Lab 9

Hüseyin Anil Gündüz

2022-07-05

In the first part of the lab, we will analytically derive the backpropagation equations for a simple RNN. Then, in the second part, we will implement forward and backward propagation functions for a simple RNN-model, and train to predict the future temperature based on past weather metrics.

Exercise 1

In this part, we derive the backpropagation equations for a simple RNN from forward propagation equations. For simplicity, we will focus on a single input sequence $\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[\tau]}$. The forward pass in a RNN with hyperbolic tangent activation at time t is given by:

$$\mathbf{h}^{[t]} = \tanh\left(\mathbf{W}\mathbf{h}^{[t-1]} + \mathbf{U}\mathbf{x}^{[t]} + \mathbf{b}\right) \tag{1}$$

$$\mathbf{y}^{[t]} = \mathbf{V}\mathbf{h}^{[t]} + \mathbf{c} \tag{2}$$

where the parameters are the bias vectors $\mathbf{b}$ and $\mathbf{c}$ along with the weight matrices $\mathbf{U}$,$\mathbf{V}$ and $\mathbf{W}$, respectively, for input-to-hidden, hidden-to-output and hidden-to-hidden connections. As we will use RNN for a regression problem in the of the exercise, we do not use an activation function in order to compute the output $\mathbf{y}^{[t]}$ (at time t).

The loss is defined as:

$$\mathcal{L} = \sum_{t=1}^{\tau} \mathcal{L}\left(\mathbf{y}^{[t]}, \hat{\mathbf{y}}^{[t]}\right) \tag{3}$$

Show that:

$$\nabla_{\mathbf{h}^{[\tau]}}\mathcal{L} = \mathbf{V}^T(\nabla_{\mathbf{y}^{[\tau]}}\mathcal{L}) \tag{4}$$

$$\nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \mathbf{W}^T\mathrm{diag}\left(1 - \left(\mathbf{h}^{[t+1]}\right)^2\right)(\nabla_{\mathbf{h}^{[t+1]}}L) + \mathbf{V}^T(\nabla_{\mathbf{y}^{[t]}}\mathcal{L}) \tag{5}$$

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{y}^{[t]}}\mathcal{L} \tag{6}$$

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_{t=1}^{\tau} \mathrm{diag}\left(1 - \left(\mathbf{h}^{[t]}\right)^2\right)\nabla_{\mathbf{h}^{[t]}}\mathcal{L} \tag{7}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_{t=1}^{\tau} (\nabla_{\mathbf{y}^{[t]}}\mathcal{L})\mathbf{h}^{[t]^T} \tag{8}$$

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_{t=1}^{\tau} \mathrm{diag}\left(1 - \left(\mathbf{h}^{[t]}\right)^2\right)\ (\nabla_{\mathbf{h}^{[t]}}\mathcal{L})\mathbf{h}^{[t-1]^T} \tag{9}$$

$$\nabla_{\mathbf{U}}\mathcal{L} = \sum_{t=1}^{\tau} \mathrm{diag}\left(1 - \left(\mathbf{h}^{[t]}\right)^2\right)(\nabla_{\mathbf{h}^{[t]}}\mathcal{L})\mathbf{x}^{[t]^T} \tag{10}$$

Hint 1 (chain rule for vector calculus): given a vector $\mathbf{x} \in \mathbb{R}^n$ and two functions $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}$, call the outputs $\mathbf{y} = f(\mathbf{x})$ and $z = g(\mathbf{y}) = g(f(\mathbf{x}))$, then the following holds:

$$\nabla_{\mathbf{x}} z = \nabla_{\mathbf{x}}\mathbf{y} \cdot \nabla_{\mathbf{y}} z \tag{11}$$

where $\nabla_{\mathbf{y}} z \in \mathbb{R}^m$ and $\nabla_{\mathbf{x}}\mathbf{y} \in \mathbb{R}^n \times \mathbb{R}^m$.

Column number	Column name	Column description
1	No	Row number
2	year	Year
3	month	Month
4	day	Day
5	hour	Hour
6	pm2.5	Pollution in PM2.5 concentration
7	DEWP	Dew Point
8	TEMP	Temperature
9	PRES	Pressure
10	cbwd	Combined wind direction
11	Iws	Cumulated wind speed
12	Is	Cumulated hours of snow
13	Ir	Cumulated hours of rain

Table 1: Features of the data.

