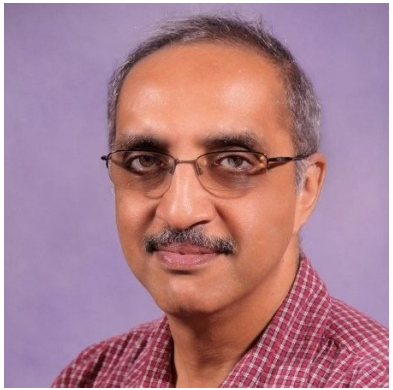
Dynamic Deep Network Models and Applications

By: Avi Linzer, Joseph Couzens, Michael Batushansky, Michael Kupferstein, Tani Gross

Our Team

- Avi Linzer - post 2nd-year student
- Joseph Couzens - post 1st-year student
- Michael Batushansky - post 1st-year student
- Michael Kupferstein - post 1st-year student
- Tani Gross - post 1st-year student

Our Mentor



- Ramesh Natarajan - Software Engineer and Scientist, Google

Project Goals

1. Develop Dynamic Deep Network [DDN] models that are capable of “sequential” or "on the fly" training/ inference.
2. Investigate approaches to DDN model adaptation in practical scenarios with evolving training data distribution, and evaluate its effectiveness.

Technical Approach

Use State Space Models (SSM's) based on Extended Kalman Filter (EKF) and its variants for neural network training, **rather than** the standard training algorithms based on gradient descent.

An "open issue" is how to match the DDN model adaptation to the gradual or rapid evolution of the new data distribution, for which we propose "human in the loop" approaches for "tweaking the uncertainties" in the model parameters.

Background

Neural network models are typically trained using datasets with static or fixed distributions.

In real-world scenarios, the underlying training data distribution can change over time due to various factors (e.g. gradually due to seasonality, trends, or rapidly due to sudden behavior shifts).

When these changes occur, the predictive performance of original models can deteriorate significantly.

DDN models offer a solution by allowing models to adapt and learn from new data as it becomes available, in a controlled way..

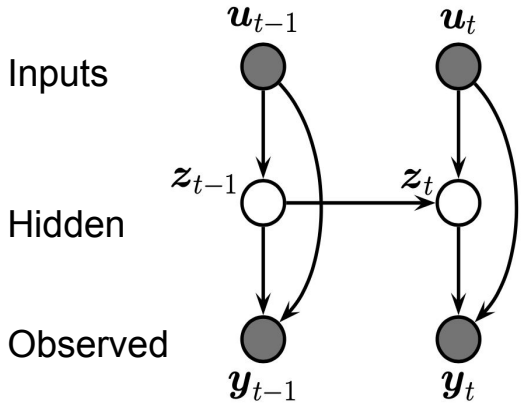
The background for DDN models in this presentation is based on:

- State Space Models (SSM's) for Linear and Nonlinear Dynamical Systems
- Training of DDN models using SSM's
- Implementation using the Jax-based Dynamax Package

Fast Overview of Mathematical Concepts

State Space Models (SSM's)

Conditional probabilistic models for sequences of hidden (**z**) and observed variables (**y**), with sequential inputs/covariates/features (**u**), where *p* denotes probability distributions and ***t*** is the sequence index.



$$p(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T} | \boldsymbol{u}_{1:T}) = \left[p(\boldsymbol{z}_1 | \boldsymbol{u}_1) \prod_{t=2}^{T} p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1}, \boldsymbol{u}_t) \right] \left[\prod_{t=1}^{T} p(\boldsymbol{y}_t | \boldsymbol{z}_t, \boldsymbol{u}_t) \right]$$

Joint Distribution — Initial state distribution — Transition / dynamics model — Observation / emission model

Specific Examples of SSMs

Name	p(z(t)|z(t-1)) - Dynamics model	p(y(t)|z(t)) - Observation model
LG-SSM	linear mean - Gaussian noise	linear mean - Gaussian noise
GLM-LG-SSM	linear mean - Gaussian noise	linear mean - Exponential family (GLM)
NLG-SSM	nonlinear mean - Gaussian noise	nonlinear mean - Gaussian noise
GLM-NLG-SSM	nonlinear mean - Gaussian noise	nonlinear mean - Exponential family

Note: The state space models in red can be used for training neural network models (e.g. using the implementation in the Jax Dynamax library).

Sequential filtering in SSMs (predict and update)

PREDICT STEP:

Dynamics model

Previous posterior belief state

$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})p(\boldsymbol{z}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{z}_{t-1}$$

Chapman-Enskog

UPDATE STEP:

Likelihood /Observation model

Prior predictive distribution

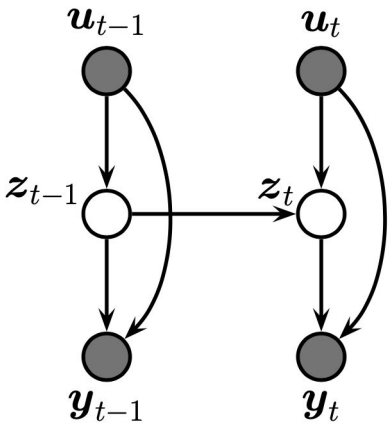
$$p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t}) = \frac{1}{Z_t}p(\boldsymbol{y}_t|\boldsymbol{z}_t)p(\boldsymbol{z}_t|\boldsymbol{y}_{1:t-1})$$

Bayes rule

Updated posterior belief state

Normalization constant

Specific Example: Linear-Gaussian SSMs (aka "Linear Dynamical Systems")



States and observations are continuous, and distributions are Gaussian with linear dependence on their parents.

$$p(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{z}_t|\mathbf{F}_t\boldsymbol{z}_{t-1} + \mathbf{B}_t\boldsymbol{u}_t, \mathbf{Q}_t)$$
$$p(\boldsymbol{y}_t|\boldsymbol{z}_t,\boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{y}_t|\mathbf{H}_t\boldsymbol{z}_t + \mathbf{D}_t\boldsymbol{u}_t, \mathbf{R}_t)$$

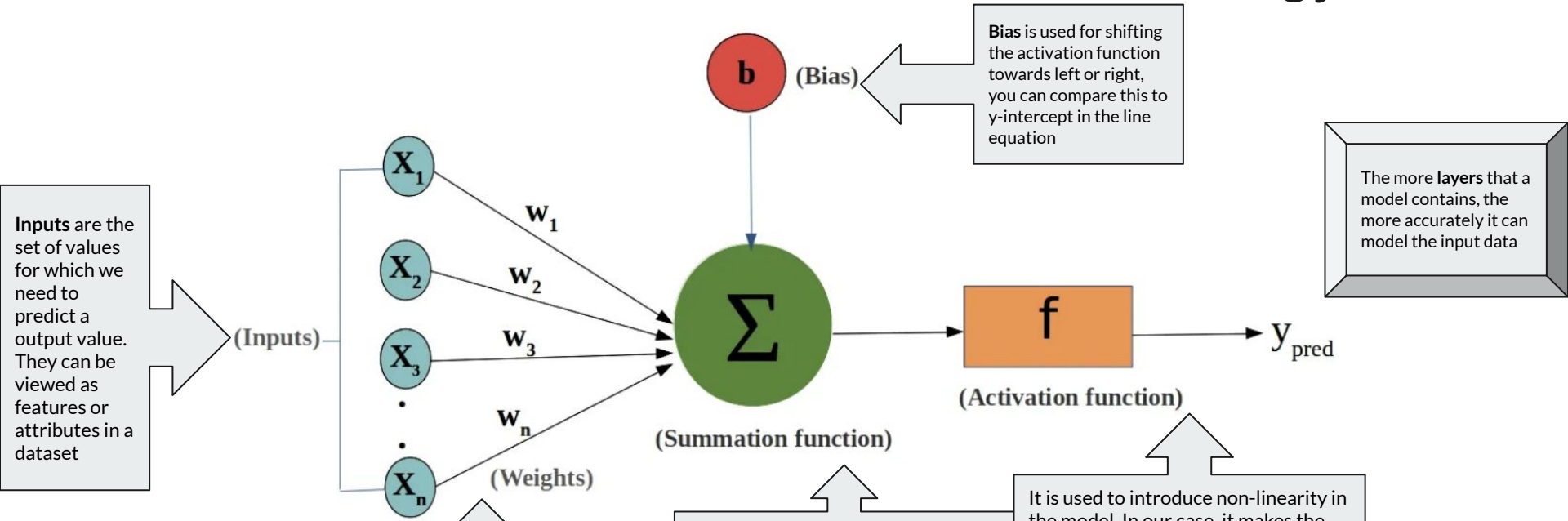
Equivalent Linear Dynamical System

$$\boldsymbol{z}_t = \mathbf{F}_t\boldsymbol{z}_{t-1} + \mathbf{B}_t\boldsymbol{u}_t + \boldsymbol{q}_t, \qquad \boldsymbol{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$$
$$\boldsymbol{y}_t = \mathbf{H}_t\boldsymbol{z}_t + \mathbf{D}_t\boldsymbol{u}_t + \boldsymbol{r}_t, \qquad \boldsymbol{r}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$$

