Machine Learning

Recurrent Neural Networks

Gabriele Russo Russo   Francesco Lo Presti

Laurea Magistrale in Ingegneria Informatica – A.Y. 2023/24

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Dipartimento di Ingegneria Civile e Ingegneria Informatica

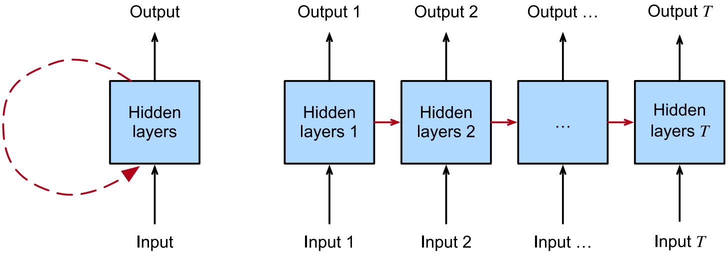
Learning with Sequences

- So far, we have focused primarily on fixed-length data
  - e.g., input vector $\boldsymbol{x}$ consisting of a fixed number of features
  - e.g., raw pixel values at each coordinate in an image

- Many tasks require dealing with sequential data
- Not only input sequences:
  - Input: time series prediction, video analysis, musical information retrieval
  - Output: image captioning, speech synthesis, music generation
  - Both: text translation

Recurrent Neural Networks

- Recurrent neural networks (RNNs): DL models that capture the dynamics of sequences via **recurrent connections**
  - cycles in the network of nodes
- RNNs are *unrolled* across time steps, with the same underlying parameters applied at each step



Remark

- RNNs work well with sequential data, but they are not the only kind of DNN used for these tasks
- e.g., CNNs have been successfully applied to sequences (e.g., WaveNet to generate synthetic audio from raw audio)
- Recently, Transformers have been shown to outperform RNNs in many cases

Hidden State

- Given an input sequence $x_1, x_2, \ldots, x_{t-1}$, we are interested in

$$P(x_t | x_{t-1}, x_{t-2}, \ldots, x_1) = ??$$

- Storing the full (or most recent) sequence of past observations would be unfeasible and would require an exponentially large number of parameters
- Therefore, we exploit a hidden state h_{t-1} to capture sequence information up to $t-1$

$$P(x_t | x_{t-1}, x_{t-2}, \ldots, x_1) \approx P(x_t | h_{t-1})$$

$$h_t = f(x_t, h_{t-1})$$

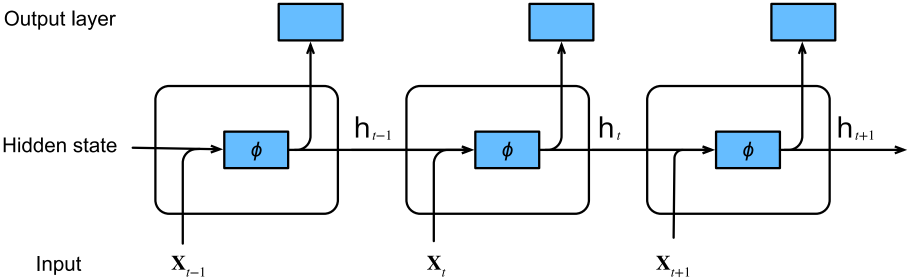
Recurrent Layers

- Recurrent (hidden) layers are characterized by hidden states
  - "hidden" refers to different concepts in this sentence
- Consider an input example at time t from the sequence: $\boldsymbol{x}_t$
- The output of the hidden layer is computed as

$$\boldsymbol{h}_t = \phi\left(\boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}\right)$$

- ...and this is the hidden state of the RNN
- Recurrent: $\boldsymbol{h}_t$ is defined in terms of $\boldsymbol{h}_{t-1}$
- $\phi()$ is usually tanh or sigmoid for RNNs

Recurrent Layers (2)

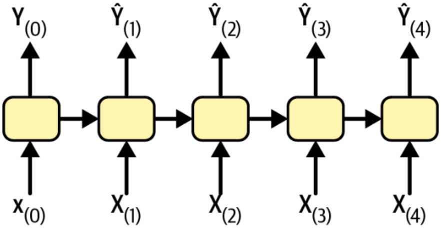


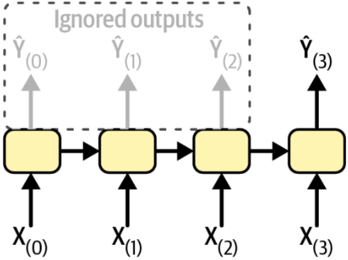
Note: alternative RNN models exist (e.g., in the model we consider hidden state and output of the layer are the same, but they could be computed differently)

