IOHMMs as a Neural Network: An Interpretable Deep Learning Model

Akshay Reddy

Abstract

This paper introduces an interpretable deep learning architecture based on Input Output Hidden Markov Models (IOHMMs), bridging probabilistic graphical models and deep learning. IOHMMs, originally proposed by Bengio and Frasconi in 1995 [1], offer an interpretable approach to time series data modeling but traditionally lack the predictive power and scalability of neural networks. Conversely, neural networks are often considered black box models, limiting their interpretability.

We present a self-contained exposition, developing an architecture from first principles to model sequential data with input-dependent transitions and emissions. This study offers an interpretable alternative for time series analysis, validated through experiments on synthetic and real-world datasets, addressing the need for transparency and explainability in applications requiring these features. We call this architecture the IOHMM-NN architecture.

Keywords: machine learning, neural networks, deep learning, interpretable ai, interpretable deep learning, time series analysis, hidden markov model, IOHMM

Definitions and Notations

Let us formally define the notations and conventions adopted throughout this work.

Consider a discrete-time sequence indexed by $t = 1, 2, \ldots, T$, where at each time step t, the system receives an input vector $\boldsymbol{u}_t \in \mathbb{R}^m$ and produces an output vector $\boldsymbol{x}_t \in \mathbb{R}^k$.

The underlying process is governed by a latent discrete state variable $z_t \in \{1, 2, \ldots, N\}$, representing the hidden state of the model at time t. The model assumes a Markovian structure, wherein the state at time t depends

on the previous state z_{t-1} and the current input $\boldsymbol{u}_t$. The true sequence of states $\{z_t\}_{t=1}^T$ is unobserved and must be inferred during training.

Apart from the number of hidden states (N), the model is parameterized by a set of weights and biases, which will be introduced in detail in the following section. These parameters are grouped as follows:

- Initial state parameters: $\boldsymbol{W}_\pi \in \mathbb{R}^{N\times m}$ and $\boldsymbol{b}_\pi \in \mathbb{R}^N$
- State transition parameters: $\boldsymbol{W}_z^{(i)} \in \mathbb{R}^{N\times m}$ and $\boldsymbol{b}_z^{(i)} \in \mathbb{R}^N$ for each state $i \in \{1,\dots,N\}$
- Emission (output) parameters: $\boldsymbol{W}_x^{(i)} \in \mathbb{R}^{k\times m}$ and $\boldsymbol{b}_x^{(i)} \in \mathbb{R}^k$ for each state $i \in \{1,\dots,N\}$

Throughout this paper, boldface symbols denote vectors or matrices, and superscripts in parentheses indicate state-specific parameters.

To summarize:

- Inputs: $\{\boldsymbol{u}_t\}_{t=1}^T$
- Outputs: $\{\boldsymbol{x}_t\}_{t=1}^T$
- Hidden States: $\{z_t\}_{t=1}^T$
- Weights: $\boldsymbol{W}_\pi, \{\boldsymbol{W}_z^{(i)}\}_{i=1}^N, \{\boldsymbol{W}_x^{(i)}\}_{i=1}^N$
- Biases: $\boldsymbol{b}_\pi, \{\boldsymbol{b}_z^{(i)}\}_{i=1}^N, \{\boldsymbol{b}_x^{(i)}\}_{i=1}^N$

Let us also define the notation we utilize for the softmax function over $\mathbb{R}^s$ $\forall s \in \mathbb{N}$:

Softmax : $\mathbb{R}^s \to \mathbb{R}^s$ with $\text{Softmax}(u)_i = \frac{e^{u_i}}{\sum_{j=1}^s e^{u_j}}$

Model Architecture

Model Formulation

The proposed model is defined by the following probabilistic assumptions:

