Introduction to Neural Networks and Deep Learning

Recurrent Neural Networks

Andres Mendez-Vazquez

August 22, 2020

Outline

Outline

In 1987 Robinson and Fallside [2]

At Cambridge University Engineering Department

- They proposed a new type of neural network based on Linear Control Theory

They took the work of Jacobs, 1974 on dynamic nets [1]

$$s_{t+1} = As_t + Bx_t$$
$$y_t = Cs_t$$

In 1987 Robinson and Fallside [2]

At Cambridge University Engineering Department

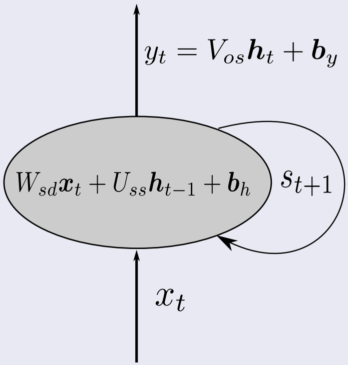
- They proposed a new type of neural network based on Linear Control Theory

They took the work of Jacobs, 1974 on dynamic nets [1]

$$
\begin{aligned}
s_{t+1} &= As_t + Bx_t \\
y_t &= Cs_t
\end{aligned}
$$

Example of this unit

We have



Furthermore

Jordan Proposed a simple recurrent network

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right) \\
\boldsymbol{y}_t &= \sigma_s \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_o \right)
\end{aligned}
$$

- h_t is a hidden state layer of dimension h.
- y_t is the output vector of dimension s.
- W, U, V parameter matrices.
- b_h and b_o bias for the linear part.
- σ_h and σ_s are activation functions.

Furthermore

Jordan Proposed a simple recurrent network

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right) \\
\boldsymbol{y}_t &= \sigma_s \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_o \right)
\end{aligned}
$$

Where

1. $\boldsymbol{x}_t$ is an input of dimension d.
2. h_t is a hidden state layer of dimension h.
3. y_t is the output vector of dimension s.
4. W, U, V parameter matrices.
5. b_h and b_o bias for the linear part.
6. σ_h and σ_s are activation functions.

Furthermore

Jordan Proposed a simple recurrent network

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right)$$
$$\boldsymbol{y}_t = \sigma_s \left(V_{os}\boldsymbol{h}_t + \boldsymbol{b}_o \right)$$

Where

1. $\boldsymbol{x}_t$ is an input of dimension d.
2. $\boldsymbol{h}_t$ is a hidden state layer of dimension h.
3. $\boldsymbol{y}_t$ is the output vector of dimension s.
4. W, U, V parameter matrices.
5. b_h and b_o bias for the linear part.
6. σ_h and σ_s are activation functions.

Furthermore

Jordan Proposed a simple recurrent network

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right) \\
\boldsymbol{y}_t &= \sigma_s \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_o \right)
\end{aligned}
$$

Where

1. $\boldsymbol{x}_t$ is an input of dimension d.
2. $\boldsymbol{h}_t$ is a hidden state layer of dimension h.
3. $\boldsymbol{y}_t$ is the output vector of dimension s.
4. W, U, V parameter matrices.
5. b_h and b_o bias for the linear part.
6. σ_h and σ_s are activation functions.

Furthermore

Jordan Proposed a simple recurrent network

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right)$$
$$\boldsymbol{y}_t = \sigma_s \left(V_{os}\boldsymbol{h}_t + \boldsymbol{b}_o \right)$$

Where

1. $\boldsymbol{x}_t$ is an input of dimension d.
2. $\boldsymbol{h}_t$ is a hidden state layer of dimension h.
3. $\boldsymbol{y}_t$ is the output vector of dimension s.
4. W, U, V parameter matrices.
5. b_h and b_o bias for the linear part.
6. σ_h and σ_s are activation functions.

Furthermore

Jordan Proposed a simple recurrent network

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right) \\
\boldsymbol{y}_t &= \sigma_s \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_o \right)
\end{aligned}
$$

Where

1. $\boldsymbol{x}_t$ is an input of dimension d.
2. $\boldsymbol{h}_t$ is a hidden state layer of dimension h.
3. $\boldsymbol{y}_t$ is the output vector of dimension s.
4. W, U, V parameter matrices.
5. b_h and b_o bias for the linear part.
6. σ_h and σ_s are activation functions.

Furthermore

Jordan Proposed a simple recurrent network

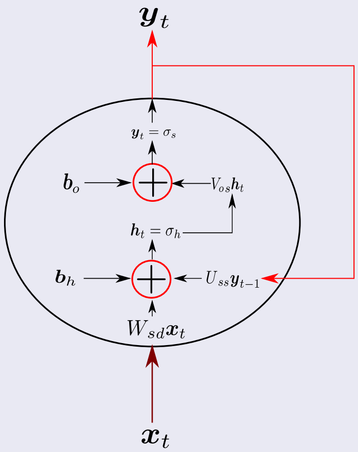
$$\begin{aligned} \boldsymbol{h}_t &= \sigma_h\left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h\right) \\ \boldsymbol{y}_t &= \sigma_s\left(V_{os}\boldsymbol{h}_t + \boldsymbol{b}_o\right) \end{aligned}$$

Where

1. $\boldsymbol{x}_t$ is an input of dimension d.
2. $\boldsymbol{h}_t$ is a hidden state layer of dimension h.
3. $\boldsymbol{y}_t$ is the output vector of dimension s.
4. W, U, V parameter matrices.
5. b_h and b_o bias for the linear part.
6. σ_h and σ_s are activation functions.

Graphically

We have



What were they used for?

Robinson and Fallside

- As with Hidden Markov Models, they were proposed for Speech Coding

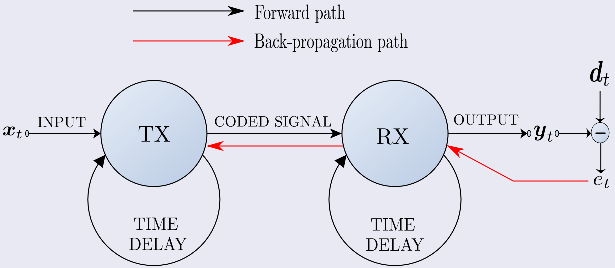
They proposed the following architecture

What were they used for?

Robinson and Fallside

- As with Hidden Markov Models, they were proposed for Speech Coding

They proposed the following architecture



Outline

Based on the State-Space Model

Basically, a linear system

- Based in a state-determined system model

Definition

- A mathematical description of the system in terms of a minimum set of variables $x_i(t)$, $i = 1, ..., n$, together with knowledge of those variables at an initial time t_0 and the system inputs for time $t \geq t_0$, are sufficient to predict the future system state and outputs for all time $t > t_0$.

Based on the State-Space Model

Basically, a linear system

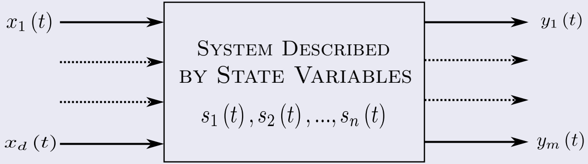
- Based in a state-determined system model

Definition

- A mathematical description of the system in terms of a minimum set of variables $x_i(t)$, $i = 1, ..., n$, together with knowledge of those variables at an initial time t_0 and the system inputs for time $t \geq t_0$, are sufficient to predict the future system state and outputs for all time $t > t_0$.

Therefore

We have a system as a block

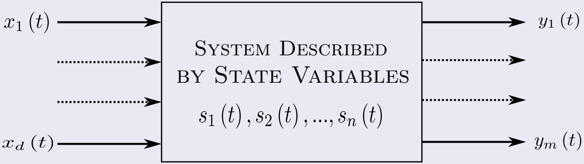


This can be expressed as a state equations

$$\begin{aligned}
\dot{s}_1 &= f_1(\boldsymbol{x}, \boldsymbol{s}, t) \\
\dot{s}_2 &= f_2(\boldsymbol{x}, \boldsymbol{s}, t) \\
\cdots &= \cdots \\
\dot{s}_n &= f_n(\boldsymbol{x}, \boldsymbol{s}, t)
\end{aligned}$$

Therefore

We have a system as a block



This can be expressed as a state equations

$$
\begin{aligned}
\dot{s}_1 &= f_1(\boldsymbol{x}, \boldsymbol{s}, t) \\
\dot{s}_2 &= f_2(\boldsymbol{x}, \boldsymbol{s}, t) \\
\cdots &= \cdots \\
\dot{s}_n &= f_n(\boldsymbol{x}, \boldsymbol{s}, t)
\end{aligned}
$$

Using Vector Notation

Assuming that we have a linear system and time invariant

- Time-Invariant $\bowtie$ $x(t+\delta)$ directly equates $y(t+\delta)$, for example

$$\alpha x(t+\delta) + \beta = y(t+\delta)$$

Therefore, using this idea

$$\dot{s}_i = a_{i1}x_1(t) + \ldots + a_{id}x_d(t) + b_{11}s_1(t) + \ldots + b_{1n}s_n(t)$$

Or in Matrix form

$$y(t) = Ax(t) + Bs(t)$$

Using Vector Notation

Assuming that we have a linear system and time invariant

- Time-Invariant $\bowtie$ $x(t+\delta)$ directly equates $y(t+\delta)$, for example

$$\alpha x(t+\delta) + \beta = y(t+\delta)$$

Therefore, using this idea

$$\dot{s}_i = a_{i1}x_1(t) + ... + a_{id}x_d(t) + b_{11}s_1(t) + ... + b_{1n}s_n(t)$$

Or in Matrix form

$$y(t) = Ax(t) + Bs(t)$$

Using Vector Notation

Assuming that we have a linear system and time invariant

- Time-Invariant $\bowtie$ $x(t+\delta)$ directly equates $y(t+\delta)$, for example

$$\alpha x(t+\delta) + \beta = y(t+\delta)$$

Therefore, using this idea

$$\dot{s}_i = a_{i1}x_1(t) + ... + a_{id}x_d(t) + b_{11}s_1(t) + ... + b_{1n}s_n(t)$$

Or in Matrix form

$$\boldsymbol{y}(t) = A\boldsymbol{x}(t) + B\boldsymbol{s}(t)$$

Then, the discretized version

We introduce an update for the state part

$$\boldsymbol{y}(t) = A\boldsymbol{x}(t) + B\boldsymbol{s}(t)$$
$$\dot{\boldsymbol{s}}(t) = C\boldsymbol{s}(t)$$

Or our discrete step equitations

$$\boldsymbol{y}(t) = A\boldsymbol{x}(t) + B\boldsymbol{s}(t)$$
$$\boldsymbol{s}(t+1) = C\boldsymbol{s}(t)$$

Then, the discretized version

We introduce an update for the state part

$$\boldsymbol{y}(t) = A\boldsymbol{x}(t) + B\boldsymbol{s}(t)$$
$$\dot{\boldsymbol{s}}(t) = C\boldsymbol{s}(t)$$

Or our discrete step equitations

$$\boldsymbol{y}(t) = A\boldsymbol{x}(t) + B\boldsymbol{s}(t)$$
$$\boldsymbol{s}(t+1) = C\boldsymbol{s}(t)$$

Outline

The Elman Network

In Elman's Equations

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right) \\
\boldsymbol{y}_t &= \sigma_y \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_y \right)
\end{aligned}
$$

We noticed something different from the linear recurrent system

- The use of activation functions to introduce the concept of non-linearity

The Elman Network

In Elman's Equations

$$
\begin{aligned}
\boldsymbol{h}_t &= \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right) \\
\boldsymbol{y}_t &= \sigma_y \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_y \right)
\end{aligned}
$$

We noticed something different from the linear recurrent system

- The use of activation functions to introduce the concept of non-linearity

Explanation

We have the following

1. The input $\boldsymbol{x}_t$ is coded by W_{sd}

$$W_{sd}\boldsymbol{x}_t$$

2. An state is generated by using the codified version of the input plus a previous state $\boldsymbol{h}_{t-1}$

$$\boldsymbol{h}_t = \sigma_h\left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h\right)$$

3. The output is generated using the new state $\boldsymbol{h}_t$

$$\boldsymbol{y}_t = \sigma_y\left(V_{os}\boldsymbol{h}_t + \boldsymbol{b}_y\right)$$

Explanation

We have the following

1. The input $\boldsymbol{x}_t$ is coded by W_{sd}

$$W_{sd}\boldsymbol{x}_t$$

2. An state is generated by using the codified version of the input plus a previous state $\boldsymbol{h}_{t-1}$

$$\boldsymbol{h}_t = \sigma_h\left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h\right)$$

3. The output is generated using the new state h_t

$$y_t = \sigma_y\left(V_{os}h_t + b_y\right)$$

Explanation

We have the following

1. The input $\boldsymbol{x}_t$ is coded by W_{sd}

$$W_{sd}\boldsymbol{x}_t$$

2. An state is generated by using the codified version of the input plus a previous state $\boldsymbol{h}_{t-1}$

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right)$$

3. The output is generated using the new state $\boldsymbol{h}_t$

$$\boldsymbol{y}_t = \sigma_y \left(V_{os}\boldsymbol{h}_t + \boldsymbol{b}_y \right)$$

Outline

We need to introduce the concept of cost function

Which as always

- It needs to comply with two properties

The cost function L must be able to be written as an average

$$L = \frac{1}{N} \sum_{x \in \mathcal{X}} C_x$$

over the cost individual cost functions C_x

This allow to apply different optimization techniques as

- Minbatch
- Stochastic Gradient Descent
- etc

We need to introduce the concept of cost function

Which as always

- It needs to comply with two properties

The cost function L must be able to be written as an average

$$L = \frac{1}{N} \sum_{x \in \mathcal{X}} C_x$$

over the cost individual cost functions C_x

This allow to apply different optimization techniques as

- Minbatch
- Stochastic Gradient Descent
- etc

We need to introduce the concept of cost function

Which as always

- It needs to comply with two properties

The cost function L must be able to be written as an average

$$L = \frac{1}{N} \sum_{x \in \mathcal{X}} C_x$$

over the cost individual cost functions C_x

This allow to apply different optimization techniques as

- Minbatch
- Stochastic Gradient Descent
- etc

Furthermore

Non dependency

- The cost function L must not be dependent on any activation values of a neural network besides the output values.

