16.2.2 Reference

- A guide to convolution arithmetic for deep learning[?]
- Distill: Deconvolution and Checkerboard Artifacts[?]

16.3 LeNet5

16.3.1 Model formulation

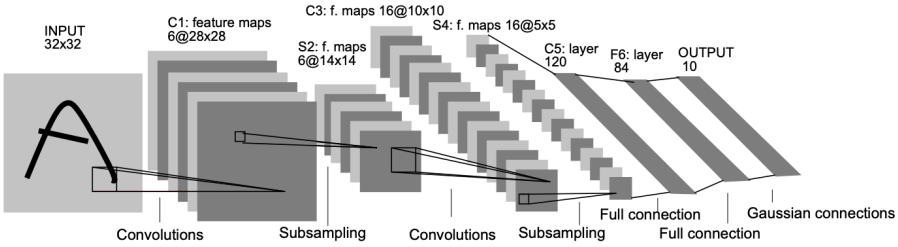


图 52: Architecture of LeNet5[?]

16.3.2 Model implement

MNIST[?] 手写体识别数据集（也可选择 Imagenet 数据集 [?]）存储方式如图，每行为一个样本，第一列为 label，第二列～最后一列为像素点

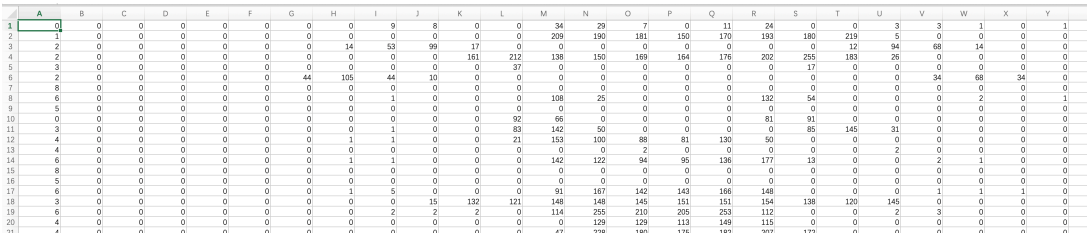


图 53: The dataset saving in practical

```
# -*- coding: utf-8 -*-
"""
Created on 2018/12/28 15:39
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of LeNet5 using TensorFlow framework.
"""
import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer
from six.moves import xrange

def input_fn(
        filenames,
        delimiter,
        epochs,
        batch_size,
        shuffle=True,
```

```
        dtype=tf.float32,
        num_parallel_calls=10,
        buffer_size_prefetch=100000,
        buffer_size_shuffle=2048):
    def map_func(line):
        columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
        label =tf.string_to_number(string_tensor=columns.values[0], out_type=tf.int32, name=None)
        feature =tf.string_to_number(string_tensor=columns.values[1: ], out_type=dtype, name=None)
        return feature, label
    dataset =tf.data.TextLineDataset(filenames=filenames).\
        map(map_func=map_func, num_parallel_calls=num_parallel_calls).\
        prefetch(buffer_size=buffer_size_prefetch)
    if shuffle ==True:
        dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
    dataset =dataset.repeat(count=epochs).batch(batch_size=batch_size, drop_remainder=False)
    iterator =dataset.make_one_shot_iterator()
    features, labels =iterator.get_next()
    return features, labels

def model_fn(features, labels, mode, params):
    """Model function of LeNet5."""
    # ----------Define hyper-parameters----------
    num_classes =params["num_classes"]
    image_height =params["image_height"]
    image_width =params["image_width"]
    image_channels =params["image_channels"]
    conv1_size =params["conv1_size"] # The height and width of filters of convolutional layer-1
    conv1_deep =params["conv1_deep"] # The number of filters of convolutional layer-1
    conv2_size =params["conv2_size"]
    conv2_deep =params["conv2_deep"]
    pooling_height =params["pooling_height"]
    pooling_width =params["pooling_width"]
    hidden_units =params["hidden_units"]
    dropout_rates =params["dropout_rates"]
    reg =params["reg"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]
    dtype =params["dtype"]
    reuse =params["reuse"]

    with tf.variable_scope(name_or_scope="inference", reuse=reuse):
        with tf.variable_scope(name_or_scope="input", reuse=reuse):
            x =tf.reshape(tensor=features, shape=[-1, image_height, image_width, image_channels])

        with tf.variable_scope(name_or_scope="layer1-conv1", reuse=reuse):
            weightsConv1 =tf.get_variable(
                name="weight",
                shape=[conv1_size, conv1_size, image_channels, conv1_deep], # The shape of (filter_height,
                                                                            filter_width, in_channels, out_channels)
                dtype=dtype,
                initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype),
                regularizer=l2_regularizer(scale=reg)
```

```
        )
        biasConv1 =tf.get_variable(
            name="bias",
            shape=[conv1_deep], # The number of filters of convolutional layer-1
            dtype=dtype,
            initializer=tf.constant_initializer(value=0.0, dtype=dtype)
        )
        conv1 =tf.nn.conv2d(
            input=x,
            filter=weightsConv1,
            strides=[1, 1, 1, 1],
            padding="SAME",
            data_format="NHWC" # it means (None, height, width, channel)
        ) # A tensor in shape of (None, image_height, image_width, conv1_deep)
        relu1 =tf.nn.relu(
            features=tf.nn.bias_add(value=conv1, bias=biasConv1, data_format="NHWC") # tf.nn.bias_add() is
                                                                                      used for add bias in CNN
        ) # A tensor in shape of (None, image_height, image_width, conv1_deep)

    with tf.variable_scope(name_or_scope="layer2-pool1", reuse=reuse):
        pool1 =tf.nn.max_pool(
            value=relu1,
            ksize=[1, pooling_height, pooling_width, 1],
            strides=[1, pooling_height, pooling_width, 1],
            padding="SAME",
            data_format="NHWC"
        ) # A tensor in shape of (None, image_height / 2, image_width / 2, conv1_deep)

    with tf.variable_scope(name_or_scope="layer3-conv2", reuse=reuse):
        weightsConv2 =tf.get_variable(
            name="weight",
            shape=[conv2_size, conv2_size, conv1_deep, conv2_deep],
            dtype=dtype,
            initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype),
            regularizer=l2_regularizer(scale=reg)
        )
        biasConv2 =tf.get_variable(
            name="bias",
            shape=[conv2_deep],
            dtype=dtype,
            initializer=tf.constant_initializer(value=0.0, dtype=dtype)
        )
        conv2 =tf.nn.conv2d(
            input=pool1,
            filter=weightsConv2,
            strides=[1, 1, 1, 1],
            padding="SAME",
            data_format="NHWC"
        ) # A tensor in shape of (None, image_height / 2, image_width / 2, conv2_deep)
        relu2 =tf.nn.relu(
            features=tf.nn.bias_add(value=conv2, bias=biasConv2, data_format="NHWC")
        )
```

```
with tf.variable_scope(name_or_scope="layer4-pool2", reuse=reuse):
    pool2 =tf.nn.max_pool(
        value=relu2,
        ksize=[1, pooling_height, pooling_width, 1],
        strides=[1, pooling_height, pooling_width, 1],
        padding="SAME",
        data_format="NHWC"
    ) # A tensor in shape of (None, image_height / 4, image_width / 4, conv2_deep)
    logits =tf.layers.flatten(inputs=pool2) # Flattens an input tensor while preserving the batch axis
                                                (axis 0). Note it will return error after
                                                when using tf.reshape

with tf.variable_scope(name_or_scope="layer5-fc", reuse=reuse):
    for i in xrange(len(hidden_units)):
        logits =tf.layers.dense(
            inputs=logits,
            units=hidden_units[i],
            activation=tf.identity,
            use_bias=True,
            kernel_initializer=tf.glorot_normal_initializer(dtype=dtype),
            kernel_regularizer=l2_regularizer(scale=reg),
            bias_initializer=tf.constant_initializer(value=0.0, dtype=dtype)
        )
        logits =tf.layers.batch_normalization(
            inputs=logits,
            axis=-1,
            momentum=0.99,
            epsilon=1e-3,
            center=True,
            scale=True,
            beta_initializer=tf.zeros_initializer(dtype=dtype),
            gamma_initializer=tf.ones_initializer(dtype=dtype),
            moving_mean_initializer=tf.zeros_initializer(dtype=dtype),
            moving_variance_initializer=tf.ones_initializer(dtype=dtype),
            training=(mode ==tf.estimator.ModeKeys.TRAIN)
        )
        logits =tf.nn.relu(features=logits)
        logits =tf.layers.dropout(
            inputs=logits,
            rate=dropout_rates[i],
            training=(mode ==tf.estimator.ModeKeys.TRAIN)
        ) # A tensor in shape of (None, hidden_units[i])
    logits =tf.layers.dense(
        inputs=logits,
        units=num_classes,
        activation=tf.identity,
        use_bias=True,
        kernel_initializer=tf.glorot_normal_initializer(dtype=dtype),
        kernel_regularizer=l2_regularizer(scale=reg),
        bias_initializer=tf.constant_initializer(value=0.1, dtype=dtype)
    ) # A tensor in shape of (None, num_classes)
```

```
        probability =tf.nn.softmax(logits=logits, axis=-1) # A tensor in shape of (None, num_classes)
        category =tf.argmax(input=logits, axis=-1) # A tensor in shape of (None, )

        predictions ={
            "probability": probability,
            "class": category
        }

    # ----------Provide an estimator spec for `ModeKeys.PREDICTION` mode----------
    if mode ==tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

    with tf.variable_scope(name_or_scope="objective", reuse=reuse):
        loss =tf.reduce_mean(
            input_tensor=tf.nn.sparse_softmax_cross_entropy_with_logits(labels=labels, logits=logits), # A
                                                                    tensor in shape of (None, )
            axis=0,
            keepdims=False
        )
        regularization =tf.reduce_sum(
            input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
            axis=0,
            keepdims=False
        )
        loss_op =loss +regularization

    with tf.variable_scope(name_or_scope="optimization", reuse=reuse):
        if (optimizer.lower() =="sgd"):
            opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
        elif (optimizer.lower() =="momentum"):
            opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
        elif (optimizer.lower() =="nesterov"):
            opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
        elif (optimizer.lower() =="adagrad"):
            opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
        elif (optimizer.lower() =="adadelta"):
            opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
        elif (optimizer.lower() =="rmsprop"):
            opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
        elif (optimizer.lower() =="adam"):
            opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
        else:
            raise ValueError("Argument <optimizer> of model_fn is invalid")

        train_op =opt.minimize(loss=loss_op, global_step=tf.train.get_global_step())

    # ----------Provide an estimator spec for `ModeKeys.TRAIN` mode----------
    if (mode ==tf.estimator.ModeKeys.TRAIN):
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

    # ----------Build evaluation metrics----------
    with tf.variable_scope(name_or_scope="evaluation"):
```

```
        eval_metric_ops ={
            "accuracy": tf.metrics.accuracy(labels=labels, predictions=category)
        }

    # ----------Provide an estimator spec for `ModeKeys.EVAL` mode----------
    if mode ==tf.estimator.ModeKeys.EVAL:
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                                                eval_metric_ops=eval_metric_ops)


def main(unused_argv):
    task =0
    filenamesTrain =["..//..//data//fashion-mnist_train.csv"]
    filenamesEval =["..//..//data//fashion-mnist_test.csv"]
    filenamesTest =["..//..//data//fashion-mnist_test.csv"]
    model_dir ="..//..//log//LeNet5"
    epochs =2000
    batch_size =128
    dtype =tf.float32
    params ={
        "num_classes": 10,
        "image_height": 28,
        "image_width": 28,
        "image_channels": 1,
        "conv1_size": 5,
        "conv1_deep": 32,
        "conv2_size": 5,
        "conv2_deep": 64,
        "pooling_height": 2,
        "pooling_width": 2,
        "hidden_units": [256, 128, 64, 32],
        "dropout_rates": [0.4, 0.3, 0.2, 0.2],
        "reg": 0.003,
        "optimizer": "Adam",
        "learning_rate": 0.001,
        "dtype": dtype,
        "reuse": False
    }
    # Create a estimator
    config =tf.estimator.RunConfig(
        log_step_count_steps=100,
        save_checkpoints_steps=10000,
        keep_checkpoint_max=1
    )
    estimator =tf.estimator.Estimator(model_fn=model_fn, model_dir=model_dir, params=params, config=config)
    # ----------Define task----------
    if (task ==0):
        train_spec =tf.estimator.TrainSpec(
            input_fn=lambda :input_fn(
                filenames=filenamesTrain,
                delimiter=",",
                epochs=epochs,
```

```
            batch_size=batch_size,
            dtype=dtype,
            shuffle=True
        ),
        max_steps=None
    )
    eval_spec =tf.estimator.EvalSpec(
        input_fn=lambda :input_fn(
            filenames=filenamesEval,
            delimiter=",",
            epochs=1,
            batch_size=batch_size,
            dtype=dtype,
            shuffle=False
        ),
        steps=None,
        start_delay_secs=120,
        throttle_secs=600
    )
    tf.estimator.train_and_evaluate(estimator=estimator, train_spec=train_spec, eval_spec=eval_spec)
elif (task ==1):
    estimator.train(
        input_fn=lambda :input_fn(
            filenames=filenamesTrain,
            delimiter=",",
            epochs=epochs,
            batch_size=batch_size,
            dtype=dtype,
            shuffle=True
        ),
        steps=None,
        max_steps=None
    )
elif (task ==2):
    estimator.evaluate(
        input_fn=lambda :input_fn(
            filenames=filenamesEval,
            delimiter=",",
            epochs=1,
            batch_size=batch_size,
            dtype=dtype,
            shuffle=False
        )
    )
elif (task ==3):
    pred_iter =estimator.predict(
        input_fn=lambda :input_fn(
            filenames=filenamesTest,
            delimiter=",",
            epochs=1,
            batch_size=batch_size,
            dtype=dtype,
```

```
            shuffle=True
        )
    )
    for element in pred_iter:
        print(element["class"])
else:
    raise ValueError("Argument <task> must be in [0, 1, 2, 3]")

if __name__ =='__main__':
    tf.logging.set_verbosity(v=tf.logging.INFO)
    tf.app.run(main=main)
```