Hint 2: draw a computational graph representing the computation performed by the RNN unrolled over time, then use this graph to compute the gradients: multiply gradients via the chain rule when traversing edges, and sum the gradients obtained along each path from the loss to the item you are differentiating against.

Exercise 2

In the third exercise, we are going to be estimating only the temperature value of the next hour from the given past 24 hours of weather-related information. Thus we will not be computing any intermediate output from the RNN and only one scalar value at the final step. Additionally, we will use mean square error as a loss function.

Given this information, show that:

$$\nabla_{\mathbf{h}^{[\tau]}}\mathcal{L} = 2(\hat{y} - y)\mathbf{V}^T \tag{12}$$

$$\nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \mathbf{W}^T \cdot \text{diag}\left(1 - \mathbf{h}^{[t+1]^2}\right) \cdot \nabla_{\mathbf{h}^{[t+1]}}\mathcal{L} \tag{13}$$

$$\nabla_{\mathbf{c}}\mathcal{L} = 2(\hat{y} - y) \tag{14}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = 2(\hat{y} - y)\mathbf{h}^{[\tau]^T} \tag{15}$$

Exercise 3

In this exercise, we will implement forward and backward propagation steps of the simple RNN and train it on a real data. We will stick to the notation the we used in the first part of the exercise.

Prepare the data

In this exercise we will develop a model that estimates the temperature of the following hour from different weather parameters in the last 24 hours. We will be using a dataset at https://raw.githubusercontent.com/jbrownlee/Datasets/master/pollution.csv. Please download it in the same folder as this notebook and name it "rnn_dataset.csv". The dataset includes the features described in Table 1.

We read this file and print out the first rows and the dimensions of file. We will use DEWP, TEMP, PRES, cbwd, Iws, Is, Ir features as input and not the pollution, since pm2.5 contains some NA values we do not want to deal with. Save the corresponding columns of the file for these features and they will be used in the rest of the assignment.

```
csv_file <- read.csv(file = 'rnn_dataset.csv', nrows=43000, stringsAsFactors = FALSE)
csv_file <- csv_file[,c(7, 8, 9, 11, 12, 13)]
head(csv_file)
```

Now we have the data, composed of 13 observations in 43,000 consecutive hours. We first arrange it in an array so that we can use as many dimensions as we need later on.

```
n_features <- ncol(csv_file)
data <- array(0, dim=dim(csv_file))
for (i in 1:n_features){
  data[,i] <- array(csv_file[[i]], dim=c(1, nrow(csv_file)))
}

# remove the last day because it is missing observations
n_samples <- 24*floor(nrow(data) / 24)
data <- data[1:n_samples,]
dim(data)
```

The data is already sorted by time, from oldest to newest observation. We then create a test set using the last 20% of days:

```
train_idx <- (
  # TODO compute the indices of the training samples
)
data_train <- data[train_idx,]
data_test <- data[-train_idx,]
```

We now standardize the data to have zero mean and unit standard deviation:

```
means <- (
  # TODO compute the mean of each column
)

stds <- (
  # TODO compute the standard deviation of each column
)

data_train_scaled <- (
  # TODO standardize the training data
)

data_test_scaled <- (
  # TODO standardize the test data
)

apply(data_train_scaled, c(2), mean)  # should be close to zero
apply(data_train_scaled, c(2), sd)    # should be close to one
```

We now create a function to return a single random sequence of 24 contiguous observations along with the temperature to predict:

```
get_random_day <- function(data) {
  # TODO sample a random index

  list(
    x=data[start_idx:(start_idx+23),],
    y=data[start_idx+24,2]
  )
}

ex <- get_random_day(data_train_scaled)

# check size is correct
stopifnot(c(24, 6) == dim(ex$x))
```

RNN implementation

Let's initialize U, W, V, b and c weights randomly:

```
hidden_state_size = 10
U <- array(rnorm(n_features * hidden_state_size, mean=0, sd=0.001),
           dim=c(hidden_state_size, n_features))
W <- array(rnorm(hidden_state_size * hidden_state_size, mean=0, sd=0.001),
           dim=c(hidden_state_size, hidden_state_size))
b <- array(rnorm(hidden_state_size, mean=0, sd=0),
           dim=c(hidden_state_size, 1))
V <- array(rnorm(hidden_state_size, mean=0, sd=0.001),
           dim=c(1, hidden_state_size))
c <- rnorm(1, mean=0, sd=0)
```