If we cannot assure this

- If not Backpropagation becomes too unstable or too complex to solve. For example

$$L = \frac{1}{N} \sum_{t=0}^{N} \left[y_t + h_t - z_t \right]^2$$

  - This gives two entry points to the network.

Furthermore

Non dependency

- The cost function L must not be dependent on any activation values of a neural network besides the output values.

If we cannot assure this

- If not Backpropagation becomes too unstable or too complex to solve. For example

$$L = \frac{1}{N}\sum_{t=0}^{N}\left[y_t + h_t - z_t\right]^2$$

  - This gives two entry points to the network.

A List of Cost Functions

The Average Quadratic Cost

$$L = \frac{1}{N}\sum_{t=0}^{N}\left[y_t - z_t\right]^2$$

- Where y_t is the output of the network and z_t is the ground truth of the output.

Here, we are interpolating functions

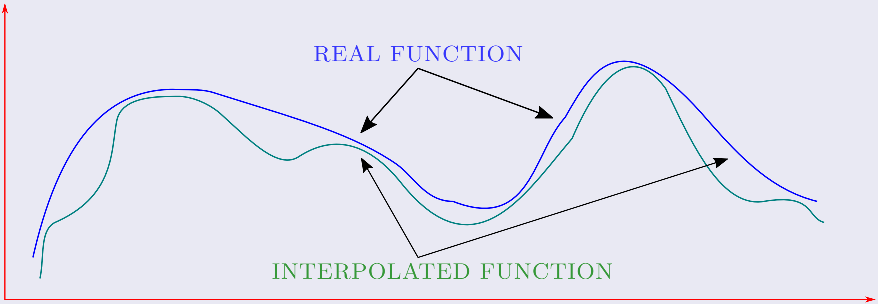
A List of Cost Functions

The Average Quadratic Cost

$$L = \frac{1}{N}\sum_{t=0}^{N}\left[y_t - z_t\right]^2$$

- Where y_t is the output of the network and z_t is the ground truth of the output.

Here, we are interpolating functions



Cross-Entropy cost

First, the Loss Function

$$L = -\sum_{i=1}^{C} z_i \log(y_i)$$

- Where y_i is the output and z_i is the ground truth for the class estimation.

Why $y_i \log(z_i)$?

- We can imagine a sequence of class probabilities $y_1, y_2, \ldots, y_m$ and the likelihood of the data and the model

$$P[data|model] = y_1^{k_1} y_2^{k_2} \cdots y_m^{k_m}$$

Cross-Entropy cost

First, the Loss Function

$$L = -\sum_{i=1}^{C} z_i \log(y_i)$$

- Where y_i is the output and z_i is the ground truth for the class estimation.

Why $y_i \log(z_i)$?

- We can imagine a sequence of class probabilities $y_1, y_2, ..., y_m$ and the likelihood of the data and the model

$$P[data|model] = y_1^{k_1} y_2^{k_2} \cdots y_m^{k_n}$$

Then

Taking the logarithm and multiplying by -1

$$-\log P\left[data|model\right] = -\sum_{i=1}^{C} k_i \log y_i$$

Then, dividing by the total number of samples

$$-\frac{1}{N}\log P\left[data|model\right] = -\sum_{i=1}^{C} \frac{k_i}{N} \log y_i = -\sum_{i=1}^{C} z_i \log y_i$$

Then

Taking the logarithm and multiplying by -1

$$-\log P\left[data|model\right] = -\sum_{i=1}^{C} k_i \log y_i$$

Then, dividing by the total number of samples

$$-\frac{1}{N}\log P\left[data|model\right] = -\sum_{i=1}^{C} \frac{k_i}{N} \log y_i = -\sum_{i=1}^{C} z_i \log y_i$$

Then

In information theory, The Kraft–McMillan theorem

- It establishes that any directly decodable coding scheme for coding a message to identify one value $x_i \in \{x_1, x_2, ..., x_n\}$

It can be seen as representing an implicit probability distribution over $\{x_1, x_2, ..., x_n\}$

$$q(x_i) = \left(\frac{1}{2}\right)^{l_i}$$

- Where l_i is the length of the code for x_i

Then

In information theory, The Kraft–McMillan theorem

- It establishes that any directly decodable coding scheme for coding a message to identify one value $x_i \in \{x_1, x_2, ..., x_n\}$

It can be seen as representing an implicit probability distribution over $\{x_1, x_2, ..., x_n\}$

$$q(x_i) = \left(\frac{1}{2}\right)^{l_i}$$

- Where l_i is the length of the code for x_i

Now

We have that

- Cross entropy can be interpreted as the expected message-length per datum when a wrong distribution q is assumed while the data actually follows a distribution p.

The expected message-length under the true distribution p is

$$\begin{aligned} E_p[l] &= -E_p\left[\frac{\ln q(x)}{\ln 2}\right] \\ &= -E_p\left[\log_2 q(x)\right] \\ &= -\sum_{x_i} p(x_i) \log_2 q(x) \\ &= H(p, q) \end{aligned}$$

Now

We have that

- Cross entropy can be interpreted as the expected message-length per datum when a wrong distribution q is assumed while the data actually follows a distribution p.

The expected message-length under the true distribution p is

$$\begin{aligned} E_p[l] &= -E_p\left[\frac{\ln q(x)}{\ln 2}\right] \\ &= -E_p\left[\log_2 q(x)\right] \\ &= -\sum_{x_i} p(x_i) \log_2 q(x) \\ &= H(p,q) \end{aligned}$$

Special Case

A special case is the binary class problem, $C = 2$

- Based on the fact that $z_1 + z_2 = 1$ and $y_1 + y_2 = 1$

$$L = -\sum_{i=1}^{2} z_i \log\left(y_i\right) = -z_1 \log\left(y_1\right) - \left(1 - z_1\right) \log\left(1 - y_1\right)$$

A problem of this

- It could be possible to have a $y_1 = 0$

Special Case

A special case is the binary class problem, $C = 2$

- Based on the fact that $z_1 + z_2 = 1$ and $y_1 + y_2 = 1$

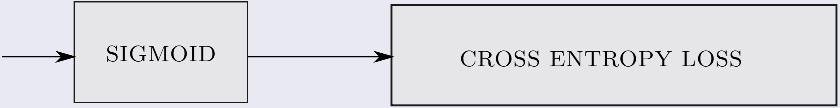
$$L = -\sum_{i=1}^{2} z_i \log(y_i) = -z_1 \log(y_1) - (1 - z_1) \log(1 - y_1)$$

A problem of this

- It could be possible to have a $y_1 = 0$

Dealing with this problem

We can use an activation function in front of it



$$y_1 = f\left(v_i\right) = \frac{1}{1+\exp\left(-v_1\right)} \quad L = -z_1 \log\left(y_1\right) - \left(1 - z_1\right) \log\left(1 - y_1\right)$$

Another Interpretation

The Loss can be expressed as

$$L = \begin{cases} -\log\left(f\left(y_1\right)\right) & \text{if } z_1 = 1 \\ -\log\left(1 - f\left(y_1\right)\right) & \text{if } z_1 = 1 \end{cases}$$

Where $z_1 = 1$

- It means that the class $C_1 = C_i$ is positive for this sample.

Another Interpretation

The Loss can be expressed as

$$L = \begin{cases} -\log\left(f\left(y_1\right)\right) & \text{if } z_1 = 1 \\ -\log\left(1 - f\left(y_1\right)\right) & \text{if } z_1 = 1 \end{cases}$$

Where $z_1 = 1$

- It means that the class $C_1 = C_i$ is positive for this sample.

The Gradient of the Binary Cross Entropy

We make a derivative with respect to y_i

$$\frac{\partial L}{\partial y_1} = z_1\left(f\left(y_1\right) - 1\right) + \left(1 - z_1\right) f\left(y_1\right)$$

In the case of the Multiclass Problem

We use two things, a softmax

$$f\left(y_i\right)=\frac{\exp\left\{y_i\right\}}{\sum_{j=1}^{C}\exp\left\{y_j\right\}}$$

As in the multiclass for the Linear Models

- The labels are one-hot, so only the positive class C_p keeps its term in the loss.

Therefore

- There is only one element of the Target vector z that is not zero, $z_i=z_p$.

In the case of the Multiclass Problem

We use two things, a softmax

$$f(y_i) = \frac{\exp\{y_i\}}{\sum_{j=1}^{C} \exp\{y_j\}}$$

As in the multiclass for the Linear Models

- The labels are one-hot, so only the positive class C_p keeps its term in the loss.

Therefore

- There is only one element of the Target vector z that is not zero, $z_i = z_p$.

In the case of the Multiclass Problem

We use two things, a softmax

$$f\left(y_i\right)=\frac{\exp\left\{y_i\right\}}{\sum_{j=1}^{C}\exp\left\{y_j\right\}}$$

As in the multiclass for the Linear Models

- The labels are one-hot, so only the positive class C_p keeps its term in the loss.

Therefore

- There is only one element of the Target vector z that is not zero, $z_i = z_p$.

We can then simplify

The cost function becomes

$$L = -\sum_{i=1}^{C} z_i \log\left(f\left(y_i\right)\right) = -log\left(\frac{\exp\left\{y_p\right\}}{\sum_{j=1}^{C}\exp\left\{y_p\right\}}\right)$$

Outline

Other Cost Functions

Exponential Cost with hyper-parameter τ

$$L = \tau \exp\left[\frac{1}{\tau}\sum_{i=1}^{N}(y_i - z_i)^2\right]$$

Hellinger Distance

$$L = \frac{1}{2}\sum_{i=1}^{N}\left(\sqrt{y_i} - \sqrt{z_i}\right)^2$$

- Here the values need to be at the interval $[0,1]$.

Other Cost Functions

Exponential Cost with hyper-parameter τ

$$L = \tau \exp\left[\frac{1}{\tau}\sum_{i=1}^{N}(y_i - z_i)^2\right]$$

Hellinger Distance

$$L = \frac{1}{2}\sum_{i=1}^{N}\left(\sqrt{y_i} - \sqrt{z_i}\right)^2$$

- Here the values need to be at the interval $[0, 1]$.

Other Cost Functions

Given Kullback-Leibler Divergence

$$D_{KL}\left(P \parallel Q\right) = \sum_i P\left(i\right) \ln \frac{P\left(i\right)}{Q\left(i\right)}$$

The Final Cost function

$$L = \sum_j \hat{y}_j \log \frac{\hat{y}_j}{y_j^{pred}}$$

Other Cost Functions

Given Kullback-Leibler Divergence

$$D_{KL}\left(P \parallel Q\right) = \sum_{i} P\left(i\right) \ln \frac{P\left(i\right)}{Q\left(i\right)}$$

The Final Cost function

$$L = \sum_{j} \hat{y}_j \log \frac{\hat{y}_j}{y_j^{pred}}$$

Outline

We have the following

Architecture with Quadratic Error

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right)$$

$$\boldsymbol{y}_t = \sigma_y \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_y \right)$$

$$L = \frac{1}{2} \sum_{t=0}^{N} \left[y_t - z_t \right]^2$$

Something Notable

- How do we train something with a recurrence forcing a dependence over time?

We have the following

Architecture with Quadratic Error

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right)$$
$$\boldsymbol{y}_t = \sigma_y \left(V_{os} \boldsymbol{h}_t + \boldsymbol{b}_y \right)$$
$$L = \frac{1}{2} \sum_{t=0}^{N} \left[y_t - z_t \right]^2$$

Something Notable

- How do we train something with a recurrence forcing a dependence over time?

Outline

Now, given the dependency over time

We can use the classic unfolding of the network [3, 4] by assuming

- W, U, V, b_h and b_o do not change under the unfolding

Unfolding?

- Assume that there are not bias correcting terms, only, W, U and V.

Now, given the dependency over time

We can use the classic unfolding of the network [3, 4] by assuming

- W, U, V, b_h and b_o do not change under the unfolding

Unfolding?

- Assume that there are not bias correcting terms, only, W, U and V.

