CSCI E-89B Introduction to Natural Language Processing

Harvard Extension School

Dmitry Kurochkin

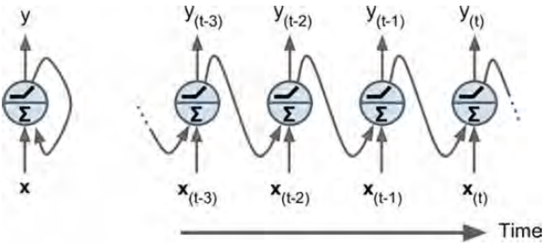
Fall 2025
Lecture 2

Contents

Contents

RNN: Recurrent Neuron

Recurrent Neuron:

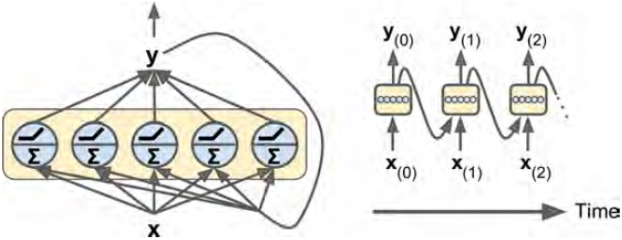


Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Contents

RNN: Layer of Recurrent Neurons

Layer of Recurrent Neurons:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Layer of Recurrent Neurons: Keras

Simple RNN in Keras:

```
n_features = 2
n_timesteps = 200

model = models.Sequential()
model.add(layers.SimpleRNN(3, activation='relu', input_shape=(n_timesteps,n_features)))
model.add(layers.Dense(1, activation='linear'))

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_6 (SimpleRNN)      (None, 3)                 18
_________________________________________________________________
dense_6 (Dense)               (None, 1)                 4
=================================================================
Total params: 22
Trainable params: 22
Non-trainable params: 0
```

Contents

Vanishing Gradient Problem

- *Explanation:*
  - Gradients become exponentially small during Backpropagation Through Time (BPTT).
  - This hinders the learning of long-term dependencies.
- *Impact:*
  - Inability to capture long-range information.
  - Challenges in training deeper networks (both in terms of temporal depth and layer depth).
  - Poor performance on tasks requiring long-term memory.

Exploding Gradient Problem

- *Explanation:*
  - Gradients can grow exponentially, causing large weight updates.
  - This leads to instability in training.
- *Impact:*
  - Unpredictable training with oscillating loss values.
  - Necessity of gradient clipping techniques.

Key Solutions for Improving RNN Training

- **Weight Initialization**
  - Proper techniques (Xavier, He initialization) help stabilize training and prevent vanishing/exploding gradients.
- **Regularization Techniques**
  - **Dropout**: Prevents overfitting by randomly dropping neurons during training.
  - **Batch Normalization**: Normalizes inputs for each mini-batch to reduce internal covariate shift.
  - **Layer Normalization**: Normalizes across the features, effective for RNNs.
- **Learning Rate Scheduling**
  - Dynamically adjusts the learning rate during training with techniques such as learning rate decay, cyclical learning rates, Adam, RMSprop.
- **Gradient Clipping**
  - Prevents exploding gradients by capping gradient values during training.

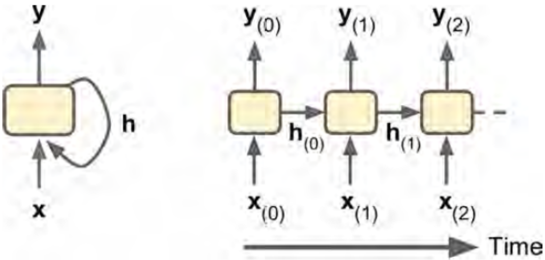
Key Solutions for Improving RNN Training (Continued)

- **Advanced RNN Architectures**
  - **LSTM (Long Short-Term Memory)**: Utilizes memory cells and gating mechanisms to retain long-term information, mitigating vanishing gradient issues.
  - **GRU (Gated Recurrent Unit)**: Simpler structure than LSTM with combined gates, offering computational efficiency while addressing gradient issues.

Contents

RNN: Memory Cells

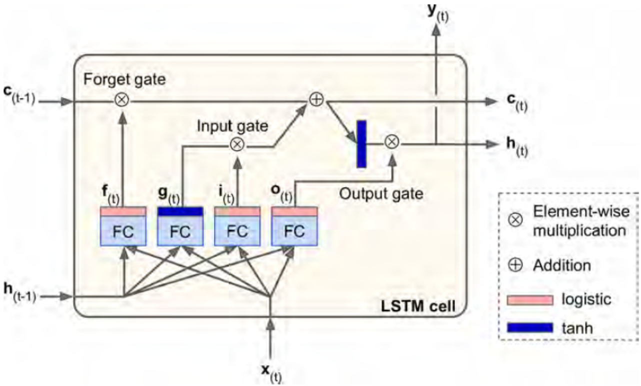
Layer of Recurrent Neurons:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Long Short-Term Memory (LSTM) Cell

LSTM Cell:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Long Short-Term Memory (LSTM) Cell

LSTM Cell Model:

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{i}\right) \\
\mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{f}\right) \\
\mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{o}\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{g}\right) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)
\end{aligned}
$$

- $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xg}$ are the weight matrices of each of the four layers for their connection to the input vector $\mathbf{x}_{(t)}$.
- $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, and $\mathbf{W}_{hg}$ are the weight matrices of each of the four layers for their connection to the previous short-term state $\mathbf{h}_{(t-1)}$.
- $\mathbf{b}_{i}$, $\mathbf{b}_{f}$, $\mathbf{b}_{o}$, and $\mathbf{b}_{g}$ are the bias terms for each of the four layers. Note that TensorFlow initializes $\mathbf{b}_{f}$ to a vector full of 1s instead of 0s. This prevents forgetting everything at the beginning of training.

Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Applications of LSTM Cells

- greatly improved speech recognition on over 4 billion Android phones (since mid 2015)
- greatly improved machine translation through Google Translate (since Nov 2016)
- greatly improved machine translation through Facebook (over 4 billion LSTMbased translations per day as of 2017)
- Siri and Quicktype on almost 2 billion iPhones (since 2016)
- generating answers by Amazon's Alexa and numerous other similar applications.

Long Short-Term Memory (LSTM) Cell: Keras

LSTM in Keras:

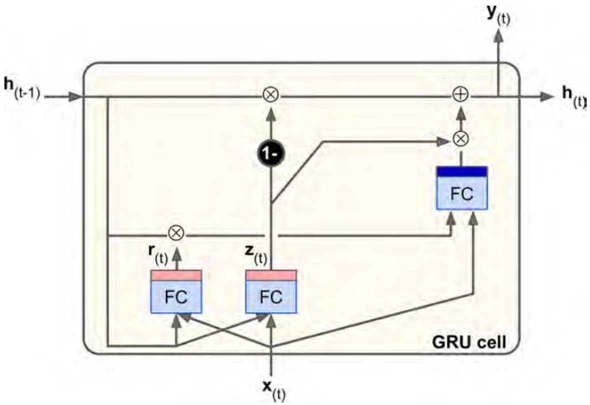
```
n_features = 2
n_timesteps = 200

model = models.Sequential()
model.add(LSTM(16, activation='relu', input_shape=(n_timesteps,n_features)))
model.add(layers.Dense(1, activation='linear'))

model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 16)                1216
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,233
Trainable params: 1,233
Non-trainable params: 0
```

Contents

Gated Recurrent Unit (GRU) Cell

GRU Cell:



Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

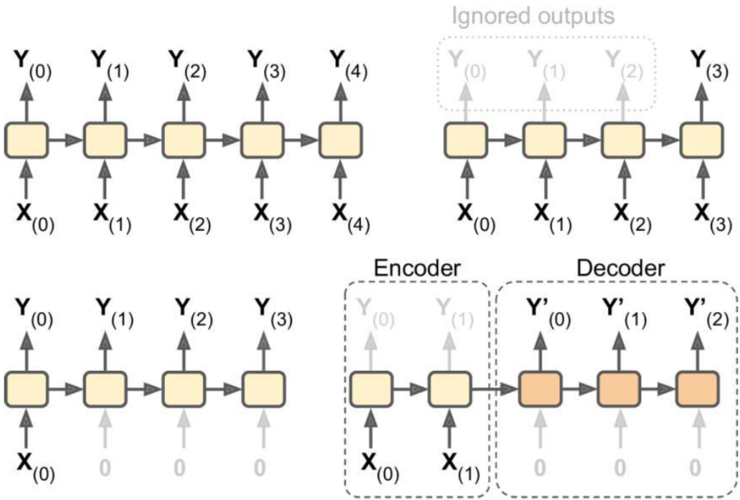
Gated Recurrent Unit (GRU) Cell

GRU Cell Model:

$$
\begin{aligned}
\mathbf{z}_{(t)} &= \sigma\left(\mathbf{W}_{xz}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{T} \cdot \mathbf{h}_{(t-1)}\right) \\
\mathbf{r}_{(t)} &= \sigma\left(\mathbf{W}_{xr}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{T} \cdot \mathbf{h}_{(t-1)}\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \left(\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}\right)\right) \\
\mathbf{h}_{(t)} &= \left(1 - \mathbf{z}_{(t)}\right) \otimes \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{z}_{(t)} \otimes \mathbf{g}_{t}\right)
\end{aligned}
$$

Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Input and Output Sequences of RNNs

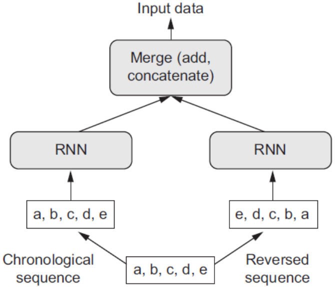


Source: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by A. Géron

Bidirectional RNNs

All RNNs (including LSTMs and RGUs) we have considered so far have "causal" structure, i.e. they are time dependent. In particular, RNNs built on reversed sequences can produce significantly different result. Moreover, prediction of state at current time t may depend on the whole input sequence (eg., speech recognition). Which order to use?

Solution: Bidirectional RNN.



Source: *Deep Learning* by F. Chollet

Bidirectional LSTM in Keras: Example

Example: One direction

```
from keras.layers import LSTM
from keras.layers.embeddings import Embedding

model = models.Sequential()
model.add(Embedding(200, 32))
model.add(LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, None, 32)          6400
_________________________________________________________________
lstm_12 (LSTM)               (None, 32)                8320
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 33
=================================================================
Total params: 14,753
Trainable params: 14,753
Non-trainable params: 0
```

Bidirectional LSTM in Keras: Example

Example: Two directions

```
from keras.layers import LSTM
from keras.layers.embeddings import Embedding

model = models.Sequential()
model.add(Embedding(200, 32))
model.add(layers.Bidirectional(LSTM(32)))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, None, 32)          6400
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                16640
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 65
=================================================================
Total params: 23,105
Trainable params: 23,105
Non-trainable params: 0
```