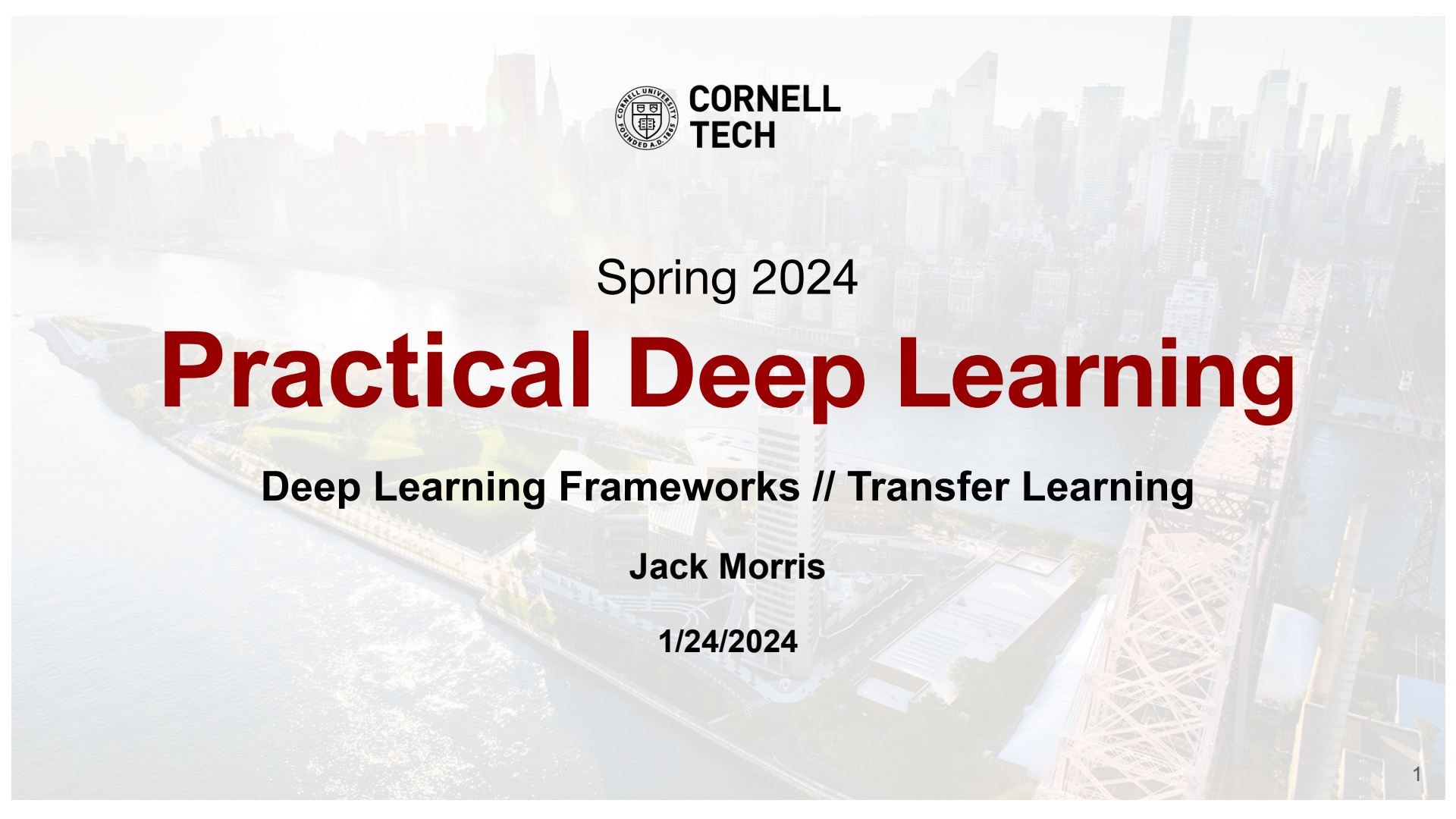


CORNELL TECH

Spring 2024

Practical Deep Learning

Deep Learning Frameworks // Transfer Learning

Jack Morris

1/24/2024

About this class

- We'll meet every Monday for 8 weeks
- No assignments, just a project at the end
- No Zoom option – please pay attention :)
- Course website https://jxmo.io/deep-learning-workshop/
- Also please ask questions on Canvas

Notes

- Anonymous feedback link: bit.ly/pdl24feedback
- Laptops are allowed (but please be respectful!)
- Will put my slides on the course website
  - https://jxmo.io/deep-learning-workshop/

Deep Learning Frameworks

Exercise

Why can’t we just use plain numpy to implement deep learning models?

Exercise

Why can’t we just use plain numpy to implement deep learning models?

(*Turn to your neighbor and discuss!*)

Deep learning frameworks … *in numpy*

```
class Network(object):

    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]
```

```
    def feedforward(self, a):
        """Return the output of the network if "a" is input."""
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a
```

Deep learning frameworks … *in PyTorch*

```
import torch

class Network(torch.nn.Module):
    def __init__(self, sizes):
        self.layers = [
            torch.nn.Linear(sizes[i], sizes[i+1]) for i in range(len(sizes))
        ]

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x
```

Deep learning frameworks … *in numpy*

```
def backprop(self, x, y):
    """Return a tuple ``(nabla_b, nabla_w)`` representing the
    gradient for the cost function C_x.  ``nabla_b`` and
    ``nabla_w`` are layer-by-layer lists of numpy arrays, similar
    to ``self.biases`` and ``self.weights``."""
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    # feedforward
    activation = x
    activations = [x] # list to store all the activations, layer by layer
    zs = [] # list to store all the z vectors, layer by layer
    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b
        zs.append(z)
        activation = sigmoid(z)
        activations.append(activation)
    # backward pass
    delta = self.cost_derivative(activations[-1], y) * \
        sigmoid_prime(zs[-1])
    nabla_b[-1] = delta
    nabla_w[-1] = np.dot(delta, activations[-2].transpose())
    # Note that the variable l in the loop below is used a little
    # differently to the notation in Chapter 2 of the book.  Here,
    # l = 1 means the last layer of neurons, l = 2 is the
    # second-last layer, and so on.  It's a renumbering of the
    # scheme in the book, used here to take advantage of the fact
    # that Python can use negative indices in lists.
    for l in xrange(2, self.num_layers):
        z = zs[-l]
        sp = sigmoid_prime(z)
        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
        nabla_b[-l] = delta
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
    return (nabla_b, nabla_w)
```

```
def update_mini_batch(self, mini_batch, eta):
    """Update the network's weights and biases by applying
    gradient descent using backpropagation to a single mini batch.
    The "mini_batch" is a list of tuples "(x, y)", and "eta"
    is the learning rate."""
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.weights = [w-(eta/len(mini_batch))*nw
                    for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb
                   for b, nb in zip(self.biases, nabla_b)]
```

Deep learning frameworks … *in PyTorch*

```
model = Network()
optimizer = torch.optim.SGD(model.parameters())

...

(y - model(x)).backward()
optimizer.step()
```

