Scaling law

2024-05-07

adaptive collection of more data

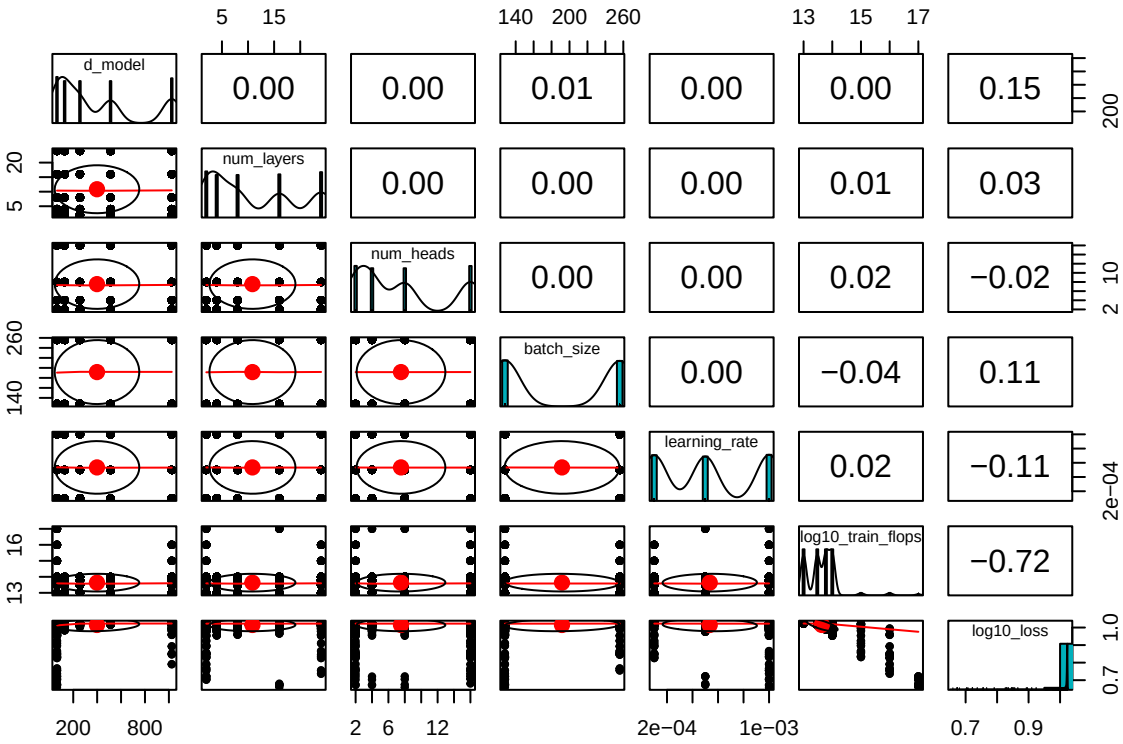
Based on the hyper param trend earlier, up to 1e16 flops the model seems to favor small batch_size, large learning rate, and small d_model. We collect following additional data

```
for d_model in [64]:
  for num_layers in [2, 16, 24]:
    for num_heads in [2, 4, 8, 16]:
      for batch_size in [128]:
        for learning_rate in [5e-4, 1e-3]:
          for train_flops in [int(1e17)]:
```

cor plots

Checking cor plots

```
pairs.panels(df,
             method = "pearson", # correlation method
             hist.col = "#00AFBB",
             density = TRUE,  # show density plots
             ellipses = TRUE # show correlation ellipses
)
```



