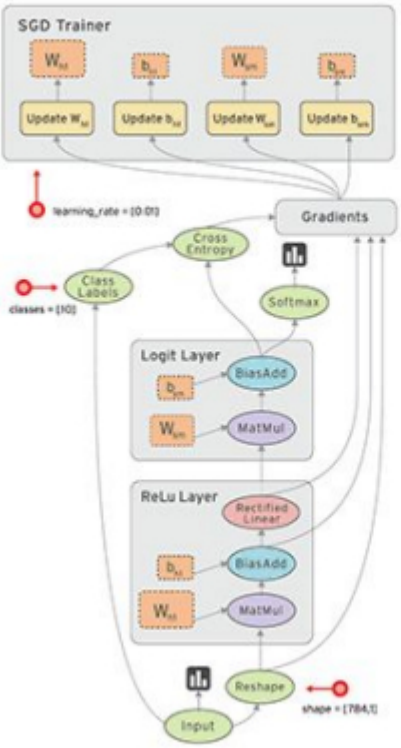
Tensorflow
Pytorch
HuggingFace

Ngoc Ba

Founder @ ProtonX
ML GDE

PROTON X

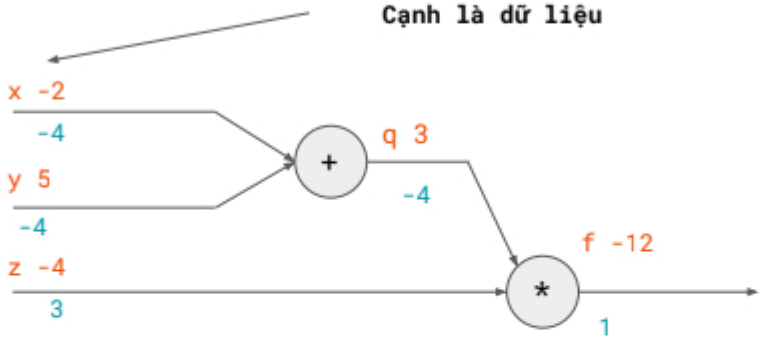


Đồ thị tính toán

Computational Graph

Đồ thị tính toán

Computational Graph



Node là phép tính

$$f(x, y, z) = (x + y)z$$

Đồ thị tính toán biểu diễn **luồng tính toán** của một phép tính bất kỳ

PROTON X

Tensorflow v1

Tensorflow v1

$$Y = W * \mathbf{x} + \mathbf{b}$$

```
in_a = tf.placeholder(dtype=tf.float32, shape=(2))
in_b = tf.placeholder(dtype=tf.float32, shape=(2))

def forward(x):
  with tf.variable_scope("matmul", reuse=tf.AUTO_REUSE):
    W = tf.get_variable("W", initializer=tf.ones(shape=(2,2)),
                        regularizer=tf.contrib.layers.l2_regularizer(0.04))
    b = tf.get_variable("b", initializer=tf.zeros(shape=(2)))
    return W * x + b

out_a = forward(in_a)
out_b = forward(in_b)

reg_loss = tf.losses.get_regularization_loss(scope="matmul")

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  outs = sess.run([out_a, out_b, reg_loss],
                  feed_dict={in_a: [1, 0], in_b: [0, 1]})
```

PROTON X

Tensorflow v2

Tensorflow v2

$$Y = W * \mathbf{x} + \mathbf{b}$$

```
W = tf.Variable(tf.ones(shape=(2,2)), name="W")
b = tf.Variable(tf.zeros(shape=(2)), name="b")

@tf.function
def forward(x):
  return W * x + b

out_a = forward([1,0])
print(out_a)
```

TF v2

PROTON X

Pytorch

Pytorch

Pytorch là một thư viện cho phép xây dựng mô hình học máy/học sâu.