Deep learning frameworks … *numpy vs PyTorch*

```
def backprop(self, x, y):
    """Return a tuple ``(nabla_b, nabla_w)`` representing the
    gradient for the cost function C_x.  ``nabla_b`` and
    ``nabla_w`` are layer-by-layer lists of numpy arrays, similar
    to ``self.biases`` and ``self.weights``."""
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    # feedforward
    activation = x
    activations = [x] # list to store all the activations, layer by layer
    zs = [] # list to store all the z vectors, layer by layer
    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b
        zs.append(z)
        activation = sigmoid(z)
        activations.append(activation)
    # backward pa
    delta = self.
        sigmoid_p
    nabla_b[-1] =
    nabla_w[-1] =
    # Note that t
    # differently
    # l = 1 means
    # second-last
    # scheme in t
    # that Python
    for l in xran
        z = zs[-l
        sp = sigm
        delta = n
        nabla_b[-l] = delta
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
    return (nabla_b, nabla_w)
```

```
    def update_mini_batch(self, mini_batch, eta):
        """Update the network's weights and biases by applying
        gradient descent using backpropagation to a single mini batch.
        The "mini_batch" is a list of tuples "(x, y)", and "eta"
        is the learning rate."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x, y)
            nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
            nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
        self.weights = [w-(eta/len(mini_batch))*nw
                        for w, nw in zip(self.weights, nabla_w)]
        self.biases = [b-(eta/len(mini_batch))*nb
                       for b, nb in zip(self.biases, nabla_b)]
```

vs.

```
import torch

class Network(torch.nn.Module):
    def __init__(self, sizes):
        self.layers = [
            torch.nn.Linear(sizes[i], sizes[i+1]) for i in range(len(sizes))
        ]

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x

model = Network()
optimizer = torch.optim.SGD(model.parameters())

...

(y - model(x)).backward()
optimizer.step()
```

Deep learning frameworks - two reasons

- (1) **Autograd**
  - To train machine learning models, we need to compute gradients
  - We can do this by hand in numpy, but it's really hard and annoying

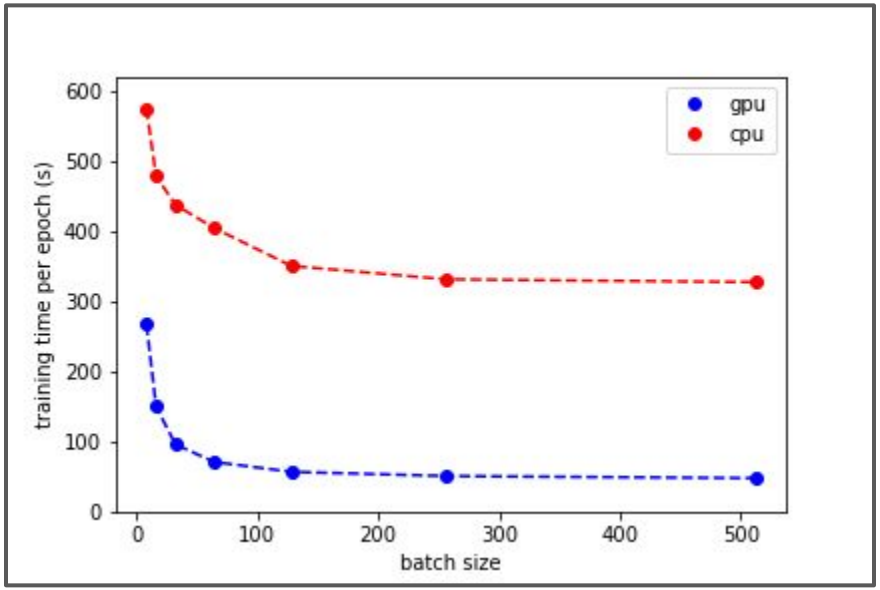
- (2) GPUs
  - Deep learning models require lots of math (mostly matrix multiplications)
  - Matrix multiplication runs much faster on GPU

Deep learning frameworks - two reasons

- (1) Autograd
  - To train machine learning models, we need to compute gradients
  - We can do this by hand in numpy, but it's really hard and annoying

- (2) **GPUs**
  - Deep learning models require lots of math (mostly matrix multiplications)
  - Matrix multiplication runs much faster on GPU

Why do we need GPUs?



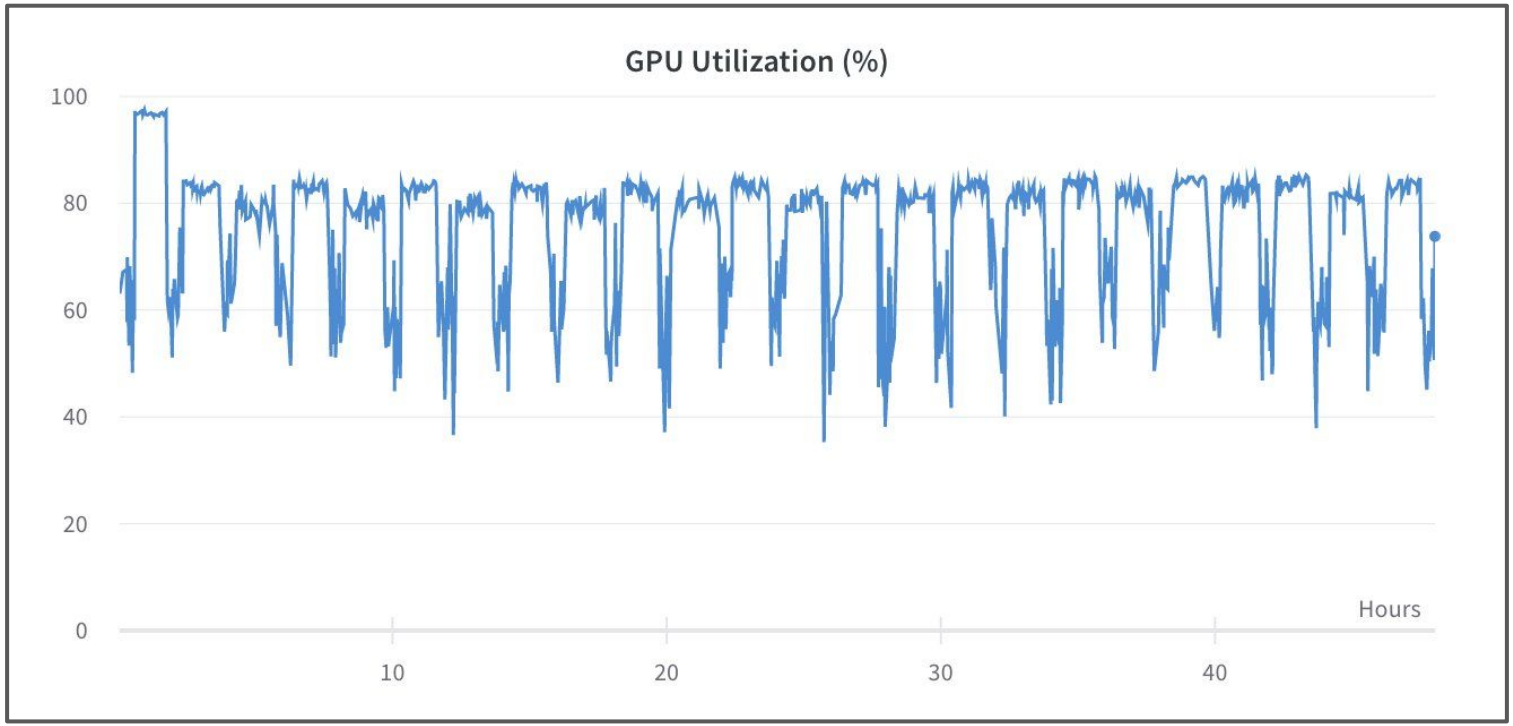
https://github.com/moritzhambach/CPU-vs-GPU-benchmark-on-MNIST