The sequential filtering algorithm here is the well-known **Kalman filter**

DDN Model Training Using GLM-NLG-SSM (with data drift)

Neural Network Architecture (Terminology)



Bias is used for shifting the activation function towards left or right, you can compare this to y-intercept in the line equation

The more **layers** that a model contains, the more accurately it can model the input data

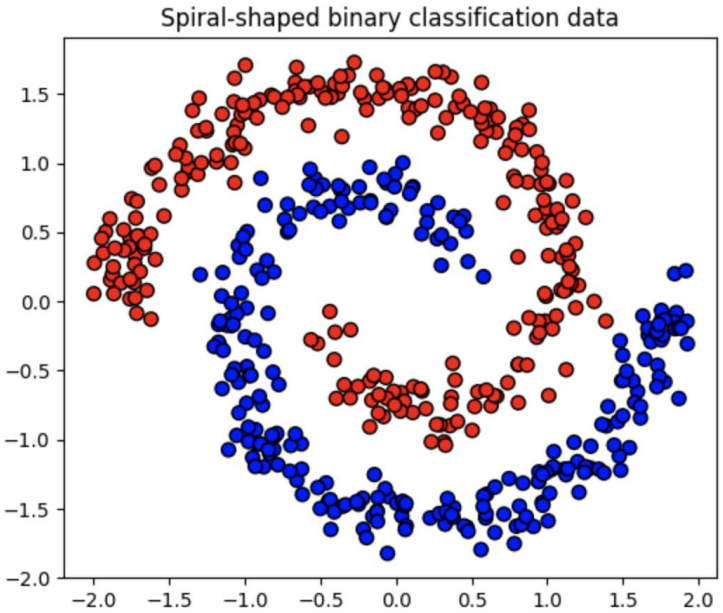
Inputs are the set of values for which we need to predict a output value. They can be viewed as features or attributes in a dataset

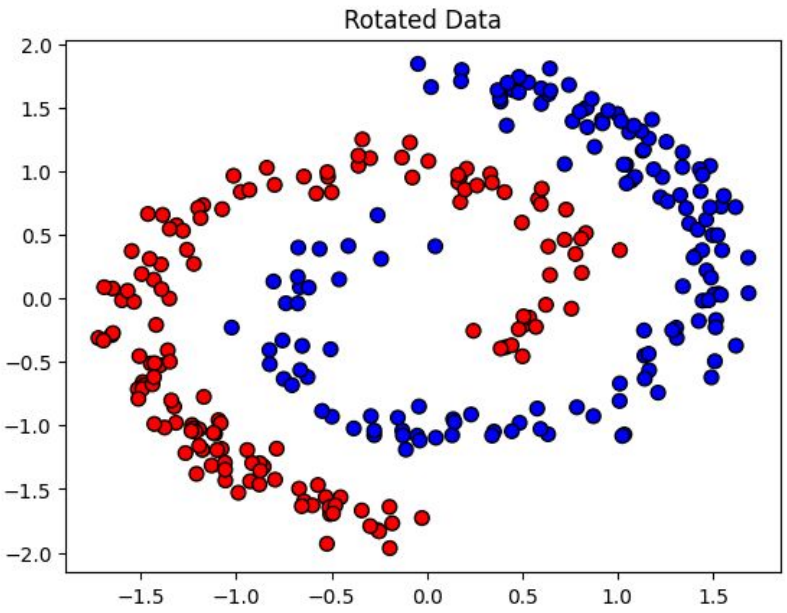
Weights are the real values that are attached with each input/feature and they convey the importance of that corresponding feature in predicting the final output.

The work of the **summation function** is to bind the weights and inputs together and calculate their sum.

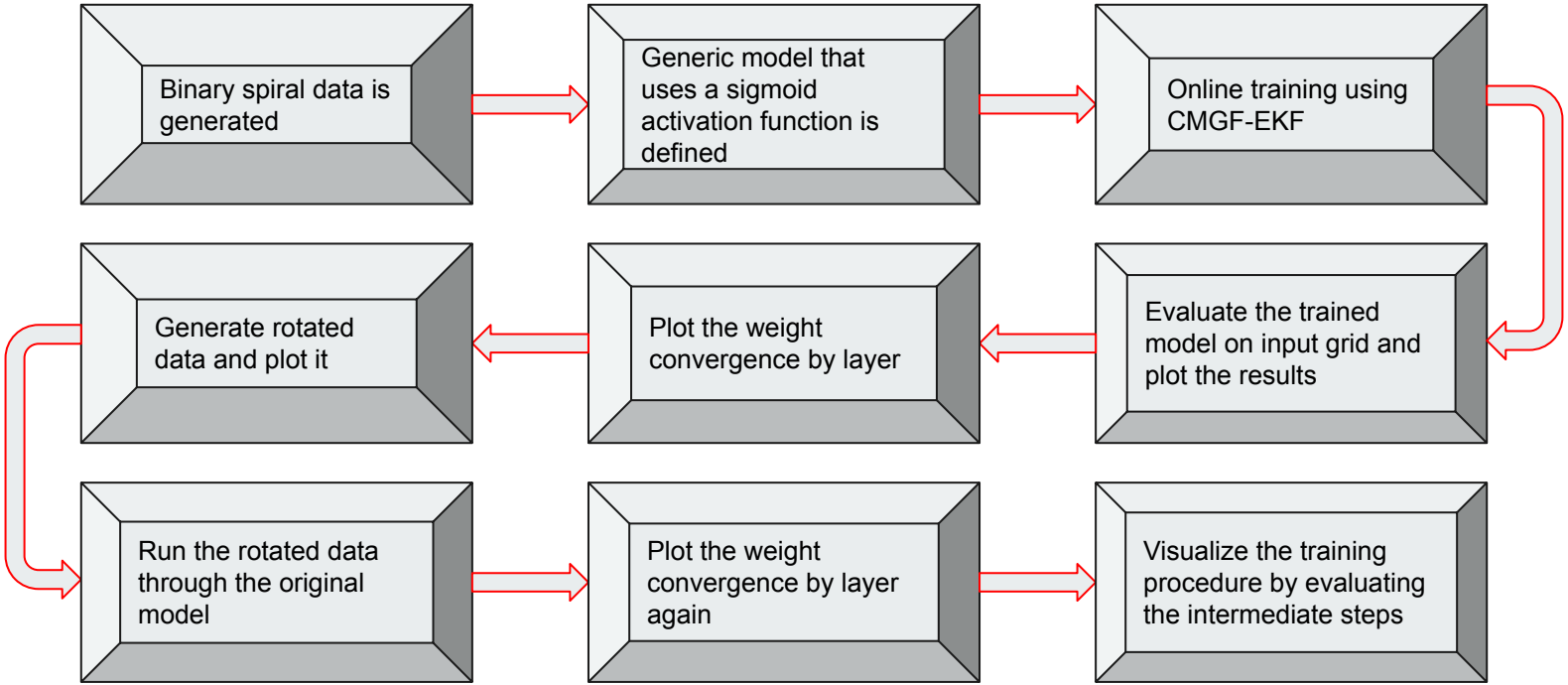
It is used to introduce non-linearity in the model. In our case, it makes the result either 1 or 0, each of which map to a class of the binary data

Initial Data and Rotated Data (Two Spirals)