16.4 AlexNet

16.4.1 Model formulation

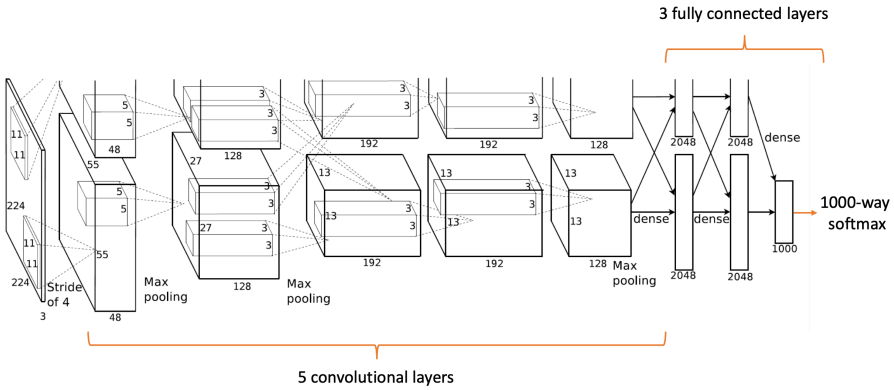


图 54: Architecture of AlexNet[?]

16.4.2 Model implement

TensorFlow 实现代码如下：

```
# Copyright (C) 2019 Tong Jia. All rights reserved.
import tensorflow as tf


def fetch_alexnet_logit_fn(inputs, params, is_training):
    """Definition of the logit inference function (before softmax operation converting into probability
                                                    distribution) of
    AlexNet[Krizhevsky et al., 2012]
    (https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf).
    When in learning phase and use the triplet loss only as loss function, the function output the final layer
                                                    and we
    don't need to add softmax layer after this.

    Parameters
    ----------
    :param inputs: (dict) -- containing mini-batch dataset following:
                        "images": tensor with shape of (None, 224, 224, num_channels);
                        "labels": tensor with shape of (None).
    :param params: (dict) -- containing all hyper-parameters of model.
                         "num_classes": (int) -- representing the number of all classes for output the
                                                                        distribution.
                         "use_dropout": (bool) -- indicating whether to use dropout in all hidden fc layers.
    :param is_training: (bool) -- indicating whether is train phase, or evaluation and inference phases.

    Returns
    ----------
    :return: (tensor) -- representing the output tensor with shape of [None, num_classes].
```

```
    """
    # Declare all hyper-parameters inside params dict.
    num_classes =params["num_classes"]
    use_dropout =params["use_dropout"]

    # Get the mini-batch images tensor from data dict.
    imgs =inputs["images"] # (None, Height = 224, Width = 224, Channel = 3).

    with tf.variable_scope(name_or_scope="conv1"):
        # Define trainable variables in 1-th convolution layer.
        in_channels =imgs.get_shape().as_list()[-1]
        out_channels =96
        kernel =tf.get_variable(
            name="kernel",
            shape=[11, 11, in_channels, out_channels],
            dtype=tf.float32,
            initializer=tf.initializers.he_normal()
        )
        bias =tf.get_variable(
            name="bias",
            shape=[out_channels],
            dtype=tf.float32,
            initializer=tf.zeros_initializer(dtype=tf.float32)
        )
        # Define the padding operation.
        paddings =tf.constant(value=[[0, 0], [3, 3], [3, 3], [0, 0]], dtype=tf.int32) # pad shape setting
        xs =tf.pad(tensor=imgs, paddings=paddings, mode="CONSTANT", constant_values=0)
        # Define the convolution operation.
        feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 4, 4, 1], padding="VALID",
                                                                data_format="NHWC")
        xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
        # Define activation function operation.
        ys =tf.nn.relu(features=xs)
        # Define local response normalization (lateral inhibition inside surrounding channels).
        ys =tf.nn.lrn(input=ys, depth_radius=2, bias=2, alpha=1e-4, beta=0.75)

    with tf.variable_scope(name_or_scope="pool1"):
        # Define the max pooling operation in 1-th pooling layer.
        xs =tf.nn.max_pool(value=ys, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding="VALID",
                                                                data_format="NHWC")

    with tf.variable_scope(name_or_scope="conv2"):
        # Define trainable variables in 2-th convolution layer. Note only 1-th convolution layer and 2-th
                                                                convolution
        # layer use local response normalization.
        in_channels =xs.get_shape().as_list()[-1]
        out_channels =256
        kernel =tf.get_variable(
            name="kernel",
            shape=[5, 5, in_channels, out_channels],
            dtype=tf.float32,
            initializer=tf.initializers.he_normal()
```

```
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_channels],
        dtype=tf.float32,
        initializer=tf.zeros_initializer(dtype=tf.float32)
    )
    # Define the padding operation.
    paddings =tf.constant(value=[[0, 0], [2, 2], [2, 2], [0, 0]], dtype=tf.int32) # pad shape setting
    xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
    # Define the convolution operation.
    feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                                                        data_format="NHWC")
    xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)
    # Define local response normalization (lateral inhibition inside surrounding channels).
    ys =tf.nn.lrn(input=ys, depth_radius=2, bias=2, alpha=1e-4, beta=0.75)

with tf.variable_scope(name_or_scope="pool2"):
    # Define the max pooling operation in 2-th pooling layer.
    xs =tf.nn.max_pool(value=ys, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding="VALID",
                                                        data_format="NHWC")

with tf.variable_scope(name_or_scope="conv3"):
    # Define trainable variables in 3-th convolution layer.
    in_channels =xs.get_shape().as_list()[-1]
    out_channels =384
    kernel =tf.get_variable(
        name="kernel",
        shape=[3, 3, in_channels, out_channels],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_channels],
        dtype=tf.float32,
        initializer=tf.zeros_initializer()
    )
    # Define the padding operation.
    paddings =tf.constant(value=[[0, 0], [1, 1], [1, 1], [0, 0]], dtype=tf.int32) # pad shape setting
    xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
    # Define the convolution operation.
    feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                                                        data_format="NHWC")
    xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)

with tf.variable_scope(name_or_scope="conv4"):
    # Define trainable variables in 4-th convolution layer. Note between 3-th and 4-th convolution layers,
```

```
                                                        there
    # doesn't exist local response normalization and pooling operation.
    in_channels =ys.get_shape().as_list()[-1]
    out_channels =384
    kernel =tf.get_variable(
        name="kernel",
        shape=[3, 3, in_channels, out_channels],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_channels],
        dtype=tf.float32,
        initializer=tf.zeros_initializer()
    )
    # Define the padding operation.
    paddings =tf.constant(value=[[0, 0], [1, 1], [1, 1], [0, 0]], dtype=tf.int32) # pad shape setting
    xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
    # Define the convolution operation.
    feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                                                        data_format="NHWC")
    xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)

with tf.variable_scope(name_or_scope="conv5"):
    # Define trainable variables in 5-th convolution layer.
    in_channels =ys.get_shape().as_list()[-1]
    out_channels =256
    kernel =tf.get_variable(
        name="kernel",
        shape=[3, 3, in_channels, out_channels],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_channels],
        dtype=tf.float32,
        initializer=tf.zeros_initializer()
    )
    # Define the padding operation.
    paddings =tf.constant(value=[[0, 0], [1, 1], [1, 1], [0, 0]], dtype=tf.int32) # pad shape setting
    xs =tf.pad(tensor=xs, paddings=paddings, mode="CONSTANT", constant_values=0)
    # Define the convolution operation.
    feature =tf.nn.conv2d(input=xs, filter=kernel, strides=[1, 1, 1, 1], padding="VALID",
                                                        data_format="NHWC")
    xs =tf.nn.bias_add(value=feature, bias=bias, data_format="NHWC")
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)
```

```
with tf.variable_scope(name_or_scope="pool5"):
    # Define the max pooling operation in 3-th pooling layer.
    xs =tf.nn.max_pool(value=ys, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding="VALID",
                                                              data_format="NHWC")
    # Flatten the tensor with shape of (None, Height, Width, Channel).
    xs =tf.layers.flatten(inputs=xs)

with tf.variable_scope(name_or_scope="fc1"):
    if use_dropout:
        # Define dropout operation in 1-th fully connected layer.
        xs =tf.layers.dropout(
            inputs=xs,
            rate=0.2,
            training=is_training
        )
    # Define trainable variables in 1-th fully connected layer.
    in_units =xs.get_shape().as_list()[-1]
    out_units =4096
    weight =tf.get_variable(
        name="weight",
        shape=[in_units, out_units],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
    bias =tf.get_variable(
        name="bias",
        shape=[out_units],
        dtype=tf.float32,
        initializer=tf.zeros_initializer(dtype=tf.float32)
    )
    # Define matrix multiplication operation.
    xs =tf.nn.xw_plus_b(x=xs, weights=weight, biases=bias)
    # Define activation function operation.
    ys =tf.nn.relu(features=xs)

with tf.variable_scope(name_or_scope="fc2"):
    if use_dropout:
        # Define dropout operation in 1-th fully connected layer.
        xs =tf.layers.dropout(
            inputs=xs,
            rate=0.2,
            training=is_training
        )
    # Define trainable variables in 2-th fully connected layer.
    in_units =ys.get_shape().as_list()[-1]
    out_units =4096
    weight =tf.get_variable(
        name="weight",
        shape=[in_units, out_units],
        dtype=tf.float32,
        initializer=tf.initializers.he_normal()
    )
```

```
        bias =tf.get_variable(
            name="bias",
            shape=[out_units],
            dtype=tf.float32,
            initializer=tf.zeros_initializer(dtype=tf.float32)
        )
        # Define matrix multiplication operation.
        xs =tf.nn.xw_plus_b(x=ys, weights=weight, biases=bias)
        # Define activation function operation.
        ys =tf.nn.relu(features=xs)

    with tf.variable_scope(name_or_scope="fc3"):
        # Define trainable variables in 3-th fully connected layer.
        in_units =ys.get_shape().as_list()[-1]
        weight =tf.get_variable(
            name="weight",
            shape=[in_units, num_classes],
            dtype=tf.float32,
            initializer=tf.variance_scaling_initializer(
                scale=1.0, mode="fan_avg", distribution="truncated_normal", dtype=tf.float32
            )
        )
        bias =tf.get_variable(
            name="bias",
            shape=[num_classes],
            dtype=tf.float32,
            initializer=tf.zeros_initializer(dtype=tf.float32)
        )
        # Define matrix multiplication operation.
        logits =tf.nn.xw_plus_b(x=ys, weights=weight, biases=bias) # [None, num_classes]

    return logits
```