Note on GPUs - The Hardware Lottery

"**Hardware lottery:** when a research idea wins because it is suited to the available software and hardware and not because the idea is superior to alternative research directions."

–Sara Hooker, The Hardware Lottery

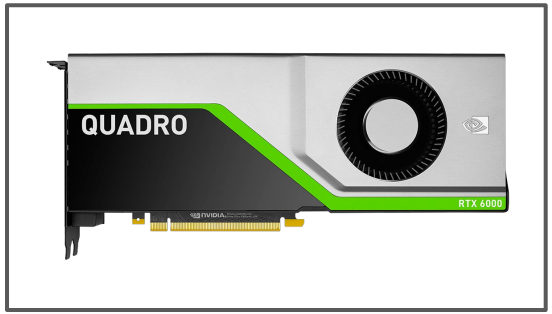
GPU Utilization



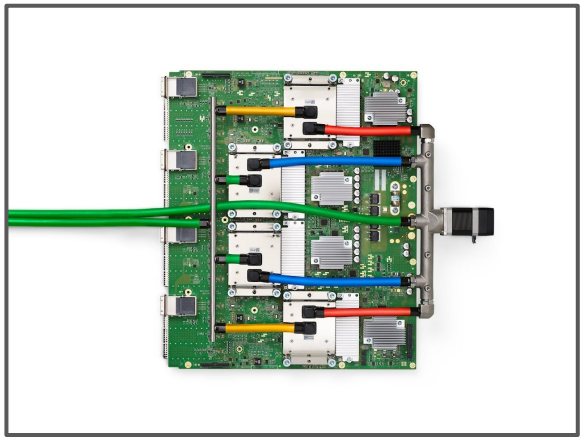
https://twitter.com/jxmnop/status/1528889386498424832/photo/1

GPUs vs TPUs

CPU



GPU

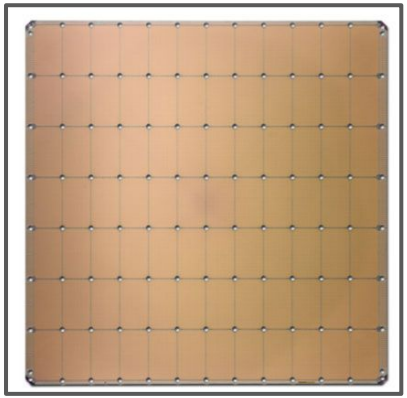


TPU

Alternatives to GPUs and TPUs

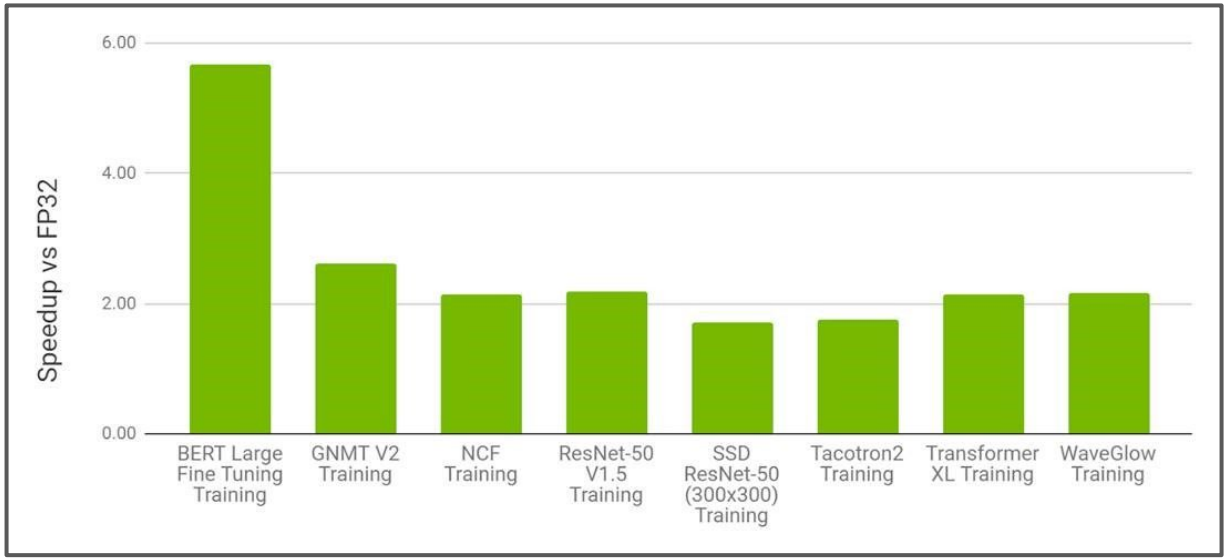
IPU
(Graphcore)

groq



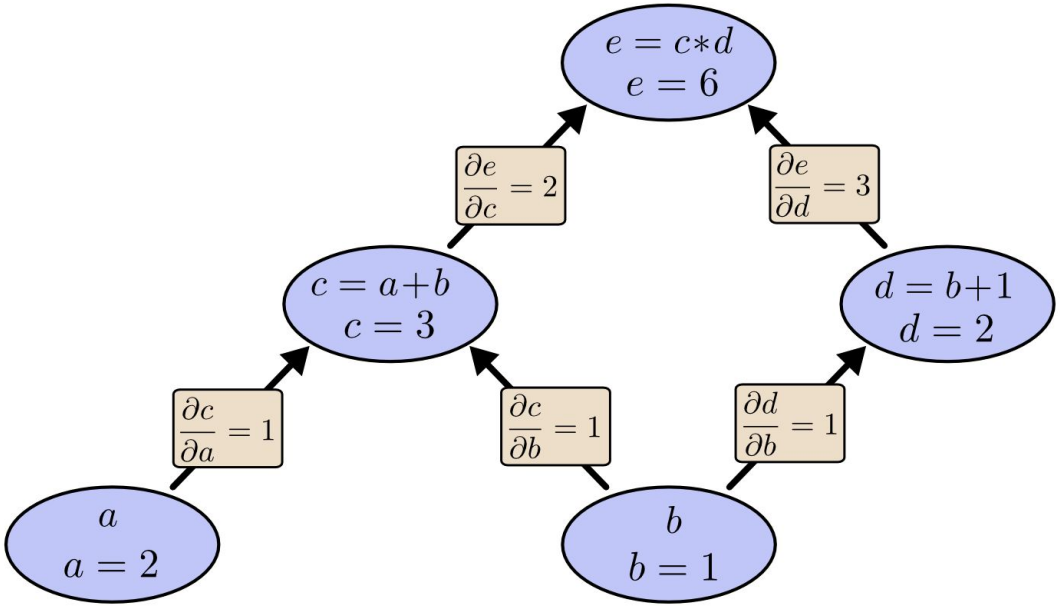
Cerebras WSE-2

Half precision (fp16 and bf16)



