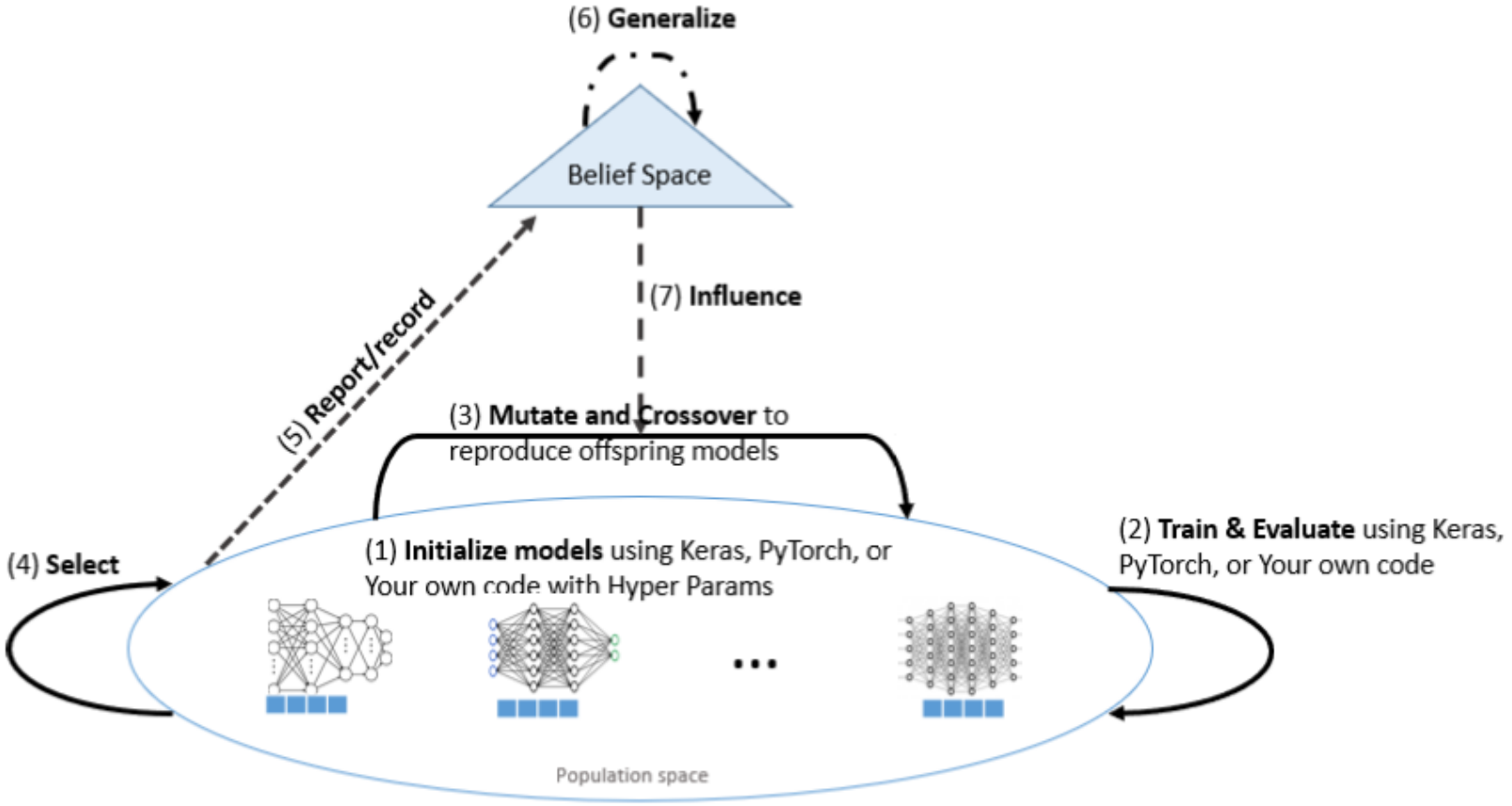
Hyper Parameter Optimization By Evolving Neural Networks