Forward pass We will now define a function for the forward propagation, which will return the prediction of the network as well as all intermediate hidden states:

```
forward_pass <- function(X, U, V, W, b, c) {
  H <- array(0, dim=c(
    # TODO compute the size of the hidden states
  ))

  # TODO compute and save all hidden states

  y <- (
    # TODO compute the output of the RNN
  )

  list(
    hidden=H,
    out=y
  )
}

fp <- forward_pass(ex$x, U, V, W, b, c)
stopifnot(c(25, 10) == dim(fp$hidden))
```

And, finally, let's compute the loss:

```
compute_loss <- function(y_pred, y_true){
  # TODO compute the loss
}

compute_loss(fp$out, ex$y)
```

Backward pass We now define functions computing the gradient for each parameter of the network separately, starting from the hidden states:

```
compute_gradient_h <- function(y_true, y_pred, hidden, V, W) {
  grad <- array(0, dim=(
    # TODO compute the dimension of the gradients
  ))

  # TODO compute the gradient of the last hidden state

  for(i in (nrow(hidden)-1):1) {
    grad[i,] <- (
      # TODO compute the gradient of the i-th hidden state
    )
```

```
  }

  grad
}

gh <- compute_gradient_h(ex$y, fp$out, fp$hidden, V, W)
stopifnot(c(25, 10) == dim(gh))
```

The bias of the output layer:

```
compute_gradient_c <- function(y_true, y_pred) {
  # TODO compute the gradient with respect to c
}

compute_gradient_c(ex$y, fp$out)
```

The bias of the recurrent layer:

```
compute_gradient_b <- function(hidden, gradient_h) {
  grad <- array(0, dim=dim(b))
  # TODO compute the gradient with respect to b
  grad
}

compute_gradient_b(fp$hidden, gh)
```

The bias of the output weights:

```
compute_gradient_V <- function(y_true, y_pred, hidden) {
  # TODO compute the gradient with respect to V
}

stopifnot(c(1, 10) == dim(compute_gradient_V(ex$y, fp$out, fp$hidden)))
```

The bias of the hidden-to-hidden weights:

```
compute_gradient_W <- function(y_true, y_pred, hidden, grad_h) {
  grad <- array(0, dim=dim(W))

  # TODO compute the gradient with respect to W

  grad
}

compute_gradient_W(ex$y, fp$out, fp$hidden, gh)
```

And, finally, the gradients of U:

```
compute_gradient_U <- function(y_true, y_pred, hidden, grad_h, X) {
  grad <- array(0, dim=dim(U))

  # TODO compute the gradient with respect to W

  grad
}

compute_gradient_U(ex$y, fp$out, fp$hidden, gh, ex$x)
```

Training step Let us now put all the functions we defined above together to execute a single training step on a randomly sampled minibatch of data:

```
train_step <- function(batch_size, lr, U, V, W, b, c) {
  loss <- 0
  gc <- 0
  gb <- 0
  gV <- 0
  gW <- 0
  gU <- 0

  for(i in 1:batch_size) {
    # TODO perform a forward pass and compute the loss

    # TODO compute and accumulate the gradients
  }

  list(
    loss / batch_size,
    U - lr * gU / batch_size,
    V - lr * gV / batch_size,
    W - lr * gW / batch_size,
    b - lr * gb / batch_size,
    c - lr * gc / batch_size
  )
}
```

Training loop We now have all components needed to train our network:

```
train_rnn <- function(steps, U, V, W, b, c) {
  losses <- c()
  for(i in 1:steps) {
    # TODO perform a training step

    # TODO update the list of losses and the parameters
  }

  list(losses=losses, U=U, V=V, W=W, b=b, c=c)
}

hh <- train_rnn(500, U, V, W, b, c)
plot(hh$losses)
```

If you did everything correctly, the loss should have converged to below 0.05:

```
stopifnot(0.05 > mean(tail(hh$losses, 25)))
```

Evaluation on the test set Let us now use the network to predict the samples in the test set and plot predicted versus true value:

```
y_trues = c()
y_preds = c()

for(i in 1:(nrow(data_test_scaled) - 24)) {
  x = data_test_scaled[i:(i+23),]
  yt = data_test_scaled[i+24,2]
  yp = forward_pass(x, hh$U, hh$V, hh$W, hh$b, hh$c)$out

  y_trues <- c(y_trues, yt)
  y_preds <- c(y_preds, yp)
}
```

```
scatter.smooth(y_trues, y_preds)
```

Neat predictions!