1. **Initial State Distribution:**
   The probability of the initial hidden state z_1 given the first input $\boldsymbol{u}_1$ is modeled as a categorical distribution parameterized by a softmax transformation. These values are stored in a vector called $\boldsymbol{\pi} \in mathbbR^N$:

$$\boldsymbol{\pi}_i = P(z_1 = i \mid \boldsymbol{u}_1) = \text{Softmax}(\boldsymbol{W}_\pi \boldsymbol{u}_1 + \boldsymbol{b}_\pi)_i, \quad i \in \{1, \ldots, N\}$$

$$\implies \boldsymbol{\pi} = \text{Softmax}(\boldsymbol{W}_\pi \boldsymbol{u}_1 + \boldsymbol{b}_\pi)$$

2. **State Transition Dynamics:**
   The conditional probability of transitioning to state z_t at time t, given the previous state z_{t-1} and current input $\boldsymbol{u}_t$, is also modeled via a softmax transformation, with parameters specific to the previous state:

$$P(z_t = i \mid \boldsymbol{u}_t, z_{t-1} = j) = \text{Softmax}(\boldsymbol{W}_z^{(j)} \boldsymbol{u}_t + \boldsymbol{b}_z^{(j)})_i, \quad i, j \in \{1, \ldots, N\}$$

3. **Emission Process:**
   The conditional expectation of the output $\boldsymbol{x}_t$ given the current input $\boldsymbol{u}_t$ and hidden state z_t is modeled as a linear transformation:

$$\mathbb{E}[\boldsymbol{x}_t \mid \boldsymbol{u}_t, z_t = i] = \boldsymbol{W}_x^{(i)} \boldsymbol{u}_t + \boldsymbol{b}_x^{(i)}, \quad i \in \{1, \ldots, N\}$$

These formulations collectively define the generative process underlying the IOHMM-NN model.

Derivation

In this section, we present a formal derivation of the inference procedure for the IOHMM-based neural network. The goal is to compute the expected emission at time t, denoted $\mathbb{E}[\boldsymbol{x}_t \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t}]$, given a sequence of input vectors. The derivation proceeds in three logical steps.

Step 1: Transition Matrix Construction

We first define the transition probability matrix at time t, $\mathbf{\Lambda}^{(t)} \in \mathbb{R}^{N\times N}$, which encodes the probability of transitioning between hidden states conditioned on the current input. The matrix elements are given by:

$$\mathbf{\Lambda}_{i,j}^{(t)} = P(z_t = i \mid z_{t-1} = j, \{\boldsymbol{u}_{t'}\}_{t'=1}^{t}) = P(z_t = i \mid \boldsymbol{u}_t, z_{t-1} = j) = \text{Softmax}(\boldsymbol{W}_z^{(j)}\boldsymbol{u}_t + \boldsymbol{b}_z^{(j)})_i$$

In matrix form:

$$\mathbf{\Lambda}^{(t)} = \begin{bmatrix} | & | & & | \\ \text{Softmax}(\boldsymbol{W}_z^{(1)}\boldsymbol{u}_t + \boldsymbol{b}_z^{(1)}) & \text{Softmax}(\boldsymbol{W}_z^{(2)}\boldsymbol{u}_t + \boldsymbol{b}_z^{(2)}) & \cdots & \text{Softmax}(\boldsymbol{W}_z^{(N)}\boldsymbol{u}_t + \boldsymbol{b}_z^{(N)}) \\ | & | & & | \end{bmatrix}$$

Step 2: Recursive Computation of State Probabilities

Let $\boldsymbol{\mu}^{(t)} \in \mathbb{R}^N$ denote the probability mass distribution over hidden states at time t:

$$\boldsymbol{\mu}_i^{(t)} = P(z_t = i \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t})$$

For $t = 1$:

$$\boldsymbol{\mu}_i^{(1)} = P(z_1 = i \mid \boldsymbol{u}_1) = \boldsymbol{\pi}_i$$

For $t > 1$:

$$\boldsymbol{\mu}_i^{(t)} = P(z_t = i \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t}) = \sum_{j=1}^{N} P(z_t = i \mid z_{t-1} = j, \boldsymbol{u}_t) P(z_{t-1} = j \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t-1})$$

