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:

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

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

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

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

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

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

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

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

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.

Solution

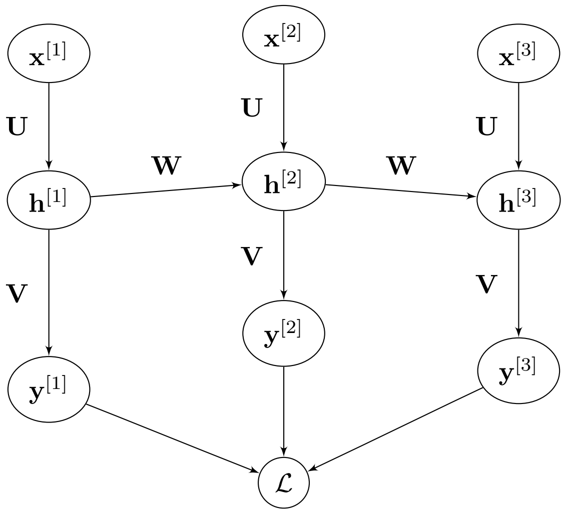


Figure 1: A simplified computational graph for three steps of a RNN. Biases omitted for simplicity.

The computational graph is shown in Figure 1. There is only one path connecting $\mathbf{h}^{[\tau]}$ to the loss:

$$\nabla_{\mathbf{h}^{[\tau]}}\mathcal{L} = \nabla_{\mathbf{h}^{[\tau]}}\mathbf{y}^{[\tau]} \cdot \nabla_{\mathbf{y}^{[\tau]}}\mathcal{L} = \mathbf{V}^T \cdot \nabla_{\mathbf{y}^{[\tau]}}\mathcal{L} \tag{12}$$

while every other hidden activation influences the loss via its associated output and the following hidden activation, thus:

$$\nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \nabla_{\mathbf{h}^{[t]}}\mathbf{y}^{[t]} \cdot \nabla_{\mathbf{y}^{[t]}}\mathcal{L} + \nabla_{\mathbf{h}^{[t]}}\mathbf{h}^{[t+1]} \cdot \nabla_{\mathbf{h}^{[t+1]}}\mathcal{L} \tag{13}$$

The first term is analogous to Eq. 12, while to find $\nabla_{\mathbf{h}^{[t]}}\mathbf{h}^{[t+1]}$ we need to apply the chain rule again:

$$\nabla_{\mathbf{h}^{[t]}}\mathbf{h}^{[t+1]} = \nabla_{\mathbf{h}^{[t]}} \tanh\left(\mathbf{W}\mathbf{h}^{[t]} + \mathbf{U}\mathbf{x}^{[t+1]} + \mathbf{b}\right) = \mathbf{W}^T \cdot \operatorname{diag}\left(1 - \mathbf{h}^{[t]^2}\right) \tag{14}$$

Therefore,

$$\nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \mathbf{V}^T \cdot \nabla_{\mathbf{y}^{[t]}}\mathcal{L} + \mathbf{W}^T \cdot \operatorname{diag}\left(1 - \mathbf{h}^{[t]^2}\right) \cdot \nabla_{\mathbf{h}^{[t+1]}}\mathcal{L} \tag{15}$$

where we do not expand $\nabla_{\mathbf{h}^{[t+1]}}\mathcal{L}$ further as that is carried over during backpropagation (it corresponds to the δ in lab 4).

We now compute the gradients with respect to the parameters of the network, starting with the easy biases. $\mathbf{c}$ is used to compute $\mathbf{y}^{[t]}$ for every t, thus:

$$\nabla_{\mathbf{c}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{c}}\mathbf{y}^{[t]} \cdot \nabla_{\mathbf{y}^{[t]}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{y}^{[t]}}\mathcal{L} \tag{16}$$

Similarly, $\mathbf{b}$ is used to compute $\mathbf{h}^{[t]}$, therefore:

$$\nabla_{\mathbf{b}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{b}}\mathbf{h}^{[t]} \cdot \nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \sum_{t=1}^{\tau} \operatorname{diag}\left(1 - \mathbf{h}^{[t]^2}\right) \cdot \nabla_{\mathbf{h}^{[t]}}\mathcal{L} \tag{17}$$

Moving on to the three weight matrices, we have:

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{V}^{[t]}}\mathbf{y}^{[t]} \cdot \nabla_{\mathbf{y}^{[t]}}\mathcal{L} \tag{18}$$

where we use $\nabla_{\mathbf{V}^{[t]}}\mathbf{y}^{[t]}$ to denote the gradient of $\mathbf{y}^{[t]}$ with respect to $\mathbf{V}$ *without* backpropagating, i.e., the contribution of $\mathbf{V}$ only at time t. In other words, you can think of $\mathbf{V}^{[1]}, \dots, \mathbf{V}^{[t]}$ as dummy variables that all equal $\mathbf{V}$. Note that we must now deal with tensors: let $\mathbf{V}^{[t]} \in \mathbb{R}^{n\times m}$, then $\nabla_{\mathbf{V}^{[t]}}\mathbf{y}^{[t]} \in \mathbb{R}^{n\times m\times n}$, so that, since $\nabla_{\mathbf{y}^{[t]}}\mathcal{L} \in \mathbb{R}^n$, $\nabla_{\mathbf{V}}\mathcal{L} \in \mathbb{R}^{n\times m}$ (the last dimension disappears due to the dot products, just like normal matrix multiplication). Let's analyze each item of the final gradient:

$$(\nabla_V \mathcal{L})_{ij} = \frac{\partial}{\partial \mathbf{V}_{ij}}\mathcal{L} \tag{19}$$

$$= \sum_{t=1}^{\tau}\sum_{k=1}^{n} \frac{\partial \mathcal{L}}{\partial y_k^{[t]}} \cdot \frac{\partial y_k^{[t]}}{\partial V_{ij}^{[t]}} \tag{20}$$

$$= \sum_{t=1}^{\tau}\sum_{k=1}^{n} \frac{\partial \mathcal{L}}{\partial y_k^{[t]}} \cdot \frac{\partial}{\partial V_{ij}^{[t]}} \sum_{\ell=1}^{m} V_{k\ell}^{[t]} h_\ell^{[t]} \tag{21}$$

$$= \sum_{t=1}^{\tau}\sum_{k=1}^{n} \frac{\partial \mathcal{L}}{\partial y_k^{[t]}} \cdot \delta_{ik} h_j^{[t]} \tag{22}$$

$$= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial y_i^{[t]}} \cdot h_j^{[t]} \tag{23}$$

Therefore, via the outer product:

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{y}^{[t]}}\mathcal{L} \cdot {\mathbf{h}^{[t]}}^T \tag{24}$$

A faster way of reaching the same result is via:

$$\nabla_{\mathbf{V}}\mathcal{L} = \sum_{t=1}^{\tau}\sum_{i=1}^{n} \nabla_{\mathbf{V}} y_i^{[t]} \cdot \nabla_{y_i^{[t]}}\mathcal{L} \tag{25}$$

and noticing that $\nabla_{\mathbf{V}} y_i^{[t]}$ is a matrix with all zeros except for row i which equals ${\mathbf{h}^{[t]}}^T$.

Moving on to $\mathbf{W}$, using the same insight, we have:

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{W}}\mathbf{h}^{[t]} \cdot \nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \sum_{t=1}^{\tau}\sum_{i=1}^{n} \nabla_{\mathbf{W}} h_i^{[t]} \cdot \nabla_{h_i^{[t]}}\mathcal{L} \tag{26}$$

where the i-th row of $\nabla_{\mathbf{W}} h_i^{[t]}$ equals, by the chain rule,

$$\nabla_{\mathbf{W}} h_i^{[t]} = \left(1 - {h_i^{[t]}}^2\right) \cdot {\mathbf{h}^{[t-1]}}^T \tag{27}$$

therefore:

$$\nabla_{\mathbf{W}}\mathcal{L} = \sum_{t=1}^{\tau}\sum_{i=1}^{n} \nabla_{\mathbf{W}} h_i^{[t]} \cdot \nabla_{h_i^{[t]}}\mathcal{L} = \sum_{t=1}^{\tau} \operatorname{diag}\left(1 - {\mathbf{h}^{[t]}}^2\right) \cdot \nabla_{\mathbf{y}^{[t]}}\mathcal{L} \cdot {\mathbf{h}^{[t-1]}}^T \tag{28}$$

