– UNDERSTANDING LSTM – A TUTORIAL INTO LONG SHORT-TERM MEMORY RECURRENT NEURAL NETWORKS

Alireza Yahyanejad

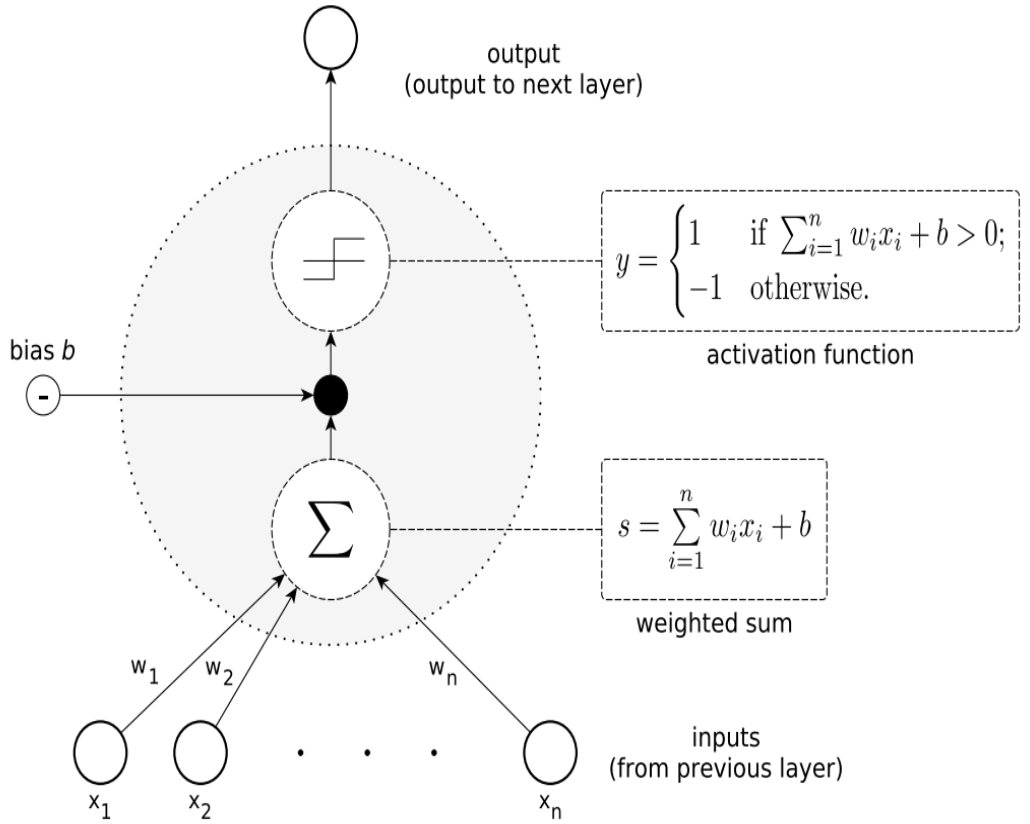
JUNE 2024

Introduction

- Most powerful dynamic classifiers
- Understanding how LSTM-RNNs evolved
- Why they work impressively well

The Perceptron

- Basic type of artificial neuron
- Number of external input links, a threshold, and a single external output link
- Additional input b, called bias
- The output of the perceptron is always Boolean



