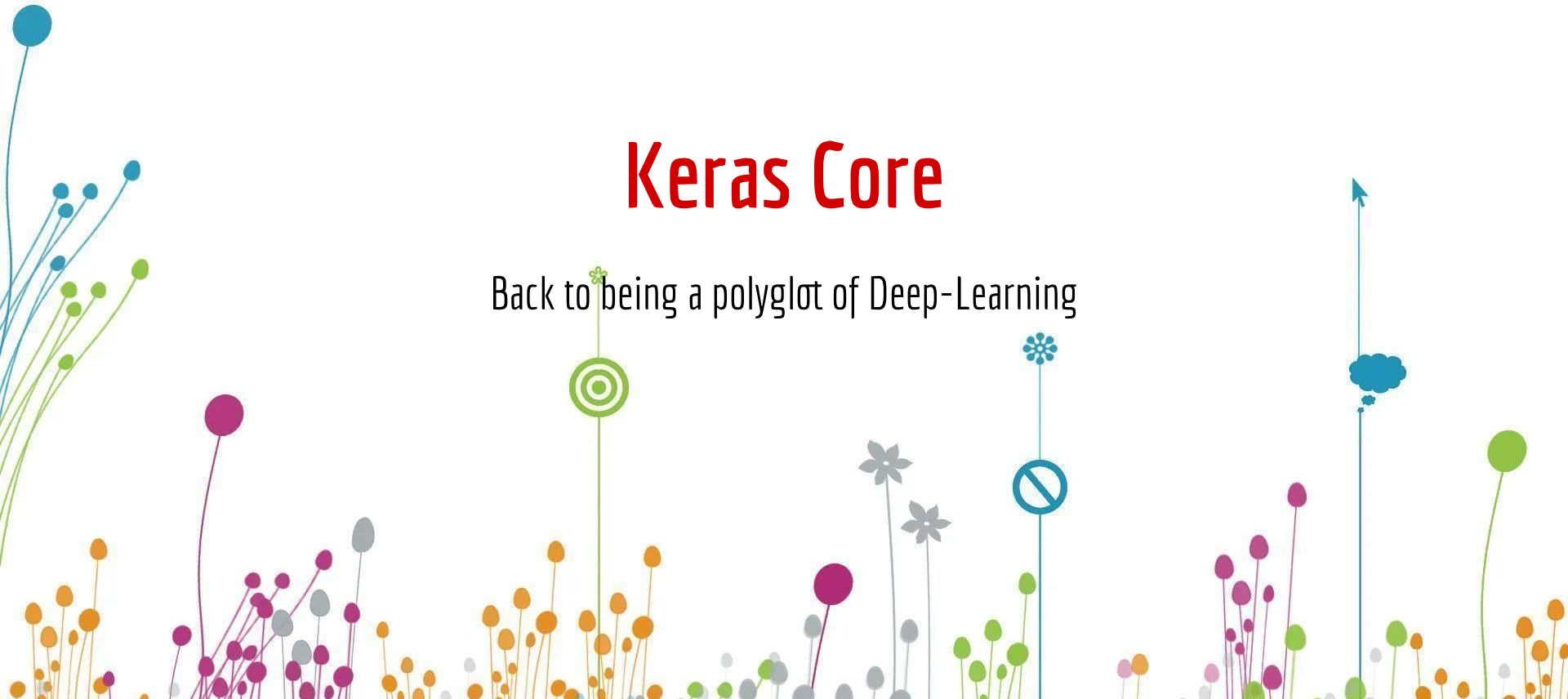
Keras Core

Back to being a polyglot of Deep-Learning



Hi!

I am Dilip Parasu.