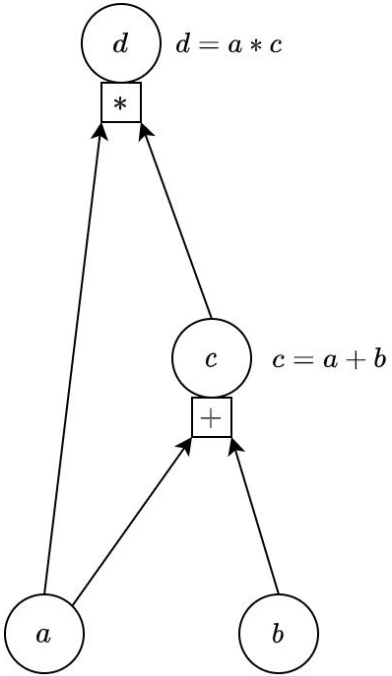

```
from torch.cuda.amp import autocast
with autocast(dtype=torch.bfloat16):
    loss, outputs = ...
```

What is autodiff? Intro to computational graph

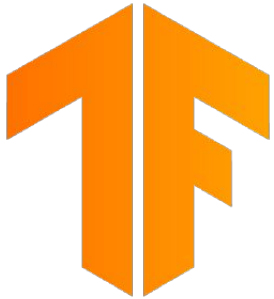


Source: [colah.github.io]

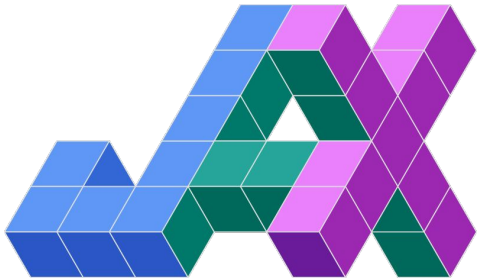
What is autodiff?



PyTorch
(Meta)



TensorFlow
(Google)



Jax
(Google)

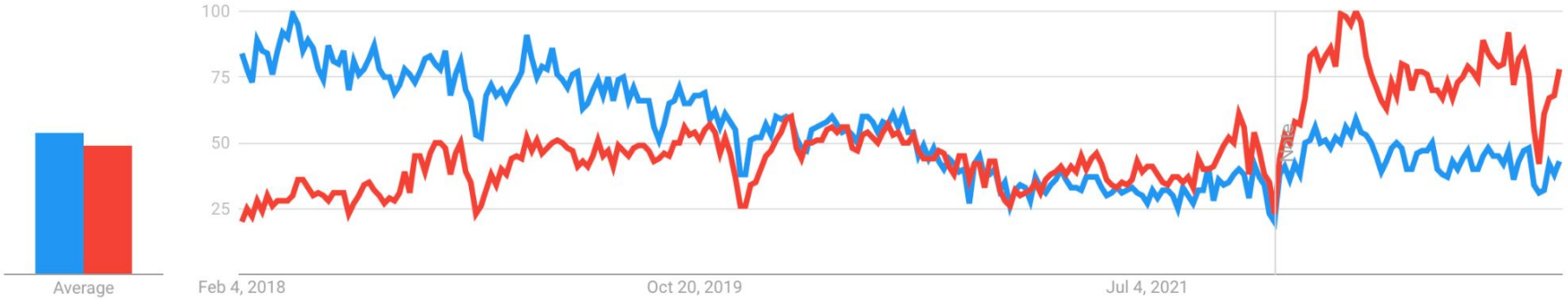
TensorFlow
Search term

pytorch
Search term

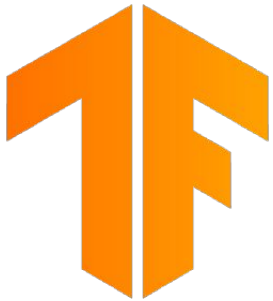
+ Add comparison

United States | Past 5 years | All categories | Web Search

Interest over time

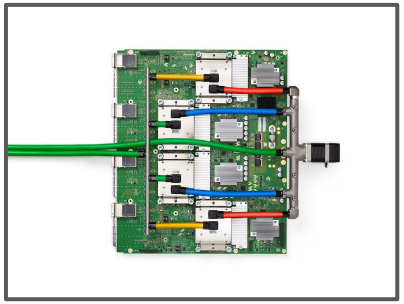


TensorFlow

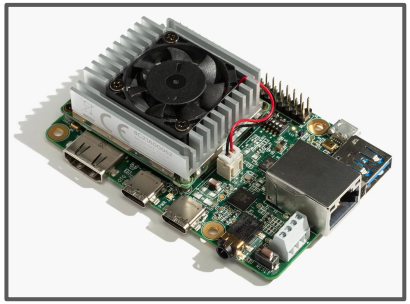


```python
# Create TF Model.
class NeuralNet(Model):
    # Set layers.
    def __init__(self):
        super(NeuralNet, self).__init__()
        # First fully-connected hidden layer.
        self.fc1 = layers.Dense(n_hidden_1, activation=tf.nn.relu)
        # First fully-connected hidden layer.
        self.fc2 = layers.Dense(n_hidden_2, activation=tf.nn.relu)
        # Second fully-connecter hidden layer.
        self.out = layers.Dense(num_classes)

    # Set forward pass.
    def call(self, x, is_training=False):
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.out(x)
        if not is_training:
            # tf cross entropy expect logits without softmax, so only
            # apply softmax when not training.
            x = tf.nn.softmax(x)
        return x

# Build neural network model.
neural_net = NeuralNet()
```

TensorFlow (Hardware)