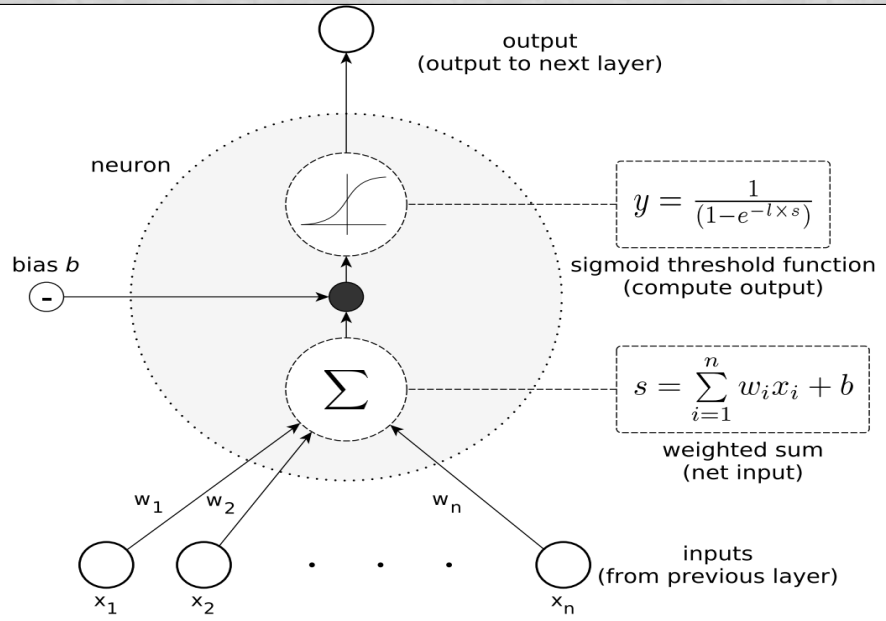
$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} W_i x_i + b > 0; \\ -1 & \text{otherwise.} \end{cases}$$

The Delta Learning Rule

- A radient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network.
- Training algorithms for perceptron:
  - Perceptron learning rule $W_i' = W_i + \Delta W_i$ with $\Delta W_i = \eta(d - y)x_i$
  - The delta learning rule: designed to handle linearly separable and linearly non-separable training examples
- Weights in the delta learning rule => gradient optimisation descent algorithm.
- The perceptron rule fails if the training examples are not linearly separable.

The Sigmoid Threshold Unit

- Similar to perceptron but uses a sigmoid function to calculate the output.
- output is "squashed" by a continuous function that ranges between 0 and 1.
- It maps a very large input domain onto a small range of outputs.
- representing non-linear functions.

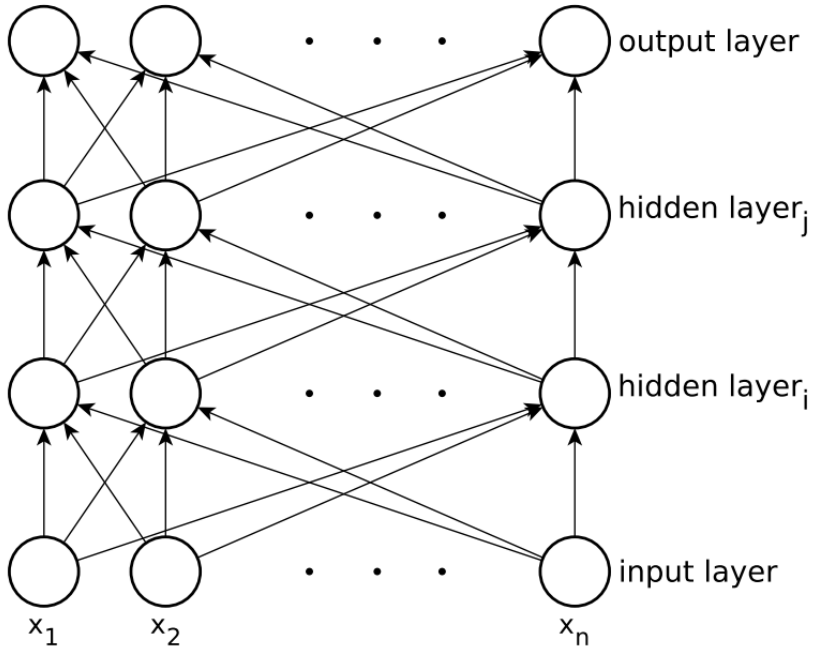


$$s = \sum_{i=1}^{n} W_i x_i + b$$

$$y = \frac{1}{(1 - e^{-l \times s})}$$

Feed-Forward Neural Networks and Backpropagation

- Hidden neurons: Neurons that are not directly connected to the environment, which are connected to other neurons.
- loop-free and fully connected: each neuron provides an input to each neuron in the following layer.
- Single-layer neural networks: The outputs of the input-layer neurons are directly connected to the neurons of the output layer.
- 1 if is above the threshold value otherwise -1.
- Multilayer feed-forward neural network input and output layers are connected via at least one hidden layer.
- Neural network learning technique is the error backpropagation algorithm. uses gradient descent to learn the weights.
- The activation function of the neuron is differentiable