Finally, in a similar way,

$$\nabla_{\mathbf{U}}\mathcal{L} = \sum_{t=1}^{\tau} \nabla_{\mathbf{U}}\mathbf{h}^{[t]} \cdot \nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \sum_{t=1}^{\tau}\sum_{i=1}^{n} \nabla_{\mathbf{U}} h_i^{[t]} \cdot \nabla_{h_i^{[t]}}\mathcal{L} = \sum_{t=1}^{\tau} \operatorname{diag}\left(1 - {\mathbf{h}^{[t]}}^2\right) \cdot \nabla_{\mathbf{y}^{[t]}}\mathcal{L} \cdot {\mathbf{x}^{[t]}}^T \tag{29}$$

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:

$$\nabla_{\mathbf{h}^{[\tau]}}\mathcal{L} = 2(\hat{y} - y)\mathbf{V}^T \tag{30}$$

$$\nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \mathbf{W}^T \cdot \text{diag}\left(1 - {\mathbf{h}^{[t+1]}}^2\right) \cdot \nabla_{\mathbf{h}^{[t+1]}}\mathcal{L} \tag{31}$$

$$\nabla_{\mathbf{c}}\mathcal{L} = 2(\hat{y} - y) \tag{32}$$

$$\nabla_{\mathbf{V}}\mathcal{L} = 2(\hat{y} - y){\mathbf{h}^{[\tau]}}^T \tag{33}$$

Solution

In the first formula we can directly expand the gradient of the loss:

$$\nabla_{\mathbf{h}^{[\tau]}}\mathcal{L} = \mathbf{V}^T \cdot \nabla_{\mathbf{y}^{[\tau]}}\mathcal{L} = \mathbf{V}^T \cdot 2(y - \hat{y}) \tag{34}$$

In the other cases, since only the last output is connected to the loss, the formulas developed above do not need to consider the paths connecting intermediate outputs. Therefore,

$$\nabla_{\mathbf{h}^{[t]}}\mathcal{L} = \nabla_{\mathbf{h}^{[t]}}\mathbf{h}^{[t+1]} \cdot \nabla_{\mathbf{h}^{[t+1]}}\mathcal{L} = \mathbf{W}^T \cdot \text{diag}\left(1 - {\mathbf{h}^{[t]}}^2\right) \cdot \nabla_{\mathbf{h}^{[t+1]}}\mathcal{L} \tag{35}$$

for the bias:

$$\nabla_{\mathbf{c}}\mathcal{L} = \nabla_{\mathbf{c}}\mathbf{y}^{[\tau]} \cdot \nabla_{\mathbf{y}^{[\tau]}}\mathcal{L} = 2(\hat{y} - y) \tag{36}$$

and for the last weight matrix:

$$\nabla_{\mathbf{V}}\mathcal{L} = \nabla_{\mathbf{V}}\mathbf{y}^{[\tau]} \cdot \nabla_{\mathbf{y}^{[\tau]}}\mathcal{L} = 2(\hat{y} - y){\mathbf{h}^{[\tau]}}^T \tag{37}$$

where $\mathbf{V}$ now has only one row since the network outputs scalars.

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.

```r
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)
```

```
##   DEWP TEMP PRES   Iws Is Ir
## 1  -21  -11 1021  1.79  0  0
## 2  -21  -12 1020  4.92  0  0
## 3  -21  -11 1019  6.71  0  0
## 4  -21  -14 1019  9.84  0  0
## 5  -20  -12 1018 12.97  0  0
## 6  -19  -10 1017 16.10  0  0
```

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.

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.

```
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)
```

```
## [1] 42984     6
```

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 <- (
  1:(24 * floor(0.8 * n_samples / 24))
)
data_train <- data[train_idx,]
data_test <- data[-train_idx,]
```

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

```
means <- (
  apply(data_train, c(2), mean)
)

stds <- (
  apply(data_train, c(2), sd)
)

data_train_scaled <- (
  t((t(data_train) - means) / stds)
)

data_test_scaled <- (
  t((t(data_test) - means) / stds)
)

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

```
## [1]  2.838959e-17 -1.164977e-16  5.289906e-15  2.707059e-17  1.258946e-17
## [6]  1.529510e-17
```

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

```
## [1] 1 1 1 1 1 1
```

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) {
  start_idx = sample.int(nrow(data) - 24, 1)

  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(
    nrow(X) + 1, nrow(U)
  ))

  for (i in 1:nrow(X)) {
    H[i + 1,] <- tanh(U %*% X[i,] + W %*% H[i,] + b)
  }

  y <- (
    (V %*% H[nrow(X) + 1,] + c)[1]
  )

  list(
    hidden=H,
    out=y
  )
}
```

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

And, finally, let's compute the loss:

```r
compute_loss <- function(y_pred, y_true){
  mean((y_true - y_pred)**2)
}

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

```
## [1] 0.2081229
```

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

```r
compute_gradient_h <- function(y_true, y_pred, hidden, V, W) {
  grad <- array(0, dim=(
    dim(hidden)
  ))

  grad[nrow(hidden),] <- 2 * (y_pred - y_true) * t(V)

  for(i in (nrow(hidden)-1):1) {
    grad[i,] <- (
      t(W) %*% diag(1 - hidden[i + 1,]**2) %*% grad[i + 1,]
    )
  }

  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:

```r
compute_gradient_c <- function(y_true, y_pred) {
  2 * (y_pred - y_true)
}

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

```
## [1] -0.9124098
```

The bias of the recurrent layer:

```r
compute_gradient_b <- function(hidden, gradient_h) {
  grad <- array(0, dim=dim(b))
  for(i in 2:nrow(hidden)) {
    grad <- grad + diag(1 - hidden[i,]**2) %*% gradient_h[i,]
  }
  grad
}

compute_gradient_b(fp$hidden, gh)
```

```
##                [,1]
##  [1,]  1.044980e-03
##  [2,]  1.615254e-03
##  [3,] -4.100665e-05
##  [4,]  1.107673e-04
##  [5,]  6.895755e-04
##  [6,] -8.515769e-05
##  [7,]  1.618337e-04
##  [8,]  1.010655e-03
##  [9,]  7.944422e-04
## [10,]  1.179439e-03
```

The bias of the output weights:

```
compute_gradient_V <- function(y_true, y_pred, hidden) {
  array(2 * (y_pred - y_true) * hidden[nrow(hidden),], dim=dim(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))

  for(i in 2:nrow(hidden)) {
    grad <- grad + diag(1 - hidden[i,]**2) %*% grad_h[i,] %*% hidden[i - 1,]
  }

  grad
}

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

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  1.748333e-06 -7.171915e-07  2.684981e-06 -8.344377e-07 -1.989023e-07
##  [2,]  2.702446e-06 -1.108586e-06  4.150251e-06 -1.289829e-06 -3.074617e-07
##  [3,] -6.869587e-08  2.842744e-08 -1.056572e-07  3.444299e-08  9.205382e-09
##  [4,]  1.853866e-07 -7.622795e-08  2.848202e-07 -8.968277e-08 -2.209987e-08
##  [5,]  1.153601e-06 -4.729018e-07  1.771419e-06 -5.484660e-07 -1.294640e-07
##  [6,] -1.423598e-07  5.807832e-08 -2.184240e-07  6.578950e-08  1.438570e-08
##  [7,]  2.707910e-07 -1.111625e-07  4.159130e-07 -1.298035e-07 -3.127983e-08
##  [8,]  1.690802e-06 -6.932874e-07  2.596427e-06 -8.050220e-07 -1.907189e-07
##  [9,]  1.329236e-06 -5.454712e-07  2.041484e-06 -6.357636e-07 -1.523579e-07
## [10,]  1.973329e-06 -8.095820e-07  3.030573e-06 -9.424779e-07 -2.250509e-07
##                [,6]          [,7]          [,8]          [,9]         [,10]
##  [1,]  4.059607e-07 -1.068121e-06 -4.770148e-07  5.175484e-07  3.773951e-07
##  [2,]  6.275018e-07 -1.651017e-06 -7.373429e-07  7.999884e-07  5.833557e-07
##  [3,] -1.568394e-08  4.124118e-08  1.956052e-08 -2.032745e-08 -1.533478e-08
##  [4,]  4.285259e-08 -1.127313e-07 -5.117433e-08  5.487292e-08  4.038501e-08
##  [5,]  2.682067e-07 -7.057058e-07 -3.137000e-07  3.415040e-07  2.483638e-07
##  [6,] -3.340343e-08  8.792104e-08  3.777804e-08 -4.215245e-08 -3.007306e-08
##  [7,]  6.278681e-08 -1.651875e-07 -7.415857e-08  8.015793e-08  5.862122e-08
##  [8,]  3.929178e-07 -1.033826e-06 -4.603482e-07  5.005275e-07  3.643684e-07
##  [9,]  3.084291e-07 -8.114842e-07 -3.633346e-07  3.934790e-07  2.873396e-07
## [10,]  4.580987e-07 -1.205289e-06 -5.387248e-07  5.841492e-07  4.261597e-07
```