Since $\mathbf{\Lambda}_{i,j}^{(t)} = P(z_t = i \mid z_{t-1} = j, \boldsymbol{u}_t)$ and $\boldsymbol{\mu}_i^{(t-1)} = P(z_{t-1} = j \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t-1})$:

$$\boldsymbol{\mu}_i^{(t)} = \sum_{j=1}^{N} \mathbf{\Lambda}_{i,j}^{(t)} \boldsymbol{\mu}_j^{(t-1)} = [\mathbf{\Lambda}^{(t)} \boldsymbol{\mu}^{(t-1)}]_i$$

In vector form:

$$\boldsymbol{\mu}^{(t)} = \begin{cases} \boldsymbol{\pi} & \text{if } t = 1 \\ \mathbf{\Lambda}^{(t)} \boldsymbol{\mu}^{(t-1)} & \text{if } t > 1 \end{cases}$$

Step 3: Expected Output Calculation

The expected value of the output at time t is:

$$\mathbb{E}[\boldsymbol{x}_t \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t}] = \sum_{i=1}^{N} \mathbb{E}[\boldsymbol{x}_t \mid \boldsymbol{u}_t, z_t = i] P(z_t = i \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t}) = \sum_{i=1}^{N} (\boldsymbol{W}_x^{(i)} \boldsymbol{u}_t + \boldsymbol{b}_x^{(i)}) \boldsymbol{\mu}_i^{(t)}$$

Auxiliary Notations

To facilitate concise mathematical expressions and avoid cumbersome summations over indices, we introduce two auxiliary operations: convolution between a tensor and a vector, and the column-wise softmax function. We also define a few key tensors and matrices for the same.

Convolution and Column-wise Softmax

Definition 1 (Convolution)**.** *Let $A \in \mathbb{R}^{p \times q \times r}$ be a three-dimensional tensor and $b \in \mathbb{R}^r$ a vector. We define the convolution $A \circ b = C \in \mathbb{R}^{p \times q}$ as:*

$$C_{i,j} = \sum_{k=1}^{r} A_{i,j,k} b_k$$

Definition 2 (Column-wise Softmax)**.** *Let $M \in \mathbb{R}^{n \times m}$ be a matrix. The column-wise softmax, denoted Col-Softmax(M), applies the softmax function independently to each column of M. For each column j:*

$$S_j = \sum_{i=1}^{n} e^{M_{i,j}}$$

$$\textit{Col-Softmax}(M)_{i,j} = \frac{e^{M_{i,j}}}{S_j}$$

In Matrix notation:

$$\textit{Col-Softmax}\left(\begin{bmatrix} | & | & & | \\ \boldsymbol{c_1} & \boldsymbol{c_2} & \cdots & \boldsymbol{c_m} \\ | & | & & | \end{bmatrix}\right) = \begin{bmatrix} | & | & & | \\ \textit{Softmax}(\boldsymbol{c_1}) & \textit{Softmax}(\boldsymbol{c_2}) & \cdots & \textit{Softmax}(\boldsymbol{c_m}) \\ | & | & & | \end{bmatrix}$$

Tensors and Matrices

We define:

- $\boldsymbol{W}^z \in \mathbb{R}^{N \times N \times m}$, where ${\boldsymbol{W}^z}_{p,q,r} = ({\boldsymbol{W}_z}^{(q)})_{p,r}$
- $\boldsymbol{W}^x \in \mathbb{R}^{k \times N \times m}$, where ${\boldsymbol{W}^x}_{p,q,r} = ({\boldsymbol{W}_x}^{(q)})_{p,r}$
- $\boldsymbol{B}^z \in \mathbb{R}^{N \times N}$, where ${\boldsymbol{B}^z}_{p,q} = ({\boldsymbol{b}_z}^{(q)})_p$
- $\boldsymbol{B}^x \in \mathbb{R}^{k \times N}$, where ${\boldsymbol{B}^x}_{p,q} = ({\boldsymbol{b}_x}^{(q)})_p$