Also known as SuperSecureHuman

 https://github.com/SuperSecureHuman

 https://linkedin.com/in/dilip-parasu

 https://supersecurehuman.github.io/

Good old Keras

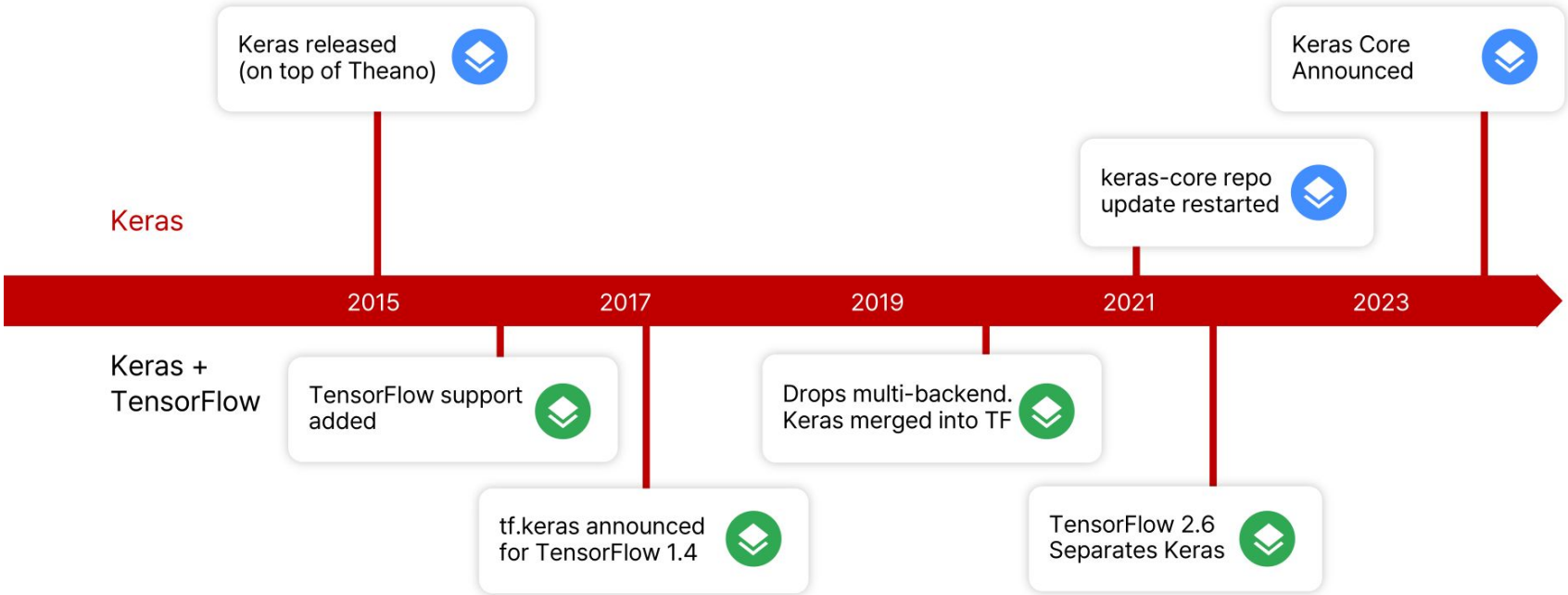
Once upon a time there existed Keras

- High performance Deep learning Library for python
- That was very easy to use
- Supported Theano, MXNET, TFv1, CNTK as backend

Then

- It became part of Tensorflow
- Dropped other backend support
- Name space was in tf.keras

From Multi to Single to Multi



Now where are we?

- Switchable backend (TF, JAX, Pytorch)
- Multi framework custom components
- Universal training loop
- Native models support
- Future proof code

```python
model = get_keras_core_model()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = torch.nn.CrossEntropyLoss()

def train_step(inputs, targets):
    # Compute loss.
    logits = model(inputs, training=True)
    loss = loss_fn(logits, targets)

    # Compute gradients.
    model.zero_grad()
    loss.backward()

    # Update weights.
    optimizer.step()
    return loss

# Iterate over epochs.
for epoch in range(num_epochs):
    # Iterate over the batches of the dataset.
    for step, (inputs, targets) in enumerate(dataset):
        loss = train_step(inputs, targets)
        print(f"Loss: {loss.detach().numpy():.4f}")
```

```python
model = get_keras_core_model()
optimizer = keras.optimizers.Adam(learning_rate=1e-3)
loss_fn = keras.losses.CategoricalCrossentropy(from_logits=True)

@tf.function(jit_compile=True)
def train_step(inputs, targets):
    # Compute loss.
    with tf.GradientTape() as tape:
        logits = model(inputs, training=True)
        loss = loss_fn(targets, logits)

    # Compute gradients.
    gradients = tape.gradient(loss, model.trainable_weights)

    # Update weights.
    optimizer.apply(gradients, model.trainable_weights)
    return loss

# Iterate over epochs.
for epoch in range(num_epochs):
    # Iterate over the batches of the dataset.
    for step, (inputs, targets) in enumerate(dataset):
        loss = train_step(inputs, targets)
        print(f"Loss: {loss.numpy():.4f}")
```