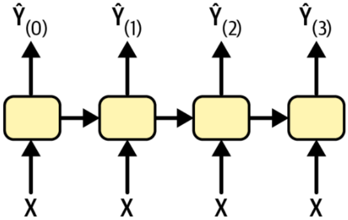
Input and Output of RNNs

- Sequence-to-sequence: a sequence of inputs produces a sequence of outputs
  - e.g., time series forecasting: you feed data over the last N days, and output the series shifted by one day into the future
- Sequence-to-vector: feed a sequence of inputs and ignore all outputs except for the last one
  - e.g., given a social media post, output a sentiment score
- Vector-to-sequence: feed the RNN the same input vector over and over again at each time step and let it output a sequence
  - e.g., given an image, produce a caption

Input and Output of RNNs (2)







Training RNNs

- Recall forward + backward propagation for feedforward NNs
- Forward propagation relatively straightforward
- Applying backpropagation in RNNs is called backpropagation through time (Werbos, 1990)
- We expand (or unroll) the computational graph of an RNN one time step at a time
  - Unrolled RNN is essentially a feedforward NN, with the same parameters repeated throughout the unrolled network
  - We can apply the chain rule as usual
  - The gradient w.r.t. each parameter must be summed across all the parameter occurrences

Backpropagation Through Time (BPTT)

Consider a simplified RNN model (identity act. function, no bias)

- $\boldsymbol{h}_t = \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1}$
- $\boldsymbol{y}_t = \boldsymbol{W}_y \boldsymbol{h}_t$
- Loss function: $\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \ell(\boldsymbol{t}_t, \boldsymbol{y}_t)$

At any time step t, it is straightforward to compute:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t} = \frac{1}{T} \frac{\partial \ell(\boldsymbol{t}_t, \boldsymbol{y}_t)}{\partial \boldsymbol{y}_t} \tag{1}$$

Backpropagation Through Time (2)

- The weights $\boldsymbol{W}_y$ are used at every time step to compute the output given $\boldsymbol{h}_t$
- Recall that $\mathcal{L}$ depends on $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$
- To compute the gradient w.r.t. $\boldsymbol{W}_y$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_y} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{W}_y} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial \ell(\boldsymbol{t}_t, \boldsymbol{y}_t)}{\partial \boldsymbol{y}_t} \boldsymbol{h}_t^\top \tag{2}$$

Backpropagation Through Time (3)

- Next, we need the gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{h}_t$
- For the final time step T, it is easy (the loss only depends on $\boldsymbol{h}_T$ via $\boldsymbol{y}_T$)

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_T} \frac{\partial \boldsymbol{y}_T}{\partial \boldsymbol{h}_T} = \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_T} \tag{3}$$

Backpropagation Through Time (4)

For any time step $t < T$, things become trickier...

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t}\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t+1}}\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} = \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t} + \boldsymbol{W}_h^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t+1}} = \tag{4}$$

$$= \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t} + \boldsymbol{W}_h^\top \left(\boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{t+1}} + \boldsymbol{W}_h^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t+2}} \right) = \tag{5}$$

$$= \sum_{i=0}^{T-t} \left(\boldsymbol{W}_h^\top \right)^i \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{t+i}} \tag{6}$$

You can see that for long sequences we need to compute large powers of $\boldsymbol{W}_h$ (eigenvalues < 1 may vanish, and those > 1 may diverge!)

Backpropagation Through Time (5)

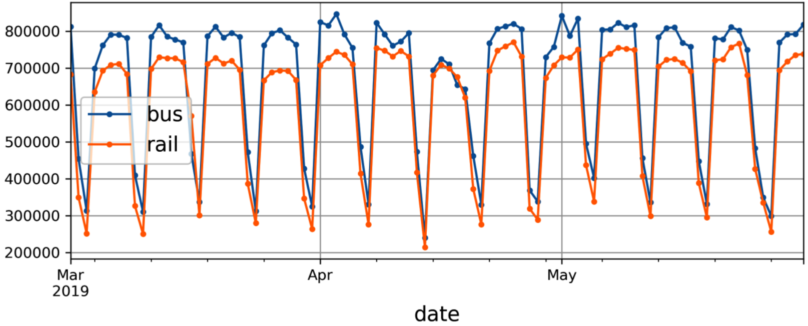
Finally, we can compute:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_h} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_h} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} \boldsymbol{h}_{t-1}^{\top} \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_x} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_x} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} \boldsymbol{x}_{t}^{\top} \tag{8}$$