Then

Given an observation sequence $\boldsymbol{x} = \{x_1, x_2, ..., x_T\}$

- where $x_i \in \mathbb{R}$, and their corresponding label $y = \{y_1, y_2, ..., y_T\}$

We remove the bias to simplify our derivations

$$h_t = \sigma_h\left(W_{sd}x_t + U_{ss}h_{t-1}\right)$$

$$y_t = \sigma_y\left(V_{os}h_t\right)$$

$$L = \frac{1}{2}\sum_{t=0}^{T}\left[z_t - y_t\right]^2$$

Then

Given an observation sequence $\boldsymbol{x} = \{x_1, x_2, ..., x_T\}$

- where $x_i \in \mathbb{R}$, and their corresponding label $y = \{y_1, y_2, ..., y_T\}$

We remove the bias to simplify our derivations

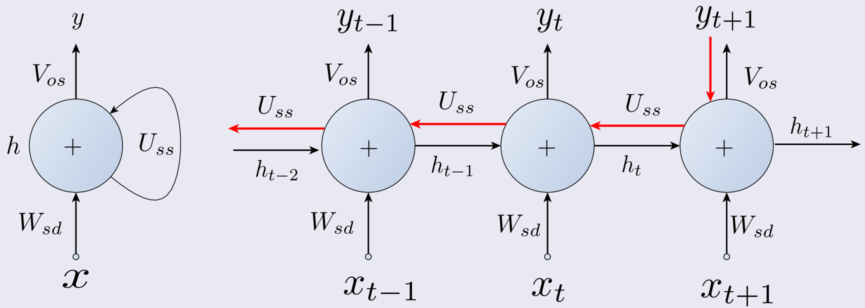
$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} \right)$$

$$y_t = \sigma_y \left(V_{os} \boldsymbol{h}_t \right)$$

$$L = \frac{1}{2} \sum_{t=0}^{T} \left[z_t - y_t \right]^2$$

Unfolding

We can then see the unfolding of the recurrence



Outline

This allows

To simplify the backpropagation process

$$\frac{\partial L}{\partial V_{os}} = \frac{1}{2}\sum_{t=0}^{T}\frac{\partial L}{\partial y_t} \times \frac{\partial y_t}{\partial V_{os}}$$

$$= \frac{1}{2}\sum_{t=0}^{T}\frac{\partial L}{\partial y_t} \times \frac{\partial y_t}{\partial net_o} \times \frac{\partial net_o}{\partial V_{os}}$$

$$= -\sum_{t=0}^{T}\left[z_t - y_t\right] \times \frac{\partial y_t}{\partial net_o} \times \frac{\partial net_o}{\partial V_{os}}$$

- Where $net_o^t = V_{os}h_t$

This allows

To simplify the backpropagation process

$$
\begin{aligned}
\frac{\partial L}{\partial V_{os}} &= \frac{1}{2}\sum_{t=0}^{T}\frac{\partial L}{\partial y_t}\times\frac{\partial y_t}{\partial V_{os}} \\
&= \frac{1}{2}\sum_{t=0}^{T}\frac{\partial L}{\partial y_t}\times\frac{\partial y_t}{\partial net_o}\times\frac{\partial net_o}{\partial V_{os}} \\
&= -\sum_{t=0}^{T}\left[z_t - y_t\right]\times\frac{\partial y_t}{\partial net_o}\times\frac{\partial net_o}{\partial V_{os}}
\end{aligned}
$$

- Where $net_o^t = V_{os}h_t$

This allows

To simplify the backpropagation process

$$
\begin{aligned}
\frac{\partial L}{\partial V_{os}} &= \frac{1}{2}\sum_{t=0}^{T}\frac{\partial L}{\partial y_t} \times \frac{\partial y_t}{\partial V_{os}} \\
&= \frac{1}{2}\sum_{t=0}^{T}\frac{\partial L}{\partial y_t} \times \frac{\partial y_t}{\partial net_o} \times \frac{\partial net_o}{\partial V_{os}} \\
&= -\sum_{t=0}^{T}\left[z_t - y_t\right] \times \frac{\partial y_t}{\partial net_o} \times \frac{\partial net_o}{\partial V_{os}}
\end{aligned}
$$

- Where $net_o^t = V_{os}\boldsymbol{h}_t$

Now, we have

We have that

$$\frac{\partial y_t}{\partial net_o} = \begin{pmatrix} \frac{\partial y_{t1}}{\partial net_{o1}} & \frac{\partial y_{t2}}{\partial net_{o1}} & \cdots & \frac{\partial y_{to}}{\partial net_{o1}} \\ \frac{\partial y_{t1}}{\partial net_{o2}} & \frac{\partial y_{t2}}{\partial net_{o2}} & \cdots & \frac{\partial y_{to}}{\partial net_{o2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{t1}}{\partial net_{oo}} & \frac{\partial y_{t2}}{\partial net_{oo}} & \cdots & \frac{\partial y_{to}}{\partial net_{oo}} \end{pmatrix}$$

Simplify!!!

Now, we have that if $i = j$

$$\frac{\partial y_{ti}}{\partial net_{oi}} = \sigma' \left(net_{oi} \right)$$

And for the rest, we have $i \neq j$

$$\frac{\partial y_{ti}}{\partial net_{oi}} = 0$$

Simplify!!!

Now, we have that if $i = j$

$$\frac{\partial y_{ti}}{\partial net_{oi}} = \sigma' \left(net_{oi} \right)$$

And for the rest, we have $i \neq j$

$$\frac{\partial y_{ti}}{\partial net_{oi}} = 0$$

Finally

We have that

$$\frac{\partial y_t}{\partial net_o} = \begin{pmatrix} \sigma_o'(net_{o1}) & 0 & \cdots & 0 \\ 0 & \sigma_o'(net_{o2}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_o'(net_{oo}) \end{pmatrix} = A$$

Now, $\frac{\partial net_o}{\partial V_{os}}$

First we have a component i

$$net_{oi} = \sum_{j=1}^{s} V_{ij} h_j$$

What happen when we derive with respect to the matrix?

$$\frac{\partial net_o}{\partial V_{os}} = \begin{bmatrix} \frac{\partial net_o}{\partial V_{11}} & \frac{\partial net_o}{\partial V_{12}} & \cdots & \frac{\partial net_o}{\partial V_{1s}} \\ \frac{\partial net_o}{\partial V_{21}} & \frac{\partial net_o}{\partial V_{22}} & \cdots & \frac{\partial net_o}{\partial V_{2s}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial net_o}{\partial V_{o1}} & \frac{\partial net_o}{\partial V_{o2}} & \cdots & \frac{\partial net_o}{\partial V_{os}} \end{bmatrix}$$

Actually

- A Tensor with three dimensions...

Now, $\frac{\partial net_o}{\partial V_{os}}$

First we have a component i

$$net_{oi} = \sum_{j=1}^{s} V_{ij} h_j$$

What happen when we derive with respect to the matrix?

$$\frac{\partial net_o}{\partial V_{os}} = \begin{bmatrix} \frac{\partial net_o}{\partial V_{11}} & \frac{\partial net_o}{\partial V_{12}} & \cdots & \frac{\partial net_o}{\partial V_{1s}} \\ \frac{\partial net_o}{\partial V_{21}} & \frac{\partial net_o}{\partial V_{22}} & \cdots & \frac{\partial net_o}{\partial V_{2s}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial net_o}{\partial V_{o1}} & \frac{\partial net_o}{\partial V_{o2}} & \cdots & \frac{\partial net_o}{\partial V_{os}} \end{bmatrix}$$

Actually

- A Tensor with three dimensions...

Now, $\frac{\partial net_o}{\partial V_{os}}$

First we have a component i

$$net_{oi} = \sum_{j=1}^{s} V_{ij} h_j$$

What happen when we derive with respect to the matrix?

$$\frac{\partial net_o}{\partial V_{os}} = \begin{bmatrix} \frac{\partial net_o}{\partial V_{11}} & \frac{\partial net_o}{\partial V_{12}} & \cdots & \frac{\partial net_o}{\partial V_{1s}} \\ \frac{\partial net_o}{\partial V_{21}} & \frac{\partial net_o}{\partial V_{22}} & \cdots & \frac{\partial net_o}{\partial V_{2s}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial net_o}{\partial V_{o1}} & \frac{\partial net_o}{\partial V_{o2}} & \cdots & \frac{\partial net_o}{\partial V_{os}} \end{bmatrix}$$

Actually

- A Tensor with three dimensions...

But something quite nice

Each of the components of net_o

- It has the previous structure

$$net_{oi} = \sum_{k=1}^{s} V_{ik} h_k$$

Then if the V_{jk} does not intervene on it

$$\frac{\partial net_{oi}}{\partial V_{jk}} = 0$$

Additionally if it intervenes

$$\frac{\partial net_{oi}}{\partial V_{jk}} = h_k$$

But something quite nice

Each of the components of net_o

- It has the previous structure

$$net_{oi} = \sum_{k=1}^{s} V_{ik} h_k$$

Then if the V_{jk} does not intervene on it

$$\frac{\partial net_{oi}}{\partial V_{jk}} = 0$$

Additionally if it intervenes

$$\frac{\partial net_{oi}}{\partial V_{jk}} = h_k$$

But something quite nice

Each of the components of net_o

- It has the previous structure

$$net_{oi} = \sum_{k=1}^{s} V_{ik} h_k$$

Then if the V_{jk} does not intervene on it

$$\frac{\partial net_{oi}}{\partial V_{jk}} = 0$$

Additionally if it intervenes

$$\frac{\partial net_{oi}}{\partial V_{jk}} = h_k$$

Therefore

It is possible to collapse the tensor into a 2D Matrix

- Given that the other information is redundant, ad we can rewrite the tensor as

$$F_{ijk} = \frac{\partial net_{oi}}{\partial V_{jk}}$$

Then, we have that

$$F_{ijk} = G_{ij} \Leftarrow \text{Better Storage!!!}$$

Therefore

It is possible to collapse the tensor into a 2D Matrix

- Given that the other information is redundant, ad we can rewrite the tensor as

$$F_{ijk} = \frac{\partial net_{oi}}{\partial V_{jk}}$$

Then, we have that

$$F_{ijk} = G_{ij} \Leftarrow \text{Better Storage!!!}$$

Therefore, given that a matrix is a tensor also

We have that two tensors, $net^{o \times o}$ and $F^{o \times s \times o}$ [5]

- We will use the contracted product of two tensors which is a generalization of the tensor-vector and tensor-matrix multiplications

Definition

- Given two tensors $A^{o \times o}$ and $B^{o \times s \times o}$

$$\langle A, B \rangle (k, j) = \sum_{i=1}^{o} A_{i,k} G_{i,j} = A_{i,i} G_{i,j} = \sigma' (net_{oi}) h_j$$

Therefore, given that a matrix is a tensor also

We have that two tensors, $net^{o\times o}$ and $F^{o\times s\times o}$ [5]

- We will use the contracted product of two tensors which is a generalization of the tensor-vector and tensor-matrix multiplications

Definition

- Given two tensors $A^{o\times o}$ and $B^{o\times s\times o}$

$$\langle A,B\rangle(k,j)=\sum_{i=1}^{o}A_{i,k}G_{i,j}=A_{i,i}G_{i,j}=\sigma'(net_{oi})\,h_j$$

Now, the term $\frac{\partial L}{\partial U_{ss}}$

Assuming our change in time step $t \rightarrow t+1$ and given

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} \right)$$

Therefore we have

$$\frac{\partial L(t+1)}{\partial U_{ss}} = \frac{\partial L(t+1)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial U_{ss}}$$

Therefore

- We can think on this as a Markovian Backpropagation

Now, the term $\frac{\partial L}{\partial U_{ss}}$

Assuming our change in time step $t \rightarrow t+1$ and given

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} \right)$$

Therefore we have

$$\frac{\partial L(t+1)}{\partial U_{ss}} = \frac{\partial L(t+1)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial U_{ss}}$$

Therefore

- We can think on this as a Markovian Backpropagation

Now, the term $\frac{\partial L}{\partial U_{ss}}$

Assuming our change in time step $t \to t+1$ and given

$$\boldsymbol{h}_t = \sigma_h\left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1}\right)$$

Therefore we have

$$\frac{\partial L\left(t+1\right)}{\partial U_{ss}} = \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial U_{ss}}$$

Therefore

- We can think on this as a Markovian Backpropagation

What if we go further

From $t-1 \rightarrow t+1$

$$\frac{\partial L\left(t+1\right)}{\partial U_{ss}} = \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_{t}} \times \frac{\partial h_{t}}{\partial U_{ss}}$$

Now, the trick if we consider all the possible derivatives from 0 to T

- We have:

$$\frac{\partial L\left(t+1\right)}{\partial U_{ss}} = \sum_{t=0}^{T} \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_{t}} \times \frac{\partial h_{t}}{\partial U_{ss}}$$

However

- How do we calculate $\frac{\partial h_{t+1}}{\partial h_k}$?

What if we go further

From $t-1 \rightarrow t+1$

$$\frac{\partial L\left(t+1\right)}{\partial U_{ss}} = \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_t} \times \frac{\partial h_t}{\partial U_{ss}}$$

Now, the trick if we consider all the possible derivatives from 0 to T

- We have:

$$\frac{\partial L\left(t+1\right)}{\partial U_{ss}} = \sum_{t=0}^{T} \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_t} \times \frac{\partial h_t}{\partial U_{ss}}$$

However

- How do we calculate $\frac{\partial h_{t+1}}{\partial h_k}$?