TPU



coral.ai

TensorFlow Lite

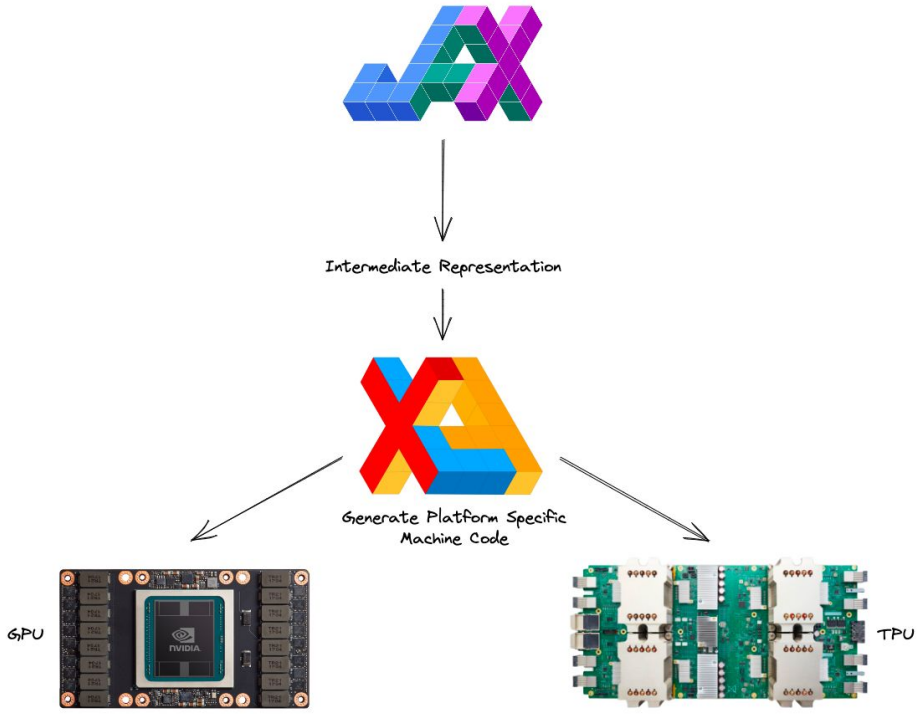
PyTorch

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

Jax

```python
from flax import linen as nn  # Linen API

class CNN(nn.Module):
  """A simple CNN model."""

  @nn.compact
  def __call__(self, x):
    x = nn.Conv(features=32, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.avg_pool(x, window_shape=(2, 2), strides=(2, 2))
    x = nn.Conv(features=64, kernel_size=(3, 3))(x)
    x = nn.relu(x)
    x = nn.avg_pool(x, window_shape=(2, 2), strides=(2, 2))
    x = x.reshape((x.shape[0], -1))  # flatten
    x = nn.Dense(features=256)(x)
    x = nn.relu(x)
    x = nn.Dense(features=10)(x)
    return x
```

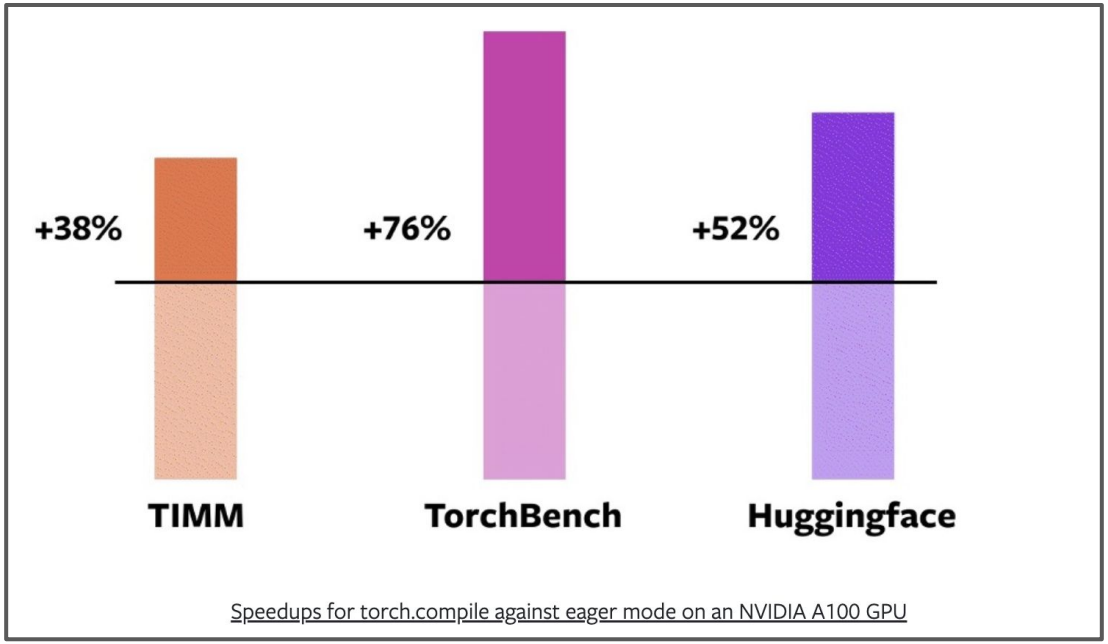
Jax


Intermediate Representation
Generate Platform Specific
Machine Code
GPU
TPU

💭 Why do we use PyTorch?

- Simple API
- Great libraries
- Need to read/use other people’s code
- Less buggy than TensorFlow
- You’ll probably use it at your job

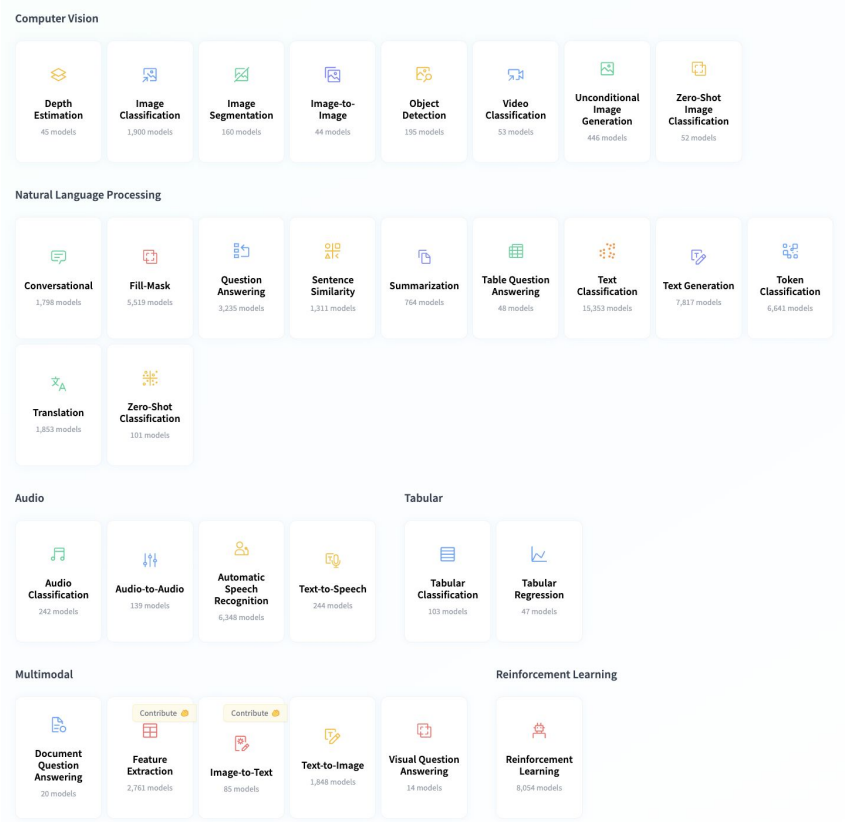
PyTorch 2.0



PyTorch 2.0 Preview (Dec. 6th, 2022)

Deep Learning Tasks

The universe of deep learning research



huggingface.co/tasks

Image Classification

Image classification is the task of assigning a label or class to an entire image. Images are expected to have only one class for each image. Image classification models take an image as input and return a prediction about which class the image belongs to.