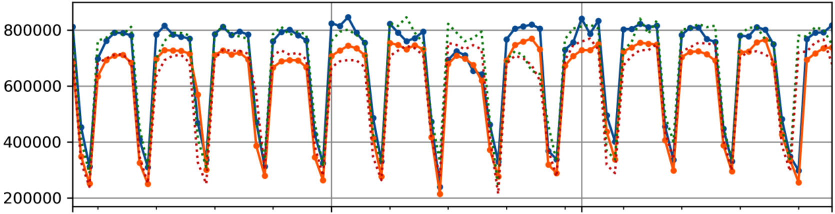
Example: Time Series Forecasting

- You have just been hired as a data scientist by Chicago's Transit Authority
- First task: build a model capable of forecasting the number of passengers that will ride on bus and rail the next day
- You have access to daily ridership data since 2001



Example: Preliminary Observations

- We are facing a multivariate time series
- Looking at data, we note that a similar pattern is clearly repeated every week (weekly seasonality)
- Exploiting this observation, we could forecast tomorrow's ridership by just copying the values from a week earlier (naive forecasting): less than 10% mean error!



Example: Using a RNN

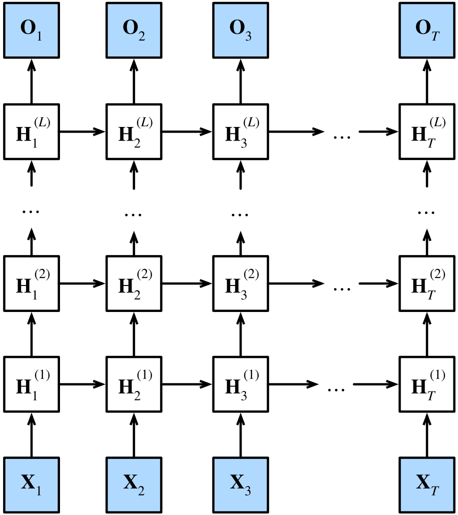
- We will use a RNN to (hopefully) outperform naive forecasting and a SARIMA model
  - SARIMA belongs to a family of widely used models for time series forecasting

rnn_timeseries.ipynb (part 1)

Deep RNNs

- So far, we defined RNNs consisting of a sequence input, a single hidden RNN layer, and an output layer
- This NN is already deep in some sense: inputs from the first time step can influence outputs 100-1000s steps later
- We may also wish to retain the ability to express complex input-output relationships at a given time step
- We can stack RNN layers on top of each other

Deep RNNs (2)



rnn_timeseries.ipynb (part 2)

Handling Long Sequences

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} = \sum_{i=0}^{T-t} \left(\boldsymbol{W}_h^\top\right)^i \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{t+i}}$$

- Long sequences cause large powers of $\boldsymbol{W}_h$ to be computed (eigenvalues < 1 may vanish, and those > 1 may diverge!)
- The problem of vanishing and exploding gradients is one of the key challenges for RNN training
- Good parameter initialization, faster optimizers, dropout can help, but do not solve the problem
- Non-saturating activ. functions (e.g., ReLU) worsen the situation (tanh is a popular choice for RNNs)
  - If activation value is increased by weights update at the first step, it is further increased at every step!

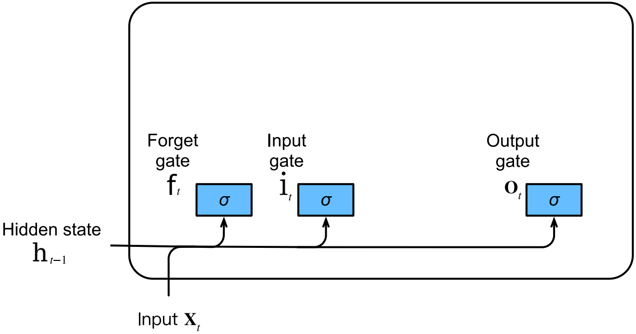
LSTM Cells

- Traversing RNNs, some information is lost at each time step
- After a while, state may contain no trace of the first inputs!
- Long short-term memory (LSTM): one of the first and most successful techniques to deal with vanishing gradients
- Published in 1997; become dominant model for sequence learning from 2011 until the rise of Transformer models in 2017
- Simple RNNs have long-term memory in the form of weights, which change slowly during training; RNNs also have short-term memory in the form of ephemeral activations, which pass from each node to successive nodes
- The LSTM model introduces an intermediate type of storage via memory cells, which replace recurrent nodes

Memory Cell

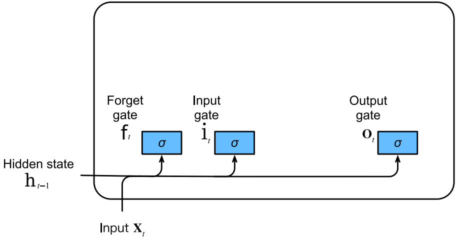
- Each memory cell is equipped with an internal state and a number of multiplicative gates
- Gates determine whether
  1. input should impact the internal state (input gate)
  2. internal state should be reset (forget gate)
  3. internal state should impact cell’s output (output gate)