Simplified Notation

Using these definitions:

- Transition matrix: $\boldsymbol{\Lambda}^{(t)} = \text{Col-Softmax}(\boldsymbol{W}^z \circ \boldsymbol{u}_t + \boldsymbol{B}^z)$
- Hidden state distribution:

$$\boldsymbol{\mu}^{(t)} = \begin{cases} \boldsymbol{\pi} & \text{if } t = 1 \\ \boldsymbol{\Lambda}^{(t)} \boldsymbol{\mu}^{(t-1)} & \text{if } t > 1 \end{cases}$$

- Expected output: $\mathbb{E}[\boldsymbol{x}_t \mid \{\boldsymbol{u}_{t'}\}_{t'=1}^{t}] = (\boldsymbol{W}^x \circ \boldsymbol{u}_t + \boldsymbol{B}^x)\boldsymbol{\mu}^{(t)}$

Forward Pass

Algorithm 1 Forward Pass

$\boldsymbol{\pi} \leftarrow \text{Softmax}(\boldsymbol{W}_\pi \boldsymbol{u}_1 + \boldsymbol{b}_\pi)$
$\boldsymbol{\mu}^{(1)} \leftarrow \boldsymbol{\pi}$
$\mathbb{E}[\boldsymbol{x}_1] \leftarrow (\boldsymbol{W}^x \circ \boldsymbol{u}_1 + \boldsymbol{B}^x)\boldsymbol{\mu}^{(1)}$
for $t = 2$ to T **do**
    $\boldsymbol{\Lambda}^{(t)} \leftarrow \text{ColSoftmax}(\boldsymbol{W}^z \circ \boldsymbol{u}_t + \boldsymbol{B}^z)$
    $\boldsymbol{\mu}^{(t)} \leftarrow \boldsymbol{\Lambda}^{(t)} \boldsymbol{\mu}^{(t-1)}$
    $\mathbb{E}[\boldsymbol{x}_t] \leftarrow (\boldsymbol{W}^x \circ \boldsymbol{u}_t + \boldsymbol{B}^x)\boldsymbol{\mu}^{(t)}$
end for

The weights can then be optimized using the backpropagation algorithm. In this paper, we utilized PyTorch, with it's in-built capability to back propagate and optimize the weights.

Data Simulation

To validate the efficacy of the proposed IOHMM-NN architecture, we conducted experiments on synthetic data derived from a toy IOHMM with predefined parameters. The goal of this section is to see whether our IOHMM-NN model can capture the parameters of our toy model.

Toy IOHMM Model

The toy model is defined as:

- Number of states: $N = 2$
- Initial State: $z_1 = 1$
- Inputs: $\boldsymbol{u}_t \sim \mathcal{N}(0, 1)$
- Transition probabilities:
  - $P(\boldsymbol{z}_t = 1 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 1) = \sigma(-\boldsymbol{u}_t)$
  - $P(\boldsymbol{z}_t = 2 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 1) = \sigma(\boldsymbol{u}_t)$
  - $P(\boldsymbol{z}_t = 1 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 2) = \sigma(-\boldsymbol{u}_t)$
  - $P(\boldsymbol{z}_t = 2 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 2) = \sigma(\boldsymbol{u}_t)$

  where $\sigma(\cdot)$ is the sigmoid function
- Outputs:

$$\boldsymbol{x}_t = \begin{cases} 5\boldsymbol{u}_t & \text{if } \boldsymbol{z}_t = 1 \\ -5\boldsymbol{u}_t & \text{if } \boldsymbol{z}_t = 2 \end{cases}$$

We simulated data for $T = 1000$ iterations, and then passed it as training data to out IOHMM-NN.

