Technical Appendix for: Kolmogorov-Arnold Networks Still Catastrophically Forget But Differently From MLP

Anonymous submission

Hardware and Software

We ran multiple (10-20) experiments in parallel on one GPU using Nvidia Multi-Process Service.

- **CPU:** Intel(R) Xeon(R) Gold 6244 CPU @ 3.60GHz
- **RAM:** 96G (each process used around 1.5G)
- **GPU:** 1x Quadro RTX 6000 (each process used roughly 1.5G)
- **OS:** 6.9.3-arch1-1

Software versions are provided in a `pyproject.toml` alongside installation instructions in `README.md`.

Best Hyper-parameters

The final/best configuration found by our hyper-parameter search can be found in `src/clkan/conf/scenario/*/*.yaml`.

Hyper-Parameter Search Space

This section defines the hyper-parameter search space that our experiments were conducted over. This matches the search space configuration defined by the `src/clkan/conf/pareto` directory.

Base

- **Optimiser**
  `optimizer` $=$ Adam
- **Learning rate:**
  `lr` $\in [0.00005, 0.1]$
  Defines the step size for the optimiser. The learning rate are sampled using a log uniform distribution.

MLP

The MLP search space inherits from the Base search space.

- **Hidden layer width:**
  `layers_hidden` $\in [1..512]$
  The number of neurons in each hidden layer. Each layer's width is sampled independently.
- **Depth**
  `n_hidden_layers` $\in \{1, 2, 3\}$
  The number of hidden layers in the model.

KAN

The KAN search space inherits from the base search space.

- **Hidden Layer Width**
  `layers_hidden` $\in [1..128]$
  The number of nodes in each hidden layer. Each layer's width is sampled independently. Smaller widths than MLP because each edge can contain many more parameters.
- **Depth**
  `n_hidden_layers` $\in \{1, 2, 3\}$
  The number of hidden layers in the model.
- **Spline order**
  `spline_order` $\in \{1, 2, 3\}$
  The order of the spline controls the smoothness of the B-spline
- **Grid resolution**
  `grid_size` $\in \{3, 6, 9, 12, 15\}$
  The grid size controls the resolution of the B-spline.
- **Grid range** (`grid_range`)
  $\in \{(-3, 3), (-2, 2), (-1, 1), (-4, 4)\}$
  Controls the domain that B-spline are non-zero in.
- **Normalisation**
  `norm` $\in \{$"none", "batch", "layer"$\}$
  The type of normalisation to apply before each hidden layer.
- **Activation penalty**
  `activation_penalty_weight`
  $\in \{0.0, 0.1, 0.5, 1.0, 2.0, 5.0\}$
  An L1 penalty on the activation values.
- **Entropy Penalty**
  `entropy_penalty_weight`
  $\in \{0.0, 0.1, 0.5, 1.0, 2.0, 5.0\}$
  EfficientKAN's (Blealtan and Dash 2024) alternative to entropy based regularisation (Liu et al. 2024).
- **First layer is linear**
  `first_layer_is_linear` $\in \{$"true", "false"$\}$
  If True, the initial layer is replaced with a linear layer. This is useful to reduce the dimensionality before using expensive splines.

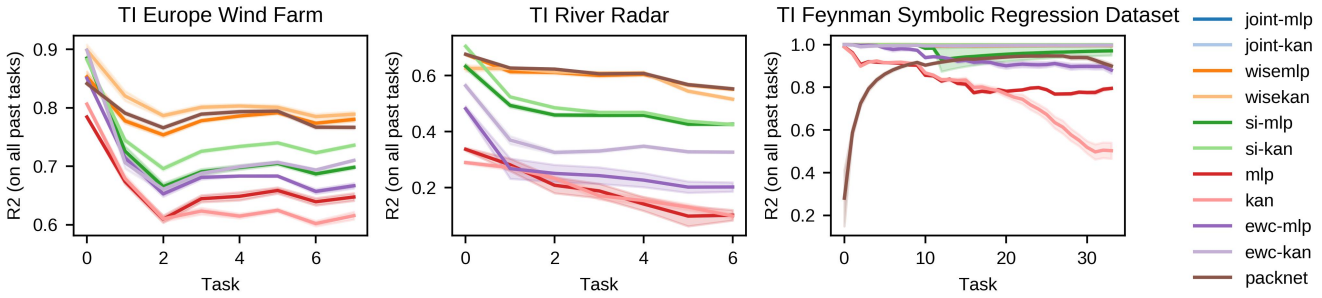

Figure 1: The figure displays R2 on all past tasks after training on each task. Error bars show the standard error for five runs.

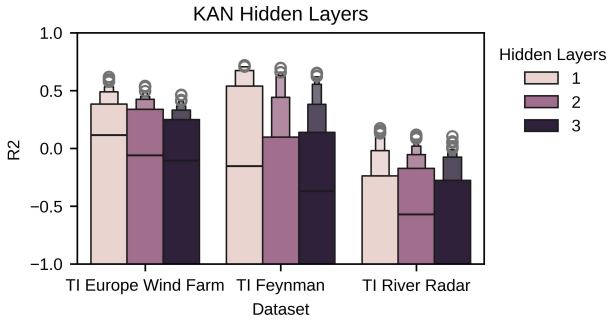

Figure 2: The boxen plot compares the R2 for different numbers of hidden layers. Other variables were randomly sampled from the search space.

