STAT 153 & 248 - Time Series Lecture Twenty Four

Spring 2025, UC Berkeley

Aditya Guntuboyina

April 24, 2025

The last topic in this course is Recurrent Neural Networks (RNNs). In order to motivate RNNs, let us first recap some models that we have already studied in this class.

1 Regression with t as covariate

The simplest and the first model that we studied was the linear regression model:

$$y_t = \beta_0 + \beta_1 t + \epsilon_t \qquad \text{with } \epsilon_t \overset{\text{i.i.d}}{\sim} N(0, \sigma^2). \tag{1}$$

We then studied at nonlinear regression. One way to make the right hand side of (1) nonlinear in t is to introduce terms involving $(t-c)_+$ for certain knots c:

$$y_t = \beta_0 + \beta_1 t + \beta_2 (t - c_1)_+ + \cdots + \beta_{k+1}(t - c_k)_+ + \epsilon_t. \tag{2}$$

Here $(t-c)_+$ is the positive part function applied to $t - c_1$. We shall also use the notation ReLU and $\sigma(\cdot)$ to denote this function (please do not confuse the function $\sigma(\cdot)$ with the standard deviation σ of ϵ_t; we shall use the same notation for both but they can be easily distinguished from the context):

$$\sigma(u) = \text{ReLU}(u) = u_+ := \max(u, 0).$$

The unknown parameters in (2) are $\beta_0, \dots, \beta_{k+1}, c_1, \dots, c_k$ and σ.

The model (2) is also a linear model but it is linear in the modified variables $1, t, (t - c_1)_+, \dots, (t - c_k)_+$ (and nonlinear in the original variable t). The vector of these modified variables:

$$(1, t, (t-c_1)_+, \dots, (t-c_k)_+)^T$$

can be called the feature vector. The model is a linear function of the feature vectors.

We now rewrite the model (2) in a slightly different form. The time t represents the covariate x_t here, so we write $x_t = t$. We shall remove the term t as it is covered by $t = (t-c)_+$ for $c = 0$ (note that $1 \le t \le n$). We also write μ_t for the mean of y_t. We shall also use r_t to denote the feature vector:

$$r_t = (\sigma(x_t - c_1), \dots, \sigma(x_t - c_k))^T$$

and s_t to denote:

$$s_t = (x_t - c_1, \dots, x_t - c_k)^T.$$

With these changes, the model (2) becomes:

$$\begin{aligned}
x_t &= t \\
s_t &= (x_t - c_1, \dots, x_t - c_k)^T \\
r_t &= \sigma(s_t) \\
\mu_t &= \beta_0 + \beta^T r_t \\
y_t &= \mu_t + \epsilon_t.
\end{aligned} \tag{3}$$

In words, the univariate covariate x_t (which is simply t) is first converted to the $k \times 1$ vector s_t in a linear fastion. Then the nonlinear function $\sigma(\cdot)$ is applied to s_t (here $\sigma(\cdot)$ is applied separately to each coordinate of s_t) to generate the feature vector r_t. Then μ_t is a linear function of r_t which serves as the mean to y_t.

2 AutoRegression

We also studied autoregression models where the covariates are simply lagged values of y_t. The simplest of these models is AR(1) where $x_t = y_{t-1}$. This is simply (1) with t replaced by $x_t = y_{t-1}$:

$$y_t = \beta_0 + \beta_1 x_t + \epsilon_t \qquad \text{with } \epsilon_t \overset{\text{i.i.d}}{\sim} N(0, \sigma^2).$$

One can create a nonlinear version of AR(1) by simply using (3) with $x_t = y_{t-1}$. We shall refer to this as Nonlinear AutoRegression of order 1: NAR(1) (there are many nonlinear autoregression models and this one is only one of them):

$$\begin{aligned}
x_t &= y_{t-1} \\
s_t &= (x_t - c_1, \dots, x_t - c_k)^T \\
r_t &= \sigma(s_t) \\
\mu_t &= \beta_0 + \beta^T r_t \\
y_t &= \mu_t + \epsilon_t.
\end{aligned} \tag{4}$$

Now let us consider the case of AR(p) for $p \geq 1$. The usual AR(p) model is simply:

$$\begin{aligned}
x_t &= (y_{t-1}, \dots, y_{t-p})^T \\
\mu_t &= \beta_0 + \beta^T x_t \\
y_t &= \mu_t + \epsilon_t.
\end{aligned} \tag{5}$$

Observe that (5) can be written in compressed form as simply $y_t = \beta_0 + \beta_1 y_{t-1} + \cdots + \beta_p y_{t-p} + \epsilon_t$ which is the usual form of AR(p).