What if we go further

From $t-1 \rightarrow t+1$

$$\frac{\partial L\left(t+1\right)}{\partial U_{ss}} = \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_t} \times \frac{\partial h_t}{\partial U_{ss}}$$

Now, the trick if we consider all the possible derivatives from 0 to T

- We have:

$$\frac{\partial L\left(t+1\right)}{\partial U_{ss}} = \sum_{t=0}^{T} \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_t} \times \frac{\partial h_t}{\partial U_{ss}}$$

However

- How do we calculate $\frac{\partial h_{t+1}}{\partial h_k}$?

We have a proposal

Given the product of functions

$$\frac{\partial h_{t+1}}{\partial h_k} = \frac{\partial h_{k+1}}{\partial h_k} \times \frac{\partial h_{k+2}}{\partial h_{k+1}} \times \cdots \times \frac{\partial h_{t+1}}{\partial h_t}$$

Here, we know that

$$\frac{\partial h_{i+1}}{\partial h_i} = \frac{\partial h_{i+1}}{\partial net_s} \times \frac{\partial net_s}{\partial h_i}$$

We have a proposal

Given the product of functions

$$\frac{\partial h_{t+1}}{\partial h_k} = \frac{\partial h_{k+1}}{\partial h_k} \times \frac{\partial h_{k+2}}{\partial h_{k+1}} \times \cdots \times \frac{\partial h_{t+1}}{\partial h_t}$$

Here, we know that

$$\frac{\partial h_{i+1}}{\partial h_i} = \frac{\partial h_{i+1}}{\partial net_s} \times \frac{\partial net_s}{\partial h_i}$$

We have that

We have given $\boldsymbol{h}_{i+1} = \sigma_h\left(W_{sd}\boldsymbol{x}_i + U_{ss}\boldsymbol{h}_i\right)$ and $net_h = W_{sd}\boldsymbol{x}_i + U_{ss}\boldsymbol{h}_i$

$$\frac{\partial h_{i+1}}{\partial net_s} = \begin{pmatrix} \sigma_h'\left(net_{h1}\right) & 0 & \cdots & 0 \\ 0 & \sigma_h'\left(net_{h2}\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_h'\left(net_{hs}\right) \end{pmatrix} = D_{i+1}$$

Finally, we have that

$$\frac{\partial net_s}{\partial h_i} = U_{ss}$$

We have that

We have given $\boldsymbol{h}_{i+1} = \sigma_h \left(W_{sd}\boldsymbol{x}_i + U_{ss}\boldsymbol{h}_i \right)$ and $net_h = W_{sd}\boldsymbol{x}_i + U_{ss}\boldsymbol{h}_i$

$$\frac{\partial h_{i+1}}{\partial net_s} = \begin{pmatrix} \sigma_h' \left(net_{h1} \right) & 0 & \cdots & 0 \\ 0 & \sigma_h' \left(net_{h2} \right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_h' \left(net_{hs} \right) \end{pmatrix} = D_{i+1}$$

Finally, we have that

$$\frac{\partial net_s}{\partial h_i} = U_{ss}$$

Then

We can aggregate over all the time

$$\frac{\partial L}{\partial U_{ss}} = \sum_{k=1}^{t} \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_k} \times \frac{\partial h_t}{\partial U_{ss}}$$

Now, we need to derive the L with respect to W_{sd}

$$\frac{\partial L\left(t+1\right)}{\partial W_{sd}} = \frac{\partial L\left(t+1\right)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}}$$

Then

We can aggregate over all the time

$$\frac{\partial L}{\partial U_{ss}} = \sum_{k=1}^{t} \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_k} \times \frac{\partial h_t}{\partial U_{ss}}$$

Now, we need to derive the L with respect to W_{sd}

$$\frac{\partial L\left(t+1\right)}{\partial W_{sd}} = \frac{\partial L\left(t+1\right)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}}$$

Now

Because h_t and x_{t+1}, we need to back-propagate to h_t

$$\frac{\partial L\left(t+1\right)}{\partial W_{sd}} = \frac{\partial L\left(t+1\right)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}} + \frac{\partial L\left(t+1\right)}{\partial h_{t}} \times \frac{\partial h_{t}}{\partial W_{sd}}$$
$$= \frac{\partial L\left(t+1\right)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}} + \frac{\partial L\left(t+1\right)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_{t}} \times \frac{\partial h_{t}}{\partial W_{sd}}$$

Then summing over all the contributions from t to 0

$$\frac{\partial L\left(t+1\right)}{\partial W_{sd}} = \sum_{k=1}^{t+1} \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_{k}} \times \frac{\partial h_{t}}{\partial W_{sd}}$$

Finally, summing over all the time

$$\frac{\partial L}{\partial W_{sd}} = \sum_{k=1}^{t+1} \frac{\partial L\left(t+1\right)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_{k}} \times \frac{\partial h_{t}}{\partial W_{sd}}$$

Now

Because h_t and x_{t+1}, we need to back-propagate to h_t

$$\begin{aligned}\frac{\partial L(t+1)}{\partial W_{sd}} &= \frac{\partial L(t+1)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}} + \frac{\partial L(t+1)}{\partial h_t} \times \frac{\partial h_t}{\partial W_{sd}} \\ &= \frac{\partial L(t+1)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}} + \frac{\partial L(t+1)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_t} \times \frac{\partial h_t}{\partial W_{sd}}\end{aligned}$$

Then summing over all the contributions from t to 0

$$\frac{\partial L(t+1)}{\partial W_{sd}} = \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_k} \times \frac{\partial h_t}{\partial W_{sd}}$$

Finally, summing over all the time

$$\frac{\partial L}{\partial W_{sd}} = \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_k} \times \frac{\partial h_t}{\partial W_{sd}}$$

Now

Because h_t and x_{t+1}, we need to back-propagate to h_t

$$\begin{aligned}\frac{\partial L(t+1)}{\partial W_{sd}} &= \frac{\partial L(t+1)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}} + \frac{\partial L(t+1)}{\partial h_t} \times \frac{\partial h_t}{\partial W_{sd}} \\ &= \frac{\partial L(t+1)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial W_{sd}} + \frac{\partial L(t+1)}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_t} \times \frac{\partial h_t}{\partial W_{sd}}\end{aligned}$$

Then summing over all the contributions from t to 0

$$\frac{\partial L(t+1)}{\partial W_{sd}} = \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_k} \times \frac{\partial h_t}{\partial W_{sd}}$$

Finally, summing over all the time

$$\frac{\partial L}{\partial W_{sd}} = \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial y_{t+1}} \times \frac{\partial y_{t+1}}{\partial h_{t+1}} \times \frac{\partial h_{t+1}}{\partial h_k} \times \frac{\partial h_t}{\partial W_{sd}}$$

Outline

Vanishing Gradients

We have a problem

$$\frac{\partial h_{k+1}}{\partial h_k} \times \frac{\partial h_{k+2}}{\partial h_{k+1}} \times \cdots \times \frac{\partial h_{t+1}}{\partial h_t}$$

You finish with a vanishing gradient using $\sigma = \frac{1}{1+\exp\{-x\}}$

- This is problematic!!!

Vanishing Gradients

We have a problem

$$\frac{\partial h_{k+1}}{\partial h_k} \times \frac{\partial h_{k+2}}{\partial h_{k+1}} \times \cdots \times \frac{\partial h_{t+1}}{\partial h_t}$$

You finish with a vanishing gradient using $\sigma = \frac{1}{1+\exp\{-x\}}$

- This is problematic!!!

Given

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f(x) = \frac{ds(x)}{dx} = \frac{e^{-x}}{\left(1 + e^{-x}\right)^2}$$

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{\left(1 + e^{-x}\right)^2} + \frac{2\left(e^{-x}\right)^2}{\left(1 + e^{-x}\right)^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Given

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f(x) = \frac{ds(x)}{dx} = \frac{e^{-x}}{\left(1+e^{-x}\right)^2}$$

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{\left(1+e^{-x}\right)^2} + \frac{2\left(e^{-x}\right)^2}{\left(1+e^{-x}\right)^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Given

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f(x) = \frac{ds(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{(1+e^{-x})^2} + \frac{2(e^{-x})^2}{(1+e^{-x})^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Network

- We could finish with

$$\lim_{k\to\infty}\left(\frac{ds(x)}{dx}\right)^k = \lim_{k\to\infty}(0.25)^k \to 0$$

A Vanishing Derivative or Vanishing Gradient

- Making quite difficult to do train a deeper network using this activation function for Deep Learning and even in Shallow Learning

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Network

- We could finish with

$$\lim_{k \to \infty} \left(\frac{ds(x)}{dx} \right)^k = \lim_{k \to \infty} (0.25)^k \to 0$$

A Vanishing Derivative or Vanishing Gradient

- Making quite difficult to do train a deeper network using this activation function for Deep Learning and even in Shallow Learning

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Network

- We could finish with

$$\lim_{k\to\infty}\left(\frac{ds(x)}{dx}\right)^k = \lim_{k\to\infty}(0.25)^k \to 0$$

A Vanishing Derivative or Vanishing Gradient

- Making quite difficult to do train a deeper network using this activation function for Deep Learning and even in Shallow Learning

For the case of vanishing gradient, we have that

Rearranging terms in $\frac{\partial h_{k+1}}{\partial h_k} \times \frac{\partial h_{k+2}}{\partial h_{k+1}} \times \cdots \times \frac{\partial h_{t+1}}{\partial h_t}$

- We have

$$\left[\prod_{k=0}^{T} \frac{\partial h_{k+1}}{\partial net_s}\right] [U_{ss}]^{T+1}$$

Then, given the sigmoid

$$\prod_{k=0}^{T} \frac{\partial h_{k+1}}{\partial net_s} = \begin{bmatrix} \prod_{k=0}^{T} \sigma_h'\left(net_{h1}^{k}\right) & 0 & \cdots & 0 \\ 0 & \prod_{k=0}^{T} \sigma_h'\left(net_{h2}^{k}\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \prod_{k=0}^{T} \sigma_h'\left(net_{hs}^{k}\right) \end{bmatrix}$$

For the case of vanishing gradient, we have that

Rearranging terms in $\frac{\partial h_{k+1}}{\partial h_k} \times \frac{\partial h_{k+2}}{\partial h_{k+1}} \times \cdots \times \frac{\partial h_{t+1}}{\partial h_t}$

- We have

$$\left[\prod_{k=0}^{T} \frac{\partial h_{k+1}}{\partial net_s}\right] [U_{ss}]^{T+1}$$

Then, given the sigmoid

$$\prod_{k=0}^{T} \frac{\partial h_{k+1}}{\partial net_s} = \begin{bmatrix} \prod_{k=0}^{T} \sigma_h' \left(net_{h1}^k\right) & 0 & \cdots & 0 \\ 0 & \prod_{k=0}^{T} \sigma_h' \left(net_{h2}^k\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \prod_{k=0}^{T} \sigma_h' \left(net_{hs}^k\right) \end{bmatrix}$$

It is clear

That you have the phenomena of vanishing gradient

- Do we have a way to fixing this?

Yes

- The use of new activation functions.

It is clear

That you have the phenomena of vanishing gradient

- Do we have a way to fixing this?

Yes

- The use of new activation functions.

Outline

Thus

The need to introduce a new function

$$f(x) = x^{+} = \max(0, x)$$

It is called ReLu or Rectifier

With a smooth approximation (Softplus function)

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

Thus

The need to introduce a new function

$$f(x) = x^{+} = \max(0, x)$$

It is called ReLu or Rectifier

With a smooth approximation (Softplus function)

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

Outline

However

Here the gradient can explode

- Thus, the need to control the gradient...

Therefore, we will use the following analysis [6]

- "The Emergence of Spectral Universality in Deep Networks" by Jeffrey Pennington, Samuel S. Schoenholz, Surya Ganguli

However

Here the gradient can explode

- Thus, the need to control the gradient...

Therefore, we will use the following analysis [6]

- "The Emergence of Spectral Universality in Deep Networks" by Jeffrey Pennington, Samuel S. Schoenholz, Surya Ganguli

We have

The following dynamic

$$\boldsymbol{h}_t = \sigma_h\left(s_t\right),\ \boldsymbol{s}_t = W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + b_h$$

Then, we have the following Jacobian

$$J = \frac{\partial h_T}{\partial h_0} = \prod_{t=1}^{L} D_t U_{SS}$$

Where as we saw it D_t is a diagonal matrix

- This Jacobian J is a matrix of dimension $s \times s$ therefore, if it is well conditioned you are not sending the projection to lower dimensionality.

We have

The following dynamic

$$\boldsymbol{h}_t = \sigma_h\left(s_t\right),\ \boldsymbol{s}_t = W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + b_h$$

Then, we have the following Jacobian

$$J = \frac{\partial h_T}{\partial h_0} = \prod_{t=1}^{L} D_t U_{SS}$$

Where as we saw it D_t is a diagonal matrix

- This Jacobian J is a matrix of dimension $s \times s$ therefore, if it is well conditioned you are not sending the projection to lower dimensionality.