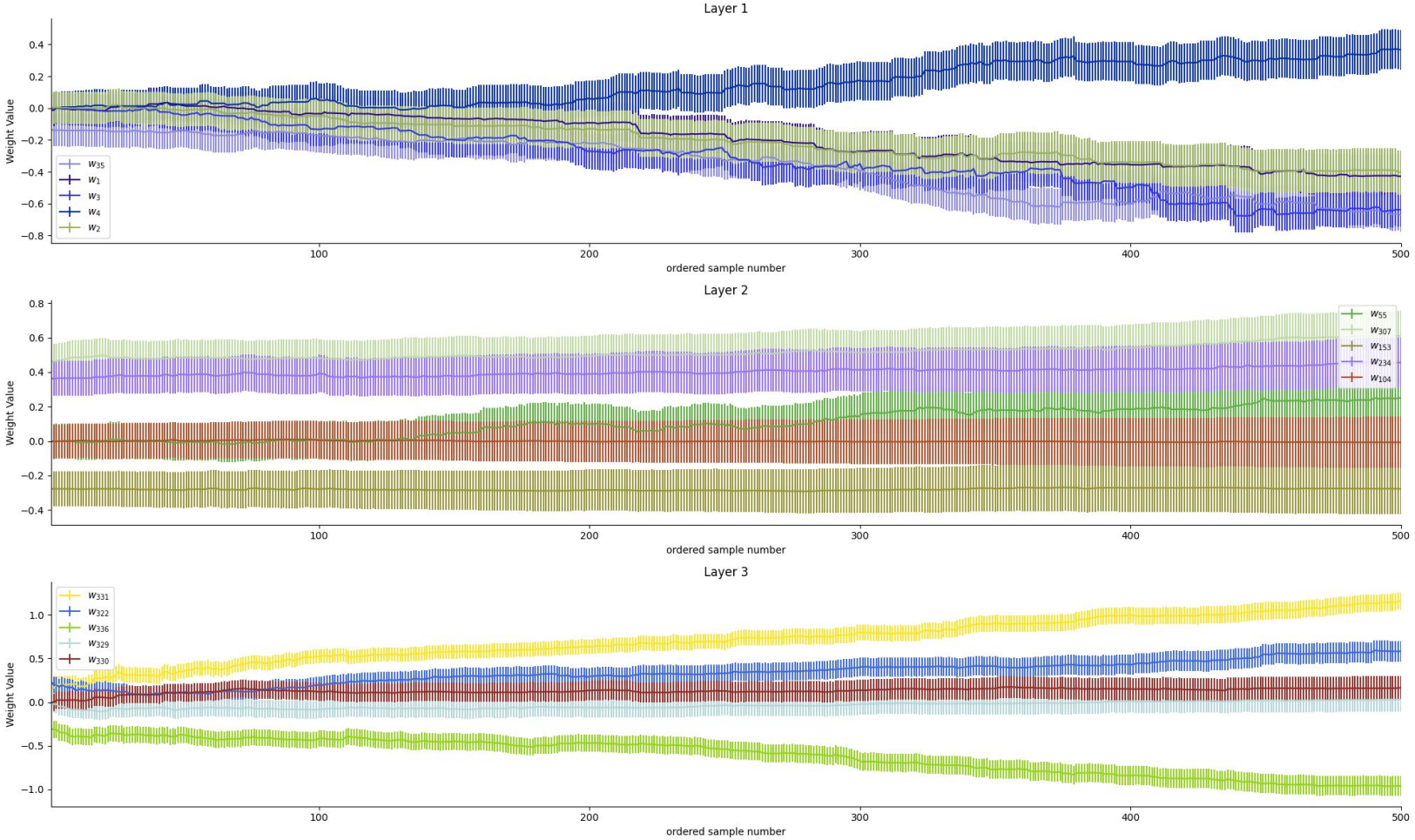




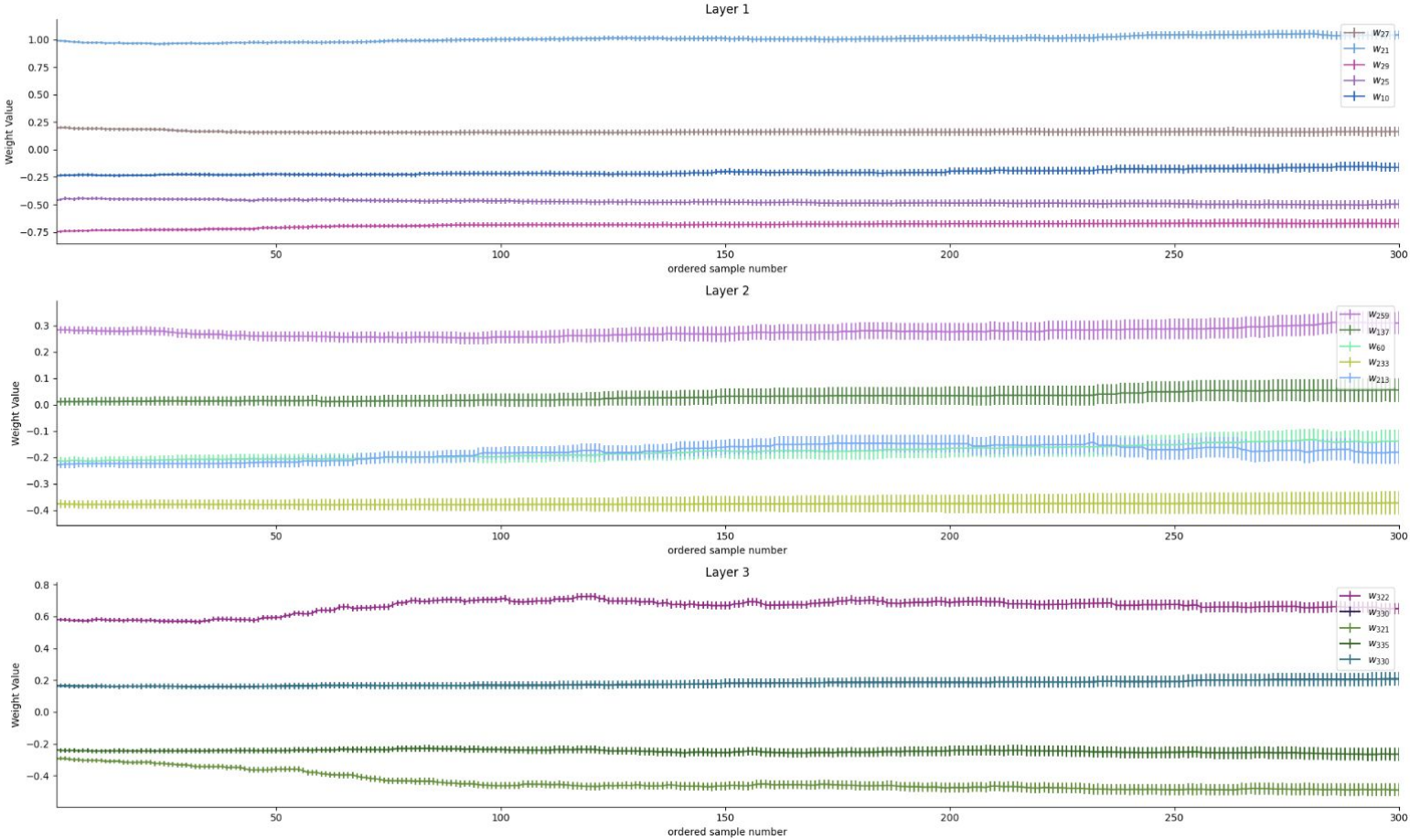
Workflow for DDN with Data Shift



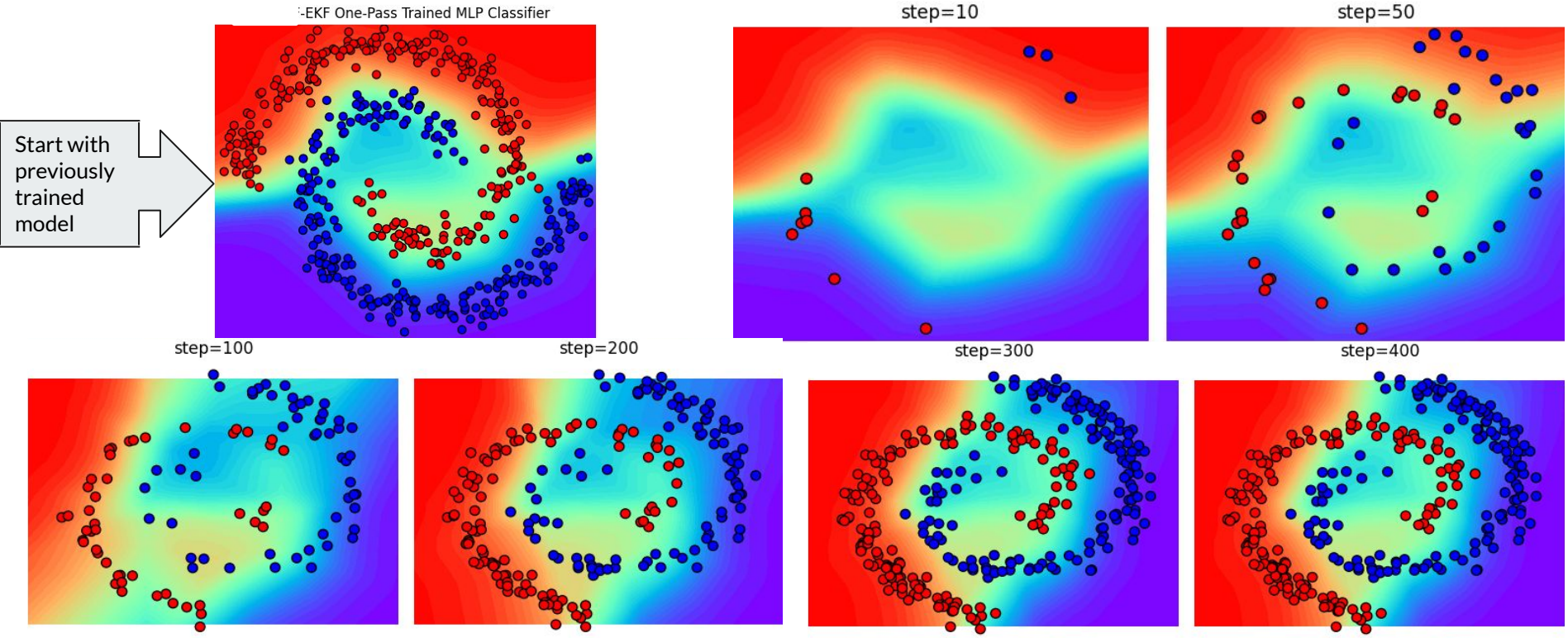
Plotting Weights by layer