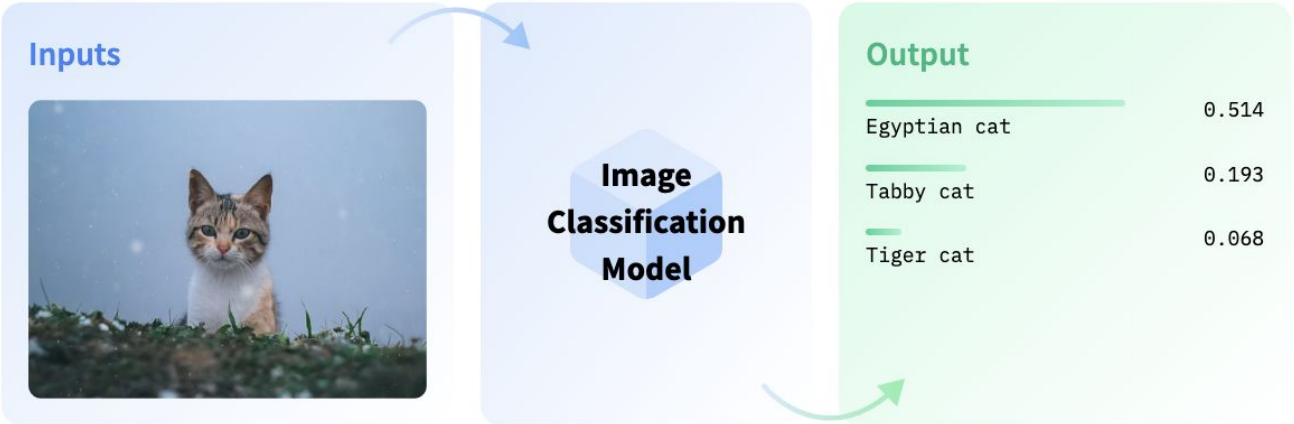
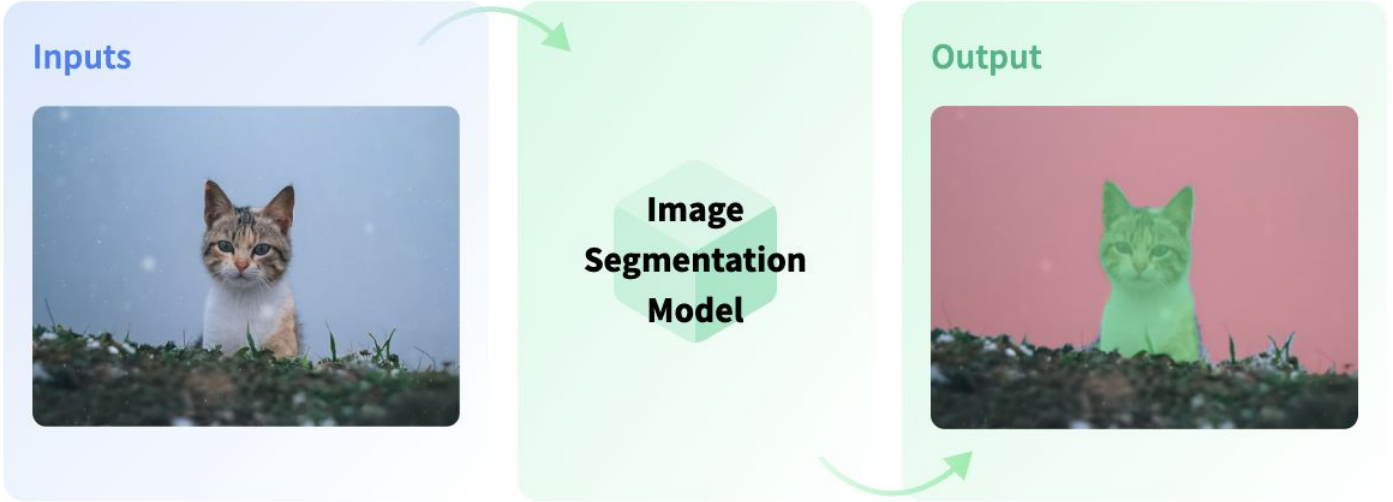


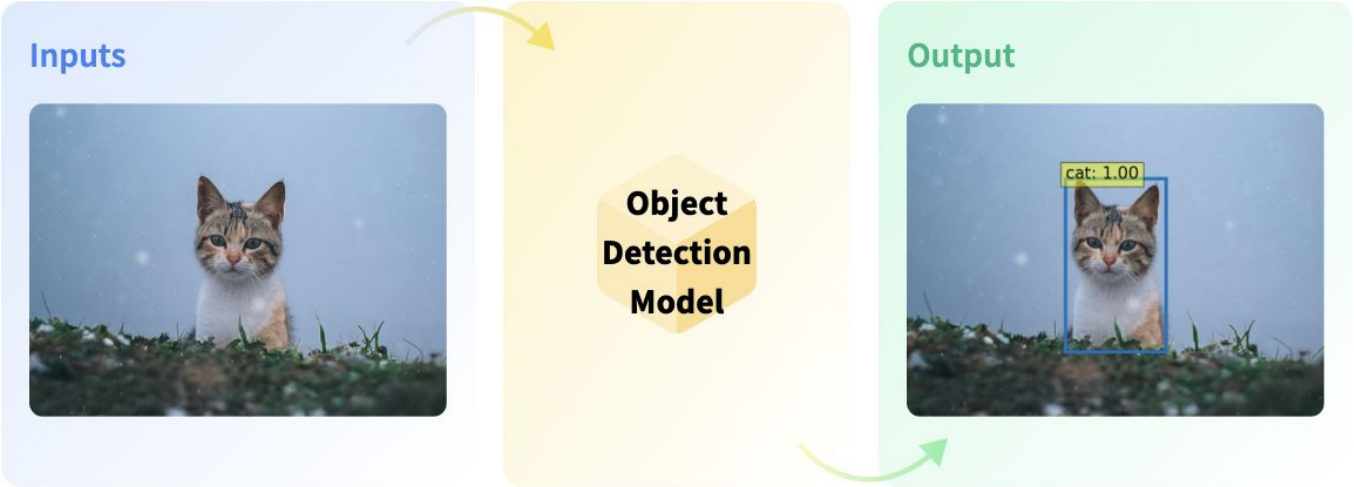
Image Segmentation

Image Segmentation divides an image into segments where each pixel in the image is mapped to an object. This task has multiple variants such as instance segmentation, panoptic segmentation and semantic segmentation.

