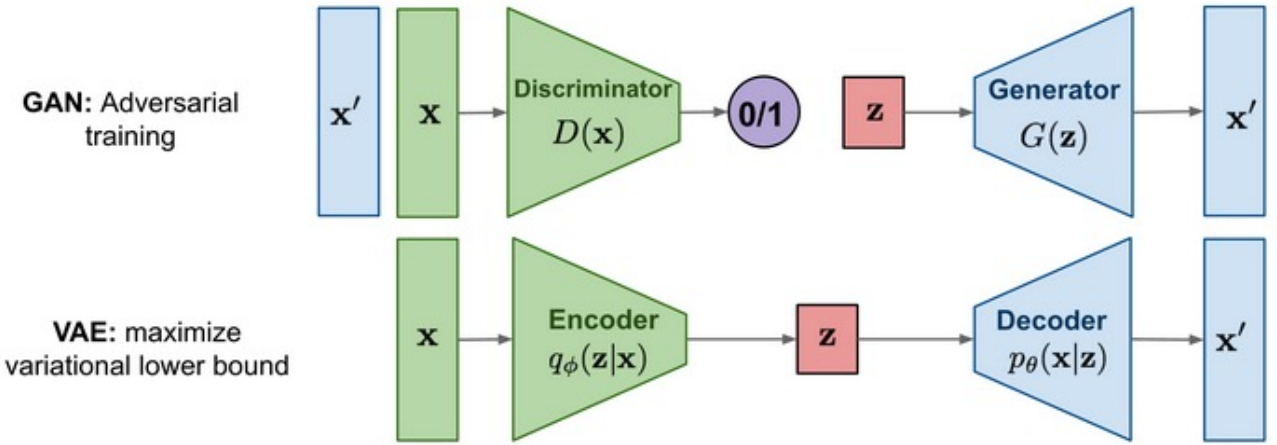
# 第二次大作业 – 图像生成

媒体计算　2023

# 图像生成的目标

- 使用神经网络模型完成图像生成
  - GAN：生成对抗网络
  - VAE：变分自编码器



# 作业要求

- 框架：Jittor
  - 基于动态编译、使用元算子和统一计算图的深度学习框架
- 网络
  - GAN：实现原版GAN或者WGAN
    - Wasserstein GAN
  - VAE：实现原版VAE或者$\beta$-VAE
    - beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework
- 数据集（任选其一）
  - MNIST
  - CIFAR-10（加分项，+1分）
- 提交截止时间：2024年1月10日

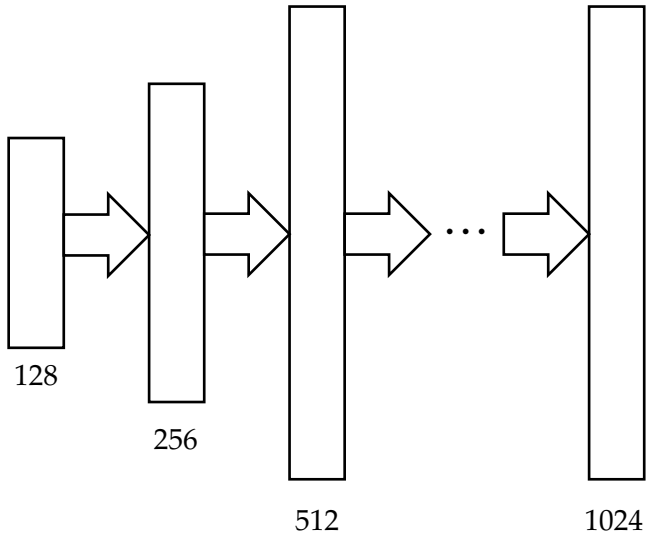
Jittor 计图

# 作业要求

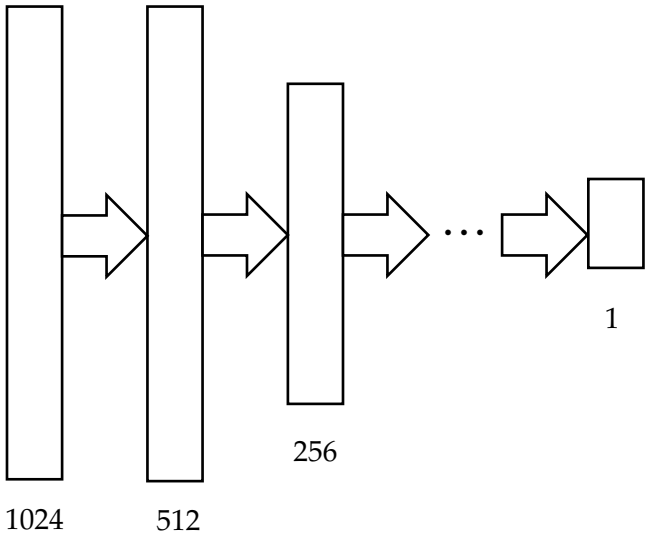
1. 在Jittor框架下实现给定的网络结构
2. 在给定的数据集上训练（任选其一）
3. 输出训练结果
4. 分析生成效果好坏及原因

