Keras with JAX

Chen Gao

2026-01-07

Table of contents

Keras with JAX

Let's start with the imports.

```python
import jax
import jax.numpy as jnp
import os
import warnings
import time
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

%config InlineBackend.figure_format = 'svg'
warnings.filterwarnings("ignore")
os.environ["KERAS_BACKEND"] = "jax"
```

```python
import keras
from keras import Sequential
from keras.layers import Dense
```

Then we can use Keras with JAX.

Data

Let's generate some data. We consider

$$y_i = f(x_i) + \epsilon_i, i = 1, \dots, n$$

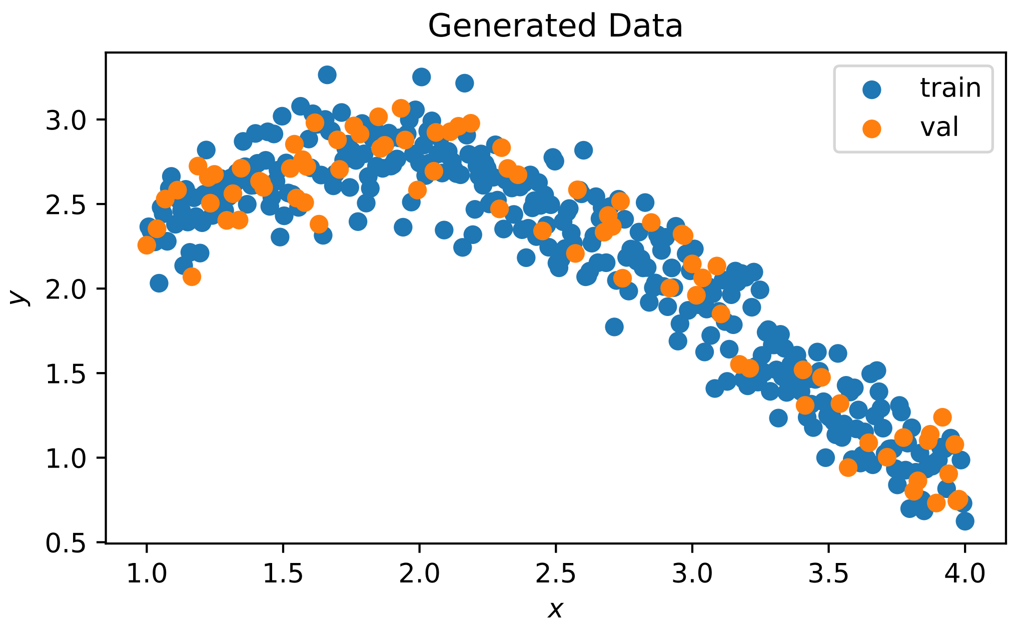
where $f(x) = \sqrt{x} + 1.5\sin(x)$.

```
def generate_data(
    key,
    x_min=1,
    x_max=4,
    data_size=400,
    seed=42,
):

    x = jnp.linspace(x_min, x_max, num=data_size)
      = 0.2 * jax.random.normal(key, shape=(data_size,))
    y = jnp.sqrt(x) + 1.5 * jnp.sin(x) +
    # Keras expects two dimensions, not flat arrays
    x = x.reshape(-1, 1)
    y = y.reshape(-1, 1)
    return x, y
```

```
key = jax.random.PRNGKey(42)
x, y = generate_data(key, data_size=400)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
plt.scatter(x_train, y_train, label="train")
plt.scatter(x_val, y_val, label="val")
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")
plt.title("Generated Data")
plt.legend()
plt.tight_layout()
plt.show()
```



Regression Model

Let's define a simple regression model.

```
def linear_model():
    model = Sequential()
    model.add(Dense(units=1))
    model.compile(optimizer=keras.optimizers.SGD(), loss="mean_squared_error")
    return model


regression_model = linear_model()
train_history = regression_model.fit(
    x_train,
    y_train,
    epochs=2000,
    batch_size=x.shape[0],
    validation_data=(x_val, y_val),
    verbose=0,
)
```