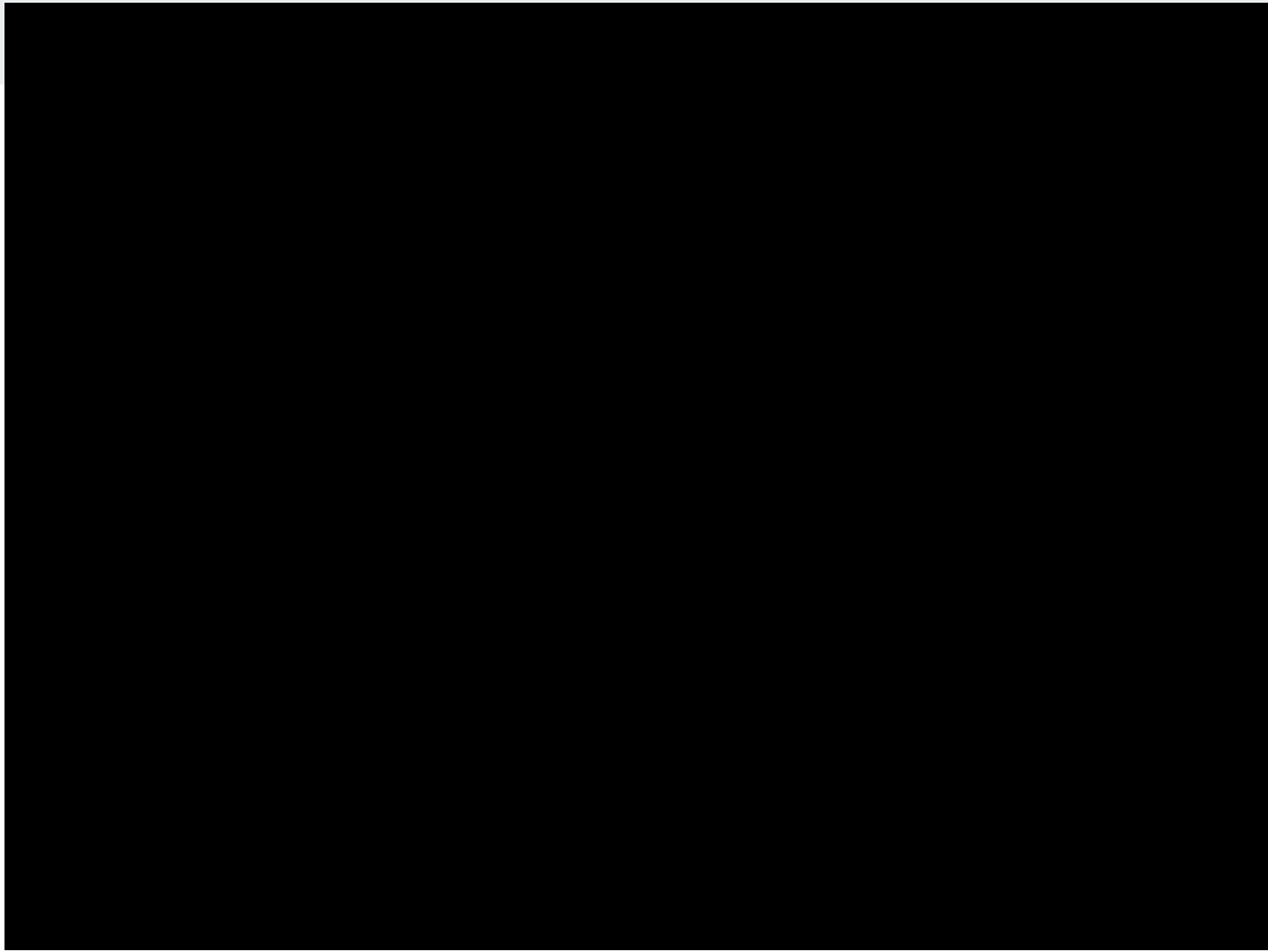
Plotting weights by layers (Post the data rotation)



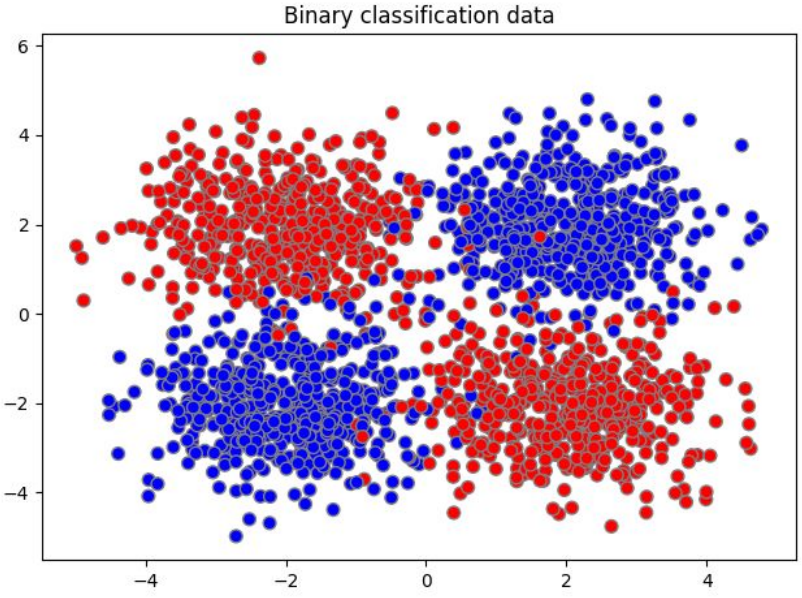
Visualize the intermediate steps in the training