Hyper Parameter Optimization (HPO)

- Hyperparameter tuning/optimization is one of the most important (and sometimes frustrating) parts of building deep learning models.
- It refers to the process of finding the best set of *hyperparameters*—parameters that are not learned during training but must be set before training begins—to maximize model performance.
- Hyperparameters can be thought of as the tuning knobs of your model
- You do not know the best value for a model hyperparameter on a given problem. You may use rules of thumb, copy values used on other similar examples, or search for the best value by trial and error

Hyper Parameters to Tune/Optimize

- Architectire related
  - Architecture type
  - Network Size: Number of hidden layers
  - Number of neurons per layer
  - Activation function
- Optimization related
  - Optimizer algorithm, Loss function
  - Batch size
  - Learning rate
  - Momentum, weight decay, gradient clipping
- Training schedule related
  - Number of epochs
  - Learning rate schedule
  - Early stopping
- Regularization related
  - Dropout rate
  - L1/L2 penalty
  - Data augmentation choices

Why Hyperparameter Tuning Matters

- Too high a learning rate → model may not converge.
- Too small batch size → noisy gradients, longer training.
- Too large model → overfitting.
- Too much regularization → underfitting.

Approaches to Hyperparameter Tuning

- **Manual Search**: Trial-and-error based on intuition and experience. Often used as a first step.
- **Grid Search**: Exhaustively tests all combinations from a predefined set of hyperparameters.Guarantees coverage, but computationally expensive.
- **Random Search**: Samples random combinations from parameter space. Much more efficient than grid search in high-dimensional spaces.
- **Bayesian Optimization** (e.g., using libraries like Optuna, Hyperopt): Builds a probabilistic model of the objective function.Selects the next set of hyperparameters to try based on previous results. More sample-efficient.
- **Population-Based / Evolutionary Methods** (Genetic algorithms, ES, etc.): Treat tuning as an evolutionary process. Good for non-smooth or irregular search spaces.
- **Automated Machine Learning (AutoML) Frameworks**: KerasTuner, Optuna, Ray Tune, AutoKeras.Can automate search and scheduling across multiple GPUs/TPUs.

A Keras Example of HPs

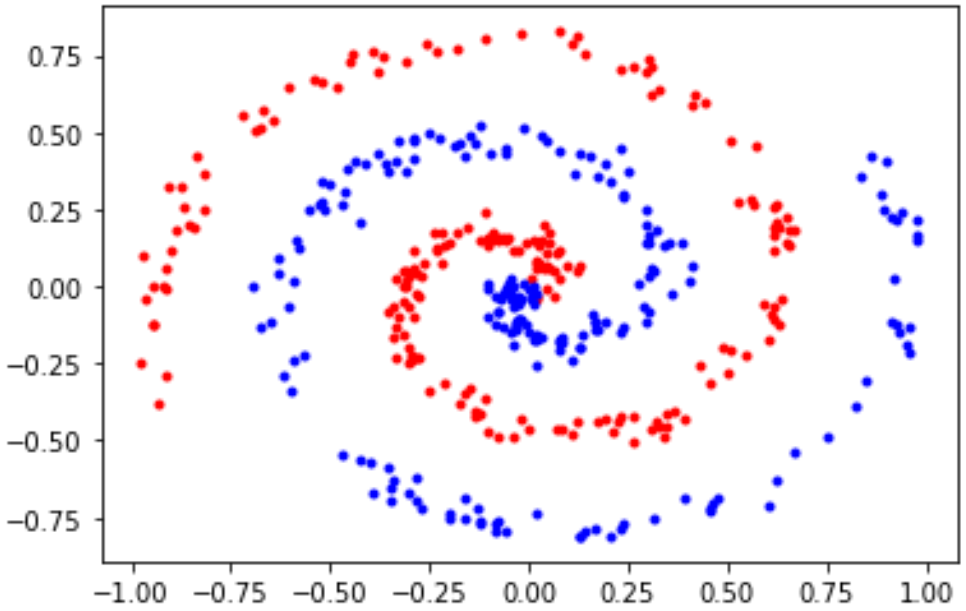
```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

model = Sequential([
    Dense(2, input_dim=2, activation='tanh'),
    Dense(1, activation='tanh')
])

model.compile(
    loss='mean_squared_error',
    optimizer=optimizers.SGD(learning_rate=0.1), # stochastic gradient decent
    metrics=['binary_accuracy']
)
model.fit(X, y, batch_size=1, epochs=2000, verbose=0)
```