Let's plot the training and validation loss.

```python
def plot_loss_history(training_history):
    # Plot MSE of training data against epoch
    epochs = training_history.epoch
    plt.plot(
        epochs,
        jnp.log(jnp.array(training_history.history["loss"])),
        label="log training loss",
    )
    # Plot MSE of validation data against epoch
    plt.plot(
        epochs,
        jnp.log(jnp.array(training_history.history["val_loss"])),
        label="log validation loss",
    )
    # Add labels
    plt.xlabel("Epoch")
    plt.ylabel("log Loss (Mean squared error)")
    plt.legend()
    plt.tight_layout()
    plt.show()


plot_loss_history(train_history)
```

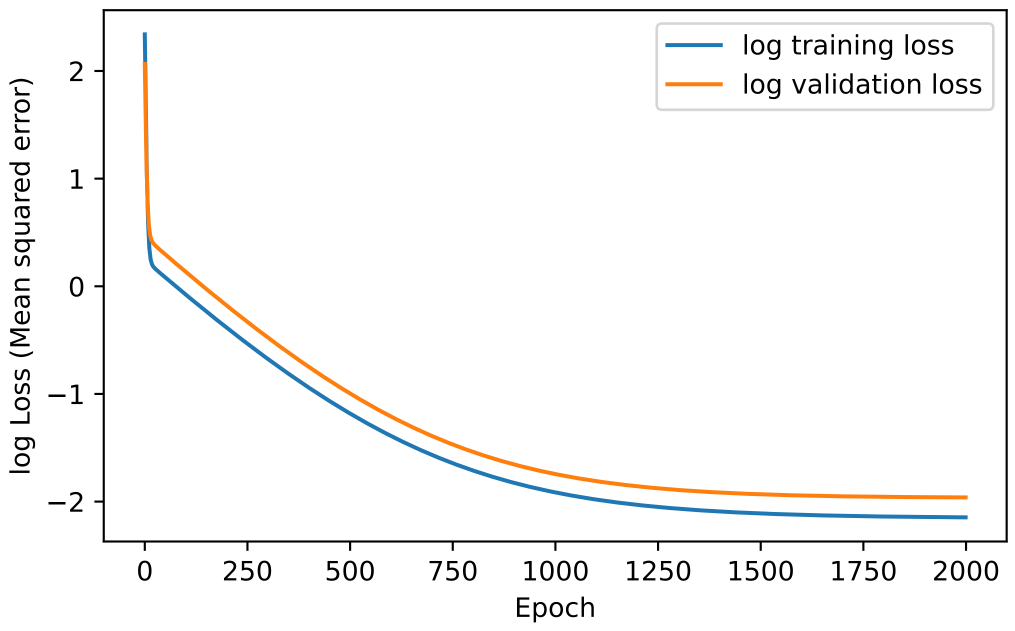


Figure 1: Keras with JAX Loss History

```
print("Testing loss on the validation set.")
regression_model.evaluate(
    x_val,
    y_val,
    verbose=2,
)
```

```
Testing loss on the validation set.
3/3 - 0s - 16ms/step - loss: 0.1407
```

```
0.14069439470767975
```

```
def plot_results(x, y, y_predict):
    plt.scatter(x, y)
    plt.plot(x, y_predict, label="fitted model", color="black")
    fx = jnp.sqrt(x) + 1.5 * jnp.sin(x)
    plt.plot(x, fx, label="true model", color="red")
    plt.xlabel(r"$x$")
    plt.ylabel(r"$y$")
    plt.legend()
    plt.tight_layout()
    plt.show()


y_pred = regression_model.predict(x, verbose=2)
plot_results(x, y, y_pred)
```

```
13/13 - 0s - 2ms/step
```