# 例子

- GAN
  - 生成器：MLP
  - 辨别器：MLP








# 例子

- GAN
  - 生成器Loss
    - 生成图片经过辨别器辨别后的结果与1之间的交叉熵
    - gen_loss = binary_cross_entropy_loss(D(G(z)), ones_vector)
  - 辨别器Loss
    - 生成图片经过辨别器辨别后的结果与0之间的交叉熵
    - 真实图片经过辨别器辨别后的结果与1之间的交叉熵
    - real_loss = binary_cross_entropy_loss(D(real), ones_vector)
    - fake_loss = binary_cross_entropy_loss(D(G(z)), zeros_vector)
    - dis_loss = real_loss + fake_loss

# 例子

- GAN
  - 训练流程：每个iteration做如下事情
    1. 训练生成器（此时固定辨别器）
       - 随机生成一个z，生成器输入z生成图片x
       - 计算生成器loss
       - 反向传播梯度
    2. 训练辨别器（此时固定生成器）
       - 辨别器输入x输出辨别结果，计算fake_loss
       - 辨别器输入真实图片输出辨别结果，计算real_loss
       - 计算辨别器loss
       - 反向传播梯度
  - 如此循环数个epoch

# 例子

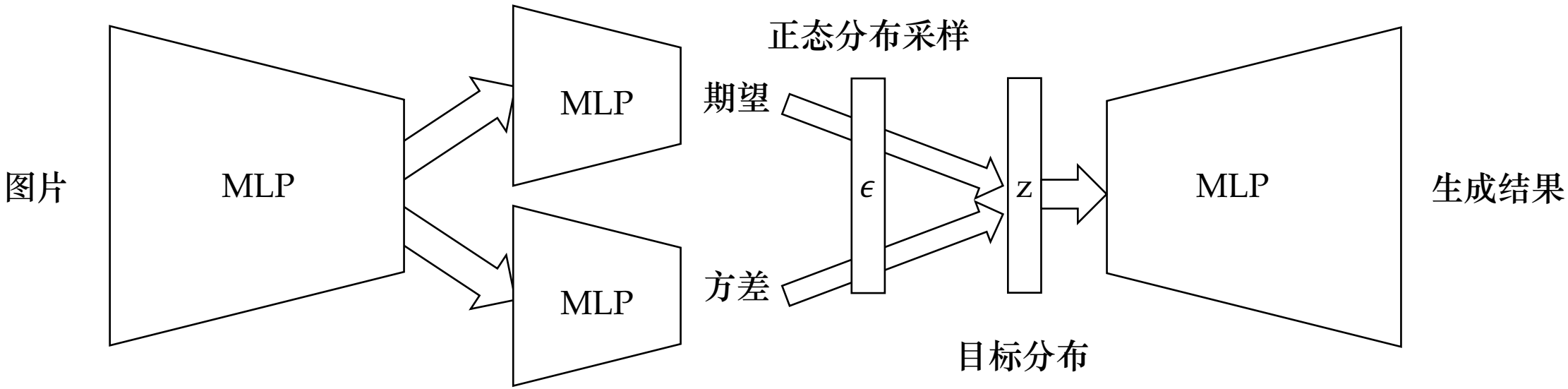
- WGAN
  - 网络结构可以与GAN类似
  - 生成器Loss
    - 具体推导可以参看原论文
    - gen_loss = - mean(D(G(z)))
  - 辨别器Loss
    - 具体推导可以参看原论文
    - dis_loss = mean(D(G(z))) - mean(D(real))

# 例子

- WGAN
  - 训练流程
    - 大体与GAN相同
    - Optimizer推荐选用RMSProp
    - 每个iteration都训练辨别器，但是每隔几个iteration才训练一次生成器
    - 每次训练辨别器之后对辨别器参数做clamp（限制在例如[-0.01,0.01]之间）