Previous Work

Using ES(1+1) 1/5 rule to find optimal hyperparameters for a NN to solve a specific problem, *Two Intertwined* Spiral Problem



Training dataset

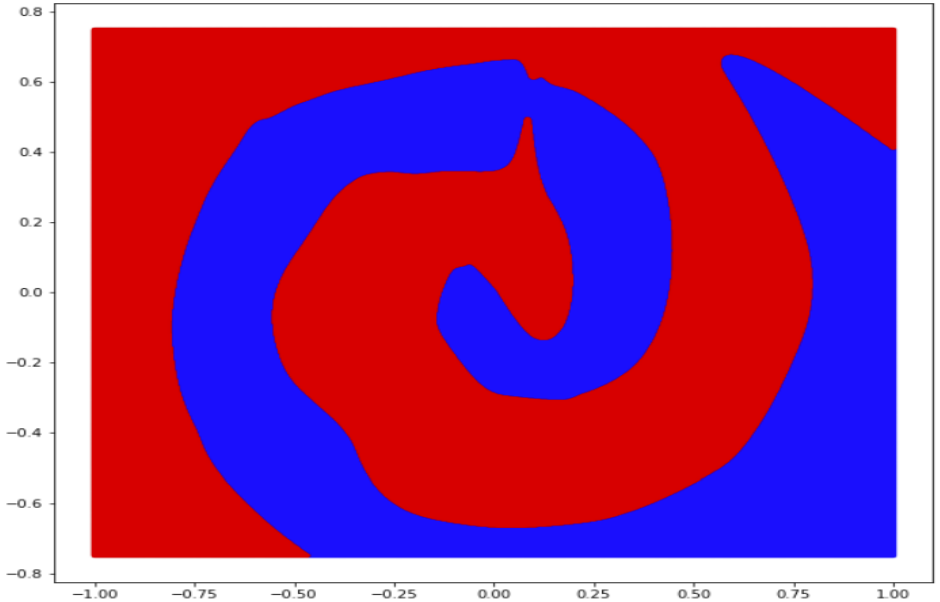


Diagram of the evolved model's prediction on each point on the given grid. Decision boundary diagram

https://ieeexplore.ieee.org/document/11103665

Hyper Parameter Heuristics (Lanham book: p105-108)

- Network Size (Number of hidden layers and number of hidden neurons for each hidden layer): over-complex model may more likely overfit. Therefore, we can directly reduce the model's complexity by removing layers and reduce the size of our model. We may further reduce complexity by decreasing the number of neurons in the fully-connected layers.
- Batch size: If input data is highly varied, then decrease the batch size. Increase the batch size for less varied and more uniform datasets
- Learning rate: If you need to increase the size of a model, generally, decrease the learning rate as well.