Feed-Forward Neural Networks and Backpropagation

- non-input unit u: $y_u = \frac{1}{1+e^{-s_u}}$ with $s_u = z_u + b_u$ and $z_u = \sum_v W_{[v,u]} X_{[v,u]}, \quad \text{with } v \in \texttt{Pre}(u)$
$= \sum_v W_{[v,u]} y_v;$

- Calculates the error e_o for each output neuron using some error function to be minimized. $e_o = (d_o - y_o)$
- Overall error of network : $E = \frac{1}{2} \sum_{o \in O} e_o^2$

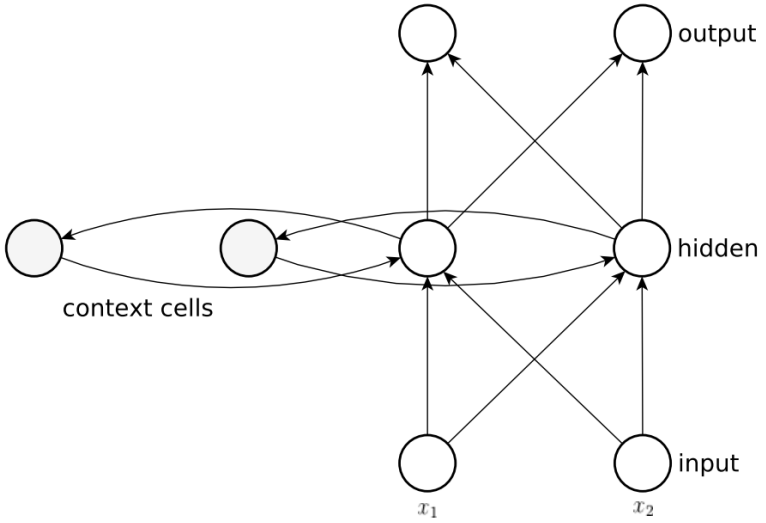
- Error signal: $\vartheta_o = (d_o - y_o) y_o (1 - y_o)$, $\Delta W_{[h,o]} = \eta \vartheta_o y_h$
$$\Delta W_{[i,h]} = -\eta \sum_o \left(\vartheta_o W_{[h,o]}\right) \frac{\partial y_h}{\partial s_h} \frac{\partial s_h}{\partial W_{[i,h]}}$$
- For a hidden unit h:
$$= \eta \sum_o \left(\vartheta_o W_{[h,o]}\right) y_h (1 - y_h) y_i.$$

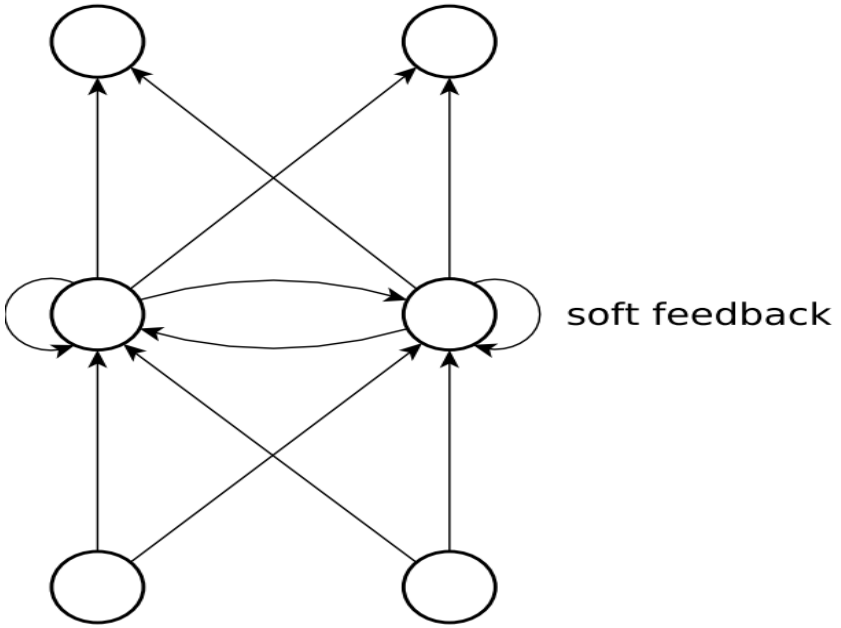
Feed-Forward Neural Networks and Backpropagation

- Error signal of the hidden unit h $\vartheta_h = \sum \left(\vartheta_o W_{[h,o]}\right) y_h(1 - y_h); \quad \text{with } o \in \mathbf{Suc}\,(h)$
- Uniform expression for weight $\Delta W_{[v,u]} = \eta \vartheta_u y_v$
- We calculate ΔW[v,u] again and again until all network outputs are within an acceptable range, or some other terminating condition is reached.