16.4.3 Reference

- Djordje Slijepcevic: Deep Learning with Tensorflow - AlexNet

16.5 VGG

16.5.1 Model formulation

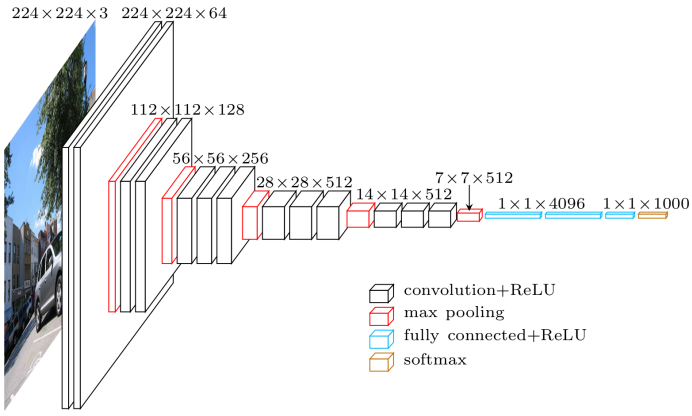


图 55: Architecture of VGG[?]

16.6 GoogLeNet

16.6.1 Model formulation

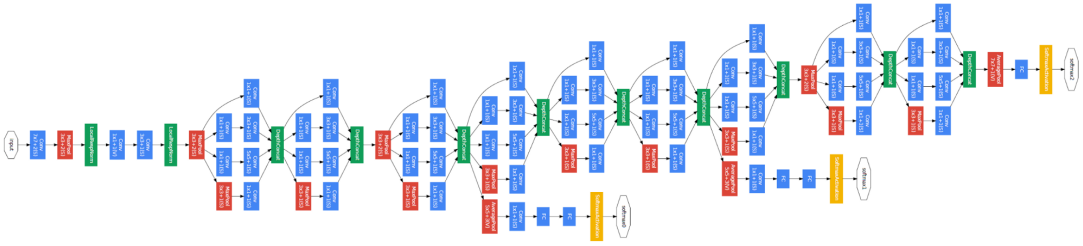


图 56: Architecture of GoogleNet[**?**]

16.7 ResNet

16.7.1 Model formulation

ResNet[?]（图像领域里程碑式的工作）的基本单元结构如下：

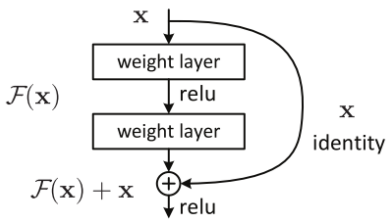


图 57: Architecture of ResNet unit

问题：为何 ResNet 相对于一般的 CNN 可以更有效地防止梯度弥散/梯度爆炸？

$$
\begin{aligned}
\because \ \boldsymbol{y} &= F(\boldsymbol{x}) + \boldsymbol{x} \\
\therefore \ \frac{\partial J}{\partial \boldsymbol{x}} &= \frac{\partial J}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \\
&= \frac{\partial J}{\partial \boldsymbol{y}} \frac{\partial}{\partial \boldsymbol{x}} (F(\boldsymbol{x}) + \boldsymbol{x}) \\
&= \frac{\partial J}{\partial \boldsymbol{y}} (\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}} + \mathbf{1}) \\
&= \underbrace{\frac{\partial J}{\partial \boldsymbol{y}} \frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}}}_{\text{left-part gradients}} + \underbrace{\frac{\partial J}{\partial \boldsymbol{y}} \mathbf{1}}_{\text{right-part gradients}}
\end{aligned}
$$

注意这里括号内的**1**为精髓，也就是说**为梯度回流开辟了一条 VIP 高速通道**。即使单元左侧部分出现梯度弥散：$\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}} \simeq 0$，单元右侧始终可以保证梯度的有效回流，从而保证了深度模型的有效训练。

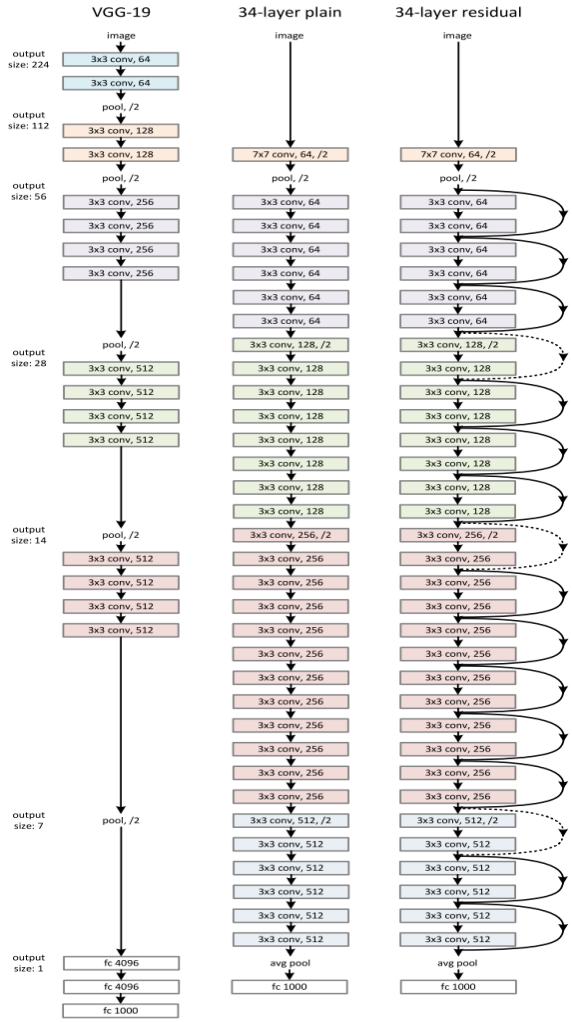


图 58: Architecture of ResNet[?] via architectures of normal CNN

16.8 DenseNet

16.8.1 Model formulation

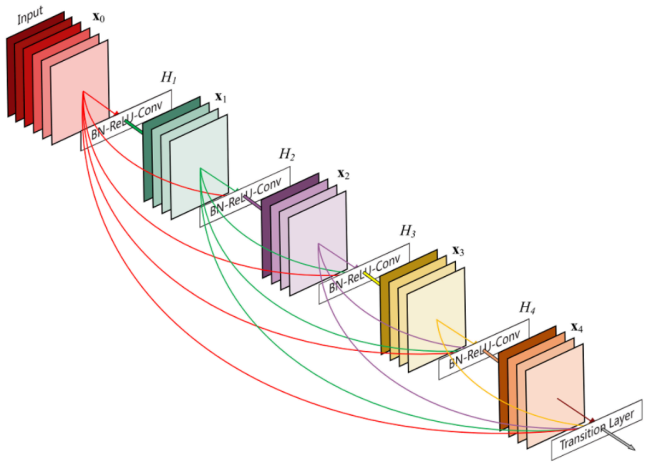


图 59: Architecture of DenseNet[**?**]