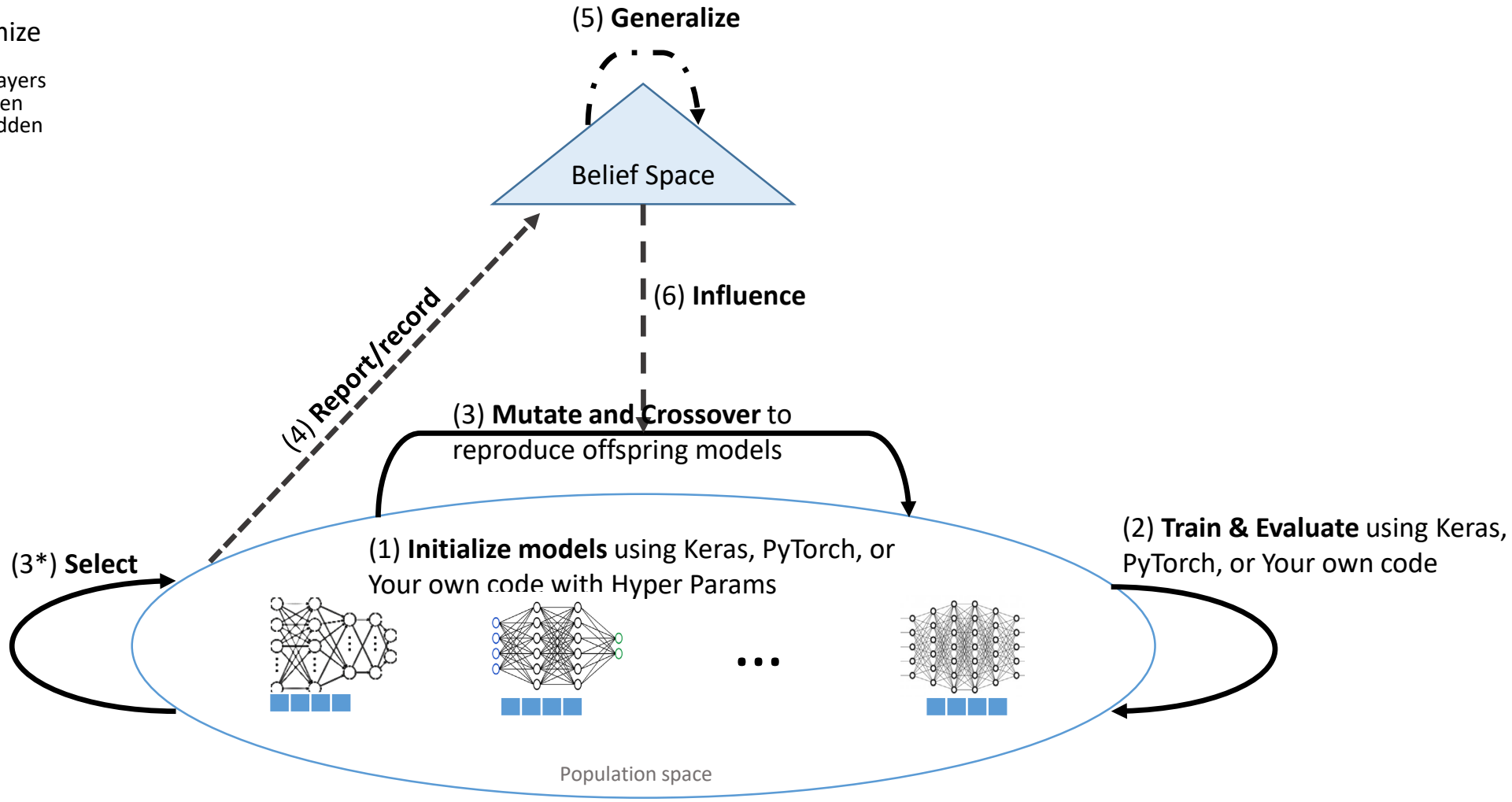
Hyper Parameter Heuristics (Lanham book: p105-108)

- Number of epochs: Always use the highest number of training iterations you think you will need. Use techniques like early stopping and/or model saving to reduce training iterations.
- Anytime you change a model's architecture, optimizer, or loss function, you need to **retune** its hyperparameters because the "best" settings depend on how the model learns under those specific conditions. Altering layers, activation functions, or connections reshapes the optimization landscape, while switching optimizers or loss functions changes the scale and behavior of gradients.

