**Pytorch and build NN models**

**521H0489 – Hồ Hữu An**

**1. Key features when compared to tensorflow**

| **Features** | **PyTorch** | **TensorFlow** |
| --- | --- | --- |
| Language | Python | Python, C++, JavaScript, C#, Ruby, Swift |
| Usability | More Python-friendly, easier to learn and debug | More mature, strong visualization capabilities |
| Computational Graph | Dynamic computational graphs, allows runtime changes | Static computational graphs, treats networks as static objects |
| Data Parallelism | Supports data parallelism, distributes work among cores | Supports data parallelism, distributed training |
| Model Serving | Lacks model serving capabilities in production | Offers TensorFlow Serving for deploying models in production |
| Visualization | Limited monitoring and visualization interfaces | TensorBoard for model graph visualization |
| Community | Active community and forums, helpful documentation | Widely used, larger developer community |
| Deployment | Experimental support for mobile platforms (iOS, Android) | Production-ready deployment options, mobile support |
| Companies | Tesla Autopilot, Uber's Pyro | Google, Uber, Microsoft, various universities |

**2. Main features of programming with pytorch**

PyTorch is an open-source machine learning framework based on the Python programming language. It offers several key features that make it a popular choice for deep learning research and development.

|  |  |
| --- | --- |
| Tensor Computation | PyTorch provides tensor computation capabilities similar to NumPy arrays. Tensors are n-dimensional arrays used for numeric computation and can be accelerated by GPUs |
| Dynamic Graph Computation | PyTorch uses dynamic computation graphs, allowing users to change network behavior on the fly. This feature enables real-time testing and debugging of code, making it easier to develop and experiment with neural network models |
| Automatic Differentiation | PyTorch supports automatic differentiation, which is essential for creating and training neural networks. It allows the framework to numerically compute the derivative of a function by making backward passes in neural networks |
| Python Support | PyTorch is based on Python, making it compatible with popular libraries and packages such as NumPy, SciPy, Numba, and Cython. This compatibility allows for seamless integration with existing Python workflows |
| Cloud Support | PyTorch is well-supported on major cloud platforms, providing easy scaling, large-scale training on GPUs, and the ability to run models in a production-scale environment |

**3. Important classes in building Neural Network models**

When building neural network models with PyTorch, there are several important classes that I will frequently use. These classes provide the necessary functionality for defining and training neural networks

|  |  |
| --- | --- |
| torch.nn.Module | This is the base class for all neural network modules in PyTorch. It provides methods for defining the structure of the network and performing forward propagation. I can create my own custom neural network models by subclassing torch.nn.Module and implementing the forward method |
| torch.nn.Linear | This class represents a linear transformation, commonly known as a fully connected layer or dense layer, in a neural network. It applies a linear transformation to the input data by multiplying it with a weight matrix and adding a bias term. The torch.nn.Linear class takes the input and output dimensions as arguments |
| torch.nn.ReLU | The ReLU (Rectified Linear Unit) activation function is widely used in neural networks to introduce non-linearity. The torch.nn.ReLU class applies the ReLU function element-wise to the input tensor |
| torch.nn.Softmax | The softmax function is commonly used in the output layer of a neural network for multi-class classification problems. It converts the output scores into probabilities that sum up to 1. The torch.nn.Softmax class applies the softmax function to the input tensor |
| torch.nn.ConvTranspose2d | This class represents a 2D transposed convolutional layer, also known as a deconvolutional layer. It performs the opposite operation of a convolution by upsampling the input. |
| torch.nn.Sigmoid | This class represents the sigmoid activation function. It applies a sigmoid function element-wise to the input. |
| torch.nn.Tanh | This class represents the hyperbolic tangent activation function. It applies a hyperbolic tangent function element-wise to the input. |
| torch.nn.Dropout | This class represents the dropout regularization technique. It randomly zeroes some of the input elements with a specified probability during training. |
| torch.nn.BatchNorm2d | This class represents the batch normalization technique applied to 2D convolutional layers. It normalizes the input along the batch dimension. |
| torch.nn.LSTM | This class represents a long short-term memory (LSTM) layer. It is commonly used for sequence modeling tasks. |
| torch.nn.GRU | This class represents a gated recurrent unit (GRU) layer. It is another type of recurrent layer commonly used for sequence modeling. |
| torch.nn.MaxPool2d | This class represents a 2D max pooling layer. It downsamples the input by taking the maximum value within a specified window. |
| torch.nn.AdaptiveAvgPool2d | This class represents an adaptive average pooling layer. It performs spatial average pooling to adaptively resize the input to a fixed size. |
| torch.nn.Embedding | This class represents an embedding layer. It maps discrete input indices to dense vectors. |
| torch.nn.CrossEntropyLoss | This class represents the cross-entropy loss function. It combines the softmax activation and the negative log-likelihood loss, commonly used for classification tasks. |
| torch.nn.MSELoss | This class represents the mean squared error loss function. It computes the mean squared difference between the input and target. |
| torch.nn.Loss | This class is the base class for all loss functions in PyTorch. It provides common functionality for computing losses. |
| torch.nn.Sequential | This class allows you to sequentially stack multiple layers together to form a neural network model. |
| torchvision.models | his module provides pre-defined models for computer vision tasks. It includes popular architectures such as ResNet, VGG, and AlexNet. |
| torch.optim | This module provides classes for various optimization algorithms. Some commonly used classes include torch.optim.SGD, torch.optim.Adam, and torch.optim.RMSprop. |