17 Recurrent Neural Networks (RNNs)

17.1 Introduction

RNN 形象化的理解：拼图

1. 眼睛看到现在手里的拼图小块，$\boldsymbol{x}_t$
2. 回忆目前最新完成的拼图场景，$\boldsymbol{h}_{t-1}$
3. 结合 1, 2 的信息做出当前时刻拼图小块应该插入到哪里，$\hat{y}_t$

图 60: Puzzle example for history memory and time series decision

注意第二步，我们在回忆历史的时候，一般不是简单回忆上一个插入的拼图小块，而是去回忆一个**模糊而宏观**的场景，在这个例子中，我们的场景就是当前时刻拼图的整体大概样貌，这就是隐层向量 $\boldsymbol{h}_{t-1}$。这也是为何不将 $\boldsymbol{h}_t$ 接上一时刻输出 y_{t-1} 的原因。

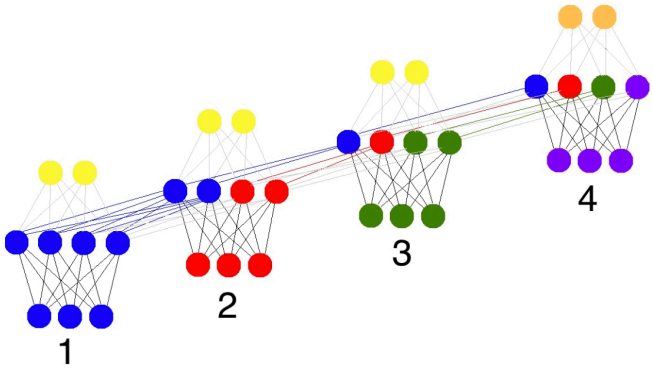


图 61: Recurrent neural network unpack in timestemp

17.2 Vanilla Recurrent Neural Network (RNN)

17.2.1 Derivation of forward-propagation in Vanilla RNN

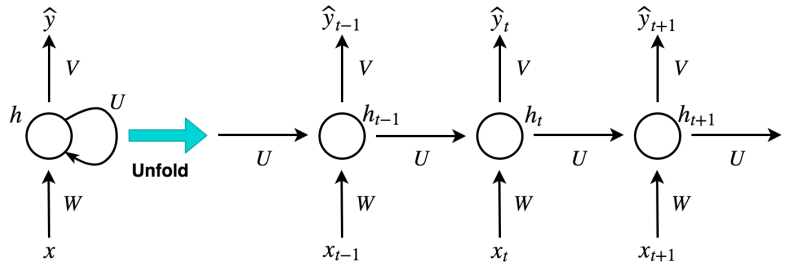


图 62: RNN forward propagation. Image is drawn under Draw.io.

以简单循环网络（Simple Recurrent Network）为例进行推导，t 时刻有：

$$\boldsymbol{h}_t = f(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b})$$
$$\widehat{\boldsymbol{y}}_t = g(\boldsymbol{V}\boldsymbol{h}_t)$$

或者更详细地表示：

$$\boldsymbol{z}_t = \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}$$
$$\boldsymbol{h}_t = f(\boldsymbol{z}_t)$$
$$\boldsymbol{o}_t = \boldsymbol{V}\boldsymbol{h}_t$$
$$\widehat{\boldsymbol{y}}_t = g(\boldsymbol{o}_t)$$

最后 $\widehat{\boldsymbol{y}}_t$ 直接对接了 t 时刻的损失函数 J_t。参数及变量解释如下：

- $\boldsymbol{x}_t \in \mathbb{R}^d$: t 时刻的输入向量（如词向量），向量长度为 d
- $\boldsymbol{W} \in \mathbb{R}^{l\times d}$
- $\boldsymbol{U} \in \mathbb{R}^{l\times l}$
- $\boldsymbol{b} \in \mathbb{R}^l$

17.2.2 Derivation of back-propagation through time (BPTT) in Vanilla RNN

为了推导的方便，假设 RNN 在每个时刻 t 都有一个监督信息，损失函数记为 J_t。那么整个序列的损失函数记为：

$$J = \sum_{t=1}^{T} J_t$$

损失 J 关于参数 $\boldsymbol{U}$ 的梯度为：

$$\frac{\partial J}{\partial \boldsymbol{U}} = \sum_{t=1}^{T} \frac{\partial J_t}{\partial \boldsymbol{U}}$$

$$
\begin{aligned}
&= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{U}} \\
&= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{U}} \\
&= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \underbrace{\left(\sum_{k=1}^{t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{U}} \right)}_{\text{BPTT}} \quad \left(\because \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{U}} = \sum_{k=1}^{t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{U}} \right) \\
&= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{U}}
\end{aligned}
$$

注意上边不要展开求 t **时刻隐层输入**到 $t-1$ **时刻隐层输出**的梯度 $\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_{t-1}}$，而是到下面最关键的步骤再展

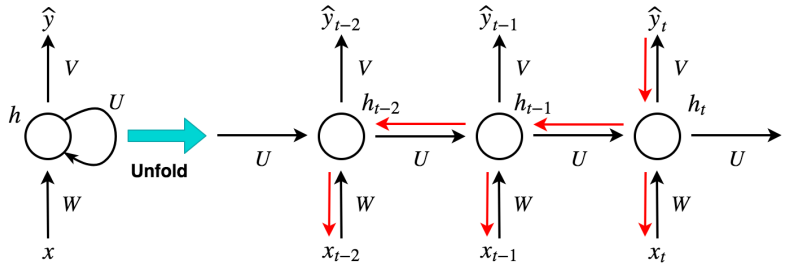


图 63: RNN backpropagation through time

开。即**保留** $\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_{t-1}}$。式子中最关键的梯度：

$$
\begin{aligned}
\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} &= \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{z}_{i-1}} \\
&= \prod_{i=k+1}^{t} \underbrace{\frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{h}_{i-1}} \frac{\partial \boldsymbol{h}_{i-1}}{\partial \boldsymbol{z}_{i-1}}}_{\text{易忽略}} \\
&= \prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}} \operatorname{diag}\left(f'(\boldsymbol{z}_{i-1})\right)
\end{aligned}
$$

代入上式中可得：

$$
\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{U}} &= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{U}} \\
&= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \left(\prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}} \operatorname{diag}\left[f'(\boldsymbol{z}_{i-1})\right] \right) \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{U}}
\end{aligned}
$$

这样推导出来是最好的展示，因为：

- $\frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t}$ 只和 t 时刻的损失函数 $J(\boldsymbol{y}_t, \widehat{\boldsymbol{y}}_t)$ 有关
- $\frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t}$ 只和 t 时刻 logit 输出 $\boldsymbol{o}_t$ 与最终输出 $\widehat{\boldsymbol{y}}_t$ 之间的激活函数 g 有关

- $\frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t}$ 只和参数 $\boldsymbol{V}$ 有关
- $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t}$ 只和 t 时刻隐层输入 $\boldsymbol{z}_t$ 与隐层输出 $\boldsymbol{h}_t$ 之间的激活函数 f 有关

同理可得：

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{W}} &= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \boldsymbol{W}} \\ &= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{W}} \\ &= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \left(\sum_{k=1}^{t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{W}} \right) \\ &= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{W}} \\ &= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \left(\prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}} \mathrm{diag}\big[f'(\boldsymbol{z}_{i-1})\big] \right) \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{W}}\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{b}} &= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \boldsymbol{b}} \\ &= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \widehat{\boldsymbol{o}}_t} \frac{\partial \widehat{\boldsymbol{o}}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{b}} \\ &= \sum_{t=1}^{T} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \left(\sum_{k=1}^{t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{b}} \right) \\ &= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{b}} \\ &= \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \left(\prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}} \mathrm{diag}\big[f'(\boldsymbol{z}_{i-1})\big] \right) \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{b}}\end{aligned}$$

17.2.3 Vanishing gradient problem in RNN

回顾 RNN 中最关键的推导：

$$\begin{aligned}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k} &= \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{z}_{i-1}} \\ &= \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{h}_{i-1}} \frac{\partial \boldsymbol{h}_{i-1}}{\partial \boldsymbol{z}_{i-1}} \\ &= \prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}} \mathrm{diag}\big(f'(\boldsymbol{z}_{i-1})\big)\end{aligned}$$

因此参数 $\boldsymbol{W}$，$\boldsymbol{U}$ 和 $\boldsymbol{b}$ 的梯度分别为：

$$\frac{\partial J}{\partial \boldsymbol{W}} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t} \frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \left(\prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}} \mathrm{diag}\big[f'(\boldsymbol{z}_{i-1})\big] \right) \frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{W}}$$

$$\frac{\partial J}{\partial \boldsymbol{U}} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t}\frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t}\frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t}\left(\prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}}\mathrm{diag}\left[f'(\boldsymbol{z}_{i-1})\right]\right)\frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{U}}$$

$$\frac{\partial J}{\partial \boldsymbol{b}} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial J_t}{\partial \widehat{\boldsymbol{y}}_t}\frac{\partial \widehat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t}\frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t}\left(\prod_{i=k+1}^{t} \boldsymbol{U}^{\mathrm{T}}\mathrm{diag}\left[f'(\boldsymbol{z}_{i-1})\right]\right)\frac{\partial \boldsymbol{z}_k}{\partial \boldsymbol{b}}$$

问题：什么是梯度弥散/梯度爆炸？ 我们定义 $\gamma = \left\|\boldsymbol{U}^{\mathrm{T}}\mathrm{diag}\left(f'(\boldsymbol{z}_{i-1})\right)\right\|$，则上述梯度中括号内的部分为 γ^{t-k}，那么当时刻 k 和时刻 t 相距很远时候：