HPO Framework for Neural Network Models using EA

■■■■ Hyper parameters to optimize

- Number of hidden layers and number of hidden neurons for each hidden layer
- learning rate
- activation function
- batch size
- loss function
- optimizer algorithm
- # of epochs
- …



* In ES, selection is done after mutation & crossover

$$\min_{X \in \mathbb{R}} f(X) \tag{1}$$

Contrarily, constrained optimization can be expressed as [59]:

$$\begin{cases} \min f(X) \\ \text{Subject to:} \\ g_i(X) \leq 0, \quad \mathrm{i} = 1, \ldots, \mathrm{m} \\ \mathrm{h_i}(X) = 0, \quad \mathrm{j} = 1, \ldots, \mathrm{p} \end{cases} \tag{2}$$

where $f(X)$ is the problem objective function, X is the domain of X, $\mathrm{g_i}(X) \leq 0, \mathrm{i} = 1, \ldots, \mathrm{m}$, and $\mathrm{h_i}(X) = 0$, $j = 1, \ldots, \mathrm{p}$, are inequality and equality constrained functions, respectively. The feasible domain D of X is as follows:

$$\mathrm{D} = \{X \in \mathrm{X}, \mathrm{g_i}(X) \leq 0, \ \mathrm{h_i}(X) = 0\} \tag{3}$$

The objective of a hyper-parameter optimization task is to get [60]:

$$X^* = \arg\min_{X \in \mathbb{R}} f(X) \tag{4}$$

The best prediction model f^* can be determined by using [61]:

$$f^* = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} L\left(f\left(\mathrm{x_i}\right), \mathrm{y_i}\right) \tag{5}$$

where L is the cost function value for each sample, n is the number of training data points, $\mathrm{x_i}$ is the feature vector of the i-th instance, y_i is the associated output. In supervised learning algorithms, there are numerous distinct loss functions, such as the square of Euclidean distance, cross-entropy, information gain, etc. [61].

https://link.springer.com/article/10.1007/s11063-024-11578-0

ES1115_HPO_XOR.ipynb

Please test this program

```
X  = np.array([[0,0],[0,1],[1,0],[1,1]])
Xt = np.array([[0.1,0],[0,0.9],[0.9,0.2],[0.9,0.9]])
y = np.array([[0],[1],[1],[0]])

# num_neurons, lr, actfun, bsize, lossfun, optmzr
def XOR_eval(param):
  model = Sequential()
  model.add(Dense(int(param[0]), input_dim=2)) # Num hidden neurons
  model.add(Activation(act_func[ round(param[2]) ]))
  model.add(Dense(1))
  model.add(Activation(act_func[ round(param[2]) ]))
  if ( param[5] == 0 ):
    optmzr = optimizers.SGD(learning_rate=param[1]) # stochastic gradient decent
  elif ( param[5] == 1 ):
    optmzr = optimizers.RMSprop(learning_rate=param[1])
  elif ( param[5] == 2 ):
    optmzr = optimizers.Adam(learning_rate=param[1])
  #model.compile(loss='mean_squared_error', optimizer=optmzr, metrics=['binary_accuracy'])
  model.compile(loss=loss_func[ round(param[4]) ],optimizer=optmzr,metrics=['binary_accuracy'])
  # binary_accuracy: Calculates how often predictions match binary labels
  model.fit(X, y, batch_size=int(param[3]), epochs=400, verbose=0)
  (loss, acc) = model.evaluate(Xt, y, verbose=0)
  return loss
```

Results

```
************************* Trial # = 1
Acceptable solution found after 2 iterations:
#neurons=6, lr=0.44000, actF=sigmoid, bsize=1, lossF=binary_crossentropy, optim=RMSprop
Eval=0.00002
************************* Trial # = 2
************************* Trial # = 3
Acceptable solution found after 22 iterations:
#neurons=3, lr=0.63000, actF=sigmoid, bsize=1, lossF=mse, optim=RMSprop
Eval=0.00002
System Success = 66.66666666666666%
Avgerage # of generations used = 20
```

HPO references

- https://medium.com/@jairiidriss/what-is-hyperparameter-optimization-477c15fe8cbe
- https://www.linkedin.com/feed/update/urn:li:activity:7322621737437716480