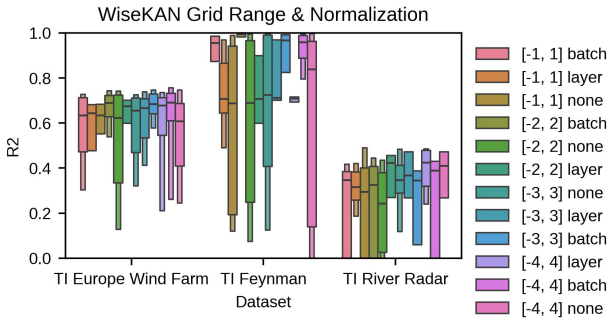

Figure 3: A boxen plot of different grid ranges and normalisation. Other variables are randomly sampled from the search space. Outliers are not plotted.

WiseMLP

The WiseMLP search space inherits from the MLP search space.

- **Start pruning after percentage:**
  `start_coef_prune_percent` $\in [0.1, 0.9]$
  The percentage after which the model starts pruning weights and biases.
- **Early stopping threshold:**
  `early_stopping_threshold` $\in [0.01, 0.3]$ The percentage difference decrease in validation metric (usually R2) that triggers early stopping.
- **Minimum Sparsity**
  `min_sparsity` $\in [0.1, 0.5]$
  If the model is below this sparsity pruning cannot be stopped by early stopping.
- **Target Sparsity**
  `target_sparsity` $\in [0.9, 1.0]$
  The target sparsity that the automated gradual pruning algorithm aims to achieve.

WiseKAN

The WiseKAN search space inherits from the KAN search space.

- **Start Pruning Edges After Percentage**
  `start_edge_prune_percent` $\in [0.1, 0.5]$
  The percentage after which the model starts pruning the spline edges.
- **Start Pruning Coefficients After Percentage**
  `start_coef_prune_percent` $\in [0.5, 0.9]$
  The percentage after which the model starts pruning the coefficients.
- **Early Stopping Threshold**
  `early_stopping_threshold` $\in [0.01, 0.3]$
  The percentage difference decrease in validation metric (usually R2) that triggers early stopping.
- **Minimum Sparsity**
  `min_sparsity` $\in [0.1, 0.5]$
  If the model is below this sparsity pruning cannot be stopped by early stopping.
- **Target Sparsity**
  `target_sparsity` $= 0.95$

The target sparsity that the automated gradual pruning algorithm aims to achieve.

EWC KAN / MLP

EWC MLP and EWC KAN inherit from MLP, and KAN search space respectively.

- **EWC Lambda**
  `ewc_lambda` $\in [0.1, 1000]$
  The strength of the EWC regularisation.

SI KAN / MLP

SI MLP and SI KAN inherit from MLP, and KAN search space respectively.

- **SI Lambda**
  `si_lambda` $\in [0.1, 1000]$
  The strength of the SI regularisation.

PackNet

PackNet inherits from the MLP search space.

- **Prune Amount**
  `prune_ratio` $\in [0.1, 0.9]$
  The proportion of remaining weights and biases that should be pruned.
- **Prune After Percentage**
  `prune_after_p_percent` $\in [0.1, 0.9]$
  Prune after this percentage of training.

Pseudo Code

Python style pseudo code explains the WiseKAN pruning procedure in Listing 1. The function agp refers to the automated gradual pruning procedure outlined by Zhu and Gupta (2017) and defined in Equation 2.

References

Blealtan; and Dash, A. 2024. An Efficient Implementation of Kolmogorov-Arnold Network. https://github.com/Blealtan/efficient-kan.

Liu, Z.; Wang, Y.; Vaidya, S.; Ruehle, F.; Halverson, J.; Soljačić, M.; Hou, T. Y.; and Tegmark, M. 2024. KAN: Kolmogorov-Arnold Networks. ArXiv:2404.19756 [cond-mat, stat].

Zhu, M.; and Gupta, S. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. ArXiv:1710.01878 [cs, stat].

Listing 1: WiseKAN Pruning Procedure

```
# Variables tracking WiseKAN state
stopped_edge_prune = False
stopped_coef_prune = False
best_valid_score = 0.0

# Decide when to start pruning edges and when to prune coefficients
start_edge_prune_epoch = start_edge_prune_percent * num_epochs
start_coef_prune_epoch = start_coef_prune_percent * num_epochs

for task in range(num_tasks):
    for epoch in range(num_epochs):
        snapshot_model = model.snapshot()

        # Get sparsity from schedule using Automated Gradual Pruning
        edge_sparsity = agp(epoch, start_edge_prune_epoch, num_epochs, target_sparsity)
        coef_sparsity = agp(epoch, start_coef_prune_epoch, num_epochs, target_sparsity)

        # Prune and fit
        if not stopped_edge_prune:
            model.prune_spline(edge_sparsity)
        if not stopped_coef_prune:
            model.prune_coef(coef_sparsity)
        model.fit_task()

        # Determine if we should stop pruning using validation set
        valid_score = model.validation_eval()
        best_valid_score = max(best_valid_score, valid_score)
        relative_diff = (valid_score - best_valid_score) / best_valid_score
        if -relative_diff > early_stopping_threshold:
            # Rollback to previous snapshot
            model = snapshot_model

            # Stop one type of pruning at a time
            if not stopped_edge_prune:
                stopped_edge_prune = True
            elif not stopped_coef_prune:
                stopped_coef_prune = True
```