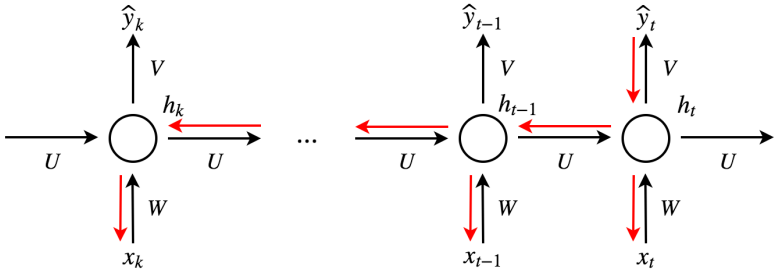


图 64: Vanishing gradient problem in RNN

- $\gamma < 1 \Rightarrow \gamma^{t-k} \simeq 0$: **梯度消失**
- $\gamma > 1 \Rightarrow \gamma^{t-k} \to +\infty$: **梯度爆炸**

并且距离越远越明显。

问题：为什么会梯度弥散/梯度爆炸？ 就是 γ 的问题，而 γ 的问题就是**激活函数**带来的问题。RNN 一般使用 tanh 函数作为隐层的激活函数，即：

$$\begin{aligned}
\because \tanh(x) &= \frac{e^{2x}-1}{e^{2x}+1} \\
\therefore \tanh'(x) &= \frac{2e^{2x}(e^{2x}+1)-(e^{2x}-1)2e^{2x}}{(e^{2x}+1)^2} \\
&= \frac{2e^{2x}e^{2x}+2e^{2x}-2e^{2x}e^{2x}+2e^{2x}}{(e^{2x}+1)^2} \\
&= \frac{4e^{2x}}{(e^{2x}+1)^2} \\
&= \frac{4e^{2x}}{e^{4x}+2e^{2x}+1}
\end{aligned}$$

$\tanh(x)$ 和其导数的图像如图所示，可以看到 $\tanh'(x)$ 无论何时都小于 1(除了 $x=0$)，而权重 $\|\boldsymbol{U}^{\mathrm{T}}\|$ 也不会太大，我们定义：

- $\|\boldsymbol{U}^{\mathrm{T}}\| \leq \gamma_u \leq 1$
- $\|\mathrm{diag}\left[f'(\boldsymbol{z}_{i-1})\right]\| \leq \gamma_f \leq 1$

那么可得：

$$\because \left\|\frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{z}_{i-1}}\right\| = \left\|\boldsymbol{U}^{\mathrm{T}}\mathrm{diag}\left[f'(\boldsymbol{z}_{i-1})\right]\right\| \leq \left\|\boldsymbol{U}^{\mathrm{T}}\right\| \cdot \left\|\mathrm{diag}\left[f'(\boldsymbol{z}_{i-1})\right]\right\| \leq \gamma_u \gamma_f \leq 1$$
$$\therefore \left\|\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k}\right\| \leq (\gamma_u \gamma_f)^{t-k}$$

如果间隔 $t-k$ 过大，那么 $\left\|\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{z}_k}\right\|$ 会趋于 0，也就是对长距离的依赖无法训练甚至崩溃。

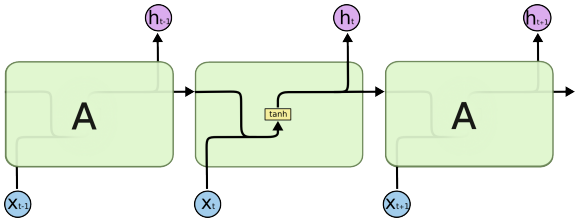


图 65: $\tanh(x)$ is the activation function of RNN hidden unit

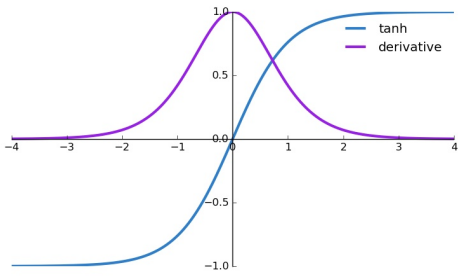


图 66: $\tanh(x)$ and its gradient

问题：梯度弥散会带来什么样的影响？ 例如句子“the clouds are in the **sky**”这样的预测问题很容易，RNN 就能解决；但是像“I grew up in France···I speak fluent **French**.”这样存在句子长期依赖的预测问题，RNN 就会因为梯度弥散而无法解决，这就是所谓的**长期依赖问题**。

17.2.4 References

- Neural Netwrok & Deep Learning
- Understanding LSTM Networks

17.3 Long Short-Term Memory Neural Network (LSTM)

17.3.1 Derivation of forward-propagation in LSTM

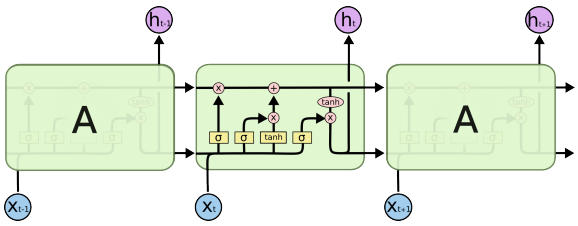
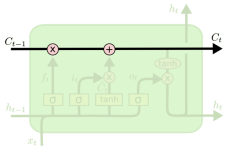
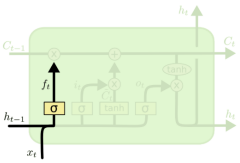


图 67: The repeating module in an LSTM contains four interacting layers.

LSTMs 核心思想 LSTMs[?] 的核心思想是**cell state**，类似于**传送带**，决定信息的流量



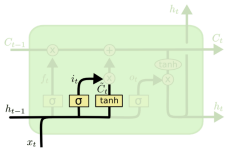
Step1: 决定从 cell state 中扔掉哪些信息



$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f\,[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f)$$

- 名称：遗忘层（forget gate layer）
- 参数：$\boldsymbol{W}_f \in \mathbb{R}^{(l+d)\times l}$, $b_f \in \mathbb{R}^l$ （l 为隐层神经元个数，d 为任意时刻输入向量 $\boldsymbol{x}_t$ 的维度，下同）
- 输入：$\boldsymbol{h}_{t-1} \in \mathbb{R}^l$, $\boldsymbol{x}_t \in \mathbb{R}^d$
- 输出：$\boldsymbol{f}_t \in \mathbb{R}^l$, $\boldsymbol{f}_{t,j} \in (0,1)$ (通过 sigmoid 函数激活)
- 作用：稍后作用于 t - 1 时刻的神经元状态（cell state）$\boldsymbol{C}_{t-1}$ 中的每一个元素（**注：每一个元素 = 每一个隐层神经元**）
- 解释：表示保留 $\boldsymbol{C}_{t-1}$ 中每一个元素的程度，即：
  - $f_{t,j} = 1$: 完全保留第 j 个隐层神经元信息
  - $f_{t,j} = 0$: 完全遗忘第 j 个隐层神经元信息

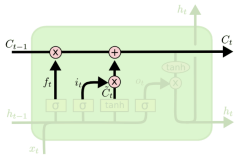
Step2: 决定从 cell state 中保存哪些信息



$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i\,[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i)$$
$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_C\,[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C)$$

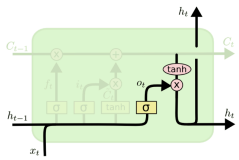
- 名称：输入层（input gate layer）
- 参数：$\boldsymbol{W}_i \in \mathbb{R}^{(l+d)\times l}$，$b_i \in \mathbb{R}^l$，$\boldsymbol{W}_C \in \mathbb{R}^{(l+d)\times l}$，$b_C \in \mathbb{R}^l$
- 输入：$\boldsymbol{h}_{t-1} \in \mathbb{R}^l$, $\boldsymbol{x}_t \in \mathbb{R}^d$
- 输出：$\boldsymbol{i}_t \in \mathbb{R}^l$，$\tilde{\boldsymbol{C}}_t \in \mathbb{R}^l$
- 作用：稍后会合并二者，对 state 创建一个 update
  - sigmoid 激活函数层：所有隐含神经元记忆新内容的程度
  - tanh 激活函数层：创建一个可能之后会被加载进 state 的候选新内容向量

Step3: 更新旧的 cell state C_{t-1} 为新的 cell state C_t



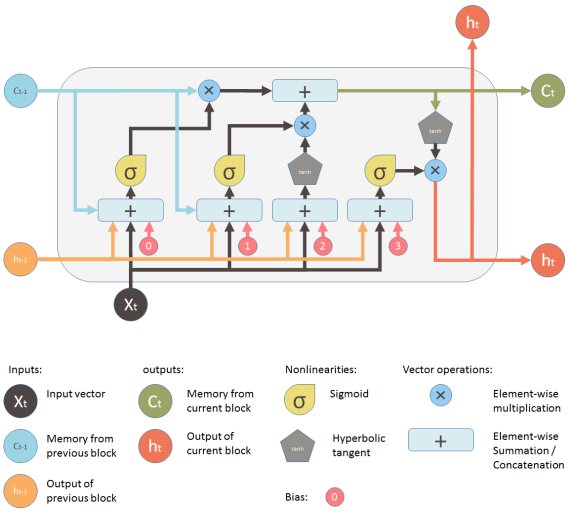
$$\boldsymbol{C}_t = \underbrace{\boldsymbol{f}_t \odot \boldsymbol{C}_{t-1}}_{\text{对旧信息的记忆}} + \underbrace{\boldsymbol{i}_t \odot \tilde{\boldsymbol{C}}_t}_{\text{对新信息的记忆}}$$

Step4: 依照当前 cell state C_t 决定最终的隐层输出内容



$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o\,[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o)$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$$