What is a natural nonlinear version of (5)? Put another way, what is a good extension of (4) for $p \geq 1$? There are multiple ways of obtaining these versions. Looking at the structure of (4), clearly $x_t = y_{t-1}$ will be replaced by $x_t = (y_{t-1}, \dots, y_{t-p})^T$. The next line gives the formula for s_t. This would need to be changed because x_t is no longer a scalar. One way to do this would be to write one version of the formula for s_t in (4) for each component of x_t. This would result in:

$$\begin{aligned}
x_t &= (y_{t-1}, \dots, y_{t-p})^T \\
s_t &= (x_{t1} - c_1^{(1)}, \dots, x_{t1} - c_k^{(1)}, x_{t2} - c_1^{(2)}, \dots, x_{t2} - c_k^{(2)}, \dots, x_{tp} - c_1^{(p)}, \dots, x_{tp} - c_k^{(p)})^T \\
r_t &= \sigma(s_t) \\
\mu_t &= \beta_0 + \beta^T r_t \\
y_t &= \mu_t + \epsilon_t.
\end{aligned} \tag{6}$$

Here $x_{t1} = y_{t-1}, \dots, x_{tp} = y_{t-p}$ denote the components of x_t. With this choice of s_t, note that μ_t becomes

$$\mu_t = \beta_0 + \beta^T r_t = \beta_0 + \beta^T \sigma(s_t) = \beta_0 + \sum_{j=1}^{p} g_j(x_{tj}) \qquad \text{where } g_j(x) := \sum_{i=1}^{k} \beta_{i,j}\sigma(x_{tj} - c_j^{(i)}).$$

In other words, we are fitting an **additive** model for y_t in terms of the covariates $x_{t1} = y_{t-1}, \dots, x_{tp} = y_{t-p}$. Additive models are popular in regression but they do not incorporate any interactions between the covariates. For example, if the true model generating the data is $y_t = 0.5 y_{t-1} y_{t-2} + \epsilon_t$, the additive model is unlikely to work well (because $(x_1, x_2) \mapsto 0.5 x_1 x_2$ is not an additive function of x_1 and x_2).

Instead of using the additive model in (6), we shall use the following model as NAR(p) (Nonlinear AutoRegression of order p). This is obtained by changing the second line of (6) to be an arbitrary linear function of x_t:

$$\begin{aligned} x_t &= (y_{t-1}, \dots, y_{t-p})^T \\ s_t &= W x_t + b \\ r_t &= \sigma(s_t) \\ \mu_t &= \beta_0 + \beta^T r_t \\ y_t &= \mu_t + \epsilon_t. \end{aligned} \tag{7}$$

Here W is a $k \times p$ matrix and b is a $k \times 1$ vector. The parameters in this model are the entries of the matrix W, the vector b, the coefficients β_0 and the components of β and finally the noise standard deviation σ.

In neural network terminology, the model (7) is called a **single-hidden layer neural network** because it first applies a linear transformation to the input x_t (via $s_t = W x_t + b$), then passes the result through the nonlinear activation function σ to get r_t, which forms the hidden layer. The output μ_t is then computed as a linear function of r_t (via $\mu_t = \beta_0 + \beta^T r_t$) and noise ϵ_t is added to explain the discrepancy between y_t and μ_t. The presence of one nonlinear transformation between the input x_t and the output μ_t, combined with otherwise linear operations, is exactly the structure of a single-hidden layer neural network.

To sum up, we take the single-hidden layer neural network model (7) to be our nonlinear generalization of AR(p).

Note that (7) can also be treated as a linear regression model but the linearity is in terms of the feature vector r_t (not in terms of the original covariate x_t). We shall refer to r_t as the feature vector at time t, it is also common to refer to it as the hidden layer output at time t.

3 Recurrent Neural Networks (RNNs)

We are now ready to define an RNN. RNN will involve one modification of the second equation in (7). Specifically, we will take s_t to be a linear function not only of x_t but also of the feature vector r_{t-1} at the previous time. This leads to (the difference relative to (7) is highlighted in blue below)

$$\begin{aligned} x_t &= (y_{t-1}, \dots, y_{t-p})^T \\ s_t &= W_r r_{t-1} + W x_t + b \\ r_t &= \sigma(s_t) \\ \mu_t &= \beta_0 + \beta^T r_t \\ y_t &= \mu_t + \epsilon_t. \end{aligned} \tag{8}$$

In Model (7), the hidden layer output r_t is computed purely from the current input x_t through a linear transformation (s_t) and the nonlinearity $\sigma(\cdot)$, so r_t depends only on x_t. In the RNN (8) however, the computation of r_t involves not just the current x_t but also the previous hidden layer output r_{t-1} through an additional linear term $W_r r_{t-1}$. This means that in the second model, the feature vector r_t is influenced both by the current input and by the feature vector from the previous step, whereas in the first model, it is influenced only by the current input.