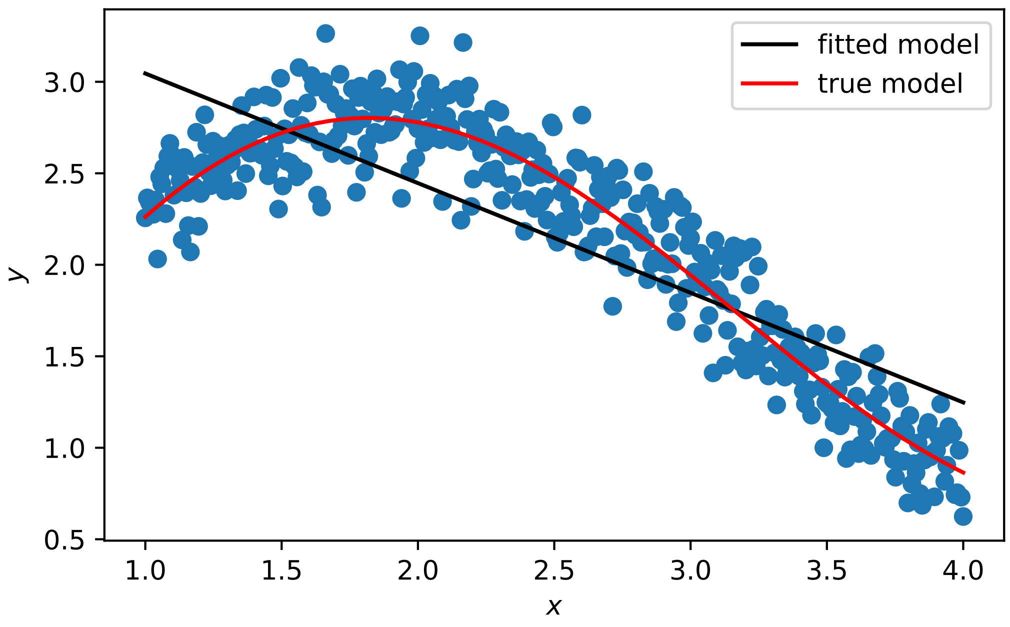


Figure 2: Keras with JAX Results

Neural Network Model

Let's define a simple neural network model.

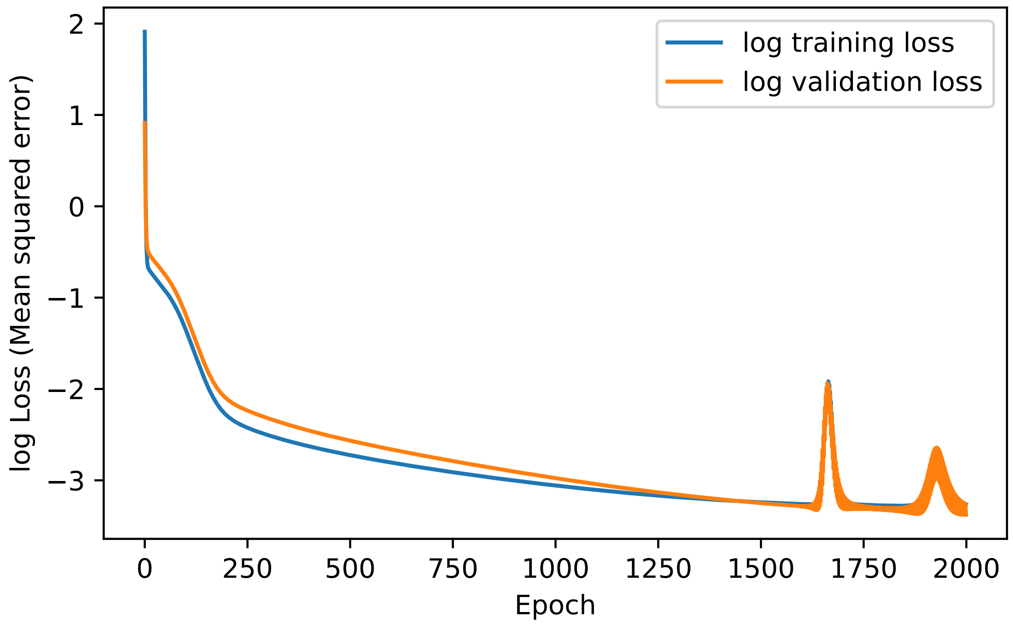
```
def nn_model(output_dim=16, num_layers=3, activation="tanh"):
    model = Sequential()
    for i in range(num_layers):
        model.add(Dense(units=output_dim, activation=activation))
    model.add(Dense(units=1))
    model.compile(
        optimizer=keras.optimizers.SGD(),
        loss="mean_squared_error",
    )
    return model
```