综合来看一个 LSTM 的 Unit 如图：



17.3.2 Derivation of back-propagation through time (BPTT) in LSTM

参数解释：

- d: 任意时刻特征向量的维度
- l: 任意时刻隐层向量的维度
- m: Batch size

重新按照深度学习的框架写正向推导：

1. 遗忘门：$\boldsymbol{f}_t = \sigma([\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]\,\boldsymbol{W}_f + \boldsymbol{b}_f)$
   - $\because [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] \in \mathbb{R}^{m\times(l+d)}$
   - $\because \boldsymbol{W}_f \in \mathbb{R}^{(l+d)\times l}$
   - $\because \boldsymbol{b}_f \in \mathbb{R}^{m\times l}$
   - $\therefore \boldsymbol{f}_t \in \mathbb{R}^{m\times l}$
2. 输入门：$\boldsymbol{i}_t = \sigma([\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]\,\boldsymbol{W}_i + \boldsymbol{b}_i), \quad \tilde{\boldsymbol{C}}_t = \tanh([\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]\,\boldsymbol{W}_C + \boldsymbol{b}_C)$
   - $\because [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] \in \mathbb{R}^{m\times(l+d)}$
   - $\because \boldsymbol{W}_i \in \mathbb{R}^{(l+d)\times l}$
   - $\because \boldsymbol{W}_C \in \mathbb{R}^{(l+d)\times l}$
   - $\because \boldsymbol{b}_i \in \mathbb{R}^{m\times l}$
   - $\because \boldsymbol{b}_C \in \mathbb{R}^{m\times l}$
   - $\therefore \boldsymbol{i}_t \in \mathbb{R}^{m\times l}$
   - $\therefore \tilde{\boldsymbol{C}}_t \in \mathbb{R}^{m\times l}$
3. 更新门：$\boldsymbol{C}_t = \underbrace{\boldsymbol{f}_t \odot \boldsymbol{C}_{t-1}}_{\text{对旧信息的记忆}} + \underbrace{\boldsymbol{i}_t \odot \tilde{\boldsymbol{C}}_t}_{\text{对新信息的记忆}}$
   - $\because \boldsymbol{f}_t \in \mathbb{R}^{m\times l}$
   - $\because \boldsymbol{i}_t \in \mathbb{R}^{m\times l}$
   - $\because \tilde{\boldsymbol{C}}_t \in \mathbb{R}^{m\times l}$
   - $\because \boldsymbol{C}_{t-1} \in \mathbb{R}^{m\times l}$
   - $\therefore \boldsymbol{C}_t \in \mathbb{R}^{m\times l}$
4. 输出门：$\boldsymbol{o}_t = \sigma([\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]\,\boldsymbol{W}_o + \boldsymbol{b}_o), \quad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$
   - $\because [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] \in \mathbb{R}^{m\times(l+d)}$
   - $\because \boldsymbol{W}_o \in \mathbb{R}^{(l+d)\times l}$
   - $\because \boldsymbol{b}_o \in \mathbb{R}^{m\times l}$
   - $\therefore \boldsymbol{o}_t \in \mathbb{R}^{m\times l}$
   - $\because \boldsymbol{C}_t \in \mathbb{R}^{m\times l}$

- $\therefore \tanh(\boldsymbol{C}_t) \in \mathbb{R}^{m \times l}$
- $\therefore \boldsymbol{h}_t \in \mathbb{R}^{m \times l}$

所有待训练参数：

- $\boldsymbol{W}_f \in \mathbb{R}^{(l+d) \times l}$
- $\boldsymbol{b}_f \in \mathbb{R}^l$
- $\boldsymbol{W}_i \in \mathbb{R}^{(l+d) \times l}$
- $\boldsymbol{W}_C \in \mathbb{R}^{(l+d) \times l}$
- $\boldsymbol{b}_i \in \mathbb{R}^l$
- $\boldsymbol{b}_C \in \mathbb{R}^l$
- $\boldsymbol{W}_o \in \mathbb{R}^{(l+d) \times l}$
- $\boldsymbol{b}_o \in \mathbb{R}^l$
- $\boldsymbol{V} \in \mathbb{R}^{l \times 1}$：隐层到输出层之间的参数

17.3.3 Model implement

```
# -*- coding: utf-8 -*-
"""
Created on 2018/12/17 03:03
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of single-layer rnn for single-timestep-output regression task.
"""
import tensorflow as tf
from tensorflow.python import rnn
from tensorflow.contrib.layers import l2_regularizer

def input_fn(
        filenames,
        delimiter,
        epochs,
        batch_size,
        dtype,
        shuffle=True,
        num_parallel_calls=10,
        buffer_size_prefetch=100000,
        buffer_size_shuffle=2048):
    def map_func(line):
        columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
        y =tf.string_to_number(string_tensor=columns.values[0], out_type=dtype)
        x =tf.string_to_number(string_tensor=columns.values[1: ], out_type=dtype)
        return x, y
    dataset =tf.data.TextLineDataset(filenames=filenames). \
        map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
        prefetch(buffer_size=buffer_size_prefetch)
    if shuffle ==True:
        dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
    dataset =dataset. \
        repeat(count=epochs). \
        batch(batch_size=batch_size, drop_remainder=False)
    iterator =dataset.make_one_shot_iterator()
    features, labels =iterator.get_next()
    return features, labels

def model_fn(features, labels, mode, params):
    # ----------Define Hyper-parameters----------
    num_steps =params["num_steps"]
    num_hidden_units =params["num_hidden_units"]
    dtype =params["dtype"]
    reg =params["reg"]
    cell =params["cell"] # A string in range of ["RNN", "LSTM", "GRU"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]
```

```
# ----------Define variables----------
with tf.variable_scope(name_or_scope="variables", reuse=tf.AUTO_REUSE):
    W = tf.get_variable(
        name="W",
        shape=[num_hidden_units, 1],
        dtype=dtype,
        initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype), # adjust
        regularizer=l2_regularizer(scale=reg)
    )
    b = tf.get_variable(
        name="bias",
        shape=[1],
        dtype=dtype,
        initializer=tf.zeros_initializer(dtype=dtype)
    )
with tf.variable_scope(name_or_scope="inference", reuse=tf.AUTO_REUSE):
    x = tf.reshape(tensor=features, shape=[-1, num_steps, 1])
    if cell.lower() =="rnn":
        basicCell =tf.nn.rnn_cell.BasicRNNCell(num_units=num_hidden_units, dtype=dtype)
    elif cell.lower() =="lstm":
        basicCell =tf.nn.rnn_cell.LSTMCell(num_units=num_hidden_units, use_peepholes=False,
                                           forget_bias=1.0,
                                           initializer=tf.orthogonal_initializer(dtype=dtype))
    elif cell.lower() =="gru":
        basicCell =tf.nn.rnn_cell.GRUCell(num_units=num_hidden_units,
                                          kernel_initializer=tf.orthogonal_initializer(dtype=dtype))
    else:
        raise ValueError("Argument <cell> of model_fn is invalid")

    outputs, states =rnn.dynamic_rnn(cell=basicCell, inputs=x, dtype=dtype) # outputs is a tensor in shape
                                                    of (None, num_steps, num_inputs)
    outputs =tf.transpose(a=outputs, perm=[1, 0, 2]) # a tensor in shape of (num_steps, None, num_inputs)
    predictions =tf.matmul(a=outputs[-1], b=W) +b # a tensor in shape of (None, 1)
    predictions =tf.reshape(tensor=predictions, shape=[-1])
    pred ={"predictions": predictions}

# ----------Provide an estimator spec for `ModeKeys.PREDICTION` mode----------
if mode ==tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=pred)

with tf.variable_scope(name_or_scope="loss", reuse=tf.AUTO_REUSE):
    loss =tf.losses.mean_squared_error(labels=labels, predictions=predictions)
    regloss =tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    ) # A scaler representing the regularization loss
    loss +=regloss

with tf.variable_scope(name_or_scope="optimizer", reuse=tf.AUTO_REUSE):
    if (optimizer.lower() =="sgd"):
        opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
```

```
    elif (optimizer.lower() =="momentum"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif (optimizer.lower() =="nesterov"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif (optimizer.lower() =="adagrad"):
        opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif (optimizer.lower() =="adadelta"):
        opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif (optimizer.lower() =="rmsprop"):
        opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif (optimizer.lower() =="ftrl"):
        opt =tf.train.FtrlOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() =="adam"):
        opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:
        raise ValueError("Argument <optimizer> of model_fn is invalid")
    train_op =opt.minimize(loss=loss, global_step=tf.train.get_global_step())

# ----------Provide an estimator spec for `ModeKeys.TRAIN` mode----------
if (mode ==tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# ----------Build evaluation metrics----------
with tf.variable_scope(name_or_scope="evaluation", reuse=tf.AUTO_REUSE):
    eval_metric_ops ={"rmse": tf.metrics.root_mean_squared_error(labels=labels, predictions=predictions)}

# ----------Provide an estimator spec for `ModeKeys.EVAL` mode----------
if mode ==tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, predictions=predictions,
                                                            eval_metric_ops=eval_metric_ops)
```

17.4 Gated Recurrent Unit (GRU)

17.4.1 Derivation of forward-propagation in GRU

GRU[?] 相对于标准 LSTMs 的异同：

- 将 LSTM 中的 forget gate 和 input gate 合并为一个 update gate
- 合并 LSTM 中的 cell state 和 hidden state
- 没有了 Bias

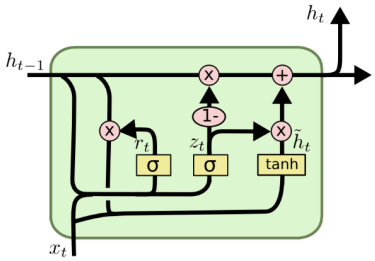


图 68: The unit of GRU[?]

- $\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right])$：新信息的进入率，老信息的遗忘率
- $\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r \left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right])$：当前时刻对 $\boldsymbol{h}_{t-1}$ 的接受程度
- $\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W} \left[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right])$: 新生产内容
- $\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t$: 最终隐含层输出

17.5 Bidirectional LSTM (Bi-LSTM)

17.5.1 Derivation of forward-propagation in Bi-LSTM

双向 LSTM[?] 实质结构为：

- 根据正向的序列数据训练一个 LSTM，在 t 时刻得到正向的隐层输出 $\overrightarrow{\boldsymbol{h}_t} \in \mathbb{R}^l$
- 根据反向的序列数据训练一个 LSTM，在 t 时刻得到反向的隐层输出 $\overleftarrow{\boldsymbol{h}_t} \in \mathbb{R}^l$
- 拼接 $\overrightarrow{\boldsymbol{h}_t}$ 与 $\overleftarrow{\boldsymbol{h}_t}$ 得到 t 时刻的隐层输出 $\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}_t}, \overleftarrow{\boldsymbol{h}_t}] \in \mathbb{R}^{2l}$
- **注意**：Bi-LSTM 相对于 LSTM 有更多的信息捕获，因此理论上效果好于 LSTM

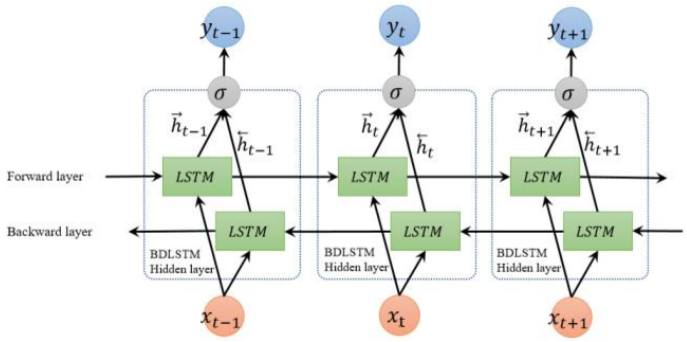


图 69: The structure of bidirectional lstm[?]

