Ouroboros

Neural Cryptanalysis of Linear Feedback Shift Registers

Stochastic Batman

February 18, 2026

1 Linear Feedback Shift Registers

Definition

A **Linear Feedback Shift Register** (LFSR) of degree n is a finite-state machine whose state at time t is a vector of bits

$$\mathbf{s}^{(t)} = \left(s_1^{(t)}, s_2^{(t)}, \ldots, s_n^{(t)}\right) \in \mathbb{F}_2^n$$

where $\mathbb{F}_2 = \{0, 1\}$ is the field of integers modulo 2, with addition defined as XOR ($\oplus$) and multiplication as AND ($\wedge$).

The state evolves by a linear recurrence over $\mathbb{F}_2$:

$$s_n^{(t+1)} = c_1 s_1^{(t)} \oplus c_2 s_2^{(t)} \oplus \cdots \oplus c_n s_n^{(t)}, \qquad c_i \in \mathbb{F}_2$$

and then the register shifts: $s_i^{(t+1)} = s_{i+1}^{(t)}$ for $i < n$. The output (keystream) at each step is the bit that falls off the end:

$$b^{(t)} = s_1^{(t)}.$$

Characteristic Polynomial and Taps

The recurrence is encoded by the *characteristic polynomial*

$$p(x) = x^n + c_{n-1}x^{n-1} + \cdots + c_1 x + 1 \in \mathbb{F}_2[x]$$

The non-zero coefficients indicate which bit positions feed back into the computation; these positions are called **taps**.

Ouroboros uses a degree-32 LFSR with the maximal-length polynomial

$$p(x) = x^{32} + x^{22} + x^2 + x + 1,$$

giving taps at positions {32, 22, 2, 1}.

Maximal Length and Periodicity

If $p(x)$ is *primitive*(can produce every single non-zero element in that field through repeated multiplication) over $\mathbb{F}_2$, the LFSR cycles through every non-zero state exactly once before repeating. The period is then

$$T = 2^n - 1$$

For $n = 32$ this gives $T = 4,294,967,295 \approx 4.3 \times 10^9$ bits - astronomically long, yet the entire sequence is determined by the n-bit seed and the tap set.

State Transition as a Matrix

Over $\mathbb{F}_2$ the one-step transition is a linear map. Writing the state as a column vector, the update rule is

$$\mathbf{s}^{(t+1)} = A\,\mathbf{s}^{(t)}, \quad A \in \mathbb{F}_2^{n\times n},$$

where A is the *companion matrix* of $p(x)$:

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ c_1 & c_2 & c_3 & \cdots & c_n \end{pmatrix}.$$

After k steps: $\mathbf{s}^{(t+k)} = A^k\,\mathbf{s}^{(t)}$, all arithmetic mod 2. This is why the sequence is completely predictable given the seed and tap set: the RNN's job is to *infer* this linear structure from the raw bitstream alone.

The Berlekamp-Massey Theorem

A classical result states that $2n$ consecutive output bits are sufficient to reconstruct both the tap polynomial and the internal state of any degree-n LFSR exactly. For our 32-bit register, 64 bits of observed output are *provably enough* to break it analytically. The neural approach instead attempts to learn this structure implicitly via gradient descent.

2 Recurrent Neural Networks: A Recap

The Elman RNN

Given a sequence of inputs $x_1, x_2, \ldots, x_T$ (scalars here, since each input is one bit), an **Elman RNN** maintains a hidden state $\mathbf{h}_t \in \mathbb{R}^H$ and produces an output $y_t \in \mathbb{R}$ at each step according to:

$$\mathbf{h}_t = \tanh(W_{xh}x_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$
$$y_t = \sigma(W_{hy}\mathbf{h}_t + b_y)$$

where the learnable parameters are:

$$W_{xh} \in \mathbb{R}^{H\times 1}, \quad W_{hh} \in \mathbb{R}^{H\times H}, \quad \mathbf{b}_h \in \mathbb{R}^H,$$
$$W_{hy} \in \mathbb{R}^{1\times H}, \quad b_y \in \mathbb{R}.$$

The initial state is set to $\mathbf{h}_0 = \mathbf{0}$.