Eager & Graph-based Execution

Distributed Training
Đào tạo phân tán

Simplicity over complexity

Hardware Accelerated Inference
Hỗ trợ xuất ONNX cho nhiều phần cứng

ROBUST ECOSYSTEM
Cộng đồng nghiên cứu sử dụng đông đảo

PROTON X

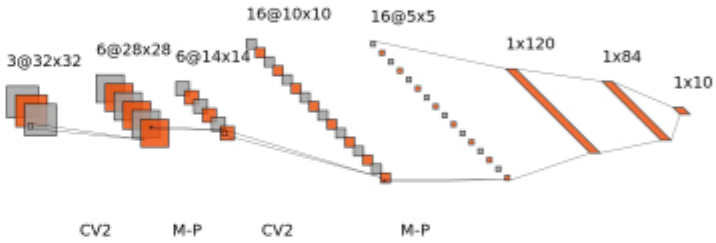
Tensorflow vs Pytorch

Tensorflow vs Pytorch

	Tensorflow	Pytorch
Eager execution vs. Static computation graph Các kiểu chạy tính toán	v1 sử dụng static computation graph. Mỗi lần tính toán cần chạy toàn bộ đồ thị. Tối ưu hơn eager execution nhưng **khó lập trình hơn.** Từ v2 đã có eager execution.	Dùng eager execution ngay từ đầu, chạy tính toán ngay lập tức
Code readability and simplicity Code đơn giản, dễ đọc	v1 lập trình đồ thị tương đối phức tạp. Từ v2 code đã thân thiện hơn.	Lập trình thân thiện, cú pháp tương tự Python.
Adoption in research vs. industry Sử dụng trong nghiên cứu và công nghiệp	Được ưu tiên hơn trong môi trường sản phẩm thực tế	Nổi tiếng và được ưu chuộng trong **cộng đồng nghiên cứu**
Popularity of libraries and ecosystems Các thư viện và hệ sinh thái	Hệ sinh thái xoay quanh xây dựng các ứng dụng - TensorFlow Hub: Lưu trữ mô hình - TensorFlow Lite: Triển khai mô hình trên các loại phần cứng khác nhau	Hệ sinh thái xoay quanh xây dựng các loại mô hình khác nhau từ thị giác máy tính, ngôn ngữ tự nhiên cho đến âm thanh. Các thư viện: torchvision và torchaudio.
GPU support and performance Hỗ trợ GPU và hiệu năng	Đều hỗ trợ Có đào tạo phân tán Có hỗ trợ các kỹ thuật tối ưu mô hình	Đều hỗ trợ Có đào tạo phân tán Có hỗ trợ các kỹ thuật tối ưu mô hình
Visualization and debugging Hiển thị và kiểm tra lỗi	TensorBoard	Các thư viện hiển thị như Matplotlib
Documentation Tài liệu	Hơi phức tạp, có nhiều API trùng nhau	Dễ đọc

LeNet

LeNet



https://youtu.be/FwFduRA_L6Q

Convolutional Neural Networks (LeCun, 1989) là một loại Neural Network đặc biệt để xử lý dữ liệu dạng lưới (grid-like topology), ví dụ là ảnh. Ảnh thường có thể coi là lưới 2D các pixels

Yann LeCun

Turing Award

PROTON X

LeNet - Tensorflow

LeNet - Tensorflow

Tensorflow

```python
import tensorflow as tf
from tensorflow.keras import layers, models

def build_lenet(input_shape, num_classes):
    model = models.Sequential()

    # Layer 1: Convolutional layer with 6 filters, 5x5 kernel, and ReLU activation
    model.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape))

    # Layer 2: Average Pooling layer with 2x2 pool size and 2x2 strides
    model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2)))

    # Layer 3: Convolutional layer with 16 filters, 5x5 kernel, and ReLU activation
    model.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='relu'))

    # Layer 4: Average Pooling layer with 2x2 pool size and 2x2 strides
    model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2)))

    # Flatten the output for the fully connected layers
    model.add(layers.Flatten())

    # Layer 5: Fully Connected layer with 120 units and ReLU activation
    model.add(layers.Dense(120, activation='relu'))

    # Layer 6: Fully Connected layer with 84 units and ReLU activation
    model.add(layers.Dense(84, activation='relu'))

    # Layer 7: Fully Connected layer with num_classes units for classification
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Define input shape and number of classes
input_shape = (32, 32, 1)  # LeNet expects input images of size 32x32 with 1 channel (grayscale)
num_classes = 10           # For example, 10 classes for digit recognition (0-9)

# Build the LeNet model
model = build_lenet(input_shape, num_classes)
```

PROTON X

LeNet - Tensorflow

LeNet - Tensorflow

Tensorflow

```
# Load and preprocess the data
(train_images, train_labels), (test_images, test_labels) = load_and_preprocess_mnist()

# Define input shape and number of classes
input_shape = (28, 28, 1) # LeNet expects input images of size 32x32 with 1 channel (grayscale)
num_classes = 10 # For example, 10 classes for digit recognition (0-9)

# Build the LeNet model
model = build_lenet(input_shape, num_classes)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
batch_size = 128
epochs = 10
model.fit(train_images, train_labels, batch_size=batch_size, epochs=epochs, validation_split=0.1)

# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(test_images, test_labels, verbose=0)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

PROTON X

Tùy biến đào tạo

Custom Training

Sử dụng **GradientTape** để tính đạo hàm

```python
for epoch in range(epochs):
    epoch_loss_avg = tf.keras.metrics.Mean()
    epoch_accuracy = tf.keras.metrics.CategoricalAccuracy()

    # Training loop
    for i in range(0, len(train_images), batch_size):
        batch_images = train_images[i:i+batch_size]
        batch_labels = train_labels[i:i+batch_size]

        with tf.GradientTape() as tape:
            # Forward pass
            logits = model(batch_images, training=True)
            loss_value = tf.keras.losses.categorical_crossentropy(batch_labels, logits)