And, finally, the gradients of U:

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

  for(i in 2:nrow(hidden)) {
    grad <- grad + diag(1 - hidden[i,]**2) %*% grad_h[i,] %*% X[i - 1,]
  }
```

```r
  grad
}

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

```
##                  [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  6.442675e-04  5.617952e-04 -9.291988e-04  2.383163e-04 -7.837573e-05
##  [2,]  9.958669e-04  8.683751e-04 -1.436283e-03  3.683742e-04 -1.211475e-04
##  [3,] -2.585113e-05 -2.138047e-05  3.606876e-05 -9.605171e-06  3.075585e-06
##  [4,]  6.870507e-05  5.906702e-05 -9.820816e-05  2.544523e-05 -8.307781e-06
##  [5,]  4.244197e-04  3.715773e-04 -6.136782e-04  1.569396e-04 -5.171962e-05
##  [6,] -5.176188e-05 -4.664735e-05  7.623521e-05 -1.909106e-05  6.387007e-06
##  [7,]  9.996948e-05  8.677899e-05 -1.437701e-04  3.699371e-05 -1.213787e-05
##  [8,]  6.224331e-04  5.441293e-04 -8.991455e-04  2.301898e-04 -7.580125e-05
##  [9,]  4.902663e-04  4.265603e-04 -7.060986e-04  1.813859e-04 -5.958485e-05
## [10,]  7.273914e-04  6.338191e-04 -1.048604e-03  2.690812e-04 -8.846040e-05
##                  [,6]
##  [1,] -1.486591e-04
##  [2,] -2.297863e-04
##  [3,]  5.833612e-06
##  [4,] -1.575777e-05
##  [5,] -9.809911e-05
##  [6,]  1.211455e-05
##  [7,] -2.302249e-05
##  [8,] -1.437759e-04
##  [9,] -1.130175e-04
## [10,] -1.677871e-04
```

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:

```r
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) {
    ex <- get_random_day(data_train_scaled)
    fp <- forward_pass(ex$x, U, V, W, b, c)
    loss <- loss + compute_loss(fp$out, ex$y)

    gh <- compute_gradient_h(ex$y, fp$out, fp$hidden, V, W)
    gc <- gc + compute_gradient_c(ex$y, fp$out)
    gb <- gb + compute_gradient_b(fp$hidden, gh)
    gV <- gV + compute_gradient_V(ex$y, fp$out, fp$hidden)
    gW <- gW + compute_gradient_W(ex$y, fp$out, fp$hidden, gh)
    gU <- gU + compute_gradient_U(ex$y, fp$out, fp$hidden, gh, ex$x)
  }

  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
  )
}
```