Writing a custom training loop for a Keras model: PyTorch, TensorFlow, JAX

```python
model = get_keras_core_model()
optimizer = keras.optimizers.Adam(learning_rate=1e-3)
loss_fn = keras.losses.CategoricalCrossentropy(from_logits=True)

# All variables must be built before training starts.
optimizer.build(model.trainable_variables)

def compute_loss_and_updates(trainable_vars, non_trainable_vars, data):
    # Stateless function to compute the loss and non-trainable var updates.
    x, y = data
    pred, non_trainable_vars = model.stateless_call(trainable_vars, non_trainable_vars, x)
    loss = loss_fn(y, pred)
    return loss, non_trainable_vars

# Function that returns the gradients for the trainable vars.
grad_fn = jax.value_and_grad(compute_loss_and_updates, has_aux=True)

@jax.jit
def train_step(state, data):
    # Stateless function that calls the grad_fn and computes trainable vars updates.
    trainable_vars, non_trainable_vars, optimizer_vars = state
    print(len(trainable_vars), len(non_trainable_vars), len(optimizer_vars))
    (loss, non_trainable_vars), grads = grad_fn(trainable_vars, non_trainable_vars, data)
    trainable_vars, optimizer_vars = optimizer.stateless_apply(
        optimizer_vars, grads, trainable_vars
    )
    return loss, (trainable_vars, non_trainable_vars, optimizer_vars)

# Prepare model state.
state = (model.trainable_variables, model.non_trainable_variables, optimizer.variables)

# Iterate over epochs.
for epoch in range(num_epochs):
    # Iterate over the batches of the dataset.
    for step, (inputs, targets) in enumerate(dataset):
        # Each train_step call is entirely stateless (no side effects).
        loss, state = train_step(state, data)
        print(f"Loss: {loss:.4f}")
```

Switchable backend

2 Lines, and you are good to go!

```python
import os
os.environ["KERAS_BACKEND"] = 'jax'
# Set the backend first always

import keras_core as keras
```

Why is this even a deal?

2 big players - Tensorflow and Pytorch with having almost 40 to 60% of share each.

Imagine now having just one codebase, and you have covered almost 100% of the users.

Sounds like flutter → IOS and Android, Single Codebase

Seamless integration with backends

Train Keras models with

- Low level JAX - `optax, jax.grad, jax.jit, jax.pmap ….`
- Low level tf - `tf.gradientTape….`
- Low level torch - `torch.optim, torch.nn`

You can use the keras model you made, directly in torch as a nn.module !

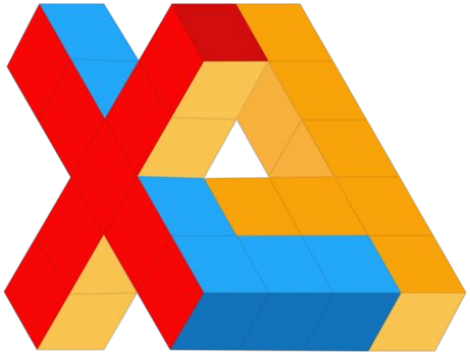
JAX enters

Blazing Speed: JAX delivers high-performance computing with GPU/TPU acceleration, XLA compilation, perfect for large-scale numerical tasks.

Effortless Gradients: JAX's automatic differentiation simplifies gradient computations.

Functional Magic: JAX promotes functional programming, enabling modular and readable code for machine learning pipelines.

Scalable Magic: JAX streamlines parallelism and distributed computing (via vmap, pmap), ideal for scaling computations across multiple devices.



What powers JAX

XLA (Accelerated Linear Algebra)

- Fast matrix operations on CPU, GPU, TPU
- JIT!

MLIR (Multi Level Intermediate Representation)

- Intermediate compilation layer for ML

OpenXLA

- Open source version of XLA and StableHLO
  - StableHLO - Operation set for high level operations HLO in ML models

What is around JAX

JAX

- The core

FLAX

- Based on Jax
- High level API for defining and Training Neural Networks

Haiku

- Another high level API

Optax

- Optimizers and Loss functions

Huggingface

- Probably the largest model source for jax now?

So what's the catch

- Super high learning curve
- Still growing, and changing docs
- Lack of 3rd party codes
- You need to care about distributed workload
- Purely Stateless

Keras helps here

- You don’t have to care about the stateless nature of jax
- All the performance optimizations taken care (up to a point)
- Using JAX while maintaining a very familiar, easy to read implementation

Introducing Keras Core:
Keras for TensorFlow, JAX, and PyTorch.

To the Notebooks


databricks
DagsHub
Nebuly AI
PyTorch
OpenAI
GitHub
ONNX
OPTUNA
scikit learn
APACHE ARROW
dask
Parquet
Streamlit
gradio
mlflow
TensorFlow
OpenXLA
Lightning AI
APACHE Spark
THANK YOU!
kaggle
RAY
DeepMind
Amazon SageMaker
Keras
OpenVINO
BENTOML
MLIR
Hugging Face
Coral
stability.ai
intel
HOROVOD
NVIDIA
GRAPHCORE
DeepSpeed
Weights & Biases
habana