注意：此处基本的模型结构 LSTM 也可以替换为 RNN，GRU 等。

17.5.2 Model implement

考虑利用 Bi-LSTM 进行时序销量预测的例子，利用 TensorFlow 实现的代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on 2018/12/19 00:00
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of single-layer Bidirectional LSTM for single-timestep-output regression task.
"""
import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer

def input_fn(
        filenames,
        delimiter,
        epochs,
        batch_size,
        dtype,
```

```
        shuffle=True,
        num_parallel_calls=10,
        buffer_size_prefetch=100000,
        buffer_size_shuffle=2048):
    def map_func(line):
        columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
        y =tf.string_to_number(string_tensor=columns.values[0], out_type=dtype)
        x =tf.string_to_number(string_tensor=columns.values[1: ], out_type=dtype)
        return x, y

    dataset =tf.data.TextLineDataset(filenames=filenames). \
        map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
        prefetch(buffer_size=buffer_size_prefetch)

    if shuffle ==True:
        dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
    dataset =dataset. \
        repeat(count=epochs). \
        batch(batch_size=batch_size, drop_remainder=False)
    iterator =dataset.make_one_shot_iterator()
    features, labels =iterator.get_next()
    return features, labels

def model_fn(features, labels, mode, params):
    # ----------Define Hyper-parameters----------
    num_steps =params["num_steps"]
    num_hidden_units =params["num_hidden_units"]
    dtype =params["dtype"]
    reg =params["reg"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]

    # ----------Define variables----------
    with tf.variable_scope(name_or_scope="variables", reuse=tf.AUTO_REUSE):
        W =tf.get_variable(
            name="W",
            shape=[2 *num_hidden_units, 1],
            dtype=dtype,
            initializer=tf.truncated_normal_initializer(mean=0.0, stddev=0.1, dtype=dtype), # adjust
            regularizer=l2_regularizer(scale=reg)
        )
        b =tf.get_variable(
            name="bias",
            shape=[1],
            dtype=dtype,
            initializer=tf.zeros_initializer(dtype=dtype)
        )

    with tf.variable_scope(name_or_scope="inference", reuse=tf.AUTO_REUSE):
        x =tf.reshape(tensor=features, shape=[-1, num_steps, 1])
        x =tf.unstack(value=x, num=num_steps, axis=1)
        lstm_cell_fw =tf.nn.rnn_cell.LSTMCell(num_units=num_hidden_units, use_peepholes=False,
```

```
                                                    forget_bias=1.0,
                                                    initializer=tf.orthogonal_initializer(dtype=dtype))
    lstm_cell_bw =tf.nn.rnn_cell.LSTMCell(num_units=num_hidden_units, use_peepholes=False,
                                                    forget_bias=1.0,
                                                    initializer=tf.orthogonal_initializer(dtype=dtype))

    outputs, states_fw, states_bw =tf.nn.static_bidirectional_rnn(cell_fw=lstm_cell_fw,
                                                    cell_bw=lstm_cell_bw, inputs=x, dtype=dtype)
    # output is a list of tensor with length of num_steps, each element of this list is a tensor in shape
                                                    of (None, 2 * num_hidden_units)
    # states_fw & states_bw are tuple of tensor with length of 2, each element of this tuple is a tensor in
                                                    shape of (None, num_hidden_units)

    predictions =tf.matmul(a=outputs[-1], b=W) +b
    predictions =tf.reshape(tensor=predictions, shape=[-1]) # a tensor in shape of (None, 1)

pred ={"predictions": predictions}

# ----------Provide an estimator spec for `ModeKeys.PREDICTION` mode----------
if mode ==tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=pred)

with tf.variable_scope(name_or_scope="loss", reuse=tf.AUTO_REUSE):
    loss =tf.losses.mean_squared_error(labels=labels, predictions=predictions)
    regloss =tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    ) # A scaler representing the regularization loss
    loss +=regloss

with tf.variable_scope(name_or_scope="optimizer", reuse=tf.AUTO_REUSE):
    if (optimizer.lower() =="sgd"):
        opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() =="momentum"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif (optimizer.lower() =="nesterov"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif (optimizer.lower() =="adagrad"):
        opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif (optimizer.lower() =="adadelta"):
        opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif (optimizer.lower() =="rmsprop"):
        opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif (optimizer.lower() =="ftrl"):
        opt =tf.train.FtrlOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() =="adam"):
        opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:
        raise ValueError("Argument <optimizer> of model_fn is invalid")

    train_op =opt.minimize(loss=loss, global_step=tf.train.get_global_step())
```

```
    # ----------Provide an estimator spec for `ModeKeys.TRAIN` mode----------
    if (mode ==tf.estimator.ModeKeys.TRAIN):
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

    # ----------Build evaluation metrics----------
    with tf.variable_scope(name_or_scope="evaluation", reuse=tf.AUTO_REUSE):
        eval_metric_ops ={
            "rmse": tf.metrics.root_mean_squared_error(labels=labels, predictions=predictions)
        }

    # ----------Provide an estimator spec for `ModeKeys.EVAL` mode----------
```

17.6 Multi Layers Recurrent Neural Networks

17.6.1 Derivation of forward-propagation in Multi-layer RNNs

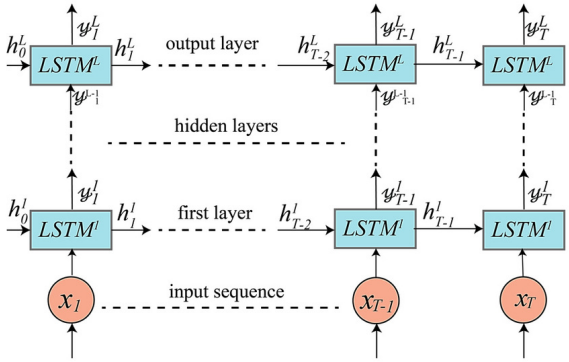


图 70: The structure of multi LSTMs

17.6.2 Model implement

```
# -*- coding: utf-8 -*-
"""
Created on 2018/12/19 01:01
author: Tong Jia
email: cecilio.jia@gmail.com
software: PyCharm
Description:
    An implement of multi-layer LSTMs for single-timestep-output regression task.
"""
import tensorflow as tf
from tensorflow.contrib.layers import l2_regularizer
from tensorflow.python import rnn
from six.moves import xrange

def input_fn(
        filenames,
        delimiter,
        epochs,
        batch_size,
        dtype,
        shuffle=True,
        num_parallel_calls=10,
        buffer_size_prefetch=100000,
        buffer_size_shuffle=2048):
    def map_func(line):
        columns =tf.string_split(source=[line], delimiter=delimiter, skip_empty=False)
        y =tf.string_to_number(string_tensor=columns.values[0], out_type=dtype)
        x =tf.string_to_number(string_tensor=columns.values[1: ], out_type=dtype)
        return x, y
```

```
    dataset =tf.data.TextLineDataset(filenames=filenames). \
        map(map_func=map_func, num_parallel_calls=num_parallel_calls). \
        prefetch(buffer_size=buffer_size_prefetch)
    if shuffle ==True:
        dataset =dataset.shuffle(buffer_size=buffer_size_shuffle)
    dataset =dataset. \
        repeat(count=epochs). \
        batch(batch_size=batch_size, drop_remainder=False)
    iterator =dataset.make_one_shot_iterator()
    features, labels =iterator.get_next()
    return features, labels

def model_fn(features, labels, mode, params):
    # ----------Define Hyper-parameters----------
    num_steps =params["num_steps"]
    num_hidden_units =params["num_hidden_units"]
    num_hidden_layers =params["num_hidden_layers"]
    dropout_rate =params["dropout_rate"]
    reg =params["reg"]
    optimizer =params["optimizer"]
    learning_rate =params["learning_rate"]
    dtype =params["dtype"]

    # ----------Define a function to create a basic rnn cell for multi-layers rnn----------
    def rnn_cell(mode):
        cell =tf.nn.rnn_cell.LSTMCell(
            num_units=num_hidden_units,
            use_peepholes=False,
            initializer=tf.orthogonal_initializer(dtype=dtype),
            forget_bias=1.0,
            state_is_tuple=True
        )
        if mode ==tf.estimator.ModeKeys.TRAIN:
            cell =tf.nn.rnn_cell.DropoutWrapper(
                cell=cell,
                input_keep_prob=1.0,
                output_keep_prob=1 -dropout_rate,
                state_keep_prob=1.0,
                dtype=dtype
            )
        return cell

    # ----------Build inference----------
    with tf.variable_scope(name_or_scope="inference", reuse=tf.AUTO_REUSE):
        x =tf.reshape(tensor=features, shape=[-1, num_steps, 1]) # A tensor in shape of (None, num_steps,
                                                                 num_inputs)
        batch_size =tf.shape(input=x)[0]
        cells =[rnn_cell(mode=mode) for _ in xrange(num_hidden_layers)] # Note the basic element must be
                                                                 created by a function
        multi_cells =tf.nn.rnn_cell.MultiRNNCell(cells=cells, state_is_tuple=True)
        initial_state =multi_cells.zero_state(batch_size=batch_size, dtype=dtype)
```

```
    outputs, states =rnn.dynamic_rnn(cell=multi_cells, inputs=x, initial_state=initial_state,
                                                        time_major=False)
    outputs =tf.transpose(a=outputs, perm=[1, 0, 2]) # output is a tensor in shape of (num_steps, None,
                                                        num_hidden_units)

    outputs =outputs[-1]

    predictions =tf.layers.dense(
        inputs=outputs,
        units=1,
        activation=tf.identity,
        use_bias=True,
        kernel_initializer=tf.truncated_normal_initializer(mean=0.0, stddev=1.0, dtype=dtype),
        bias_initializer=tf.zeros_initializer(dtype=dtype),
        kernel_regularizer=l2_regularizer(scale=reg)
    )
    predictions =tf.reshape(tensor=predictions, shape=[-1]) # A tensor inshape of (None, )
    pred ={"predictions": predictions}