Model (7) is a standard single-hidden layer feedforward neural network where the hidden layer r_t depends only on the current input. In contrast, the second model RNN (8) introduces a **recurrent** connection by adding a term $W_r r_{t-1}$ to the hidden layer input, meaning that r_t now depends not only on the current input x_t but also on the previous hidden state r_{t-1}. This recurrence creates a form of memory across time steps, making the second model a recurrent neural network (RNN), while the first model has no memory and treats each input independently.

The matrix W_r is $k \times k$ so it is a square matrix. The parameters now include $W_r, W, b, \beta_0, \beta$ (along with the noise standard deviation σ). Typically k will be larger than p. Model (8) also requires an initialization of r_t usually done by $r_0 = 0$.

In the model (7), the feature vector r_t depends only on x_t. On the other hand, in (8), r_t depends on all the inputs: $x_t, x_{t-1}, \dots, x_1$ (or $x_t, x_{t-1}, \dots, x_{p+1}$ in case $x_t = (y_{t-1}, \dots, y_{t-p})^T$ is not defined for $t \le p$; below we assume that the inputs x_t are defined for all $t = 1, 2, \dots$ without loss of generality; in a time series setting, this can be arranged by rearranging the time index). To see how r_t depends on $x_t, x_{t-1}, \dots$, note that

$$r_1 = \sigma(W x_1 + b) \qquad \text{because } r_0 = 0$$

$$r_2 = \sigma\left(W_r \sigma(W x_1 + b) + W x_2 + b\right),$$

$$r_3 = \sigma\left(W_r \sigma\left(W_r \sigma(W x_1 + b) + W x_2 + b\right) + W x_3 + b\right),$$

$$r_4 = \sigma(W_r \sigma\left(W_r \sigma\left(W_r \sigma(W x_1 + b) + W x_2 + b\right) + W x_3 + b\right) + W x_4 + b). \tag{9}$$

From the above, r_t clearly depends on all of $x_1, \dots, x_t$. But the strength of the dependence of r_t on x_s varies with s.

RNNs can have stability issues because the formula for r_t involves the product of a possibly large number of terms where the matrix W_r appears multiple times (e.g., see the formula (9) for r_4 above). Imagining W_r to be a scalar (just for the sake of making this argument), then two things can happen: it can be strictly larger than 1 in magnitude or strictly smaller than 1 in magnitude (it cannot be exactly equal to 1 in magnitude because these parameters are learning by a training algorithm and it is unlikely that this algorithm will output an estimate of W_r that is exactly equal to 1 in magnitude). If W_r is strictly larger than 1 in magnitude, then multiple appearances of W_r in products will blow them up, causing r_t to explode for moderate and large t. On the other hand, if W_r is strictly smaller than 1 in magnitude, then the products will be very small, and this leads to r_t depending mainly on x_s for which s is close to t (the implication is that RNNs cannot capture long-range dependence). When W_r is a matrix (instead of a scalar), this argument will still hold but, instead of magnitude, we need to use the spectral radius of W_r (spectral radius of a square matrix is defined as the largest magnitude of any eigenvalue).

The nonlinear activation function $\sigma(\cdot)$ also appears multiple times in the formula for r_t, (see again the formula (9) for r_4). To solve stability problems, it is customary in RNNs to take σ to be the hyperbolic tangent function (instead of ReLU). The hyperbolic tangent function is given by

$$\sigma(u) := \frac{e^u - e^{-u}}{e^u + e^{-u}}.$$

Unlike the ReLU function (which can take arbitrarily large positive values), the hyperbolic tangent activation function always takes values between -1 and 1. This helps the RNN be more stable.

We will discuss the RNNs more next week (along with related models such as GRU and LSTM).

4 Parameter Estimation via PyTorch

Given a time series dataset $y_1, \dots, y_n$, we estimate the parameters of these models simply by least squares. Specifically the parameters are estimated by minimizing:

$$\sum_{t=1}^{n} (y_t - \mu_t)^2. \tag{10}$$

Note that μ_t depends on the parameters $W, W_r \dots$ so that these parameters need to be chosen so that the sum of squares above is as small as possible. Minimization of (10) is done in an iterative fashion using a simple algorithm such as gradient descent. This requires calculation of gradients which is done efficiently in PyTorch. This also requires an initial value of the parameters.

5 Additional Optional Reading

1. Read the wikipedia article for Recurrent Neural Networks: `https://en.wikipedia.org/wiki/Recurrent_neural_network`. Our RNN model (8) is referred to as the Elman network in this wiki article.
2. For more on RNNs, I recommend the paper `https://arxiv.org/abs/1808.03314`.