STAT 153 & 248 - Time Series
Lecture Twenty Five

Spring 2025, UC Berkeley

Aditya Guntuboyina

April 29, 2025

1 Nonlinear AutoRegression

In the last lecture, we discussed nonlinear forms of autoregression for an observed time series $y_1, \dots, y_n$. For each t, we take $x_t = (y_{t-1}, \dots, y_{t-p})^T$ for some integer $p \geq 1$. x_t can be called the covariate at time t corresponding to the response value y_t. In the context of recurrent neural network models, x_t is referred to as the input at time t.

The usual (linear) autoregression AR(p) model corresponds to:

$$\mu_t = \beta_0 + \beta^T x_t. \tag{1}$$

The loss function is $\sum_t (y_t - \mu_t)^2$, and the parameters β_0, β are estimated by minimizing the loss.

In nonlinear autoregression, we change the formula (1) into a nonlinear function of x_t. When $p = 1$, one simple nonlinear AR(1) model is:

$$\mu_t = \beta_0 + \beta_1 x_t + \beta_2 (x_t - c_1)_+ + \dots + \beta_{k+1}(x_t - c_k)_+.$$

We simplify this slightly by dropping x_t (because $x_t = (x_t - c_0)_+ + c_0$ for all t provided c_0 is smaller than all the observed values of x_t; we will not lose anything by dropping x_t). This leads to

$$\mu_t = \beta_0 + \beta_1 (x_t - c_1)_+ + \dots + \beta_k (x_t - c_k)_+.$$

We rewrite this equation using the following notation:

$$\begin{aligned} s_t &= (x_t - c_1, \dots, x_t - c_k)^T \\ r_t &= \sigma(s_t) \\ \mu_t &= \beta_0 + \beta^T r_t. \end{aligned} \tag{2}$$

s_t is a linear function of x_t which maps the scalar x_t to the $k \times 1$ vector s_t. $\sigma(\cdot)$ denotes the ReLU function applied pointwise to the input. So r_t is obtained by apply the ReLU function to each coordinate of s_t. Finally μ_t is a linear function of r_t (we shall sometimes refer to μ_t as the output corresponding to the input x_t).

When $p \geq 1$, there are multiple ways of generalizing (2). One simple way is to consider the following "additive" model (below $x_t^{(i)} = y_{t-i}$ denotes the i^{th} coordinate of x_t)

$$\begin{aligned} s_t^{(i)} &= (x_t^{(i)} - c_1^{(i)}, \ldots, x_t^{(i)} - c_k^{(i)})^T \qquad \text{for } 1 \leq i \leq p \\ s_t &= \begin{pmatrix} s_t^{(1)} \\ \cdot \\ \cdot \\ \cdot \\ s_t^{(p)} \end{pmatrix} \\ r_t &= \sigma(s_t) \\ \mu_t &= \beta_0 + \beta^T r_t \end{aligned} \tag{3}$$

This is called an additive model because μ_t can be written as an additive sum of separate functions of $x_t^{(i)}$ for $i = 1, \ldots, p$. A different (i.e., non-additive) generalization of (2) is the single-hidden layer neural network defined as follows.

$$\begin{aligned} s_t &= Wx_t + b \\ r_t &= \sigma(s_t) \\ \mu_t &= \beta_0 + \beta^T r_t \end{aligned} \tag{4}$$

Here s_t is again $k \times 1$, W is $k \times p$ and b is $p \times 1$. We shall refer to (4) as the NonLinear AR model of order p. The total number of parameters here is $kp + k + k + 1 = kp + 2k + 1$. When p increases by 1, the number of parameters in (4) increases by k. On the other hand, in the usual (linear), AR(p) model, the number of parameters increases only by 1 when p increases by 1. So these models have a tendency to become high-dimensional faster than the linear AR(p) models.

Note that (4) 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.

2 Recurrent Neural Network (RNN)

RNN will involve one modification of the first equation in (4). Specifically, in an RNN, 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 following set of equations defining the RNN:

$$\begin{aligned} r_0 &= 0 \\ s_t &= W_r r_{t-1} + Wx_t + b \\ r_t &= \sigma(s_t) \\ \mu_t &= \beta_0 + \beta^T r_t \end{aligned} \tag{5}$$

This formula can also be written as

$$\begin{aligned} r_0 &= 0 \\ r_t &= \sigma(W_r r_{t-1} + Wx_t + b) \\ \mu_t &= \beta_0 + \beta^T r_t \end{aligned} \tag{6}$$

In Model (4), 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 (5) 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 (4) 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 (5) 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 in the RNN are $W_r, W, b, \beta_0, \beta$. Typically k will be larger than p. Model (5) also requires an initialization of r_t usually done by $r_0 = 0$.

In the model (4), the feature vector r_t depends only on x_t. On the other hand, in (5), 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

$$
\begin{aligned}
r_1 &= \sigma(Wx_1 + b) \qquad \text{because } r_0 = 0 \\
r_2 &= \sigma\left(W_r \sigma(Wx_1 + b) + Wx_2 + b\right), \\
r_3 &= \sigma\left(W_r \sigma\left(W_r \sigma(Wx_1 + b) + Wx_2 + b\right) + Wx_3 + b\right), \\
r_4 &= \sigma(W_r \sigma\left(W_r \sigma\left(W_r \sigma(Wx_1 + b) + Wx_2 + b\right) + Wx_3 + b\right) + Wx_4 + b). \qquad (7)
\end{aligned}
$$

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.

