REI 603M - Design Pattern Presentation
Hyperparamter tuning

Andri Freyr Viðarsson

University of Iceland

2021

Introduction

- In machine learning model parameters are generated during training and make up the prediction function
  - Linear Models: coefficients and biases
  - Decision Trees: breakpoints
  - Neural Networks: weights that connect neurons
- Hyperparameters affect model architecture and how the models learn in the training phase
  - Tree Ensemble Models: number of trees, max depth of each tree
  - Neural Networks: number of hidden layers, number of neurons in each layer, number of epochs, batch size, learning rate

Problem description

Hyperparameter tuning

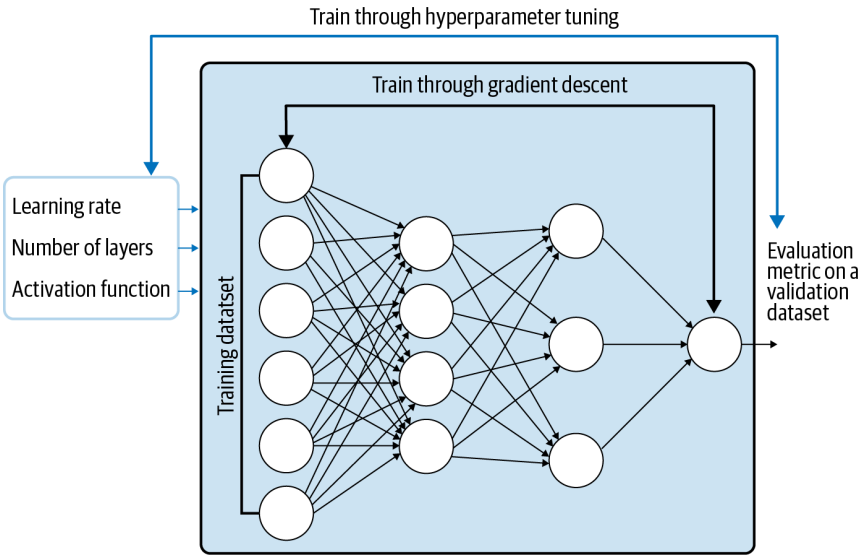
The problem at hand is to find a systematic way of choosing optimal or near optimal set of hyperparameters to use in model training.

- We often find ourselves manually tuning hyperparameters using trial and error or some form of intuition.
- In the following slides simple and more advanced methods to optimize hyperparameters are presented.

Grid search

- The simplest structured form of hyperparameter tuning is grid search
- For each hyperparameter we want to tune a list of values is chosen. Then all combinations of values from those lists are tried out and the best combination is selected.
- Grid search can take a long time to run
  - For each combination of hyperparemters a new model is trained.
  - Not efficient to try all combinations.

Solution



- Use an approach that *learns* to find the optimal hyperparameters
- Think of hyperparameter tuning as an outer optimization loop

Complexity

- Hyperparameters fall into two groups
  - Model architecture parameters:
    - Control the underlying mathematical function of the machine learning model
  - Model training parameters:
    - Control the training loop and the how the gradient descent optimizer works
- We cannot describe the process of finding optimal parameters with a differentiable function.
- To *learn* the optimal hyperparameters nonlinear optimization methods that apply to nondifferentiable problems have to be used.

Bayesian optimization

Bayesian Statistics

In Bayesian data analysis prior knowledge about an event is combined with observed data to form a probability distribution distribution that describes the event[1]. By assuming some sampling distribution $p(y|\theta)$ and some prior distribution $p(\theta)$ we obtain the posterior $p(\theta|y)$ by applying Bayes' Theorem

$$p(\theta|y) = \frac{p(\theta)p(y|\theta)}{p(y)}.$$

Where y is the observed data, θ are unknown parameters and

$$p(y) = \begin{cases} \sum_\theta p(\theta)p(y|\theta), & \text{if } \theta \text{ is discrete} \\ \int_\theta p(\theta)p(y|\theta)d\theta, & \text{if } \theta \text{ is continuous.} \end{cases}$$

[1] https://en.wikipedia.org/wiki/Bayesian_statistics

- Bayesian optimization is a technique that is used to optimize black-box functions, first introduced in the 1970s, first applied to hyperparameter tuning in 2012.

Bayesian optimization step by step

1. Select hyperparameters we want to tune and a range of values for each parameter.
2. Define the *objective function* as the process of training our model on data.
3. Emulate the *objective function* using Bayesian inference, define this function as the *surrogate function*, the inputs to the function are model hyperparameters and the output is an optimization metric.
4. Determine optimal hyperparameters by calling the *surrogate function* repeatedly.
5. Run a full training cycle using the optimal parameters and feed results back to the *surrogate function*.
6. Repeat steps 3-5 for a number of trials.

Code Implementation with `keras-tuner`

```
def build_model(hp):
    model = keras.Sequential([
      keras.layers.Flatten(input_shape=(28, 28)),
      keras.layers.Dense(hp.Int('first_hidden', 128, 256, step=32), activation='relu'),
      keras.layers.Dense(hp.Int('second_hidden', 16, 128, step=32), activation='relu'),
      keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(
      optimizer=tf.keras.optimizers.Adam(
        hp.Float('learning_rate', .005, .01, sampling='log')),
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'])

    return model

tuner = kt.BayesianOptimization(
    build_model,
    objective='val_accuracy',
    max_trials=30
)

tuner.search(x_train, y_train, validation_split=0.1, epochs=10)
best_hps = tuner.get_best_hyperparameters(num_trials = 1)[0]

# Build and train model with optimal parameters
model = tuner.hypermodel.build(best_hps)
history = model.fit(x_train, y_train, epochs = n_epochs, validation_split = 0.1)
```

Trade-offs and Alternatives

Using `keras-tuner` doesn't scale well to complex models because of the likelihood of machine error or other failure. The book recommends to use a fully managed service to perform hyperparameter tuning. The Google Cloud AI Platform offers a hyperparameter tuning service that is used internally at Google.

Genetic Algorithms

Genetic algorithms are roughly based on Charles Darwin's evolutionary theory of natural selection. They solve search problems by repeatedly narrowing down the search space by "survival of the fittest" according to some metric.
Applying this approach to hyperparameter tuning is done by first selecting a search space, then generating random combinations of parameters from the search space. After running trials with the combinations, values from the best combinations according to the evaluation metric are chosen to define the new search space. The new search space is then used to generate new combinations. This process is repeated until some pre defined requirement is satisfied.

Questions