We have

The following dynamic

$$\boldsymbol{h}_t = \sigma_h\left(s_t\right), \boldsymbol{s}_t = W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + b_h$$

Then, we have the following Jacobian

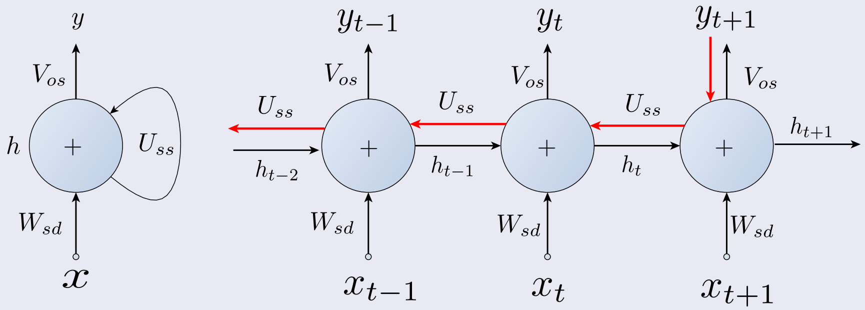
$$J = \frac{\partial h_T}{\partial h_0} = \prod_{t=1}^{L} D_t U_{SS}$$

Where as we saw it D_t is a diagonal matrix

- This Jacobian J is a matrix of dimension $s \times s$ therefore, if it is well conditioned you are not sending the projection to lower dimensionality.

A Trick

A RNN can be seen as a deep neural network



Remember the structure of the layer

The following dynamic

$$\boldsymbol{h}_t = \sigma_h\left(s_t\right), \ \boldsymbol{s}_t = W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + b_h$$

Therefore, we have that

$$s_{it} = \sum_j W_{ij} x_j^t + \sum_k U_{ik} h_k^{t-1} + b_i$$

We assume the following about the temporal layer weights

$$\left[U_{ss}, W_{sd}\right] \sim N\left(0, \frac{\rho_w^2}{N}\right), b_h \sim N\left(0, \rho_b^2\right)$$

- Here $N = s$ the state dimension.

Remember the structure of the layer

The following dynamic

$$\boldsymbol{h}_t = \sigma_h\left(s_t\right), \ \boldsymbol{s}_t = W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + b_h$$

Therefore, we have that

$$s_{it} = \sum_j W_{ij} x_j^t + \sum_k U_{ik} h_k^{t-1} + b_i$$

We assume the following about the temporal layer weights

$$\left[U_{ss}, W_{sd}\right] \sim N\left(0, \frac{\rho_w^2}{N}\right), b_h \sim N\left(0, \rho_b^2\right)$$

- Here $N = s$ the state dimension.

Remember the structure of the layer

The following dynamic

$$\boldsymbol{h}_t = \sigma_h\left(s_t\right), \; \boldsymbol{s}_t = W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} + b_h$$

Therefore, we have that

$$s_{it} = \sum_j W_{ij}x_j^t + \sum_k U_{ik}h_k^{t-1} + b_i$$

We assume the following about the temporal layer weights

$$\left[U_{ss}, W_{sd}\right] \sim N\left(0, \frac{\rho_w^2}{N}\right), b_h \sim N\left(0, \rho_b^2\right)$$

- Here $N = s$ the state dimension.

Outline

Now, assume that

Now, consider the evolution of a single input through the network $\boldsymbol{x}_{it}$

- Since the weights and biases are independent with zero mean

$$E\left[s_{it}\right] = 0$$

The second moment of the Gaussian random variable

$$E\left[s_{it}s_{jt}\right] = q^t\delta_{ij}$$

Now, assume that

Now, consider the evolution of a single input through the network $\boldsymbol{x}_{it}$

- Since the weights and biases are independent with zero mean

$$E\left[s_{it}\right] = 0$$

The second moment of the Gaussian random variable

$$E\left[s_{it}s_{jt}\right] = q^{t}\delta_{ij}$$

Where the second moment

Of a Gaussian Distribution is

$$\int_{-\infty}^{\infty} s^2 \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(s-\mu)}{2\sigma^2}\right\} ds$$

Here we have

Here q is the variance of the pre-activations in the t^{th} layer due to an input $\boldsymbol{x}_t$

$$q^t = \frac{\rho_w^2}{\sqrt{2\pi}} \int \sigma_h^2 \left(\sqrt{q^{t-1}}\boldsymbol{s}_{it-1}\right) \exp\left\{-\frac{1}{2}\boldsymbol{s}_{it}^2\right\} d\boldsymbol{s}_{it} + \rho_b^2$$

They describe the pass through the recursion of the RNN

- For any choice of ρ_w^2 and ρ_b^2 and a bounded ϕ the previous equation converges to a specific fix point.

This recursion has a fixed point

$$q^* = \frac{\rho_w^2}{\sqrt{2\pi}} \int \sigma_h^2 \left(\sqrt{q^*}\boldsymbol{s}_{it-1}\right) \exp\left\{-\frac{1}{2}\boldsymbol{s}_{it}^2\right\} d\boldsymbol{s}_{it} + \rho_b^2$$

Here we have

Here q is the variance of the pre-activations in the t^{th} layer due to an input $\boldsymbol{x}_t$

$$q^t = \frac{\rho_w^2}{\sqrt{2\pi}} \int \sigma_h^2 \left(\sqrt{q^{t-1}} \boldsymbol{s}_{it-1} \right) \exp\left\{ -\frac{1}{2} \boldsymbol{s}_{it}^2 \right\} d\boldsymbol{s}_{it} + \rho_b^2$$

They describe the pass through the recursion of the RNN

- For any choice of ρ_w^2 and ρ_b^2 and a bounded ϕ the previous equation converges to a specific fix point.

This recursion has a fixed point

$$q^* = \frac{\rho_w^2}{\sqrt{2\pi}} \int \sigma_h^2 \left(\sqrt{q^*} \boldsymbol{s}_{it-1} \right) \exp\left\{ -\frac{1}{2} \boldsymbol{s}_{it}^2 \right\} d\boldsymbol{s}_{it} + \rho_b^2$$

Here we have

Here q is the variance of the pre-activations in the t^{th} layer due to an input $\boldsymbol{x}_t$

$$q^t = \frac{\rho_w^2}{\sqrt{2\pi}} \int \sigma_h^2 \left(\sqrt{q^{t-1}} \boldsymbol{s}_{it-1} \right) \exp\left\{ -\frac{1}{2} \boldsymbol{s}_{it}^2 \right\} d\boldsymbol{s}_{it} + \rho_b^2$$

They describe the pass through the recursion of the RNN

- For any choice of ρ_w^2 and ρ_b^2 and a bounded ϕ the previous equation converges to a specific fix point.

This recursion has a fixed point

$$q^* = \frac{\rho_w^2}{\sqrt{2\pi}} \int \sigma_h^2 \left(\sqrt{q^*} \boldsymbol{s}_{it-1} \right) \exp\left\{ -\frac{1}{2} \boldsymbol{s}_{it}^2 \right\} d\boldsymbol{s}_{it} + \rho_b^2$$

A Fixed Point

Definition

- In mathematics, a fixed point of a function is an element of the function's domain that is mapped to itself by the function.

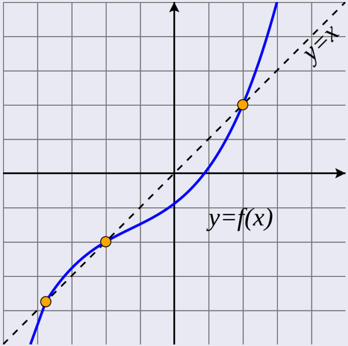
Example

A Fixed Point

Definition

- In mathematics, a fixed point of a function is an element of the function's domain that is mapped to itself by the function.

Example



Therefore

We have that

- It the input $\boldsymbol{x}_0$ is chosen so that $q^1 = q^*$ the dynamics start at the fixed point and the distribution of D_t is independent of t.

Not only that

- $q^1 \neq q^*$ a few layers is often sufficient to approximately converge to a fixed point.

So when t is large

- So it is a good approximation to assume $q^l = q^*$.

Therefore

We have that

- It the input $\boldsymbol{x}_0$ is chosen so that $q^1 = q^*$ the dynamics start at the fixed point and the distribution of D_t is independent of t.

Not only that

- $q^1 \neq q^*$ a few layers is often sufficient to approximately converge to a fixed point.

So when t is large

- So it is a good approximation to assume $q^l = q^*$.

Therefore

We have that

- It the input $\boldsymbol{x}_0$ is chosen so that $q^1 = q^*$ the dynamics start at the fixed point and the distribution of D_t is independent of t.

Not only that

- $q^1 \neq q^*$ a few layers is often sufficient to approximately converge to a fixed point.

So when t is large

- So it is a good approximation to assume $q^t = q^*$.

Additionally

The independence of the weights and biases implies

- The covariance between different pre-activations in the same layer will be given by

$$E\left[z_{it;a}z_{jt;b}\right]=q_{ab}^{t}\delta_{ij}$$

Therefore

$$q_{ab}^{t}=\rho_{w}^{2}\int\sigma_{h}\left(u_{1}\right)\sigma_{h}\left(u_{2}\right)Dz_{1}Dz_{2}+\rho_{b}^{2}$$

- Where $Dz=\frac{1}{\sqrt{2\pi}}\int\exp\left\{-\frac{1}{2}s^{2}\right\}ds$
- $u_{1}=\sqrt{q_{aa}^{t-1}}$
- $u_{2}=\sqrt{q_{bb}^{t-1}}\left[c_{ab}^{t-1}s_{1}+\sqrt{1-\left(c_{ab}^{t-1}\right)^{2}}z_{2}\right]$
- $c_{ab}^{t}=\frac{q_{ab}^{t}}{\sqrt{q_{aa}^{t}q_{bb}^{t}}}$

Additionally

The independence of the weights and biases implies

- The covariance between different pre-activations in the same layer will be given by

$$E\left[z_{it;a} z_{jt;b}\right] = q^t_{ab}\delta_{ij}$$

Therefore

$$q^t_{ab} = \rho^2_w \int \sigma_h\left(u_1\right)\sigma_h\left(u_2\right) Dz_1 Dz_2 + \rho^2_b$$

- Where $Dz = \frac{1}{\sqrt{2\pi}} \int \exp\left\{-\frac{1}{2}s^2\right\} ds$
- $u_1 = \sqrt{q^{t-1}_{aa}}$
- $u_2 = \sqrt{q^{t-1}_{bb}} \left[c^{t-1}_{ab} s_1 + \sqrt{1 - \left(c^{t-1}_{ab}\right)^2} z_2\right]$
- $c^t_{ab} = \frac{q^t_{ab}}{\sqrt{q^t_{aa} q^t_{bb}}}$

Therefore

Therefore, we can look at the variance of the Jacobian Matrix elements

$$\chi = \frac{1}{N}\left\langle Tr\left[\left(D_t U_{SS}\right)^T D_t U_{SS}\right]\right\rangle = \sigma_w^2 \int \left[\sigma_h'\left(\sqrt{q^*}\boldsymbol{s}_{it}\right)\right]^2 \exp\left\{-\frac{1}{2}\boldsymbol{s}_{it}^2\right\} d\boldsymbol{s}_{it}$$

Then

$\chi\left(\rho_w, \rho_b\right)$

- It separates $\left(\rho_w, \rho_b\right)$ plane into two regions.

When $\chi > 1$

- Forward signal propagation expands and folds space in a chaotic manner and gradients explode

When $\chi < 1$

- Forward signal propagation contracts in an ordered manner and gradients exponentially vanishes

Then

$\chi\left(\rho_w, \rho_b\right)$

- It separates $\left(\rho_w, \rho_b\right)$ plane into two regions.

When $\chi > 1$

- Forward signal propagation expands and folds space in a chaotic manner and gradients explode

When $\chi < 1$

- Forward signal propagation contracts in an ordered manner and gradients exponentially vanishes

Then

$\chi\left(\rho_w, \rho_b\right)$

- It separates $\left(\rho_w, \rho_b\right)$ plane into two regions.

When $\chi > 1$

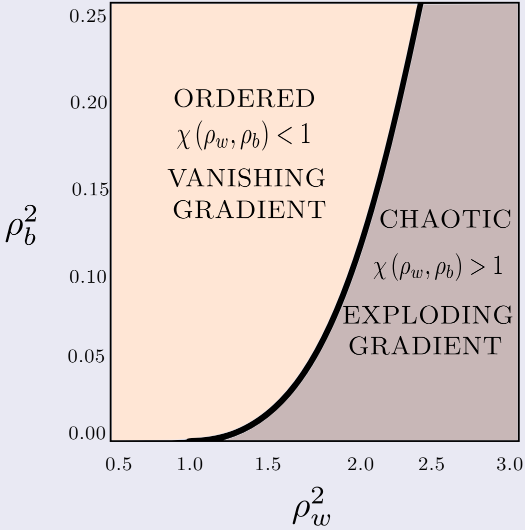
- Forward signal propagation expands and folds space in a chaotic manner and gradients explode

When $\chi < 1$

- Forward signal propagation contracts in an ordered manner and gradients exponentially vanishes

This Regions establish the stability of the network

We have the following



Therefore

It is clear that

- When we choose same $\rho_b = \rho_w$ we have a convergence of the network

Having other values

- It requires a careful choosing of the values

Therefore

It is clear that

- When we choose same $\rho_b = \rho_w$ we have a convergence of the network

Having other values

- It requires a careful choosing of the values

Outline

Another Problem

Although, the Vanishing and Exploding Gradients

- They are a problem for the RNN's

If we use the full BPTT

- We confront limitations on the amount of Memory and Hardware available

Thus a popular strategy

- It is the Truncated BPTT [7, 8]

Another Problem

Although, the Vanishing and Exploding Gradients

- They are a problem for the RNN's

If we use the full BPTT

- We confront limitations on the amount of Memory and Hardware available

Thus a popular strategy

- It is the Truncated BPTT [7, 8]

Another Problem

Although, the Vanishing and Exploding Gradients

- They are a problem for the RNN's

If we use the full BPTT

- We confront limitations on the amount of Memory and Hardware available

Thus a popular strategy

- It is the Truncated BPTT [7, 8]

Therefore

They proposed using a truncation on the BPTT

- To solve the problem with the Vanishing and Exploding Gradient

What is Truncated BPTT?