From the above (e.g., see the formula (7) for r_4), it is clear that the formula for r_t will involve products of a large number of terms where the matrix W_r appears multiple times. This can lead to stability problems when W_r is large. For example, imagine that W_r is a scalar which 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. When W_r is a matrix, this will happen when the spectral radius of W_r (defined as the largest modulus of any eigenvalue of W_r) is strictly larger than one. This is a regime which needs to be avoided to prevent instability.

The nonlinear activation function $\sigma(\cdot)$ also appears multiple times in the formula for r_t, (see again the formula (7) for r_4). For better stability, 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.

So the RNN is given by:

$$\begin{aligned} r_0 &= 0 \\ s_t &= W_r r_{t-1} + W x_t + b \\ r_t &= \sigma_{\text{tanh}}(s_t) \\ \mu_t &= \beta_0 + \beta^T r_t \end{aligned} \tag{8}$$

or

$$\begin{aligned} r_0 &= 0 \\ r_t &= \sigma_{\text{tanh}}(W_r r_{t-1} + W x_t + b) \\ \mu_t &= \beta_0 + \beta^T r_t \end{aligned} \tag{9}$$

Here the activation function σ_{tanh} is the tanh activation function given by

$$\sigma_{\text{tanh}}(u) := \frac{e^u - e^{-u}}{e^u + e^{-u}}.$$

The only difference between (8) and (5) (and also (9) and (6)) is the tanh activation function.

Use of the tanh activation, as well as requiring that W_r does not have spectral radius strictly larger than 1 makes the RNN stable. However, if W_r has spectral radius strictly smaller than 1 (note that we can ignore the case where the spectral radius is exactly equal to one, because W_r and other parameters of the RNN are learned by a training algorithm and it is unlikely that this algorithm will output an estimate of W_r with spectral radius exactly equal to 1), then the RNN has the problem of "lack of long memory". This means that r_t effectively depends only on those inputs x_u which are somewhat close to t. To see this, let us calculate the derivative of r_t with respect to x_u for $u \leq t$. The formula is given by:

$$\frac{\partial r_t}{\partial x_u} = \sigma'(s_t) W_r \sigma'(s_{t-1}) W_r \dots \sigma'(s_{u+1}) W_r \sigma'(s_u) W \qquad \text{for } u \leq t. \tag{10}$$

Here $\frac{\partial r_t}{\partial x_u}$ denotes the $k \times p$ Jacobian Matrix of derivatives of r_t with respect to x_u. On the right hand side in (10), $\sigma'(s_t)$ should be interpreted as $k \times k$ diagonal matrices whose diagonal entries are obtained by applying the $\sigma'(u) = \frac{d}{du}\sigma(u)$ function to each element of s_t ($\sigma'(s_{t-1}), \dots$ are similarly defined as $k \times k$ diagonal matrices). This diagonal interpretation of $\sigma'(u)$ aligns with the Jacobian of the pointwise interpretation of $\sigma(u)$ when u is a vector.

As a concrete example,

$$\frac{\partial r_4}{\partial x_4} = \sigma'(s_4) W \quad \frac{\partial r_4}{\partial x_3} = \sigma'(s_4) W_r \sigma'(s_3) W \quad \frac{\partial r_4}{\partial x_2} = \sigma'(s_4) W_r \sigma'(s_3) W_r \sigma'(s_2) W$$
$$\frac{\partial r_4}{\partial x_1} = \sigma'(s_4) W_r \sigma'(s_3) W_r \sigma'(s_2) W_r \sigma'(s_1) W$$

Note that these gradient formulae are with respect to inputs x_u, and not with respect to the parameters (which is crucial to parameter estimation). In the formula (10), it is clear that when u is much smaller than t, many more terms appear in the right hand side of (10) compared to the case when u is closer to t. Note here that σ is the tanh activation function:

$$\sigma(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}} \text{ so that } \sigma'(u) = 1 - \sigma^2(u) \in (0, 1].$$

Thus each $\sigma'(\cdot)$ term will add a fractional multiplier to $\partial r_t / \partial x_u$. The number of these fractional multipliers will increase as u decreases in (10).

Further the matrix W_r also plays a key role in (10). For the model equation in (8) to be stable, W_r needs to have spectral radius (defined as the largest modulus of any eigenvalue) to be strictly smaller than one. In that case, each additional W_r multiplier will bring the whole term down, leading to $\partial r_t/\partial x_u$ being small when u is much smaller than t.

This points to the following shortcoming of RNNs that more sophisticated models such as GRUs and LSTMs attempt to fix. We want r_t to represent the ideal summary of $x_1, \dots, x_t$ that is relevant for the output y_t. However, in an RNN, r_t effectively only depends on those inputs x_u which are somewhat close to t. In this sense, the RNN can be thought of as not having a very long memory.