Training Methodology and Hyperparameters

The model was trained with:

- Optimizer: Adam
- Learning Rate: 0.1
- Loss Function: MSE/L2 Loss
- Epochs: 1000

Results

The learned weights were:

$$\boldsymbol{W}_z^{(1)} = \begin{bmatrix} 0.3053 \\ -0.0069 \end{bmatrix}, \quad \boldsymbol{W}_z^{(2)} = \begin{bmatrix} -0.4369 \\ -0.3805 \end{bmatrix}$$

$$\boldsymbol{W}_\pi = \begin{bmatrix} -1.7116 \\ 0.9817 \end{bmatrix}$$

$$\boldsymbol{W}_x^{(1)} = 4.8856, \quad \boldsymbol{W}_x^{(2)} = -4.9564$$

Upon decoding the weights, we get the following:

Output Function

The trained model's output function:

$$\boldsymbol{x}_t = \begin{cases} 4.8856\boldsymbol{u}_t & \text{if } \boldsymbol{z}_t = 1 \\ -4.9564\boldsymbol{u}_t & \text{if } \boldsymbol{z}_t = 2 \end{cases}$$

This is very close to the output function of the toy model. Looking at the values, we can confidently say that the IOHMM-NN was able to capture the structure of the two states in our toy model.

Note: Since the labeling of z_t as 1 and 2 is arbitrary, it is possible that the values 4.8856 and -4.9564 could have been interchanged. In that case, we would have needed to switch the labeling for our analysis.

Initial State

While the IOHMM-NN model cannot determine that the initial state is 1, we can calculate the probability distribution of z_1. Setting $\boldsymbol{u}_1$ to -1.1258 (as per the simulated data), we get:

$$\boldsymbol{\pi} = \begin{bmatrix} 0.9540 \\ 0.0460 \end{bmatrix}$$

Transition Probabilities

Decoding $\boldsymbol{W}^z$, we get the following transition probability functions:

- $P(\boldsymbol{z}_t = 1 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 1) = \sigma(0.3122\boldsymbol{u}_t)$
- $P(\boldsymbol{z}_t = 2 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 1) = \sigma(-0.3122\boldsymbol{u}_t)$

- $P(\boldsymbol{z}_t = 1 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 2) = \sigma(-0.05641\boldsymbol{u}_t)$
- $P(\boldsymbol{z}_t = 2 \mid \boldsymbol{u}_t, \boldsymbol{z}_{t-1} = 2) = \sigma(0.05641\boldsymbol{u}_t)$

We can see that the IOHMM-NN model was not able to accurately capture the transition probability function of our toy model. This is most likely since:

1. The data is essentially a random sequence generated by the toy model, but the IOHMM outputs the expected value of the sequence. The noise introduced in the random sequence generation could have caused the issue with the transition function.
2. The IOHMM outputs the weights for a softmax function. However, there are multiple weights that can result in the same sigmoid function. This could have resulted in poor optimization of the $\boldsymbol{W}^z$ weights.

Overall, we can see that the IOHMM-NN was able to accurately capture the underlying structure of the toy model. This provides evidence that our architecture can be used to interpret time series data, with its weights providing insight into the underlying structure of the data.

Predicting Google Stock Market Data

To evaluate the practical applicability of the IOHMM-NN, we conduct an experiment on real-world financial time series data. Specifically, we aim to predict the daily stock values of Google (Alphabet Inc.) using historical data as input to our model and compare its performance against a standard Recurrent Neural Network (RNN).

Dataset

The dataset used in this experiment consists of Google's stock data spanning from 2004 to 2022. The data is divided into two segments: a training set from 2004 to 2020 and a test set from 2020 to 2022. Each data point includes the following features:

- Open price
- Low price
- High price
- Close price
- Adjusted close price

The input to the model at each time step t consists of the aforementioned five features over the past three days, resulting in an input vector $\boldsymbol{u}_t \in \mathbb{R}^{15}$. The output $\boldsymbol{x}_t \in \mathbb{R}^5$ represents the predicted values for the same five features for the current day.