- In general, this should be regarded as a heuristic technique for simplifying the computation.
  - Which it is a good approximation true gradient

Therefore

They proposed using a truncation on the BPTT

- To solve the problem with the Vanishing and Exploding Gradient

What is Truncated BPTT?

- In general, this should be regarded as a heuristic technique for simplifying the computation.
  - Which it is a good approximation true gradient

The Algorithm

Truncated BPTT

1. for $t = 1$ to T do:
2. Run the RNN for one step, computing h_t and y_t
3. if t divides k_1 then
4. Run BPTT from t to $t - k_2$

Something Notable

1. It was first used by Elman [9]
2. Also Mikolov et al. [10] used the TBPTT to train RNN on word-level language modeling.

The Algorithm

Truncated BPTT

1. for $t = 1$ to T do:
2. Run the RNN for one step, computing h_t and y_t
3. if t divides k_1 then
4. Run BPTT from t to $t - k_2$

Something Notable

1. It was first used by Elman [9]
2. Also Mikolov et al. [10] used the TBPTT to train RNN on word-level language modeling.

Outline

Outline

Initialization of the Hidden State

This is the classic problem in RNN

- How to initialize the h_s hidden state?

There are two main mehtods

1. Initialize h_s to the zero vector.
2. Adaptive noisy initialization of h_s
3. Find the steady state

Initialization of the Hidden State

This is the classic problem in RNN

- How to initialize the h_s hidden state?

There are two main mehtods

1. Initialize h_s to the zero vector.
2. Adaptive noisy initialization of h_s
3. Find the steady state

The Simplest One

We can simply initialize h_s

- To a zero state

Quite simple and easy to apply

- However do we have something better?

The Simplest One

We can simply initialize h_s

- To a zero state

Quite simple and easy to apply

- However do we have something better?

Adaptive noisy initialization

It is proposed by Zimmermann et al. [11]

- They proposed to use the residual error once the back-propagation was done for $\boldsymbol{h}_0$

This is done

- By disturbing h_0 with a noise term Θ which follows the distribution of the residual error.

Adaptive noisy initialization

It is proposed by Zimmermann et al. [11]

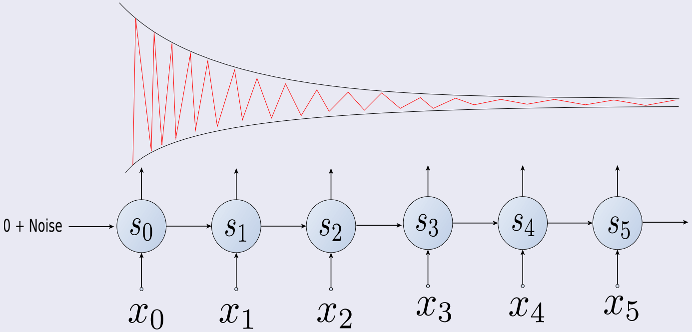
- They proposed to use the residual error once the back-propagation was done for $\boldsymbol{h}_0$

This is done

- By disturbing $\boldsymbol{h}_0$ with a noise term Θ which follows the distribution of the residual error.

Adaptive Noise

The network tries to stabilize the output



Example of this initializations

Source https://r2rt.com/non-zero-initial-states-for-recurrent-neural-networks.html

What about the Weight Parameters?

We could simply initialize them to zero

- Denger Will Robinson!!!

A simple example with the following feed-forward architecture

$$
\begin{aligned}
w &= \sigma_1 \left(W_{hi} x \right) \\
y &= \sigma_2 \left(W_{oh} w \right) \\
L &= \frac{1}{2} \left[y - z \right]^2
\end{aligned}
$$

What about the Weight Parameters?

We could simply initialize them to zero

- Denger Will Robinson!!!

A simple example with the following feed-forward architecture

$$\boldsymbol{w} = \sigma_1 \left(W_{hi} \boldsymbol{x} \right)$$
$$\boldsymbol{y} = \sigma_2 \left(W_{oh} \boldsymbol{w} \right)$$
$$L = \frac{1}{2} \left[\boldsymbol{y} - \boldsymbol{z} \right]^2$$

Therefore

We have by back-propagation

$$\Delta W_{ho} = \left[\sigma_2'\left(W_{oh}\sigma_1\left(W_{hi}\boldsymbol{x}_1\right)\right) - \boldsymbol{z}\right]\sigma_2'\left(W_{oh}\sigma_1\left(W_{hi}\boldsymbol{x}\right)\right)W_{oh}\sigma_1'\left(W_{hi}\boldsymbol{x}\right)\boldsymbol{x}$$

Therefore

$$\Delta W_{ho} = 0$$

Therefore

We have by back-propagation

$$\Delta W_{ho} = \left[\sigma_2'\left(W_{oh}\sigma_1\left(W_{hi}\boldsymbol{x}_1\right)\right) - \boldsymbol{z}\right]\sigma_2'\left(W_{oh}\sigma_1\left(W_{hi}\boldsymbol{x}\right)\right)W_{oh}\sigma_1'\left(W_{hi}\boldsymbol{x}\right)\boldsymbol{x}$$

Therefore

$$\Delta W_{ho} = 0$$

Therefore

Not a good idea

- What else we can do?

We have heuristics as the Gaussian initialization

$$w_{ij} \sim N\left(0, \sigma^2\right)$$

Therefore

Not a good idea

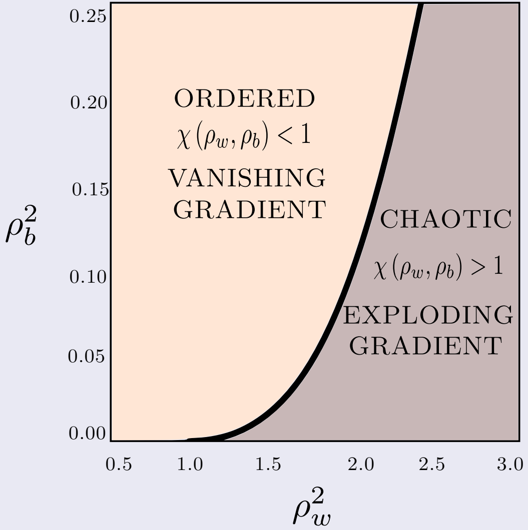
- What else we can do?

We have heuristics as the Gaussian initialization

$$w_{ij} \sim N\left(0, \sigma^2\right)$$

Do you remember?

We have the following



Furthermore

We have heuristics

- For Relu — We multiply the randomly generated values of W by:

$$\sqrt{\frac{2}{size^{l-1}}}$$

For tanh — The heuristic is called Xavier initialization

$$\sqrt{\frac{2}{size^{l-1}}}$$

Other common one

$$\sqrt{\frac{2}{size^{l-1}+size^{l}}}$$

Furthermore

We have heuristics

- For Relu — We multiply the randomly generated values of W by:

$$\sqrt{\frac{2}{size^{l-1}}}$$

For tanh — The heuristic is called Xavier initialization

$$\sqrt{\frac{2}{size^{l-1}}}$$

Other common one

$$\sqrt{\frac{2}{size^{l-1}+size^{l}}}$$

Furthermore

We have heuristics

- For Relu — We multiply the randomly generated values of W by:

$$\sqrt{\frac{2}{size^{l-1}}}$$

For tanh — The heuristic is called Xavier initialization

$$\sqrt{\frac{2}{size^{l-1}}}$$

Other common one

$$\sqrt{\frac{2}{size^{l-1}+size^{l}}}$$

Outline

History of LSTM

They were introduced by

- LSTM was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [12]

An attempt to deal with the vanishing and exploding gradient

- By introducing Constant Error Carousel (CEC) units

Properties

- In 1999, Felix Gers and his advisor Jürgen Schmidhuber and Fred Cummins introduced the forget gate (also called "keep gate") into LSTM architecture.
  - It enables the LSTM to reset its own state

History of LSTM

They were introduced by

- LSTM was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [12]

An attempt to deal with the vanishing and exploding gradient

- By introducing Constant Error Carousel (CEC) units

Properties

- In 1999, Felix Gers and his advisor Jürgen Schmidhuber and Fred Cummins introduced the forget gate (also called "keep gate") into LSTM architecture.
  - It enables the LSTM to reset its own state

History of LSTM

They were introduced by

- LSTM was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [12]

An attempt to deal with the vanishing and exploding gradient

- By introducing Constant Error Carousel (CEC) units

Properties

- In 1999, Felix Gers and his advisor Jürgen Schmidhuber and Fred Cummins introduced the forget gate (also called "keep gate") into LSTM architecture.
  - It enables the LSTM to reset its own state

Long Short Term Memory (LSTM)

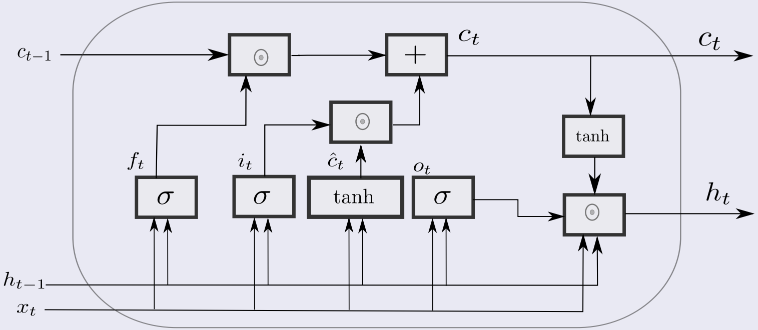
We have the following Architecture (Component wise product $\odot$)

$$
\begin{aligned}
\boldsymbol{f}_t &= \sigma \left[W_f \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t \right] + \boldsymbol{b}_f \right] \text{ (Forget Gate)} \\
\boldsymbol{i}_t &= \sigma \left[W_i \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t \right] + \boldsymbol{b}_i \right] \text{ (Input/Update Gate)} \\
\boldsymbol{o}_t &= \sigma \left[W_o \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t \right] + \boldsymbol{b}_o \right] \text{ (Output Gate)} \\
\hat{\boldsymbol{c}}_t &= \tanh \left[W_o \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t \right] + \boldsymbol{b}_c \right] \text{ (Intermediate Cell Gate)} \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t \text{ (Cell State Gate)} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh \left(\boldsymbol{c}_t \right) \text{ (Hidden State)}
\end{aligned}
$$

- Where σ is a sigmoid function.

Graphically

We have that



Here the interesting part

In the RNN

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd}\boldsymbol{x}_t + U_{ss}\boldsymbol{h}_{t-1} \right)$$

But Here

$$\begin{aligned} \boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t \text{ (Cell State Gate)} \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh\left(\boldsymbol{c}_t\right) \end{aligned}$$

You need the forget term, the input term ant the intermediate cell

- To update the state

Here the interesting part

In the RNN

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} \right)$$

But Here

$$\begin{aligned} \boldsymbol{c}_t =& \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t \text{ (Cell State Gate)} \\ \boldsymbol{h}_t =& \boldsymbol{o}_t \odot \tanh\left(\boldsymbol{c}_t \right) \end{aligned}$$

You need the forget term, the input term ant the intermediate cell

- To update the state

Here the interesting part

In the RNN

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{ss} \boldsymbol{h}_{t-1} \right)$$

But Here

$$\begin{aligned} \boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t \text{ (Cell State Gate)} \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh \left(\boldsymbol{c}_t \right) \end{aligned}$$

You need the forget term, the input term ant the intermediate cell

- To update the state

You can see

Something Notable

- The cell keeps track of the dependencies between the elements in the input sequence and the state

The input gate

- It is in charge of how much of the input flows into the cell gate

$$i_t = \sigma\left[W_i\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_i\right]$$

You can see

Something Notable

- The cell keeps track of the dependencies between the elements in the input sequence and the state

The input gate

- It is in charge of how much of the input flows into the cell gate

$$\boldsymbol{i}_t = \sigma\left[W_i\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_i\right]$$

What is the meaning?

We have that

- The sigmoid layer decides what values to update

They impact the term $i_t \odot \hat{c}_t$

- Making possible to decide how to control the cell intermediate values

What is the meaning?