Let's train the model.

```
nn_model = nn_model()
train_history = nn_model.fit(
    x_train,
    y_train,
    epochs=2000,
    batch_size=x.shape[0],
    validation_data=(x_val, y_val),
    verbose=0,
)
```

Let's plot the training and validation loss.

```
plot_loss_history(train_history)
```



```
print("Testing loss on the validation set.")
nn_model.evaluate(
    x_val,
    y_val,
    verbose=2,
)
```

```
Testing loss on the validation set.
3/3 - 0s - 33ms/step - loss: 0.0342

0.03416486829519272
```

And let's plot the results.

```
y_pred = nn_model.predict(x, verbose=2)
plot_results(x, y, y_pred)
```

```
13/13 - 0s - 7ms/step
```

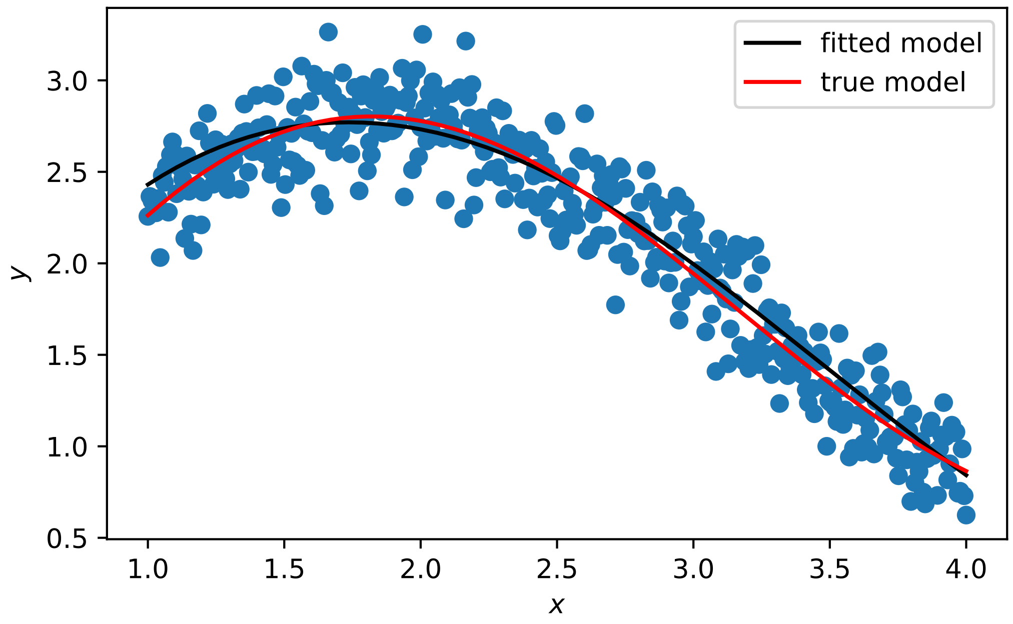


Figure 3: Keras with JAX Neural Network Results

It easy to see that Figure 3 is much better than Figure 2. And a benefit of using keras is that it's very easy to add more layers and more units to the model.