Recurrent Neural Networks

- Dynamic systems
- Have circular connections between higher- and lower-layer neurons and optional self-feedback.
- build a memory of time series events.
- Elman network:
  - Outputs of the hidden layer are saved in so-called 'context cells'.
  - Hidden layer receives input both from the input layer and the context cells.
  - Trained with standard error backpropagation
- Training RNNs: backpropagation through time and real-time recurrent learning.
- BPTT: offline learning algorithm, collect the data and then build the model.
- RTRL: online learning algorithm, data is collected online from the system and the model is learned during collection.





Backpropagation Through Time

- A gradient-based technique for training certain types of recurrent neural networks
- It can be used to train Elman networks. The algorithm was independently derived by numerous researchers.
- Output u at time τ: $y_u(\tau) = \mathtt{f}_u(z_u(\tau))$, f_u is differentiable, non-linear squashing function of the unit $u \in U$.
- weighted input: $$z_u(\tau+1) = \sum_l W_{[u,l]} X_{[l,u]}(\tau+1), \quad \text{with } l \in \mathtt{Pre}\,(u)$$
$$= \sum_v W_{[u,v]} y_v(\tau) + \sum_i W_{[u,i]} y_i(\tau+1)$$
- Summed error: $E_{total}(t',t) = \sum_{\tau=t'}^{t} E(\tau)$ where: $E(\tau) = \frac{1}{2}\sum_{u\in U}(e_u(\tau))^2$ $e_u(\tau) = \begin{cases} d_u(\tau) - y_u(\tau) & \text{if } u \in T(\tau), \\ 0 & \text{otherwise.} \end{cases}$
- Error signal: $\vartheta_u(\tau) = \begin{cases} \mathtt{f}'_u(z_u(\tau))e_u(\tau) & \text{if } \tau = t, \\ \mathtt{f}'_u(z_u(\tau))\left(\sum_{k\in U} W_{[k,u]}\vartheta_k(\tau+1)\right) & \text{if } t' \le \tau < t \end{cases}$

Backpropagation Through Time

- After the backpropagation computation is performed down to time t' => ΔW[u,v]

$$\Delta W_{[u,v]} = -\eta \frac{\partial E_{total}(t',t)}{\partial W_{[u,v]}} \quad \text{with} \quad \frac{\partial E_{total}(t',t)}{\partial W_{[u,v]}} = \sum_{\tau=t'}^{t} \vartheta_u(\tau) \frac{\partial z_u(\tau)}{\partial W_{[u,v]}}$$

$$= \sum_{\tau=t'}^{t} \vartheta_u(\tau) X_{[u,v]}(\tau).$$

Real-Time Recurrent Learning

- No need error propagation.
- Need sensitivity of the output.
- Memory depends on network size not input size.
- $E(\tau) = \frac{1}{2}\sum_{k\in U}(d_k(\tau) - y_k(\tau))^2$ where d_k(τ) is a label for non input k at time τ.

- Weight changes: $\Delta W_{[u,v]} = \sum_{\tau=t'+1}^{t} \Delta W_{[u,v]}(\tau)$ and $\Delta W_{[u,v]}(\tau) = -\eta \sum_{k\in U} \frac{\partial E(\tau)}{\partial y_k(\tau)} \frac{\partial y_k(\tau)}{\partial W_{[u,v]}}$

$$= -\eta \sum_{k\in U} (d_k(\tau) - y_k(\tau)) \left(\frac{\partial y_k(\tau)}{\partial W_{[u,v]}}\right).$$

- Second factor: $p^k_{uv}(\tau) = \frac{\partial y_k(\tau)}{\partial W_{[u,v]}}$, measures the sensitivity of the output of unit k at time τ

to a small change in the weight W[u,v]