We have that

- The sigmoid layer decides what values to update

They impact the term $\boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t$

- Making possible to decide how to control the cell intermediate values

Now

The forget gate

- How much of the previous cell gate time value remains in the cell at time t

$$\boldsymbol{f}_t = \sigma\left[W_f\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_f\right]$$

Actually

- It uses previous state and input

Then the sigmoid actually can be interpreted as

- Sigmoid: value 0 and 1 – "completely forget" vs. "completely keep"

Now

The forget gate

- How much of the previous cell gate time value remains in the cell at time t

$$\boldsymbol{f}_t = \sigma \left[W_f \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t \right] + \boldsymbol{b}_f \right]$$

Actually

- It uses previous state and input

Then the sigmoid actually can be interpreted as

- Sigmoid: value 0 and 1 – "completely forget" vs. "completely keep"

Now

The forget gate

- How much of the previous cell gate time value remains in the cell at time t

$$\boldsymbol{f}_t = \sigma \left[W_f \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t \right] + \boldsymbol{b}_f \right]$$

Actually

- It uses previous state and input

Then the sigmoid actually can be interpreted as

- Sigmoid: value 0 and 1 – "completely forget" vs. "completely keep"

Furthermore

The output gate

- It controls the extent to which the value in the cell is used to compute the actual state

Which impacts the term $f_t \odot c_{t-1}$

- Based on the previous cell state

Thus a type of control

- Between the previous cell state and the new cell state

Furthermore

The output gate

- It controls the extent to which the value in the cell is used to compute the actual state

Which impacts the term $\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}$

- Based on the previous cell state

Thus a type of control

- Between the previous cell state and the new cell state

Furthermore

The output gate

- It controls the extent to which the value in the cell is used to compute the actual state

Which impacts the term $\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}$

- Based on the previous cell state

Thus a type of control

- Between the previous cell state and the new cell state

Finally

We have the update of the cell as

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t$$

Basically

- Apply forget operation to previous internal cell state.
- Add new candidate values, scaled by how much we decided to update

We can see as

- Drop old information and add new information about subject's gender.

Finally

We have the update of the cell as

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t$$

Basically

- Apply forget operation to previous internal cell state.
- Add new candidate values, scaled by how much we decided to update

We can see as

- Drop old information and add new information about subject's gender.

Finally

We have the update of the cell as

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t$$

Basically

- Apply forget operation to previous internal cell state.
- Add new candidate values, scaled by how much we decided to update

We can see as

- Drop old information and add new information about subject's gender.

Thus at the output layer and update state

We have

$$\begin{aligned} \boldsymbol{o}_t &= \sigma\left[W_o\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_o\right] \text{ (Output Gate)} \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh\left(\boldsymbol{c}_t\right) \text{ (Hidden State)} \end{aligned}$$

Therefore, we have that

- Sigmoid layer: decide what linear combination of state/input to output

Additionally, we have that the tanh squashes the values between -1 and 1

- The output is used to filter a version of cell state!!!

Thus at the output layer and update state

We have

$$
\begin{aligned}
\boldsymbol{o}_t &= \sigma\left[W_o\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_o\right] \text{ (Output Gate)} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh\left(\boldsymbol{c}_t\right) \text{ (Hidden State)}
\end{aligned}
$$

Therefore, we have that

- Sigmoid layer: decide what linear combination of state/input to output

Additionally, we have that the tanh squashes the values between -1 and 1

- The output is used to filter a version of cell state!!!

Thus at the output layer and update state

We have

$$\begin{aligned}\boldsymbol{o}_t =& \sigma \left[W_o \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t \right] + \boldsymbol{b}_o \right] \text{ (Output Gate)} \\ \boldsymbol{h}_t =& \boldsymbol{o}_t \odot \tanh \left(\boldsymbol{c}_t \right) \text{ (Hidden State)}\end{aligned}$$

Therefore, we have that

- Sigmoid layer: decide what linear combination of state/input to output

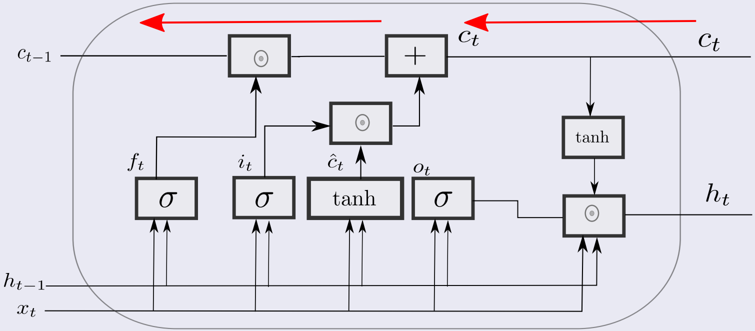
Additionally, we have that the tanh squashes the values between -1 and 1

- The output is used to filter a version of cell state!!!

Something nice about LSTM

Quite nice

- Backpropagation from c_t to c_{t-1} requires only elementwise multiplication!



LSTM Remarks

First

- It maintains a separate cell state from what is outputted

Second

- Use gates to control the flow of information
  - Forget gate tries to get rid of irrelevant information
  - Selectively update cell state
  - Output gate returns a filtered version of the cell state

Third

- Backpropagation from c_t to c_{t-1} requires only elementwise multiplication!

LSTM Remarks

First

- It maintains a separate cell state from what is outputted

Second

- Use gates to control the flow of information
  - Forget gate tries to get rid of irrelevant information
  - Selectively update cell state
  - Output gate returns a filtered version of the cell state

Third

- Backpropagation from c_t to c_{t-1} requires only elementwise multiplication!

LSTM Remarks

First

- It maintains a separate cell state from what is outputted

Second

- Use gates to control the flow of information
  - Forget gate tries to get rid of irrelevant information
  - Selectively update cell state
  - Output gate returns a filtered version of the cell state

Third

- Backpropagation from c_t to c_{t-1} requires only elementwise multiplication!

Achievements

LSTM achieved record results in natural language text compression

- http://www.mattmahoney.net/dc/text.html#1218

Unsegmented connected handwriting recognition

- Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J. (May 2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868

Finally

- Won the ICDAR handwriting competition (2009)

Achievements

LSTM achieved record results in natural language text compression

- http://www.mattmahoney.net/dc/text.html#1218

Unsegmented connected handwriting recognition

- Graves, A., Liwicki, M., Fernández, S., Bertolami, R.; Bunke, H., Schmidhuber, J. (May 2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868

Finally

- Won the ICDAR handwriting competition (2009)

Achievements

LSTM achieved record results in natural language text compression

- http://www.mattmahoney.net/dc/text.html#1218

Unsegmented connected handwriting recognition

- Graves, A., Liwicki, M., Fernández, S., Bertolami, R.; Bunke, H., Schmidhuber, J. (May 2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868

Finally

- Won the ICDAR handwriting competition (2009)

Right now

Something Notable

- As of 2016, major technology companies including Google, Apple, and Microsoft were using LSTM as fundamental components in new products.

Outline

History

They were proposed as a simplification of the LSTM

- In 2014, Kyunghyun Cho et al. put forward a simplified variant called Gated recurrent unit (GRU)

Something Notable

- The GRU is like a long short-term memory (LSTM) with forget gate...
  - but has fewer parameters than LSTM, as it lacks an output gate

History

They were proposed as a simplification of the LSTM

- In 2014, Kyunghyun Cho et al. put forward a simplified variant called Gated recurrent unit (GRU)

Something Notable

- The GRU is like a long short-term memory (LSTM) with forget gate...
  - but has fewer parameters than LSTM, as it lacks an output gate

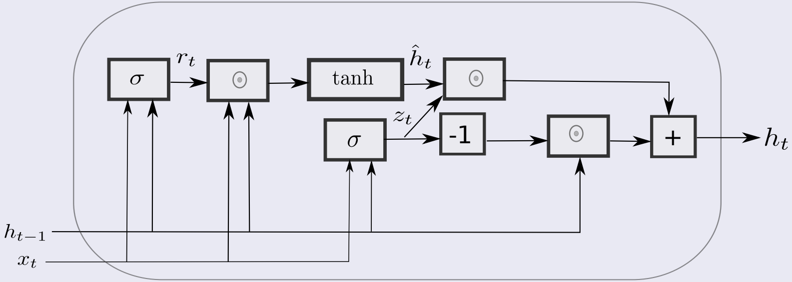
Gated Recurrent Units

Architecture

$$
\begin{aligned}
\boldsymbol{z}_t &= \sigma\left[W_z\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_z\right] \text{ (Update Gate)} \\
\boldsymbol{r}_t &= \sigma\left[W_r\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_r\right] \text{ (Reset Gate)} \\
\hat{\boldsymbol{h}}_t &= \tanh\left[W_o\left[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_h\right] \\
\boldsymbol{h}_t &= (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \hat{\boldsymbol{h}}_t
\end{aligned}
$$

Graphically, we have the architecture

GRU Architecture



Main Observations

There is a gate used to combine the state $\boldsymbol{h}_{t-1}$,

- The $\boldsymbol{z}_t$ gate that basically uses the information of the input and the previous state to decide how to update

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \hat{\boldsymbol{h}}_t$$

The intermediate step $\hat{h}_t$

- A bounded version of the possible state h_t

Main Observations

There is a gate used to combine the state $\boldsymbol{h}_{t-1}$,

- The $\boldsymbol{z}_t$ gate that basically uses the information of the input and the previous state to decide how to update

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \hat{\boldsymbol{h}}_t$$

The intermediate step $\hat{\boldsymbol{h}}_t$

- A bounded version of the possible state $\boldsymbol{h}_t$

Next

We have that a reset gate

$$\boldsymbol{r}_t = \sigma\left[W_r\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_r\right]$$

- To update

$$\hat{\boldsymbol{h}}_t = \tanh\left[W_o\left[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_h\right]$$

However

It has been shown that

- As shown by Gail Weiss, Yoav Goldberg, Eran Yahav, the LSTM is "strictly stronger" than the GRU

LSTM can perform unbounded counting[13]

- The GRU cannot.
  - It simulates a counting machine used for theoretical CS

Denny Britz, Anna Goldie, Minh-Thang Luong, Quoc Le of Google Brain

- LSTM cells consistently outperform GRU cells in "the first large-scale analysis of architecture variations for Neural Machine Translation."

However

It has been shown that

- As shown by Gail Weiss, Yoav Goldberg, Eran Yahav, the LSTM is "strictly stronger" than the GRU

LSTM can perform unbounded counting[13]

- The GRU cannot.
  - It simulates a counting machine used for theoretical CS

Denny Britz, Anna Goldie, Minh-Thang Luong, Quoc Le of Google Brain

- LSTM cells consistently outperform GRU cells in "the first large-scale analysis of architecture variations for Neural Machine Translation."

However

It has been shown that

- As shown by Gail Weiss, Yoav Goldberg, Eran Yahav, the LSTM is "strictly stronger" than the GRU

LSTM can perform unbounded counting[13]

- The GRU cannot.
  - It simulates a counting machine used for theoretical CS

Denny Britz, Anna Goldie, Minh-Thang Luong, Quoc Le of Google Brain

- LSTM cells consistently outperform GRU cells in "the first large-scale analysis of architecture variations for Neural Machine Translation."

Outline

Given that we want to do sequence modeling

Stock Options

Predict next phrase

- Question: If I am a man ?
  - Prediction: you are homo sapiens

Given that we want to do sequence modeling

Stock Options

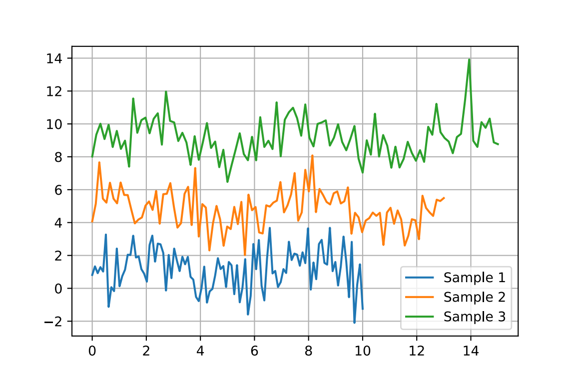
Predict next phrase

- Question: If I am a man ?
  - Prediction: you are homo sapiens

What do we have in this sequences of data?

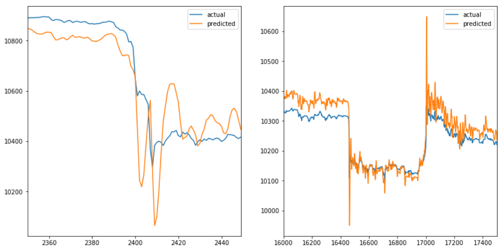
Sequences have different lengths

- We need to handle variable-length sequences



Furthermore

We need to track long-term dependencies



Not only that

Maintain information about order

- "We have a mother living in Yucatan, Mexico"

Share parameters across the sequence

- Do you remember the state h_t?

Not only that

Maintain information about order

- "We have a mother living in Yucatan, Mexico"

Share parameters across the sequence

- Do you remember the state $\boldsymbol{h}_t$?

However

There is a need to increase their power

- Given the amounts of data we have right know

Then there is a tendency to start using the Recurrent Neural Networks

- As cells to be stacked for bigger systems [14, 15]

This is based in the following idea [16]

- Hypothesis, hierarchical model can be exponentially more efficient at representing some functions than a shallow one.

However

There is a need to increase their power

- Given the amounts of data we have right know

Then there is a tendency to start using the Recurrent Neural Networks

- As cells to be stacked for bigger systems [14, 15]

This is based in the following idea [16]

- Hypothesis, hierarchical model can be exponentially more efficient at representing some functions than a shallow one.

However

There is a need to increase their power

- Given the amounts of data we have right know

Then there is a tendency to start using the Recurrent Neural Networks

- As cells to be stacked for bigger systems [14, 15]

This is based in the following idea [16]

- Hypothesis, hierarchical model can be exponentially more efficient at representing some functions than a shallow one.

In the case of RNN's

Certain Transitions are not Deep

- They are only results of a **linear projection** followed by an element-wise nonlinearity.