This "lack of long memory" problem with RNNs can be fixed by use of GRUs and LSTMs.

3 GRU (Gated Recurrent Unit)

Consider again the RNN formula (9). The basic problem with this is that r_t depends on r_{t-1} through the term $W_r r_{t-1}$. If W_r is a matrix with spectral radius less than 1 (which it needs to be for stability purposes), then the multiplier $W_r r_{t-1}$ can be thought of as "reducing" r_{t-1} by a factor of W_r. If this formula is applied repeatedly, then very soon the dependence of r_t on r_u will be very small. In order to avoid this, one needs to prevent r_t from depending on r_{t-1} only through $W_r r_{t-1}$.

This leads to the following idea. First construct a potential version $\tilde{r}_t$ of r_t in the same way as (9):

$$\tilde{r}_t = \sigma(W_r r_{t-1} + W x_t + b). \tag{11}$$

This $\tilde{r}_t$ only depends on r_{t-1} through $W_r r_{t-1}$. The two natural options for r_t now are:

1. $r_t = \tilde{r}_t$: in this case, we are back to the RNN (9).
2. $r_t = r_{t-1}$: in this case, r_t is exactly equal to r_{t-1}, which means that the current input x_t is ignored.

The idea behind GRU is to take a "convex-like" combination of these two options in the following way:

$$r_t = z_t r_{t-1} + (1 - z_t)\tilde{r}_t.$$

This would be exactly a convex combination if z_t were a scalar in the interval $[0, 1]$. But we allow z_t to be a vector interpreting the multiplication as pointwise. Thus it is better to write

$$r_t = z_t \odot r_{t-1} + (1 - z_t) \odot \tilde{r}_t. \tag{12}$$

The next thing to specify z_t. In GRU, we take

$$z_t = \sigma_{\text{sigmoid}}(W_r^z r_{t-1} + W^z x_t + b^z) \qquad \text{where } \sigma_{\text{sigmoid}}(u) := \frac{1}{1 + e^{-u}}. \tag{13}$$

Because σ_{sigmoid} takes values between 0 and 1, the above formula ensures that the components of z_t take values in $[0, 1]$ so that (12) represents a convex combination at the level of each individual component. Further (13) implies that z_t is also determined by r_{t-1} and x_t. The parameters W_r^z, W^z, b controlling the formula (13) are also unknown and they will be estimated along with all the other parameters of the model.

z_t is sometimes referred to as a gate. It controls the closeness of r_t to r_{t-1} and $\tilde{r}_t$.

r_{t-1} appears in two places in the formula (12): in the term $z_t \odot r_{t-1}$ as well as in the formula (11) for $\tilde{r}_t$. It might be redundant to have r_{t-1} appear in both these places. To address this, GRU modifies (11) by using one more gate as follows:

$$\tilde{r}_t := \sigma(W_r(r_{t-1} \odot g_t) + W x_t + b),$$

where g_t controls the extent to which r_{t-1} is used in the formula for $\tilde{r}_t$. Similar to (13), the gate g_t is specified via

$$g_t = \sigma_{\text{sigmoid}}(W_r^g r_{t-1} + W^g x_t + b^g). \tag{14}$$

Putting all the formulae together, we get the following specification of the GRU model:

$$\begin{aligned}
r_0 &= 0 \\
g_t &= \sigma_{\text{sigmoid}}(W_r^g r_{t-1} + W^g x_t + b^g) \\
z_t &= \sigma_{\text{sigmoid}}(W_r^z r_{t-1} + W^z x_t + b^z) \\
\tilde{r}_t &:= \sigma_{\tanh}(W_r(r_{t-1} \odot g_t) + W x_t + b) \\
r_t &= z_t \odot r_{t-1} + (1 - z_t) \odot \tilde{r}_t \\
\mu_t &= \beta_0 + \beta^T r_t.
\end{aligned} \tag{15}$$

z_t is called the update gate while g_t is called the reset gate. The unknown parameters in this model (which need to be estimated from the data) are $W_r^g, W^g, b^g, W_r^z, W^z, b^z, W_r, W, b, \beta_0, \beta$.

This is a more sophisticated model compared to the RNN model (9). In fact, (9) is a special case of (15) corresponding to $g_t = 1$ and $z_t = 0$. The presence of the gates g_t and z_t can alleviate the lack of long memory problem that was an issue with the RNNs.

4 LSTM (Long Short Term Memory)

LSTM is another modification to the basic RNN for enabling long memory. It also uses gates and has one more gate compared to the GRU. Instead of a recursion directly between r_{t-1} and r_t, the LSTM recursions are between the pairs $(s_{t-1}, r_{t-1}) \to (s_t, r_t)$. We will look at the formulas for LSTM in the next lecture.

5 Additional Optional Reading

1. For more on GRUs, read `https://en.wikipedia.org/wiki/Gated_recurrent_unit`.
2. For more on LSTMs, read `https://en.wikipedia.org/wiki/Long_short-term_memory`.
3. A clear description of LSTM is given here: `https://www.youtube.com/watch?v=YCzL96nL7j0&t=870s`