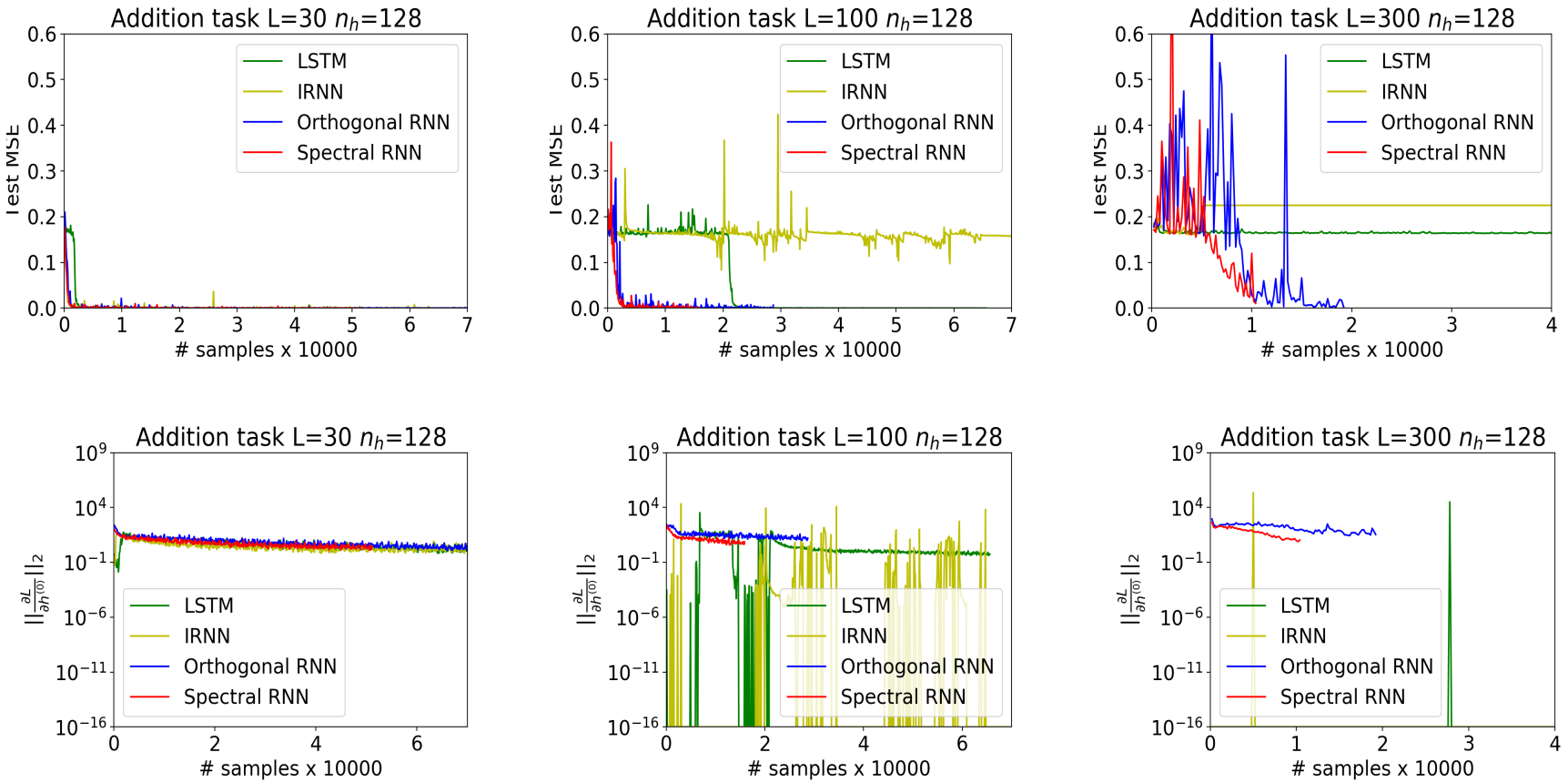


Figure: RNNs on addition task with L layers & n_h hidden dimension.

Outline

Speech recognition task

- Google speech command data set
  - 65K training examples, each one a WAVE audio sampled to be a vector of 3920 length
  - Twelve different labels: silence, unknown, “yes”, “no”, “up”, “down”, “left”, “right”, “on”, “off”, “stop”, or “go”
- Data preprocessing: Each instance is split into L pieces and then fed into the RNN models piece by piece.