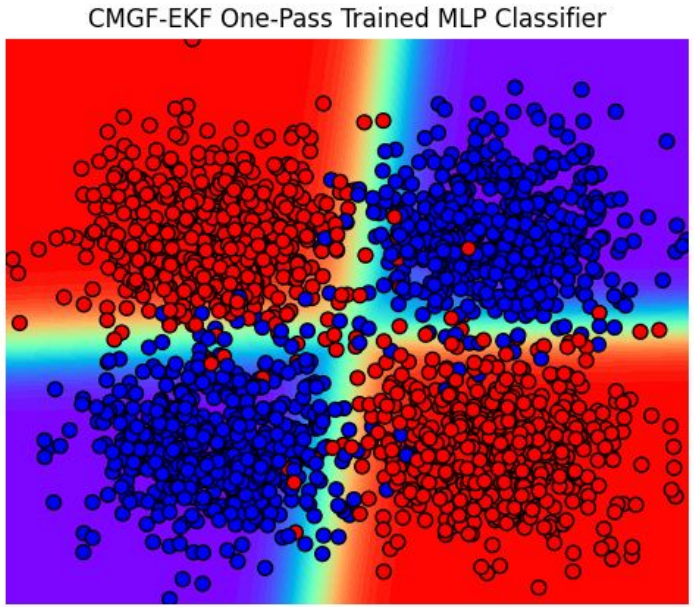


Animation of DDN-Classifier initial training followed by adaptation to new data

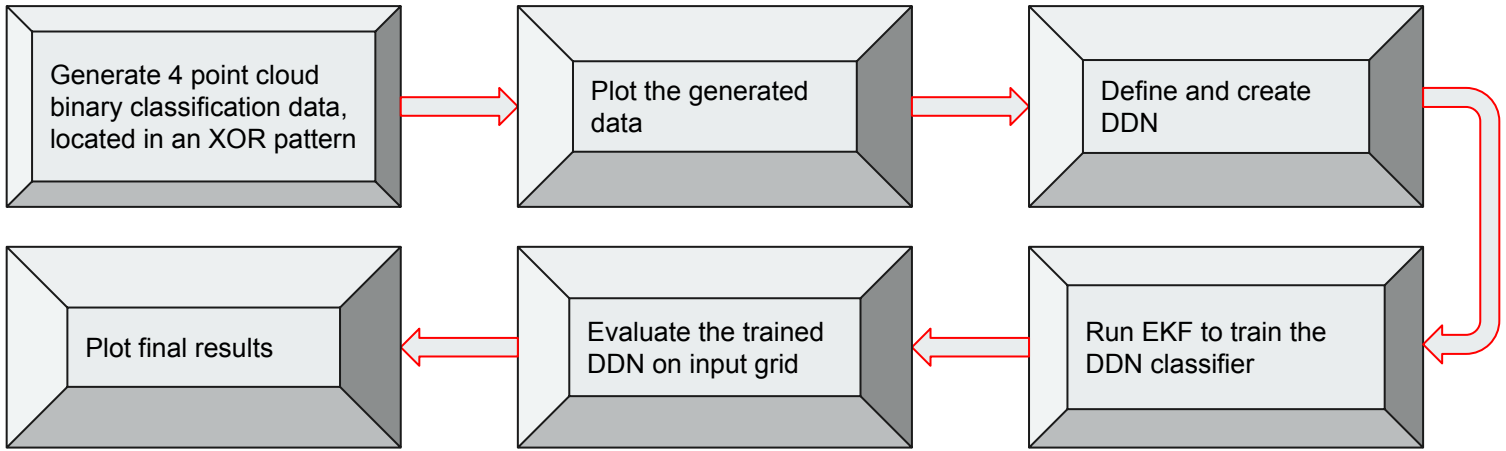


XOR Data Pattern and Predicted Probability

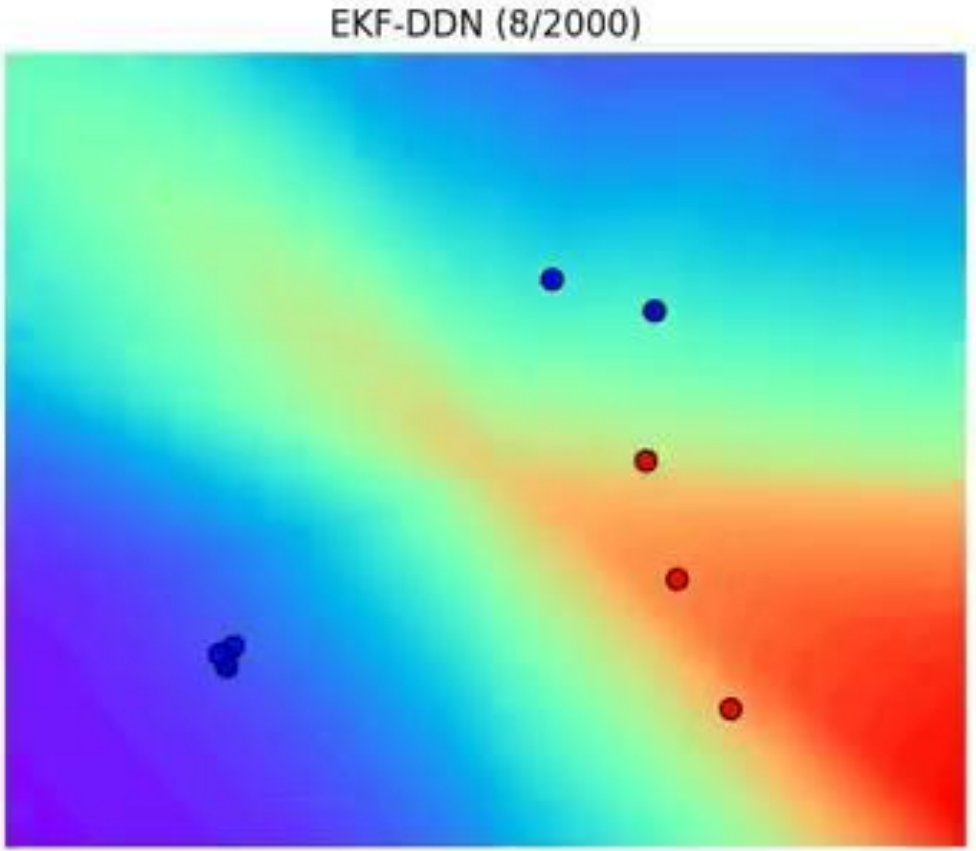