Activation Functions

Hyperbolic tangent is used for the hidden layer because it is zero-centered, has gradient in $(-1, 1)$, and gives the network the ability to represent both excitation and inhibition:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$

Sigmoid is used at the output to produce a valid probability:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in (0, 1).$$

Loss Function

Since the network predicts the probability that the next bit is 1, we use **Binary Cross-Entropy** (BCE):

$$\mathcal{L}(y_t, \hat{y}_t) = -\Big[y_t \log \hat{y}_t + (1 - y_t) \log(1 - \hat{y}_t)\Big],$$

where $y_t \in \{0, 1\}$ is the true next bit and $\hat{y}_t$ is the predicted probability. The total loss over a sequence of length T is

$$\mathcal{L}_{\text{total}} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}(y_t, \hat{y}_t).$$

3 Backpropagation Through Time (BPTT)

Overview

BPTT unrolls the RNN across T time steps and treats the result as a deep feedforward network, then applies the chain rule. Gradients flow backwards from the loss at each step all the way to the parameters.

Output Layer Gradient

The fused BCE + sigmoid gradient has a particularly clean form. Let $z_t = W_{hy}\mathbf{h}_t + b_y$ so that $\hat{y}_t = \sigma(z_t)$. Then:

$$\frac{\partial \mathcal{L}}{\partial z_t} = \hat{y}_t - y_t.$$

This is the gradient used directly in the code as `dy`.

From this, the parameter gradients for the output layer are:

$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = (\hat{y}_t - y_t)\, \mathbf{h}_t^\top,$$
$$\frac{\partial \mathcal{L}}{\partial b_y} = \hat{y}_t - y_t.$$

Hidden Layer Gradient

The gradient of the loss w.r.t. $\mathbf{h}_t$ receives two contributions: one from the output at step t, and one propagated from step $t+1$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = W_{hy}^{\top} (\hat{y}_t - y_t) + W_{hh}^{\top} \boldsymbol{\delta}_{t+1},$$

where $\boldsymbol{\delta}_t$ is the *delta* flowing into the hidden pre-activation.

Since $\mathbf{h}_t = \tanh(\mathbf{a}_t)$ with $\mathbf{a}_t$ the pre-activation, the chain rule through tanh gives:

$$\boldsymbol{\delta}_t = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \odot \big(1 - \mathbf{h}_t^2\big),$$

where $\odot$ denotes element-wise multiplication and $1 - \mathbf{h}_t^2$ is the element-wise derivative of tanh.

The remaining parameter gradients follow:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{xh}} &= \boldsymbol{\delta}_t \, x_t^{\top}, \\
\frac{\partial \mathcal{L}}{\partial W_{hh}} &= \boldsymbol{\delta}_t \, \mathbf{h}_{t-1}^{\top}, \\
\frac{\partial \mathcal{L}}{\partial \mathbf{b}_h} &= \boldsymbol{\delta}_t, \\
\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t-1}} &= W_{hh}^{\top} \boldsymbol{\delta}_t \qquad \leftarrow \text{propagated to previous step.}
\end{aligned}$$

Gradients accumulate by summing across all T steps before updating weights.

Gradient Clipping

Repeated matrix products W_{hh}^{T} in deep unrollings cause gradients to grow exponentially with T (*exploding gradients*). The code applies element-wise clipping:

$$g \leftarrow \mathrm{clip}(g, \, -c, \, c) = \max\big(-c, \, \min(c, \, g)\big), \quad c = 5.0.$$

Weight Initialisation

Weights are initialised with **Xavier (Glorot) uniform** initialisation:

$$W_{ij} \sim \mathcal{U}\left(-\frac{1}{\sqrt{n_{\text{in}}}}, \frac{1}{\sqrt{n_{\text{in}}}}\right),$$

where n_{in} is the fan-in of the layer. This keeps the variance of activations roughly constant across layers at initialisation, avoiding saturation of tanh from the first forward pass.

Parameter Update (SGD)

After accumulating and clipping gradients, a vanilla SGD step is applied:

$$\theta \leftarrow \theta - \eta \, \nabla_{\theta} \mathcal{L}, \quad \eta = 0.005.$$

4 The Cryptanalysis Objective

Let $\mathcal{S} = (b^{(0)}, b^{(1)}, b^{(2)}, \ldots)$ be the LFSR keystream. The network is trained on the supervised task

$$\hat{y}_t \approx \mathbb{P}\left(b^{(t+1)} = 1 \mid b^{(0)}, \ldots, b^{(t)}\right).$$

Since the LFSR is deterministic, this probability is degenerate: it is either 0 or 1. Perfect prediction corresponds to the network having implicitly learned the characteristic polynomial $p(x)$ and the tap set $\{32, 22, 2, 1\}$ from raw observations alone.

The hidden state $\mathbf{h}_t \in \mathbb{R}^H$ can be interpreted as the network's learned *proxy* for the LFSR's internal state $\mathbf{s}^{(t)} \in \mathbb{F}_2^{32}$. If $H \geq 32$, there is sufficient capacity to represent the full register, and we would expect accuracy to approach 100% given enough training.