# 例子

- VAE
  - 编码器：MLP
  - 解码器：MLP



# 例子

- VAE
  - MSE Loss
    - 生成图片和输入图片的MSE Loss
  - KL散度

$$-\frac{1}{2}\sum_{i}^{n}\left(1+\log\sigma_i^2-\mu_i^2-\sigma_i^2\right)$$

  - 总Loss
    - Loss = MSE Loss + KL Loss

# 例子

- VAE
  - 训练流程
    1. 选取输入图片
    2. 计算生成图片
    3. 计算VAE Loss
    4. 反向传播
  - 生成流程
    1. 生成正态分布
    2. 输入解码器
    3. 生成图片

# 例子

- $\beta$-VAE
  - 修改Loss中KL散度项的系数即可

- 以上只是简单的例子，欢迎使用更好的网络结构、训练方法等获得更好的结果

# 参考资料

- GAN
  - https://zhuanlan.zhihu.com/p/497361373
  - https://lilianweng.github.io/posts/2017-08-20-gan/
  - https://spaces.ac.cn/archives/4439

- VAE
  - https://zhuanlan.zhihu.com/p/497876029
  - https://lilianweng.github.io/posts/2018-08-12-vae/
  - https://spaces.ac.cn/archives/5253

# 评分标准

- 满分30分
- GAN（12分）
  - 实现原版GAN并在数据集上测试（9分）
  - 实现Wasserstein GAN并在数据集上测试（12分）
- VAE（12分）
  - 实现原版VAE并在数据集上测试（9分）
  - 实现$\beta$-VAE并在数据集上测试（12分）
  - 探究不同的$\beta$对训练结果有什么影响（加分项，+3分）
- 代码与报告（6分）
  - 清晰有注释的代码
  - 报告中完整报告训练过程和实验过程
    - 以动图或者视频的形式展示训练过程中测试结果的变化
  - 展示实验结果，评价并分析（实验结果适当合理即可，没有硬性质量要求）

# 作业要求

- 参考的实现需要在实验报告中写明
- 请勿提交其他课程的作业
- 严禁抄袭、复制他人代码，一旦发现后果自负

- 提交截止时间：2024年1月10日

# Jittor 计图 框架简介

Jittor 计图

# 简介

- Jittor：基于即时编译和元算子的高性能深度学习框架
  - 前端语言为Python，采用模块化的设计，类似于PyTorch，Keras
  - 后端使用高性能语言编写，如CUDA，C++

# 安装

- Jittor目前支持Linux、MacOS、Windows操作系统，并且在Linux和Windows上支持GPU计算
- 在三个平台上都支持直接使用pip安装，以Ubuntu为例：

```
sudo apt install python3.7-dev libomp-dev
python3.7 -m pip install jittor
python3.7 -m jittor.test.test_example
```

- 如果你有NVIDIA显卡，希望能够使用GPU计算，还需要额外安装CUDnn加速相关的依赖
- 安装Jittor可能需要一些额外的依赖项，具体内容可以参考：
  - https://cg.cs.tsinghua.edu.cn/jittor/download/

# 安装

- Jittor还支持通过Docker的方式进行安装，例如：

```
# CPU only(Linux)
docker run -it --network host jittor/jittor
# CPU and CUDA(Linux)
docker run -it --network host --gpus all jittor/jittor-cuda
# CPU only(Mac and Windows)
docker run -it -p 8888:8888 jittor/jittor
```

- 关于Docker安装的详细教程，可以参考：
  - https://cg.cs.tsinghua.edu.cn/jittor/tutorial/2020-5-15-00-00-docker/

# 类型与算子

- Jittor的基本数据类型称为Var，为了高效采用异步运算的方式进行。访问数据可以通过Var.data进行。一个例子如下：

```
import jittor as jt
a = jt.float32([1,2,3])
print (a)
print (a.data)
# Output: float32[3,]
# Output: [ 1. 2. 3.]
```

- Jittor的算子和numpy类似，支持四则运算重载之外也支持大量函数
  - 所有算子jittor.xxx(Var, ...)都具有别名Var.xxx(...)，例如：

```
c.max() # alias of jt.max(a)
c.add(a) # alias of jt.add(c, a)
c.min(keepdims=True) # alias of jt.min(c, keepdims=True)
```

- 关于Jittor支持的全部操作，可以运行help(jittor.ops)

# 常用网络接口

- Jittor的各种网络接口在jittor.nn模块下，下面列出一些常用的网络结构
  - 全连接层：

```
class jittor.nn.Linear(in_features, out_features, bias=True)
```