Speech recognition task

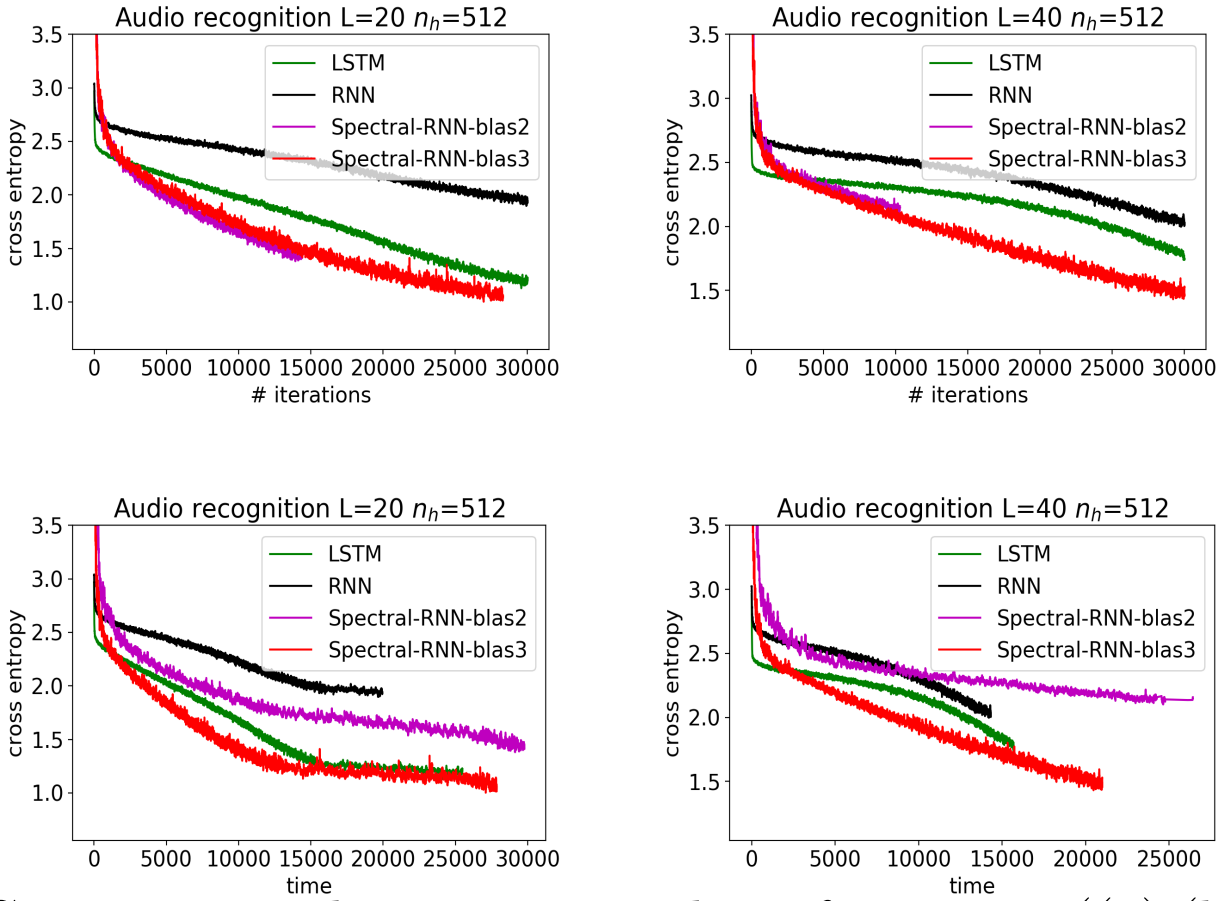


Figure: Cross entropy loss w.r.t. number of iteration ((a),(b)) and time ((c),(d)) with temporal length $L = 20$ and 40

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

[1] J. Zhang, Q. Lei, I. S. Dhillon. *Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parametrization*. ICML, 2018.

[2] R. Pascanu, T. Mikolov, and Y. Bengio. *On the difficulty of training recurrent neural networks*. In ICML (2013).

[3] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, and N. Thorat,*Google's multilingual neural machine translation system: enabling zero-shot translation*. arXiv preprint arXiv:1611.04558. (2016).

[4] P. Bartlett, D. J. Foster, and M. Telgarsky. *Spectrally-normalized margin bounds for neural networks*. arXiv preprint arXiv:1706.08498, (2017).