Workflow for DDN for XOR Dataset

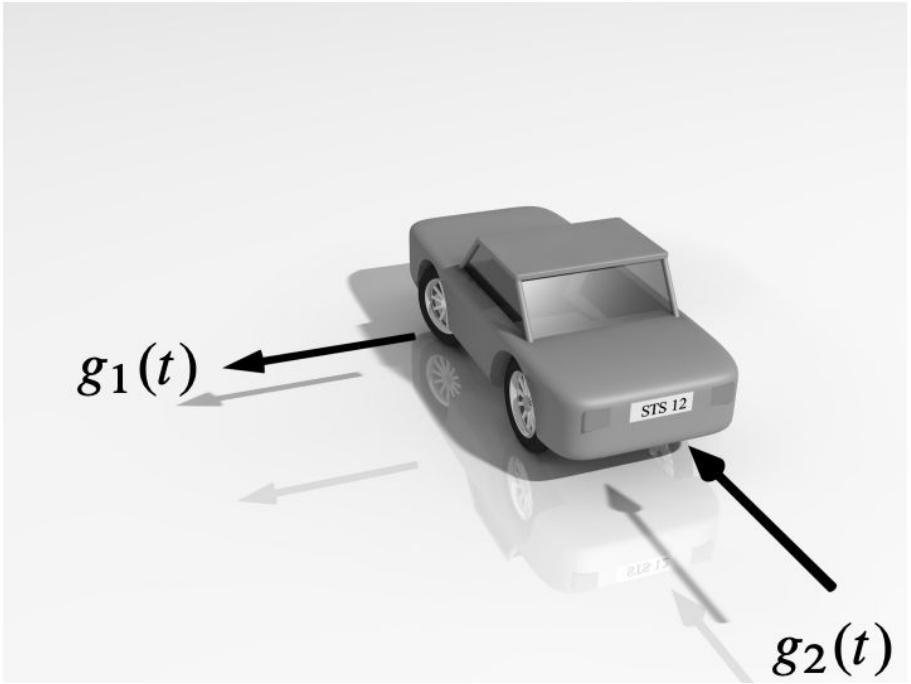


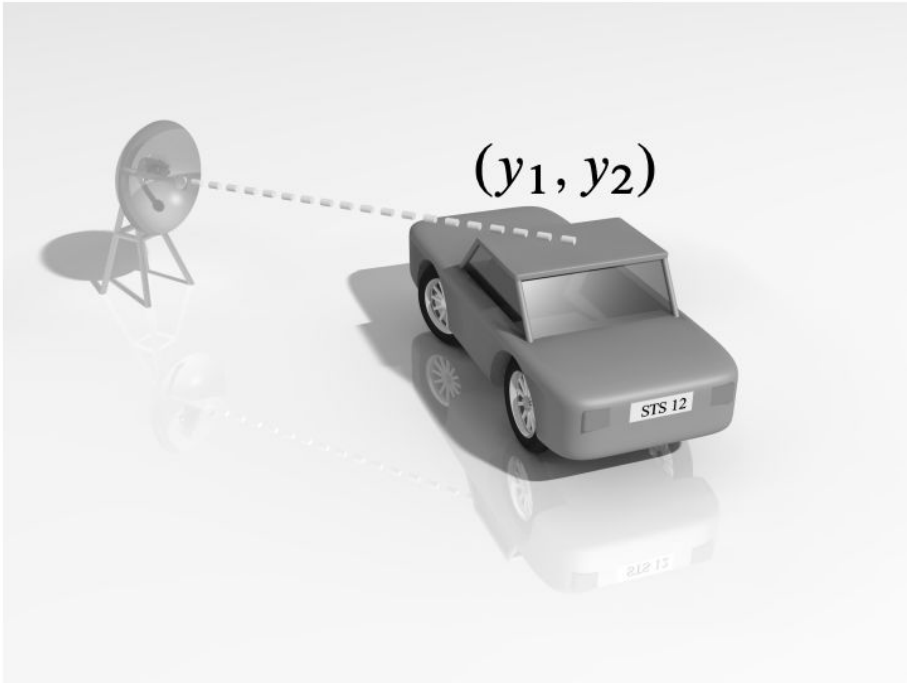
Animation of DDN-Classifier training



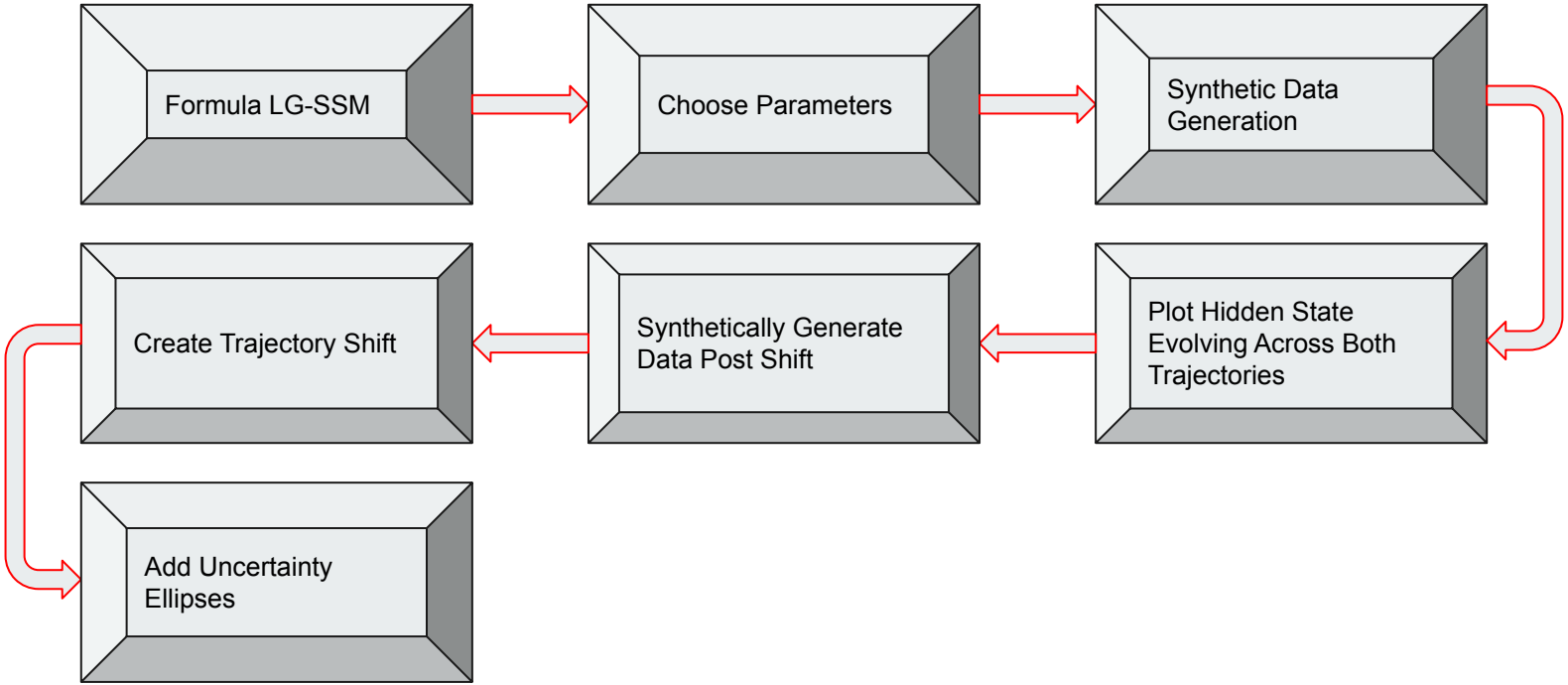
Multiple sensor object tracking for data drift application