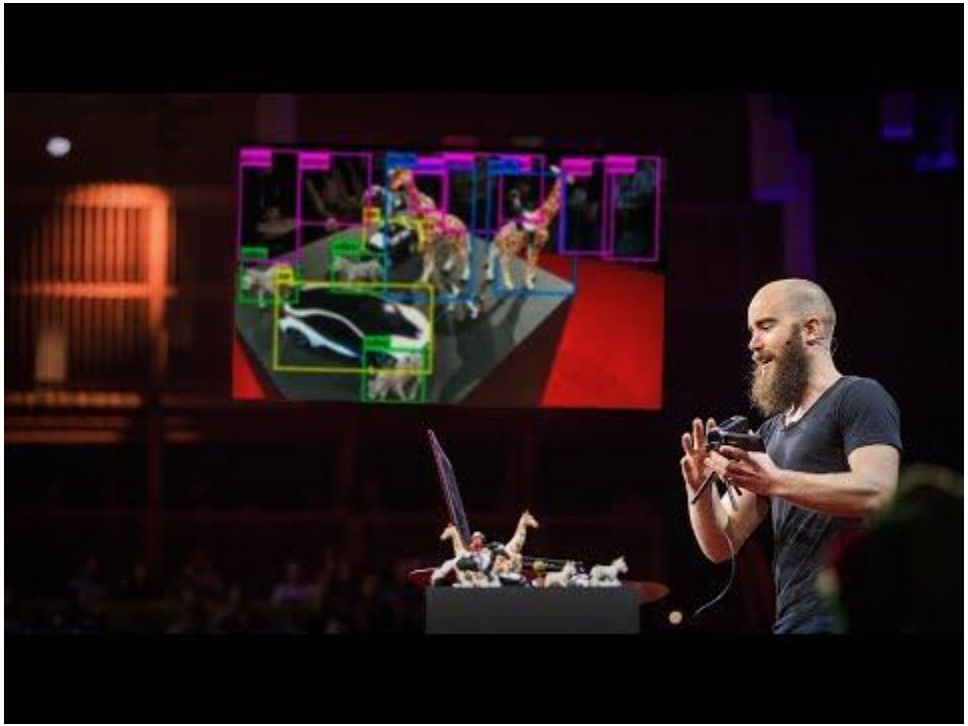


Object Detection

Object Detection models allow users to identify objects of certain defined classes. Object detection models receive an image as input and output the images with bounding boxes and labels on detected objects.

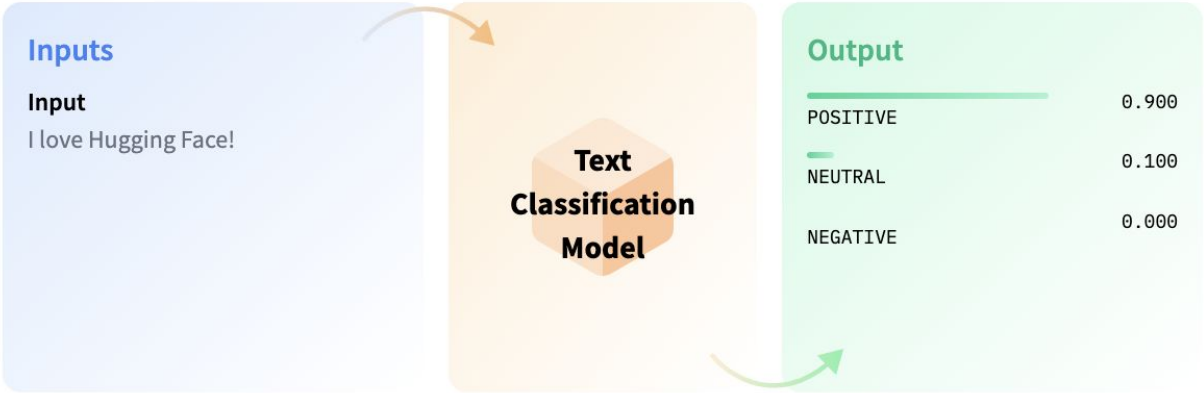


Object Detection



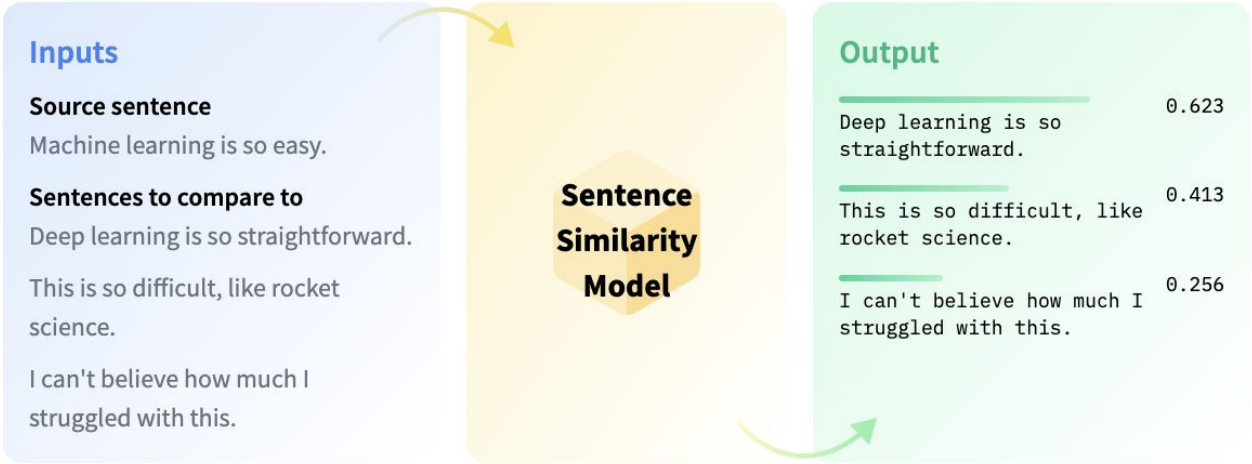
Text Classification

Text Classification is the task of assigning a label or class to a given text. Some use cases are sentiment analysis, natural language inference, and assessing grammatical correctness.



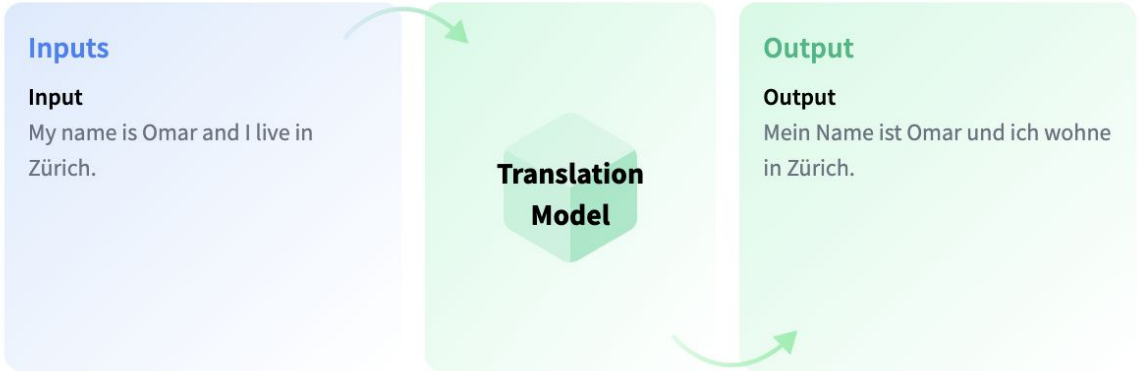
Sentence Similarity

Sentence Similarity is the task of determining how similar two texts are. Sentence similarity models convert input texts into vectors (embeddings) that capture semantic information and calculate how close (similar) they are between them. This task is particularly useful for information retrieval and clustering/grouping.