LSTM Gates



- 3 fully connected layers with sigmoid activation compute the values of the input, forget, and output gates
- Values of the gates range in $(0, 1)$
  - e.g., forget gate determines *how much* of the current state should be kept

LSTM Gates (2)



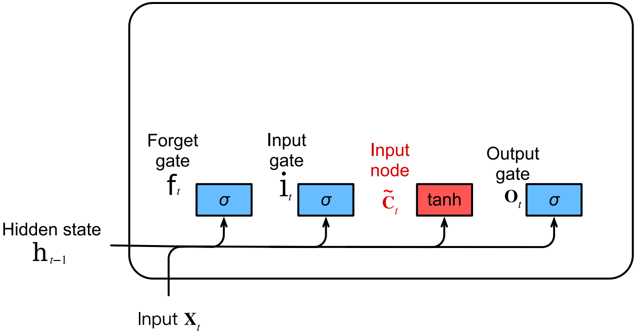
- Consider $\boldsymbol{x} \in \mathbb{R}^d$ and h hidden units
- Hidden state is $\boldsymbol{h}_{t-1} \in \mathbb{R}^h$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{x}_t \boldsymbol{W}_{xi} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) \tag{9}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{x}_t \boldsymbol{W}_{xf} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \tag{10}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{x}_t \boldsymbol{W}_{xo} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o) \tag{11}$$

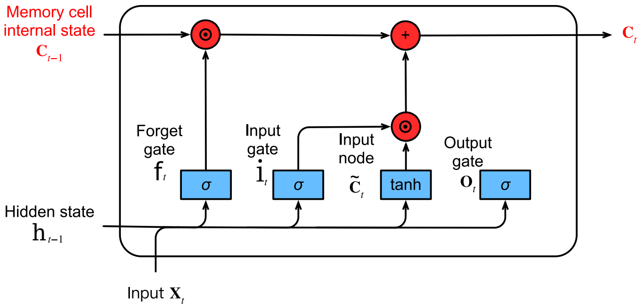
Input Node



- $\tilde{\boldsymbol{c}}_t \in (-1, 1)^h$ denotes the cell input node

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{x}_t \boldsymbol{W}_{xc} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c)$$

Cell Internal State

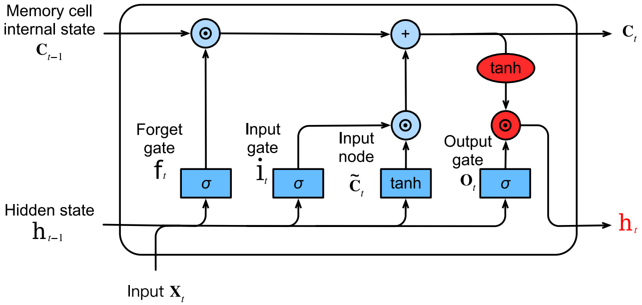


- $\boldsymbol{c}_t \in \mathbb{R}^h$ is the cell internal state

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

where $\odot$ denotes the element-wise product

Hidden State



- The hidden state $\boldsymbol{h}_t \in (-1, 1)^h$ is the output of the cell, as seen by next layer

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot tanh(\boldsymbol{c}_t)$$

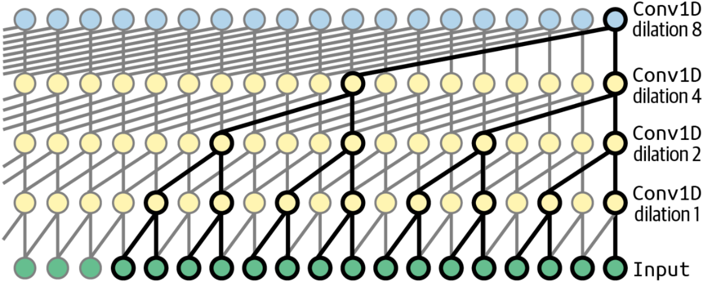
Example: Time Series Forecasting

- Last part of the notebook uses LSTM for the ridership forecasting
- Note that LSTM do not provide significant benefits in this simple example, but they generally work much better than simple RNNs in many tasks

rnn_timeseries.ipynb (part 3)

WaveNet

- CNN proposed in 2016; excellent performance on 1D audio sequences (e.g., to generate synthetic voices)
- Idea: stacking 10 convolutional layers to obtain a large enough receptive field, able to capture long-term patterns at higher layers
- More details: Residual blocks and gated activations



References

- D2L: 9.*, 10.1, 10.3
- Hands-on ML: Chapter 15