2D Object Tracking With Noise

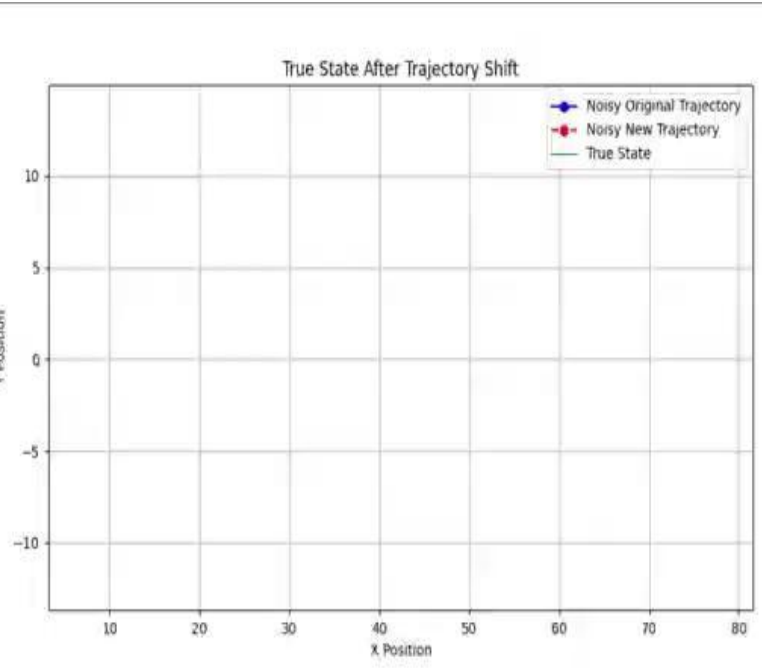


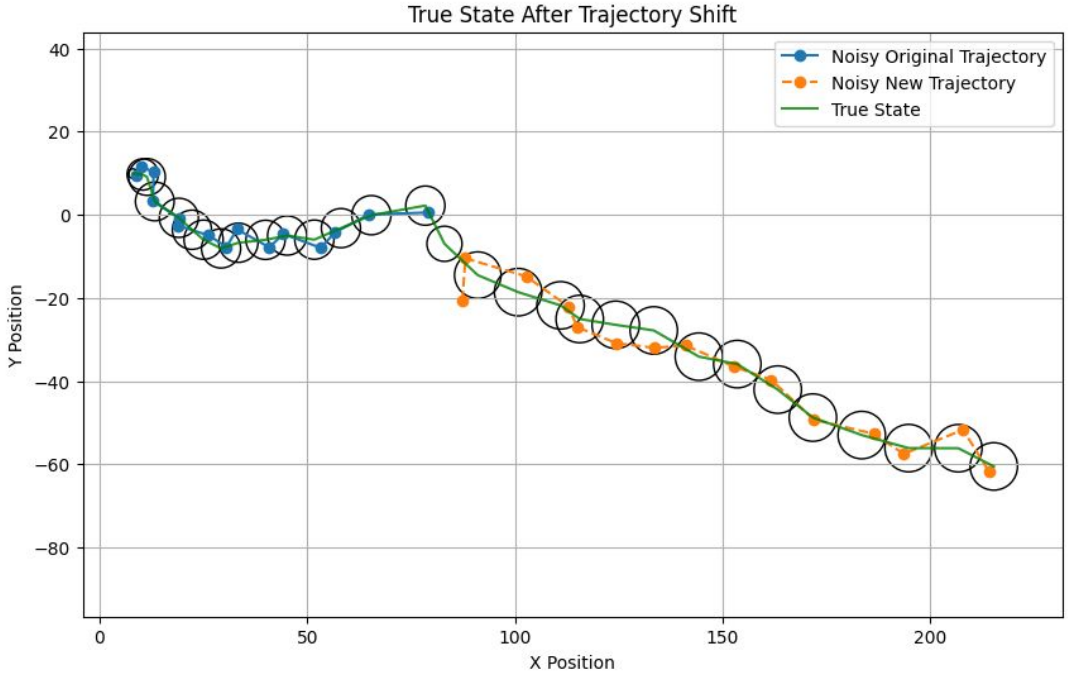


Workflow for Data drift application for sensor based object tracking

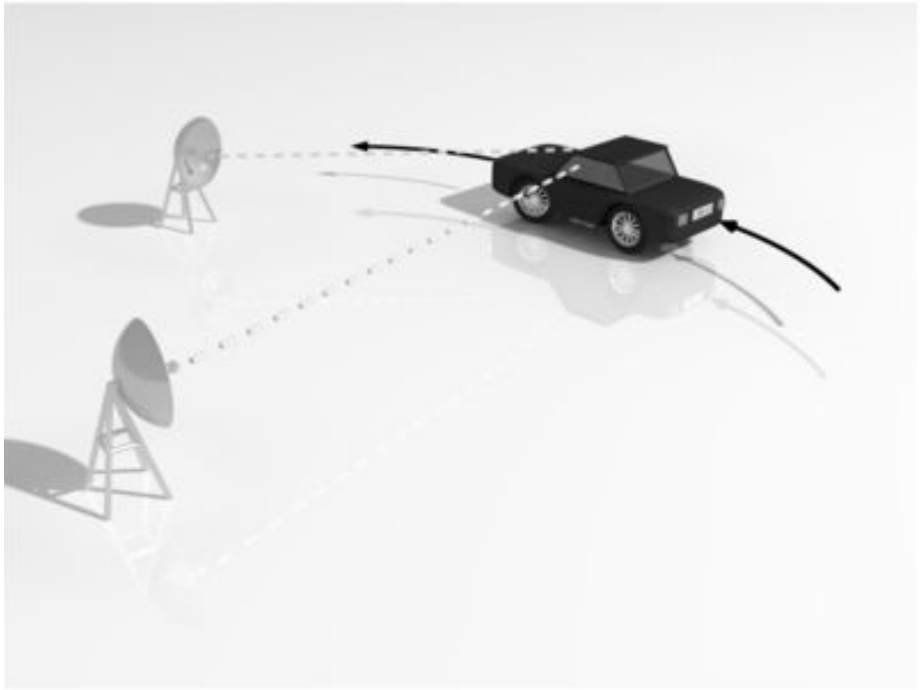


Plot of True state after trajectory shift

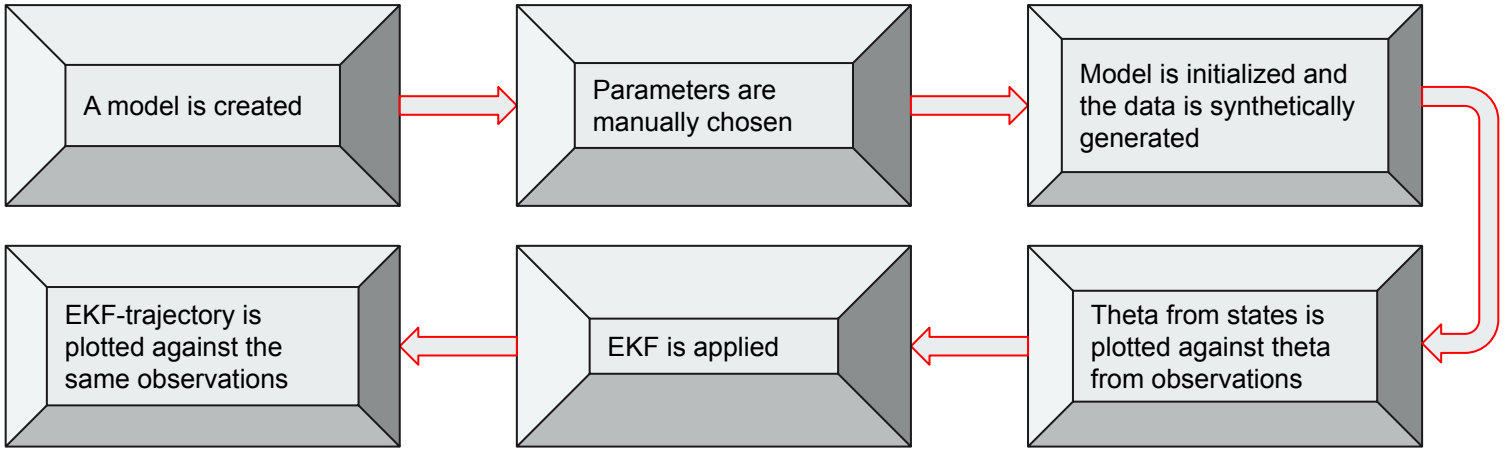




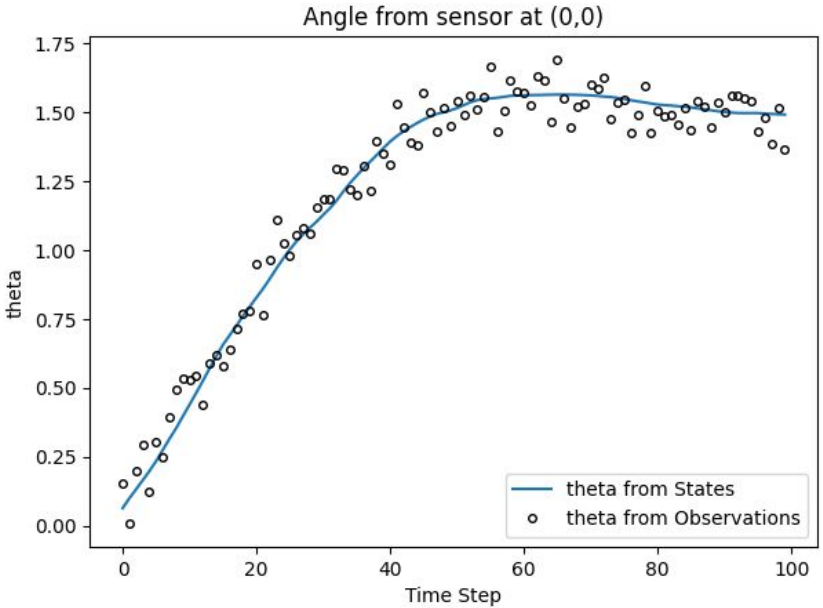
Scenario with 1 object, 2 angular measurement sensors