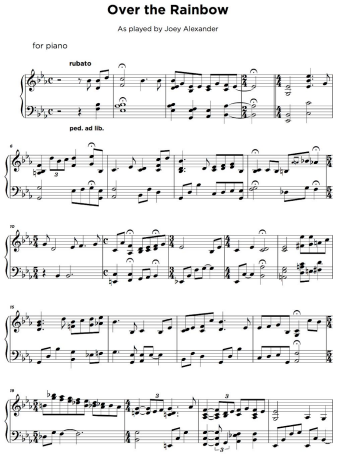


Translation

Translation is the task of converting text from one language to another.

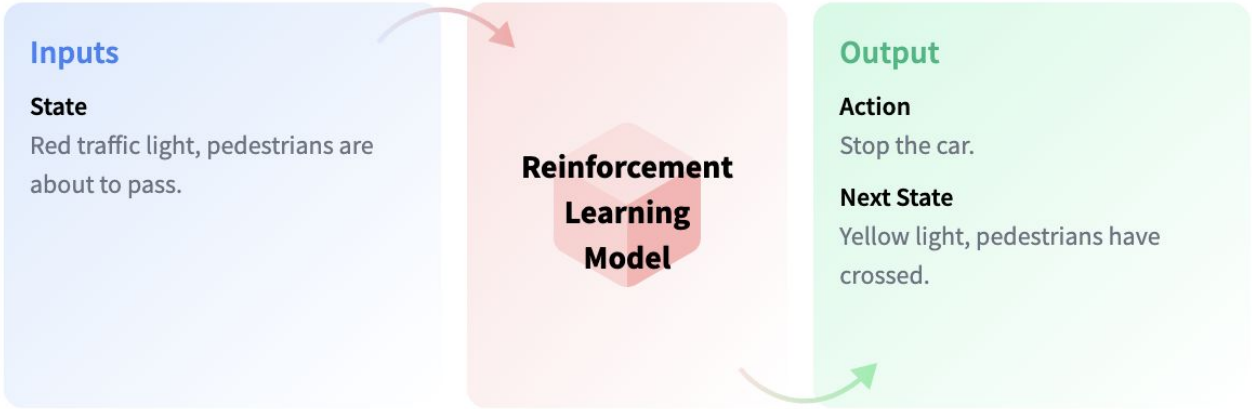


🎼 Music transcription



Reinforcement Learning

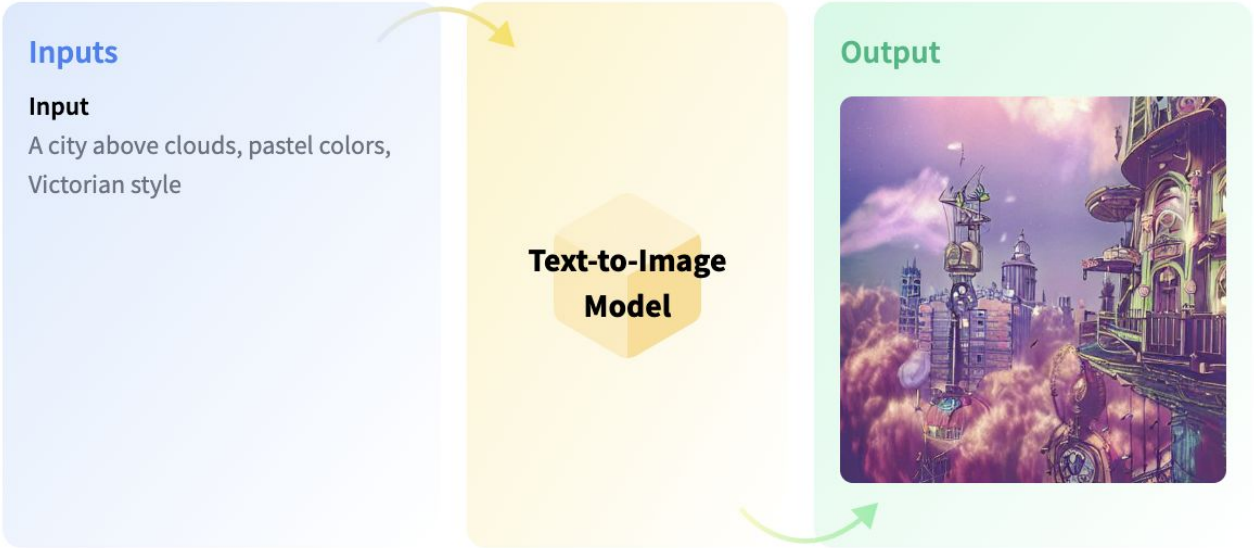
Reinforcement learning is the computational approach of learning from action by interacting with an environment through trial and error and receiving rewards (negative or positive) as feedback



Deep learning tasks: 🤖 **Reinforcement learning**

Text-to-Image

Generates images from input text. These models can be used to generate and modify images based on text prompts.

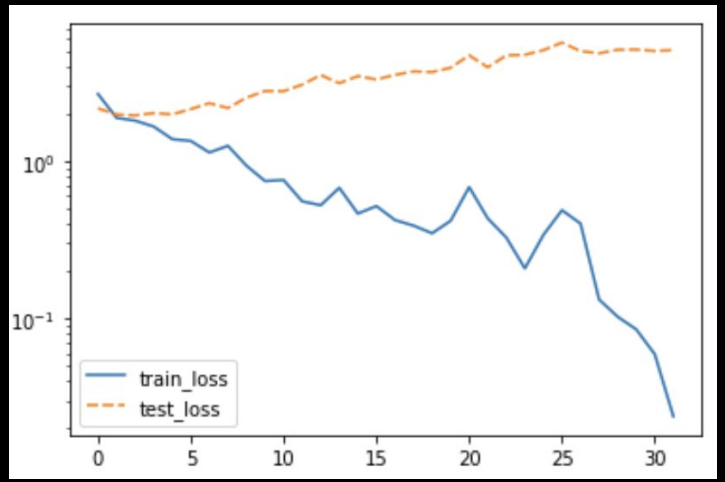


Transfer Learning

Puzzle (15 mins) → Transfer Learning Tutorial

bit.ly/pdl24puzzle2

Bad graph (current)



Good graph