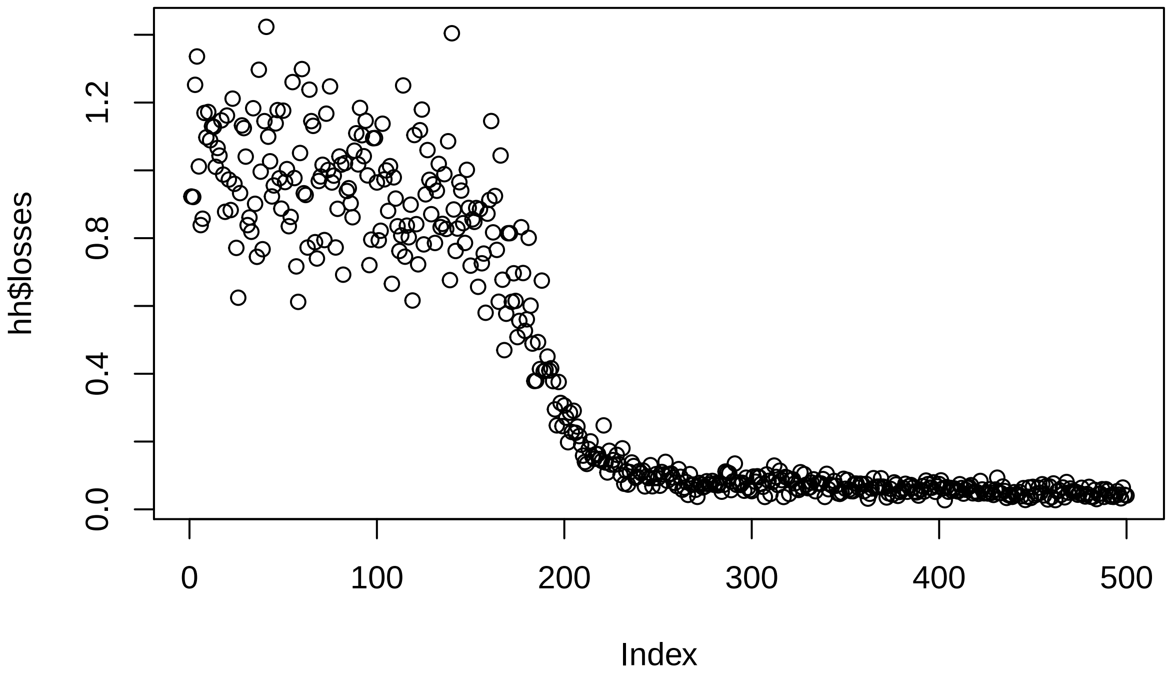
Training loop We now have all components needed to train our network:

```
train_rnn <- function(steps, U, V, W, b, c) {
  losses <- c()
  for(i in 1:steps) {
    ts <- train_step(32, 0.01, U, V, W, b, c)

    losses <- c(losses, ts[[1]])
    U <- ts[[2]]
    V <- ts[[3]]
    W <- ts[[4]]
    b <- ts[[5]]
    c <- ts[[6]]
  }

  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)))
```

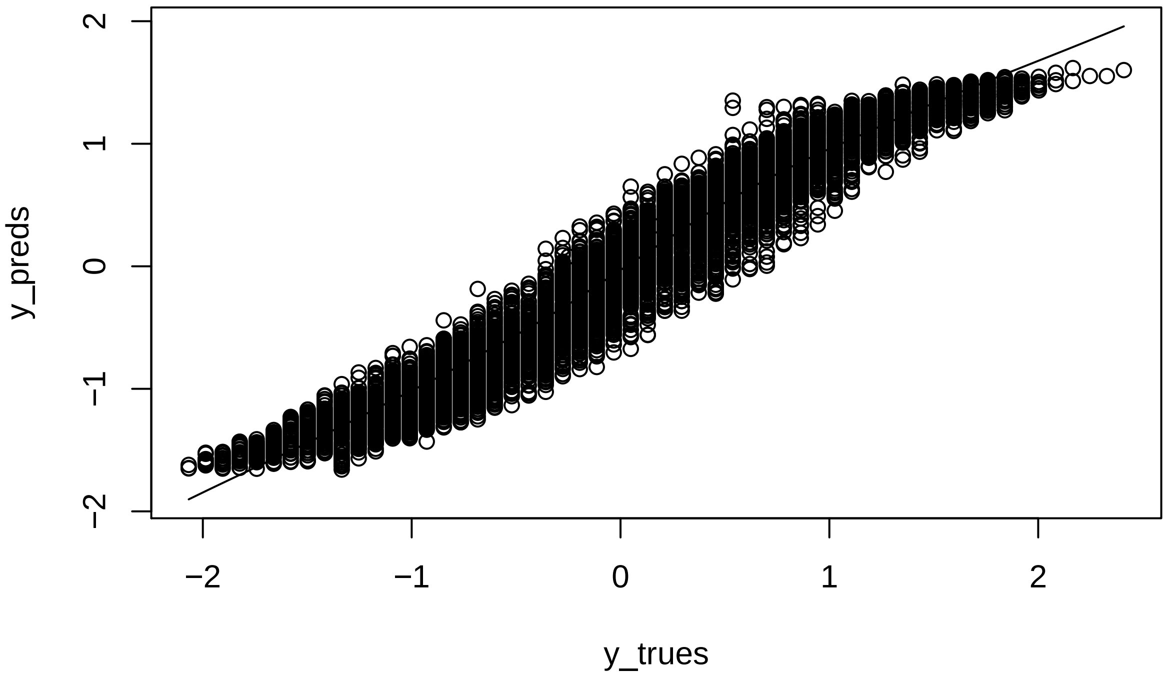
Evaluation on the test set Let us now use the network to predict the samples in the test set and plot predicted versus true value:

```
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!