Model Configuration

The two models trained and evaluated were:

1. **IOHMM-NN**: The IOHMM-NN configured with 5 hidden states. Bias terms were included in this experiment.
2. **RNN**: A vanilla Recurrent Neural Network (RNN) implemented with a hidden state dimension of 5.

Training Parameters

Both models were trained using the following hyperparameters:

- Epochs: 1000
- Loss: MSE/L2
- Optimizer: Adam
- Learning rate: 0.1

Evaluation Metrics

The performance of both models was evaluated using the following metrics:

1. Mean Squared Error (MSE): Measures the average squared difference between the predicted and actual values.

2. R-squared (R^2): Represents the proportion of variance in the dependent variable that is predictable from the independent variables.

Results

Metric	Model	Training	Test
MSE	IOHMM-NN	433.26	7770.72
	RNN	80.67	1374.04
R^2	IOHMM-NN	99.63%	97.78%
	RNN	99.93%	99.61%

Discussion

The results indicate that the RNN outperforms the IOHMM in terms of both MSE and R^2 on both the training and test datasets. This suggests that, for this particular task, the RNN is better able to capture the underlying patterns in the data.

One potential reason for the IOHMM's comparatively lower performance could be the choice of activation function. The IOHMM relies heavily on the softmax function, which, while suitable for producing probability distributions, may introduce representational inflexibility and saturation issues when used in intermediary layers. Specifically:

- **Representational Inflexibility**: The softmax function forces all outputs to sum to one, which can limit the expressive power of the hidden states.

- **Saturation**: When one input to the softmax function is significantly larger than the others, the output approaches a one-hot encoded vector. This can effectively shut off other neurons, making the gradients sensitive to small changes in the dominant neuron and causing gradients for non-dominant neurons to vanish.

In contrast, the RNN typically employs activation functions like $tanh$ or $ReLU$, which do not suffer from these limitations.

Conclusion

In this study, we introduced an interpretable deep learning architecture based on Input Output Hidden Markov Models (IOHMM-NN). Our experiments on both synthetic and real-world datasets reveal key characteristics of this model.

The IOHMM-NN architecture demonstrates a strong capability in capturing the underlying structure of time series data, as evidenced by its performance on the synthetic data model. The learned weights offer insights into the dynamics of the system, allowing for interpretation in terms of hidden Markov chains and state-specific linear models.

However, our experiments on the Google Stock data also highlight limitations of the IOHMM-NN architecture, particularly in predictive power when compared to standard RNNs. The reliance on the softmax function in intermediary layers may lead to representational inflexibility and saturation issues, hindering its ability to model complex temporal dependencies.

Despite these limitations, the IOHMM-NN architecture offers significant advantages in scenarios where interpretability is paramount. Its ability to represent time series data as an interpretable hidden Markov chain, with each state characterized by a linear model, makes it a valuable tool for understanding and explaining complex sequential data. This is particularly relevant in sectors where transparency and explainability are critical, such as finance, economics, and healthcare.

The IOHMM-NN architecture addresses key concerns that have traditionally limited the adoption of neural networks in certain domains

- **Regulatory requirements**: The transparent nature of the model facilitates compliance with regulations mandating interpretable data models.
- **Causal reasoning**: The model's structure allows for causal reasoning and understanding of the relationships between inputs, hidden states, and outputs.
- **Trust and explainability**: The model's interpretability fosters trust among decision-makers by providing clear explanations of its predictions

In conclusion, the IOHMM-NN architecture presents a viable alternative to traditional black-box neural networks in situations where interpretability and explainability are prioritized over raw predictive accuracy. Its unique ability to bridge the gap between probabilistic graphical models and deep learning makes it a valuable tool for a wide range of applications.

The code for this paper is available at:
www.github.com/InfinityAkshay/IOHMM_NN.

References

[1] Y. Bengio and P. Frasconi, "Input-output HMMs for sequence processing," IEEE Transactions on Neural Networks, vol. 7, no. 5, pp. 1231-1249, Sept. 1996.