Real-Time Recurrent Learning

- Output: $y_k(t+1) = \mathtt{f}_k(z_k(t+1))$

- Weighted input: $$z_k(t+1) = \sum_l W_{[k,l]} X_{[k,l]}(t+1), \quad \text{with } l \in \mathtt{Pre}(k)$$
$$= \sum_{v \in U} W_{[k,v]} y_v(t) + \sum_{i \in I} W_{[k,i]} y_i(t+1).$$

- $p^k_{uv}(t+1)$= $$\mathtt{f}'_k(z_k(t+1)) \left[\delta_{uk} X_{[u,v]}(t+1) + \sum_{l \in U} W_{[k,l]} p^l_{uv}(t) \right]$$

- Kronecker delta: $$\delta_{uk} = \begin{cases} 1 & \text{if } u = k \\ 0 & \text{if } otherwise, \end{cases}$$

- $p^k_{uv}(\tau)$ uses the values of W[u,v] at t′ , and not values in-between t′ and τ .

Long Short-Term Neural Networks

- Gradient based method, solution for vanishing error problem.
- Can learn bridge minimal time lags of more than 1,000 discrete time steps.
- Solution uses constant error carousels => enforce a constant error flow within special cells.
- local error: $\vartheta_u(\tau) = \mathtt{f}'_u(z_u(\tau))W_{[u,u]}\vartheta_u(\tau+1)$
- To ensure a constant error flow through u: $\mathtt{f}'_u(z_u(\tau))W_{[u,u]} = 1.0$
- By integration: $\mathtt{f}_u(z_u(\tau)) = \dfrac{z_u(\tau)}{W_{[u,u]}}$
- f_u must be linear, and that u's activation must remain constant over time:

$$y_u(\tau+1) = \mathtt{f}_u(z_u(\tau+1)) = \mathtt{f}_u(y_u(\tau)W_{[u,u]}) = y_u(\tau)$$

- This is ensured by using the identity function f_u = id, and by setting W[u,u] = 1.0. This preservation of error is called the constant error carousel (CEC)

LSTM – Memory Blocks

- CEC connects to both itself and other units in neural network.
- These connections can lead to conflicting weight update signals.
- To manage these connections, LSTM introduces input and output gates.
- Input gates scale signals from the network to the memory cell using a sigmoid activation function, protecting the cell from irrelevant signals.
- Output gates control access to the memory cell contents, shielding other cells from disturbances.
- These gates allow or deny the constant error flow through the CEC, resulting in a more complex LSTM unit known as a memory block.

Training LSTM-RNNs - the Hybrid Learning Approach

- Use BPTT to train network components located after cells.
- Use RTRL to train network components located before and including cells.
- Latter units work with RTRL because there are some partial derivatives that need to be computed during every step.
- Each step has a forward pass and a backward pass.
- Forward pass: output/activation of all units are calculated.
- Backward pass: the calculation of the error signals for all weights is performed.

The Forward Pass

- m_c be the c-th memory cell in memory block.
- In LSTM, memory block m is associated with one input gate in_m and one output gate out_m .
- activation y_{in_m} of the input gate in_m : $y_{\mathtt{in}_m}(\tau+1) = \mathtt{f}_{\mathtt{in}_m}(z_{\mathtt{in}_m}(\tau+1))$
- input gate input z_{in_m} (τ + 1) = $\sum_{v\in U} W_{[\mathtt{in}_m,v]} y_v(\tau) + \sum_{i\in I} W_{[\mathtt{in}_m,i]} y_i(\tau+1)$
- activation of the output gate out_m : $y_{\mathtt{out}_m}(\tau+1) = \mathtt{f}_{\mathtt{out}_m}(z_{\mathtt{out}_m}(\tau+1))$
- output gate input: z_{out_m}(τ + 1) = $\sum_{v\in U} W_{[\mathtt{out}_m,v]} y_v(\tau) + \sum_{i\in I} W_{[\mathtt{out}_m,i]} y_i(\tau+1)$
- f_{in_m} = f_{out_m} = f, where $\mathtt{f}(s) = \frac{1}{1+e^{-s}}$, its range [0,1]. So, the input for the memory cell will only be able to pass if the signal at the input gate is sufficiently close to '1'.
- weighted input z_{m_c} for the memory cell m_c : z_{m_c} (τ + 1) = $\sum_{v\in U} W_{[m_c,v]} y_v(\tau) + \sum_{i\in I} W_{[m_c,i]} y_i(\tau+1)$
- $s_{m_c}(\tau+1) = s_{m_c}(\tau) + y_{\mathtt{in}_m}(\tau+1)\mathrm{g}(z_{m_c}(\tau+1))$ with s_{m_c} (0) = 0