  - 卷积层：

```
class jittor.nn.Conv(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1,
bias=True)  [源代码]
```

  - 池化层：

```
class jittor.nn.Pool(kernel_size, stride=None, padding=0, dilation=None, return_indices=None, ceil_mode=False,
count_include_pad=True, op='maximum')  [源代码]
```

  - Dropout层：

```
class jittor.nn.Dropout(p=0.5, is_train=False)  [源代码]
```

# 常用网络接口

- jittor.nn模块下也有一些实用的方法函数，类似pytorch的functional模块
  - 一般来说网络层类的首字母改成小写就有对应的方法，例如：
  - 卷积：

```
jittor.nn.conv2d(x, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)  [源代码]
```

  - ReLU激活函数：

```
jittor.nn.relu(x)  [源代码]
```

  - SoftMax：

```
jittor.nn.softmax(x, dim=None)  [源代码]
```

# 常用网络接口

- 各种Loss也在jittor.nn模块下，例如：
  - L1 Loss：

```
jittor.nn.l1_loss(output, target)  [源代码]
```

  - 交叉熵：

```
jittor.nn.cross_entropy_loss(output, target, weight=None, ignore_index=None, reduction='sum')  [源代码]
```

  - MSE Loss：

```
jittor.nn.mse_loss(output, target)  [源代码]
```

- 更多信息可以参考Jittor的官方文档：
  - https://cg.cs.tsinghua.edu.cn/jittor/assets/docs/index.html

# 创建自己的模型

- 将前述的各种层结构组合起来可以创建自己的模型，这一过程类似PyTorch
- 模型类需要继承jittor.nn.Module
- 与PyTorch不同的是，前向计算的函数名为execute
- 一个简单的例子如下：

```
class Model(Module):
    def __init__(self):
        self.layer1 = nn.Linear(1, 10)
        self.relu = nn.ReLU()
        self.layer2 = nn.Linear(10, 1)
    def execute (self,x) :
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        return x
```

# 创建自己的模型

- 可以通过简单的函数存取模型的参数

```
class Net(nn.Module):
...
net = Net()
net.save('net.pkl')
net.load('net.pkl')
```

- 同时load函数也支持加载pytorch产生的pth文件
- 在jittor.models模块下也有许多经常被使用的网络结构
  - 例如AlexNet，ResNet等

# 数据集

- 数据集相关定义在jittor.dataset模块下
- 对于自定义数据集类，需要继承jittor.dataset.Dataset类，并覆写__getitem__(self, index)函数
- 通过enumerate的方式来获取数据集中的数据
- 具体实现的时候需要注意是否shuffle、batch大小等内容，一个具体的例子可以参照：
  - https://cg.cs.tsinghua.edu.cn/jittor/tutorial/2020-3-17-09-53-mnistclassification/
- jittor.dataset模块下还有一些常用的数据集
  - 例如：CIFAR10、MNIST

# 训练过程

- 定义自己的模型，设计自己的数据集
- 对每个batch使用loss函数获取loss
- 使用optimizer进行梯度下降
  - 优化器在jittor.optim模块下，支持Adam、SGD等优化器
  - 对loss进行梯度下降的操作：optimizer.step(loss)

# 训练过程

- 一个简单的训练例子：
  - https://cg.cs.tsinghua.edu.cn/jittor/tutorial/2020-3-17-09-52-example/

```
model = Model()
learning_rate = 0.1
optim = nn.SGD (model.parameters(), learning_rate)

for i,(x,y) in enumerate(get_data(n)):
    pred_y = model(x)
    loss = jt.sqr(pred_y - y)
    loss_mean = loss.mean()
    optim.step (loss_mean)
    print(f"step {i}, loss = {loss_mean.data.sum()}")
```

- 一个更为完整的例子：
  - https://cg.cs.tsinghua.edu.cn/jittor/tutorial/2020-3-17-09-53-mnistclassification/

# 更多文档

- Jittor官网
  - https://cg.cs.tsinghua.edu.cn/jittor/
- 官方教程
  - https://cg.cs.tsinghua.edu.cn/jittor/tutorial/
- 安装指南
  - https://cg.cs.tsinghua.edu.cn/jittor/download/
- 官方文档
  - https://cg.cs.tsinghua.edu.cn/jittor/assets/docs/index.html

# Thanks

联系方式

徐昆 xukun@tsinghua.edu.cn

助教 鄢涤非 ydf22@mails.tsinghua.edu.cn