They are

- Hidden-to-hidden $h_{t-1} \to h_t$
- Hidden-to-output $h_t \to y_t$
- Input-to-hidden $x_{t-1} \to h_t$

Meaning

- They are all shallow in the sense that there exists no intermediate, nonlinear hidden layer.

In the case of RNN's

Certain Transitions are not Deep

- They are only results of a **linear projection** followed by an element-wise nonlinearity.

They are

- Hidden-to-hidden $\boldsymbol{h}_{t-1} \rightarrow \boldsymbol{h}_t$
- Hidden-to-output $\boldsymbol{h}_t \rightarrow \boldsymbol{y}_t$
- Input-to-hidden $\boldsymbol{x}_{t-1} \rightarrow \boldsymbol{h}_t$

Meaning

- They are all shallow in the sense that there exists no intermediate, nonlinear hidden layer.

In the case of RNN's

Certain Transitions are not Deep

- They are only results of a **linear projection** followed by an element-wise nonlinearity.

They are

- Hidden-to-hidden $\boldsymbol{h}_{t-1} \to \boldsymbol{h}_t$
- Hidden-to-output $\boldsymbol{h}_t \to \boldsymbol{y}_t$
- Input-to-hidden $\boldsymbol{x}_{t-1} \to \boldsymbol{h}_t$

Meaning

- They are all shallow in the sense that there exists no intermediate, nonlinear hidden layer.

Outline

Bengio et al. [17]

Gave the following Hypothesis

- In sampling algorithms (Markov Chains and MCMC techniques) suffer from a fundamental problem
  - Given unconnected or weakly connected regions of distributions

We have that

- it is difficult for the Markov chain to jump from one mode of the distribution to another, when these are separated by large low-density regions

This means that we have a slow mixing of samples

- In order to represent distributions

Bengio et al. [17]

Gave the following Hypothesis

- In sampling algorithms (Markov Chains and MCMC techniques) suffer from a fundamental problem
  - Given unconnected or weakly connected regions of distributions

We have that

- it is difficult for the Markov chain to jump from one mode of the distribution to another, when these are separated by large low-density regions

This means that we have a slow mixing of samples

- In order to represent distributions

Bengio et al. [17]

Gave the following Hypothesis

- In sampling algorithms (Markov Chains and MCMC techniques) suffer from a fundamental problem
  - Given unconnected or weakly connected regions of distributions

We have that

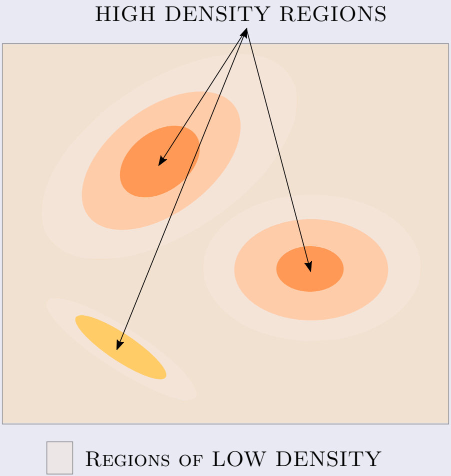
- it is difficult for the Markov chain to jump from one mode of the distribution to another, when these are separated by large low-density regions

This means that we have a slow mixing of samples

- In order to represent distributions

Example

A Big Problem



The Main Problem

We have that

- Slow mixing means that many consecutive samples tend to be correlated
  - They belong to the same mode of the mixture

Why?

- Jumping around in the MCMC method is quite slow and scarce

The Main Problem

We have that

- Slow mixing means that many consecutive samples tend to be correlated
  - They belong to the same mode of the mixture

Why?

- Jumping around in the MCMC method is quite slow and scarce

Implications in Learning Algorithms

Given that some form of sampling is at the core of many learning algorithms

- For example, to estimate the log-likelihood gradient

Therefore, at the beginning of learning

- Mixing is therefore initially easy

However as the model improves

- its corresponding distribution sharpens and mixing becomes slower

Implications in Learning Algorithms

Given that some form of sampling is at the core of many learning algorithms

- For example, to estimate the log-likelihood gradient

Therefore, at the beginning of learning

- Mixing is therefore initially easy

However as the model improves

- its corresponding distribution sharpens and mixing becomes slower

Implications in Learning Algorithms

Given that some form of sampling is at the core of many learning algorithms

- For example, to estimate the log-likelihood gradient

Therefore, at the beginning of learning

- Mixing is therefore initially easy

However as the model improves

- its corresponding distribution sharpens and mixing becomes slower

Basically

We have slow downs on the learning in shallow models

Outline

Therefore

We need to build deeper structures to reach more capabilities

- For example the vector representation of documents

Here a extra layer of representation can be used for doing representation

- For Example, Mikolov et al. [18]

Therefore

We need to build deeper structures to reach more capabilities

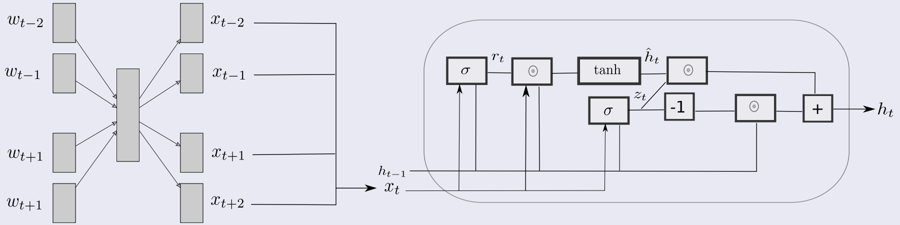
- For example the vector representation of documents

Here a extra layer of representation can be used for doing representation

- For Example, Mikolov et al. [18]

Basically a shallow network before the main architecture

An Encoder Layer before a GRU



The equations

They will look like

$$\begin{aligned}
\boldsymbol{z}_t =& \sigma\left[W_z\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_z\right] \text{ (Update Gate)} \\
\boldsymbol{r}_t =& \sigma\left[W_r\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_r\right] \text{ (Reset Gate)} \\
\hat{\boldsymbol{h}}_t =& \tanh\left[W_o\left[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_h\right] \\
\boldsymbol{h}_t =& (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \hat{\boldsymbol{h}}_t \\
\boldsymbol{x} =& \sigma\left(W_{oh}\boldsymbol{y}\right) \\
\boldsymbol{y} =& \sigma\left(W_{hi}\boldsymbol{w}\right)
\end{aligned}$$

Outline

Deep Transition Architectures

In a deep transition RNN (DT-RNN)

- At each time step the next state is computed by the sequential application of multiple transition layers.

For example in Nematus system [19]

- They use GRU transitions blocks under independent trainable parameters

With a Caveat

- The hidden state output is used as the input state on the next one

Deep Transition Architectures

In a deep transition RNN (DT-RNN)

- At each time step the next state is computed by the sequential application of multiple transition layers.

For example in Nematus system [19]

- They use GRU transitions blocks under independent trainable parameters

With a Caveat

- The hidden state output is used as the input state on the next one

Deep Transition Architectures

In a deep transition RNN (DT-RNN)

- At each time step the next state is computed by the sequential application of multiple transition layers.

For example in Nematus system [19]

- They use GRU transitions blocks under independent trainable parameters

With a Caveat

- The hidden state output is used as the input state on the next one

For example, at the encoder phase

For the i^{th} source word in the forward direction, we have $\boldsymbol{h}_i = \boldsymbol{h}_{i,L_s}$

$$
\begin{aligned}
\boldsymbol{h}_{i,1} &= GRU_1\left(\boldsymbol{x}_1, \boldsymbol{h}_{i-1,L_s}\right) \\
\boldsymbol{h}_{i,k} &= GRU_k\left(0, \boldsymbol{h}_{i,k-1}\right) \text{ for } 1 < k \leq L_s
\end{aligned}
$$

The sequence word is reversed and you have a backward state then

$$C = \left[\overrightarrow{\boldsymbol{h}}_{i,L_s}, \overleftarrow{\boldsymbol{h}}_{i,L_s}\right]$$

For example, at the encoder phase

For the i^{th} source word in the forward direction, we have $\boldsymbol{h}_i = \boldsymbol{h}_{i,L_s}$

$$\boldsymbol{h}_{i,1} = GRU_1\left(\boldsymbol{x}_1, \boldsymbol{h}_{i-1,L_s}\right)$$
$$\boldsymbol{h}_{i,k} = GRU_k\left(0, \boldsymbol{h}_{i,k-1}\right) \text{ for } 1 < k \leq L_s$$

The sequence word is reversed and you have a backward state then

$$C \equiv \left[\overrightarrow{\boldsymbol{h}}_{i,L_s}, \overleftarrow{\boldsymbol{h}}_{i,L_s}\right]$$

Then

Decoder phase uses the outputs from the previous GRU and something called attention (We will look at this latter)

$$
\begin{aligned}
\boldsymbol{s}_{j,1} &= GRU_1\left(\boldsymbol{y}_{j-1}, \boldsymbol{s}_{j-1}, L_t\right)\\
\boldsymbol{s}_{j,2} &= GRU_2\left(ATT, \boldsymbol{s}_{j-1}, L_t\right)\\
\boldsymbol{s}_{j,k} &= GRU_k\left(0, L_t\right) \text{ for } 2 < k \leq L_t
\end{aligned}
$$

Then, the target word state $s_j = s_{j,L_t}$

- It is used by a feed-forward neural network to predict the current target network

Then

Decoder phase uses the outputs from the previous GRU and something called attention (We will look at this latter)

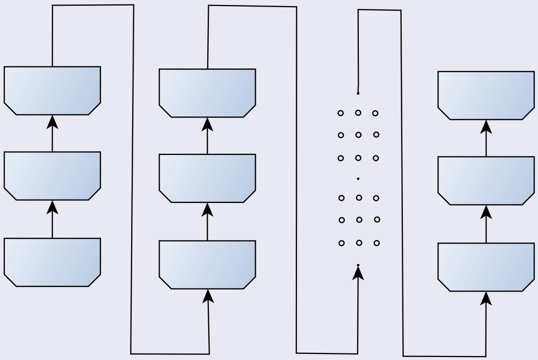
$$\boldsymbol{s}_{j,1} = GRU_1\left(\boldsymbol{y}_{j-1}, \boldsymbol{s}_{j-1}, L_t\right)$$
$$\boldsymbol{s}_{j,2} = GRU_2\left(ATT, \boldsymbol{s}_{j-1}, L_t\right)$$
$$\boldsymbol{s}_{j,k} = GRU_k\left(0, L_t\right) \text{ for } 2 < k \leq L_t$$

Then, the target word state $s_j \equiv s_{j,L_t}$

- It is used by a feed-forward neural network to predict the current target network

Deep Transition Decoder

We have the following depiction of the architecture



Outline

There are many other examples

Basically

- We are far from the classic methods as
  1. Autoregressive integrated moving average (ARMA)
  2. Auto Regressive Integrated Moving Average (ARIMA)
  3. etc

These RNN architectures are taking the prediction of time series

- To another level!!!

There are many other examples

Basically

- We are far from the classic methods as
  1. Autoregressive integrated moving average (ARMA)
  2. Auto Regressive Integrated Moving Average (ARIMA)
  3. etc

These RNN architectures are taking the prediction of time series

- To another level!!!

- O. L. R. Jacobs, "Introduction to control theory," 1974.
- A. Robinson and F. Fallside, *The utility driven dynamic error propagation network*.
  University of Cambridge Department of Engineering, 1987.
- P. J. Werbos *et al.*, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- G. Chen, "A gentle tutorial of recurrent neural network with error backpropagation," *arXiv preprint arXiv:1610.02583*, 2016.
- B. W. Bader and T. G. Kolda, "Algorithm 862: Matlab tensor classes for fast algorithm prototyping," *ACM Trans. Math. Softw.*, vol. 32, Dec. 2006.
- J. Pennington, S. S. Schoenholz, and S. Ganguli, "The emergence of spectral universality in deep networks," *arXiv preprint arXiv:1802.09979*, 2018.

R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent," *Backpropagation: Theory, architectures, and applications*, vol. 433, 1995.

R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural computation*, vol. 2, no. 4, pp. 490–501, 1990.

J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association*, 2010.

H.-G. Zimmermann, C. Tietz, and R. Grothmann, "Forecasting with recurrent neural networks: 12 tricks," in *Neural Networks: Tricks of the Trade*, pp. 687–707, Springer, 2012.

- S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- G. Weiss, Y. Goldberg, and E. Yahav, “On the practical computational power of finite precision rnns for language recognition,” *CoRR*, vol. abs/1805.04908, 2018.
- R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, “How to construct deep recurrent neural networks,” *arXiv preprint arXiv:1312.6026*, 2013.
- A. V. M. Barone, J. Helcl, R. Sennrich, B. Haddow, and A. Birch, “Deep architectures for neural machine translation,” *arXiv preprint arXiv:1707.07631*, 2017.
- Y. Bengio *et al.*, “Learning deep architectures for ai,” *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
- Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai, “Better mixing via deep representations,” in *International conference on machine learning*, pp. 552–560, 2013.

T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

R. Sennrich, O. Firat, K. Cho, A. Birch, B. Haddow, J. Hitschler, M. Junczys-Dowmunt, S. Läubli, A. V. M. Barone, J. Mokry, *et al.*, "Nematus: a toolkit for neural machine translation," *arXiv preprint arXiv:1703.04357*, 2017.