The Forward Pass

- non-linear squashing function for the cell input: $\mathrm{g}(z) = \frac{4}{1+e^{-z}} - 2$ with range [-2,2]
- Output: $y_{m_c}(\tau+1) = y_{\mathrm{out}_m}(\tau+1)\mathrm{h}(s_{m_c}(\tau+1))$
- non-linear squashing function: $\mathrm{h}(z) = \frac{2}{1+e^{-z}} - 1$ with range [–1, 1].
- activation of the output unit o: $y_o(\tau+1) = \mathrm{f}_o(z_o(\tau+1))$ where $z_o(\tau+1) = \sum_{u \in U-G} W_{[o,u]} y_u(\tau+1)$

And G is the set of gate units.

Forget Gates

- In a standard LSTM, the self-connection has a fixed weight of '1' to preserve the cell state.
- Cell states s_m tend to grow linearly in continuous input streams, causing the memory cell to lose its memorizing capability.
- Manually resetting cell states isn't practical for continuous inputs.
- An adaptive forget gate can address this issue by resetting the internal state when needed.
- Replace the fixed weight '1.0' of the self-connection with a multiplicative.
- Initialize input/output gate biases with negative values and forget gate weights with positive values.
- Initially, the forget gate activation is close to '1.0', allowing the cell to behave like a standard LSTM memory cell and prevent premature forgetting.

Backward Pass

- LSTM combines elements from BPTT and RTRL.
- Units are separated into two types:
  - Units with weight changes computed using BPTT:
    - Output units
    - Hidden units
    - Output gates
  - Units with weight changes computed using RTRL:
    - Input gates
    - Forget gates
    - Cells

Complexity

- B, C, In, Out be the number of memory blocks, memory cells in each block, input units and output units, respectively.
- Complexity of these connections: O(B^2. C^2)
- Complexity of connections from input units: O (In. B. S)
- Complexity of connections from output units: O (Out. B. S)
- LSTM's computational complexity per step and weight is O(1) because the number of weight updates is bounded by the number of connections.

Strengths and limitations of LSTM-RNNs

- LSTM excels in tasks requiring long-term memory with limited data.
- This is due to memory blocks, which:
  - Use input and output gates to control access and prevent irrelevant information from entering or leaving.
  - Include a forget gate to reset cell states when information becomes irrelevant.
  - Enable continuous prediction by allowing cells to forget previous states, preventing biases.
- Increasing network size uniformly is not effective. Modularisation is suggested for effective learning, but the process is not clearly defined.

Bidirectional LSTM

- A type of LSTM that consists of two LSTMs: one that processes the input sequence in a forward direction and another in a backward direction.
- Structure:
  - Two separate LSTM layers: one for forward pass and another for backward pass.
  - Outputs from both LSTMs are concatenated or combined at each time step.
- Advantages:
  - Captures dependencies from both past and future contexts in the sequence.
  - Improves performance for tasks where context from both directions is beneficial (e.g., speech recognition, text translation).
- Applications: Natural Language Processing (NLP), Speech recognition, Time series prediction

Grid LSTM

- An extension of the standard LSTM to handle multi-dimensional data by creating a grid structure of LSTM cells.
- Structure:
  - Multiple LSTM cells arranged in a grid, allowing interactions between cells along different dimensions.
  - Each cell can influence and be influenced by neighboring cells in different directions.
- Advantages:
  - Efficiently captures multi-dimensional dependencies.
  - Suitable for data with spatial and temporal dimensions.
- Applications: Image and video processing, Multidimensional time series analysis