        # Compute gradients
        gradients = tape.gradient(loss_value, model.trainable_variables)

        # Update weights
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

        # Update metrics
        epoch_loss_avg.update_state(loss_value)
        epoch_accuracy.update_state(batch_labels, logits)

    # Print progress for each epoch
    print(f"Epoch {epoch + 1}/{epochs}, Loss: {epoch_loss_avg.result():.4f}, Accuracy: {epoch_accuracy.result():.4f}")
```

PROTON X

LeNet - Pytorch

LeNet - Pytorch

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        # 1 input image channel (black & white), 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

Pytorch

PROTON X

LeNet - Pytorch

LeNet - Pytorch

Pytorch

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

Tính đạo hàm
Và cập nhật

https://pytorch.org/tutorials/beginner/introyt/introyt1_tutorial.html

Hugging Face

Thư viện tập trung vào các công nghệ Xử lý Ngôn ngữ Tự nhiên (NLP) và các mô hình dựa trên Transformer.

Thư Viện Transformers
Cho phép xây dựng nhanh các mô hình dùng kiến trúc Transformer

Model Hub
BERT, GPT, RoBERTa

Tokenizers
Đa dạng các bộ tách từ

https://huggingface.co/docs/transformers/index

HuggingFace

HuggingFace

Xây dựng quy trình phân loại văn bản

```
from transformers import AutoTokenizer, AutoModelForMaskedLM
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModelForMaskedLM.from_pretrained("bert-base-uncased")
```

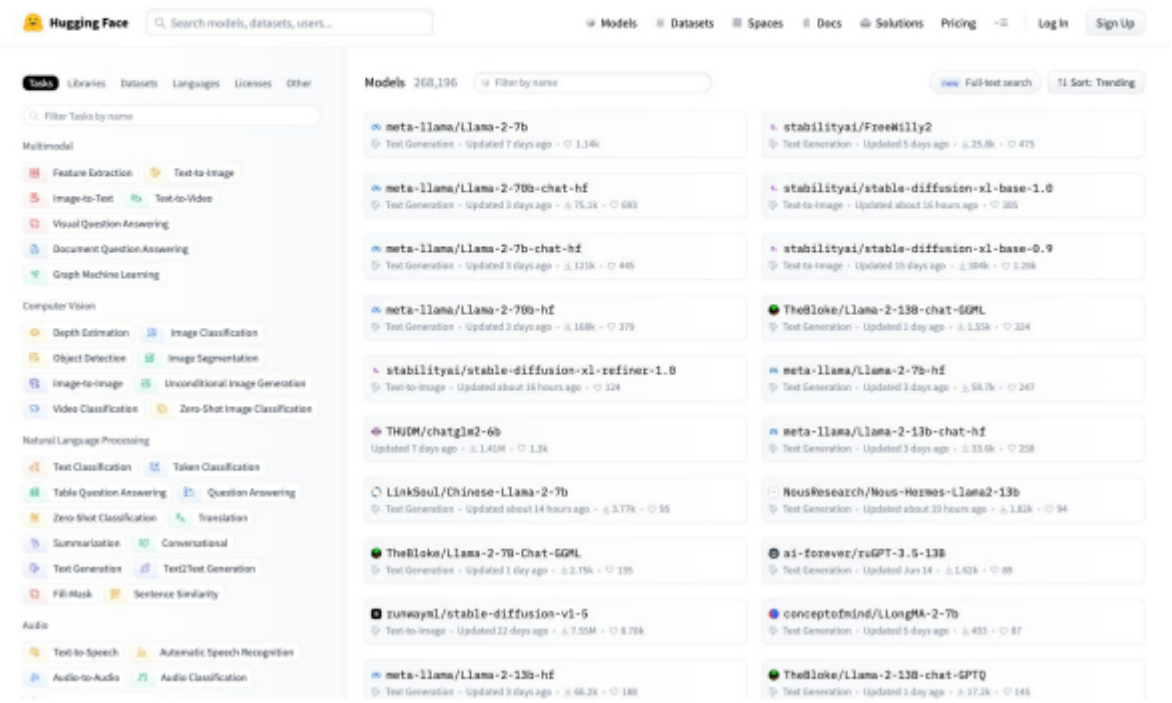
```
classifier = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)
classifier("Nous sommes très heureux de vous présenter la bibliothèque 🤗 Transformers.")

[{'label': '5 stars', 'score': 0.7273}]
```

PROTON X

HuggingFace

HuggingFace

Hub với số lượng mô hình đa dạng



PROTON X

Tổng kết

Summary

Đồ thị tính toán

Pytorch vs Tensorflow

HuggingFace