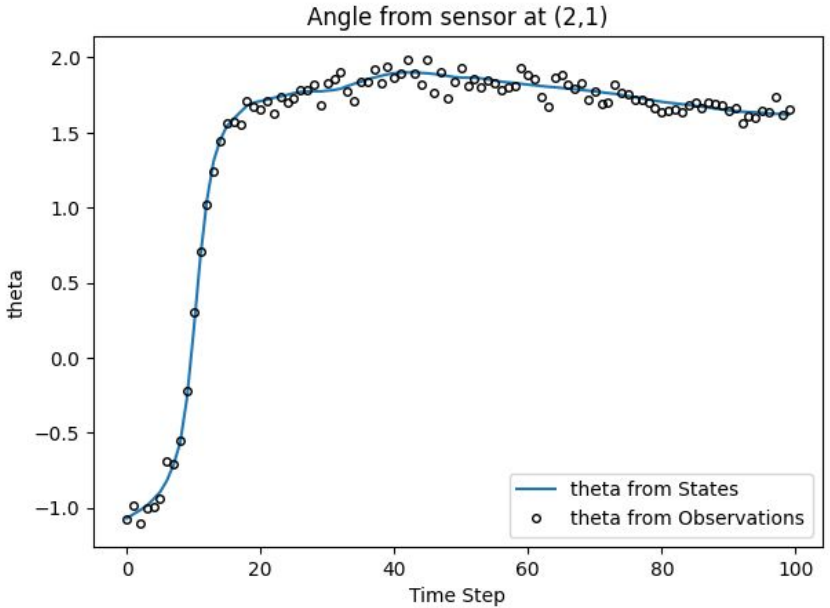


Workflow Multiple sensor object tracking

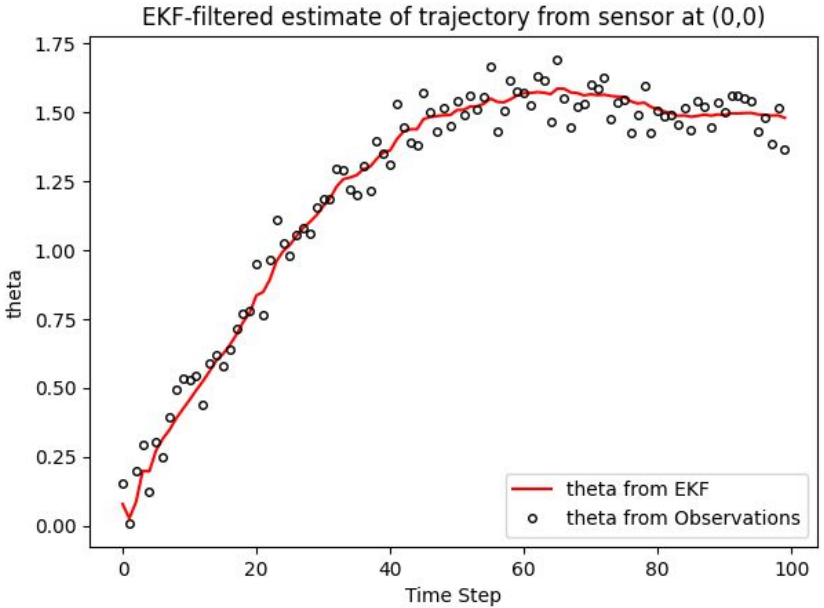


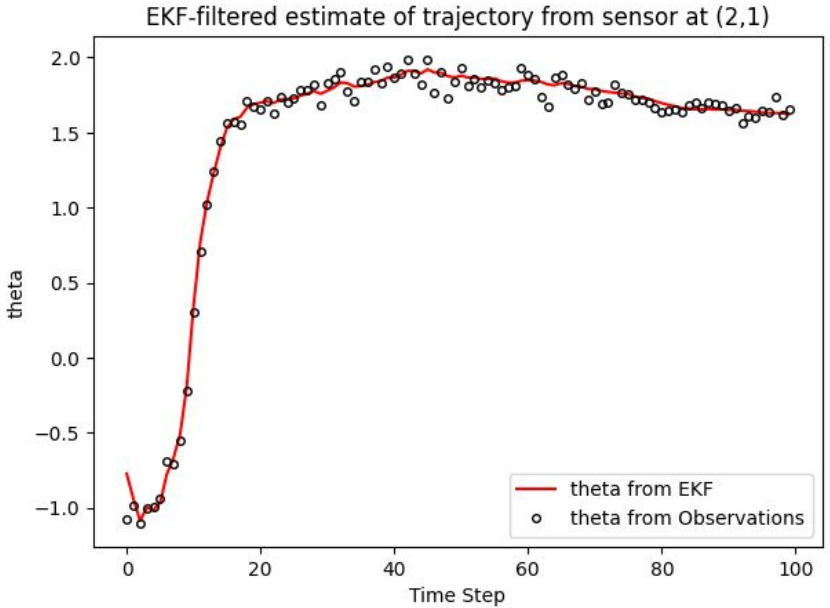
Synthetic Data True State and Observations



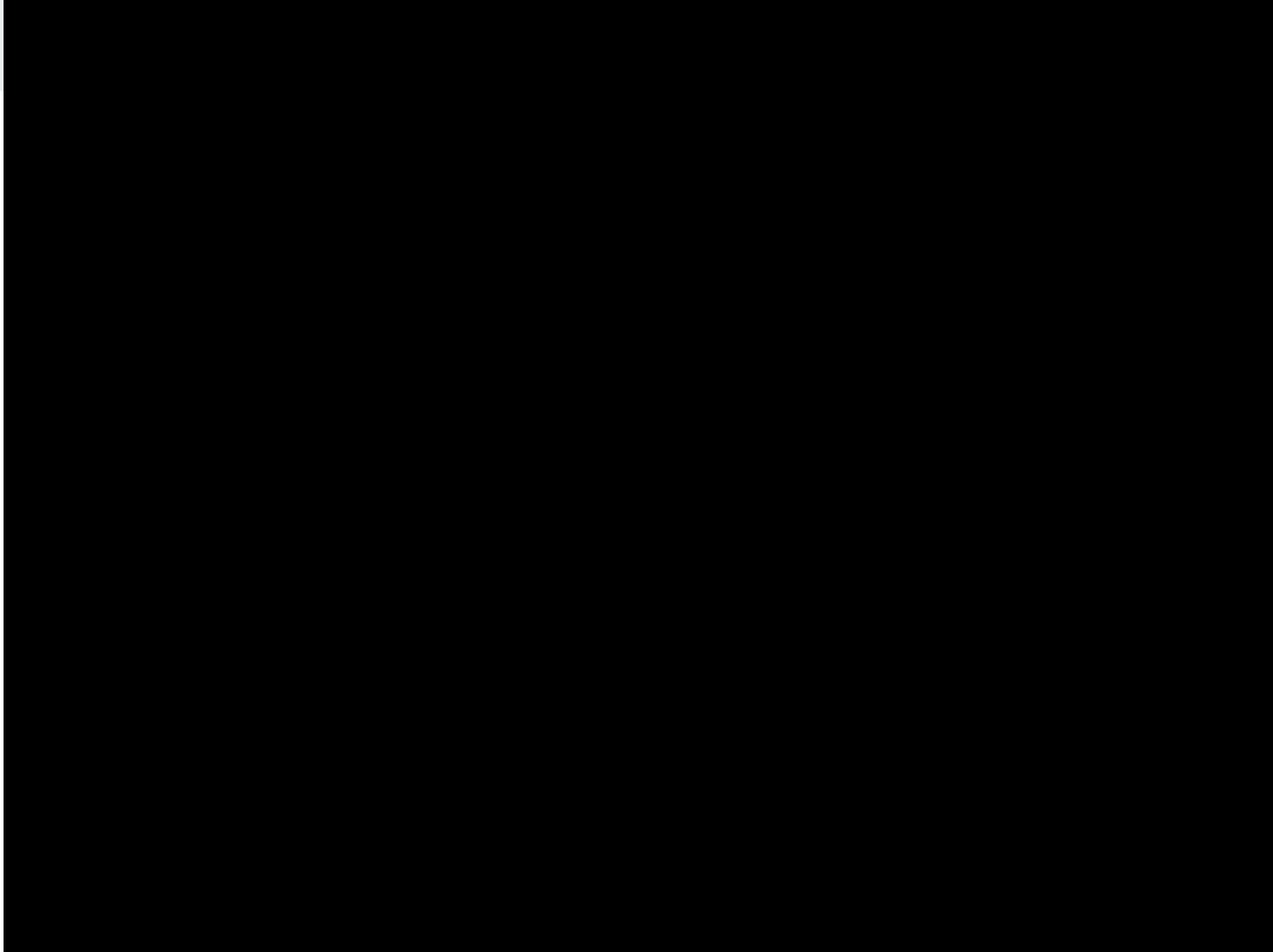


Estimated State from EKF Algorithm





Visualization of the State Estimation Using EKF



Summary

- Dynamic Neural Networks (DDNs) can be trained to adapt to evolving data drift and changing data distributions in the training data
- This approach is based on the Extended Kalman filter (EKF) which directly provides not only the model weights, but also the model weight uncertainties (unlike the standard training methods, e.g. gradient descent).
- The approach has the valuable property that entire model from the old data is not discarded, but only the parts that need modification are adapted to the new training data (this is typically just the final few layers of the neural network)
- The rate of adaptation of different parts of the model to the new data can be finely controlled by “tweaking the corresponding prior uncertainties” at the data transition points, and for this we recommend using the “human-in-the-loop” approach.