argmin cor

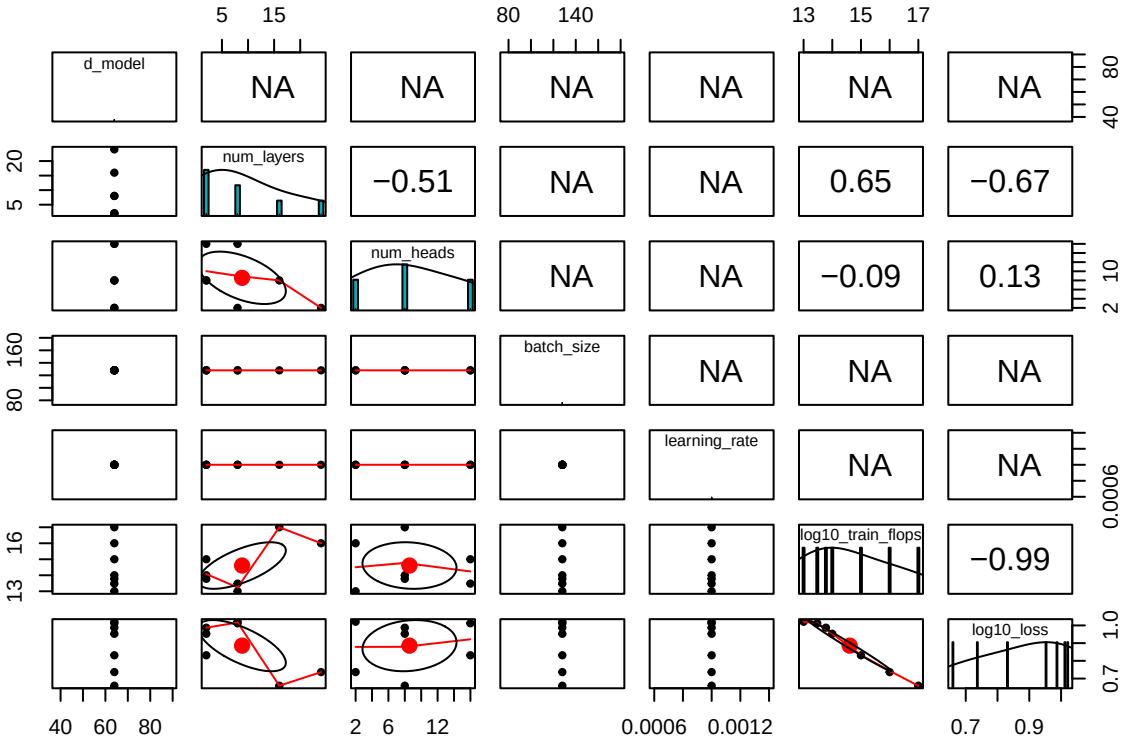
Argmin cor again.

```
# Function to find the row with the smallest loss for each unique log10_train_flops
get_min_loss_per_flop = function(df) {
  # Find unique values of log10_train_flops
  unique_flops = unique(df$log10_train_flops)

  # Initialize an empty data.table to store results
  result = data.table()

  # Iterate over each unique log10_train_flops value
  for (flop in unique_flops) {
    # Subset the data.table for the current log10_train_flops
    subset_dt = df[log10_train_flops == flop]

    # Find the row with the smallest loss within the subset
    min_loss_row = subset_dt[which.min(log10_loss)]

    # Bind the row with the smallest loss to the result data.table
    result = rbind(result, min_loss_row)
  }

  return(result)
}

# Assuming df is your data.table
argmin_df = get_min_loss_per_flop(df)
argmin_df
```

```
##    d_model num_layers num_heads batch_size learning_rate log10_train_flops
## 1:      64          8         2        128         0.001          13.00000
## 2:      64          8        16        128         0.001          13.47712
## 3:      64          2         8        128         0.001          13.77815
## 4:      64          2         8        128         0.001          14.00000
## 5:      64          2        16        128         0.001          15.00000
## 6:      64         24         2        128         0.001          16.00000
## 7:      64         16         8        128         0.001          17.00000
##    log10_loss
## 1:  1.0208374
## 2:  1.0120776
## 3:  0.9870799
## 4:  0.9527515
## 5:  0.8313305
## 6:  0.7365705
## 7:  0.6596009
```

We observe the model consistently favors small batch_size, large learning rate, and small d_model.

```
pairs.panels(argmin_df,
             method = "pearson", # correlation method
             hist.col = "#00AFBB",
             density = TRUE,  # show density plots
             ellipses = TRUE # show correlation ellipses
)
```



Fitting linear regression

```
library(data.table)

# Assuming argmin_df is already your data.table
# Step 1: Remove the first row where log10_train_flops is 13
argmin_df_filtered = argmin_df[log10_train_flops != 13.00000 | !seq_len(.N) == 1]

# Step 2: Fit a linear model to the filtered data
model = lm(log10_loss ~ log10_train_flops, data = argmin_df_filtered)

# Step 3: Predict log10_loss for log10_train_flops = 19 (log10 of 1e19)
predicted_loss = predict(model, newdata = data.frame(log10_train_flops = 19))

# Print the predicted value
10**predicted_loss
```

```
##        1
## 2.743575
```

Conclusion

We choose `num_layers=16, d_model=64, num_heads=8, batch_size=128, learning rate = 1e-3`. We predict the losos to be `2.7`.