Stacked LSTM

- A deep LSTM architecture where multiple LSTM layers are stacked on top of each other.
- Structure:
  - Several LSTM layers, with the output of one layer serving as the input to the next layer.
- Advantages:
  - Learns hierarchical feature representations.
  - Increases the model's capacity to capture complex patterns.
- Applications: NLP tasks (e.g., language modeling, text generation), Speech recognition, Complex time series forecasting

Multidimensional LSTM (MD-LSTM)

- A generalization of LSTM to handle multi-dimensional input data.
- Structure:
  - LSTM cells that operate across multiple dimensions simultaneously (e.g., 2D or 3D grids).
- Advantages:
  - Effectively captures dependencies across multiple dimensions.
  - More suited for data that naturally exists in higher dimensions.
- Applications: Image processing (e.g., handwriting recognition), Video analysis, Multidimensional time series forecasting

Grid LSTM Blocks

- A modular approach to Grid LSTM, where the grid structure is divided into smaller blocks.
- Structure:
  - Each block contains multiple LSTM cells arranged in a grid.
  - Blocks can be processed in parallel or sequentially, enhancing computational efficiency.
- Advantages:
  - Scalable and flexible for large-scale data.
  - Enhances parallel processing capabilities.
- Applications: Large-scale image and video processing, Complex spatial-temporal data analysis

Gated Recurrent Unit (GRU)

- A simplified version of LSTM that uses fewer gates and simplifies the hidden state representation.
- Structure:
  - Combines the forget and input gates into an update gate.
  - Uses a reset gate to control the flow of information.
- Advantages:
  - Computationally more efficient than LSTM due to fewer parameters.
  - Often performs similarly or better than LSTM on various tasks.
- GRU units do not have a memory cell like LSTM.

Gated Recurrent Unit (GRU)

Activation $y_{\text{res}_u}(\tau+1) = f_{\text{res}_u}(s_{\text{res}_u}(\tau+1))$

f_{res_u} is the squashing function of the reset gate, typically a sigmoid function.
$s_{\text{res}_u}(\tau+1) = z_{\text{res}_u}(\tau+1) + b_{\text{res}_u}$ represents the state of the reset gate res_u at time $\tau+1$.
$z_{\text{res}_u}(\tau+1) = \sum_{h\in H} W[\text{res}_u, h] y_h(\tau) + \sum_{i\in I} W[\text{res}_u, i] y_i(\tau+1)$ denotes the weighted input of the reset gate at time $\tau+1$.

Activation $y_{\text{upd}_u}(\tau+1) = f_{\text{upd}_u}(s_{\text{upd}_u}(\tau+1))$

f_{upd_u} represents the squashing function of the update gate, typically a sigmoid function.
$s_{\text{upd}_u}(\tau+1) = z_{\text{upd}_u}(\tau+1) + b_{\text{upd}_u}$ indicates the state of the update gate upd_u at time $\tau+1$.
$z_{\text{upd}_u}(\tau+1) = \sum_{h\in H} W[\text{upd}_u, h] y_h(\tau) + \sum_{i\in I} W[\text{upd}_u, i] y_i(\tau+1)$ defines the weighted input of the update gate at time $\tau+1$.

Associated candidate activation

$$\widetilde{y}_u(\tau+1) = \mathtt{f}_u\left(\underbrace{\sum_{i\in I} W_{[u,i]} y_i(\tau+1)}_{\text{External input at time } \tau+1} + \underbrace{y_{\mathtt{res}_u}(\tau+1) \sum_{h\in H} \left(W_{[u,h]} y_h(\tau)\right)}_{\text{Gated recurrent connection}} + \underbrace{b_u}_{\text{Bias}}\right)$$

Applications of LSTM-RNN

- **Early Learning Tasks:** LSTM excels in recalling numbers, learning languages, and precise timing.
- **Cognitive Learning Tasks:** LSTM-RNNs are effective in speech, handwriting, and machine translation.
- **Speech Recognition:** BLSTM-CTC outperforms HMMs in speech recognition.
- **Handwriting Recognition:** BLSTM-CTC excels in online and offline handwriting recognition.
- **Machine Translation:** RNN Encoder-Decoder improves machine translation.
- **Image Processing:** LSTM models enhance image classification and description.
- **Other Learning Tasks:** LSTM-RNNs are used for protein prediction, music generation, and network security.

Thanks for your attention