# ----------Provide an estimator spec for `ModeKeys.PREDICTION` mode----------
if mode ==tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode, predictions=pred)

# ----------Build loss----------
with tf.variable_scope(name_or_scope="loss", reuse=tf.AUTO_REUSE):
    loss =tf.losses.mean_squared_error(labels=labels, predictions=predictions)
    regloss =tf.reduce_sum(
        input_tensor=tf.get_collection(key=tf.GraphKeys.REGULARIZATION_LOSSES),
        axis=0,
        keepdims=False
    ) # A scaler representing the regularization loss
    loss +=regloss

# ----------Build optimizer----------
with tf.variable_scope(name_or_scope="optimizer", reuse=tf.AUTO_REUSE):
    if (optimizer.lower() =="sgd"):
        opt =tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() =="momentum"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=False)
    elif (optimizer.lower() =="nesterov"):
        opt =tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9, use_nesterov=True)
    elif (optimizer.lower() =="adagrad"):
        opt =tf.train.AdagradOptimizer(learning_rate=learning_rate, initial_accumulator_value=0.1)
    elif (optimizer.lower() =="adadelta"):
        opt =tf.train.AdadeltaOptimizer(learning_rate=learning_rate, rho=0.95, epsilon=1e-8)
    elif (optimizer.lower() =="rmsprop"):
        opt =tf.train.RMSPropOptimizer(learning_rate=learning_rate, decay=0.9, epsilon=1e-10)
    elif (optimizer.lower() =="ftrl"):
        opt =tf.train.FtrlOptimizer(learning_rate=learning_rate)
    elif (optimizer.lower() =="adam"):
        opt =tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8)
    else:
```

```
        raise ValueError("Argument <optimizer> of model_fn is invalid")

    train_op =opt.minimize(loss=loss, global_step=tf.train.get_global_step())

# ----------Provide an estimator spec for `ModeKeys.TRAIN` mode----------
if (mode ==tf.estimator.ModeKeys.TRAIN):
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

# ----------Build evaluation metrics----------
with tf.variable_scope(name_or_scope="evaluation", reuse=tf.AUTO_REUSE):
    eval_metric_ops ={
        "rmse": tf.metrics.root_mean_squared_error(labels=labels, predictions=predictions)
    }

# ----------Provide an estimator spec for `ModeKeys.EVAL` mode----------
```

17.7 Bi-LSTM with Multi Layers LSTMs

17.7.1 Forward propagation

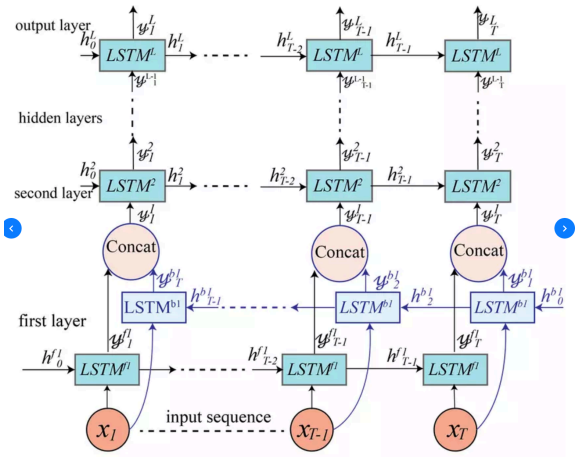


图 71: The stack of BiLSTM with multi LSTMs

18 Sequence to Sequence Learning (Seq2Seq)

18.1 Transformer（未完成）

18.1.1 Introduction

在 NLP 任务中，序列编码是个重要的事情，以 NLP 为例，一般做法是：

1. 句子分词
2. 词 one-hot 编码
3. 转换为词向量

因此每个句子都变成一个矩阵 $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ 其中：

- n: 句子长度
- d: 词向量维度

现在要做的就是编码 $\boldsymbol{X}$，并从中抽取信息。通用建模方法 (seq2seq 中的 encoder 或 decoder) 包括三类：

- RNN[**?**]
  - 序列计算限制了模型的并行性
  - 即便是有了 LSTMs 和 GRU，RNN 也是通过注意力机制（attention mechanism）来解决长距离依赖的问题
  - 那么一个假设，如果 attention 机制能够帮助我们获取任何的状态（state），或许我们就不再需要 RNN
- CNN[**?**]
- Pure Attention (e.g. Transformer[**?**])

18.1.2 Model formulation

Transformer[?] 的结构如图所示，包括了 Encoder（左边）和 Decoder（右边）两部分 以下将从宏观到

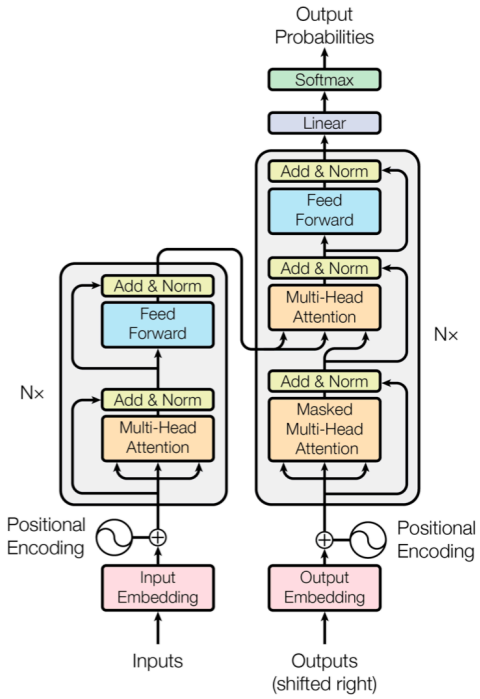


图 72: Architecture of Transformer

微观逐一介绍 Transformer 模型的细节。

A High-Level Look

1. 首先可将模型看作一个黑盒子。譬如在机器翻译任务场景中，模型使用一种语言的句子作为输入，并将其翻译为另一种语言作为输出。

图 73: Transformer model for machine translation task as a black box

2. 稍微深入一些，将中间黑盒子拆开，其包括了 Encoders 和 Decoders 两部分结构。

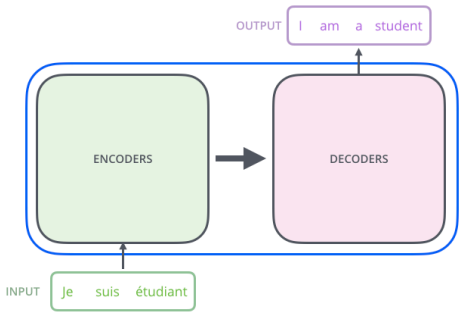


图 74: Unpack the black box as both encoders component and decoders component

3. 再深入一些，Encoders 部分为 N 个**完全相同**基本 encoder 单元的堆叠，Decoders 部分也是 N 个**完全相同**基本 decoder 单元的堆叠（论文中设置 $N = 6$）。

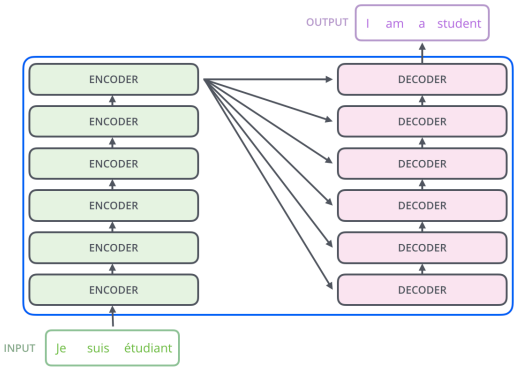


图 75: The encoding component is a stack of encoders, and the decoding component is a stack of decoders with the same number.

4. 进一步深入到基本 encoder 单元结构内部，任意一个基本 encoder 单元由下向上包含两层：Self-Attention 子层和 Feed Forward Neural Network 子层：

   - **Self-Attention Layer**: encoder 的输入首先进入 self-attention layer，在编码一个特定单词时，self-attention layer 帮助 encoder**为每一个编码单词“注意”其他单词**。
   - **Feed Forward Neural Network Layer**: 获取 self-attention layer 的输出，进行进一步的特征提取。

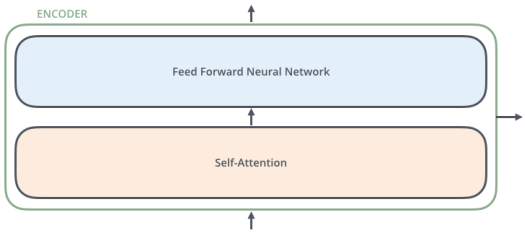


图 76: Basic encoder unit contains two layers: self-attention layer & feed forward nerual netowrk layer

5. 深入到基本 decoder 单元结构内部，稍微不同于基本 encoder 单元，其多了一个 Encoder-Decoder-Attention 层，即由下向上包含了三层：

   - **Self-Attention Layer**
   - **Encoder-Decoder Attention Layer**
   - **Feed Forward Neural Network Layer**

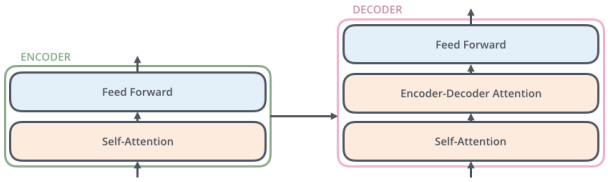


图 77: Basic decoder unit contains three layers: self-attention layer, encoder-decoder attention layer & feed forward nerual netowrk layer

到此为止，已经介绍了 Transformer 模型所有的组成成分，下边将从宏观角度切换到深入细节，跟踪查看 tensors 从模型输入到输出的全过程。

Bringing The Tensors Into The Picture

- 一般 NLP 任务，我们首先需要利用 embedding lookup 将单词转换为向量。

图 78: Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

- 所有 encoder 的通用抽象输入模式：接收一系列长度为 512 的词向量，其中：
  - 最底层的 encoder：接收一系列长度为 512 的 word embeddings

– 其他 encoders：接收一系列上一层 encoder 输出的，长度为 512 的向量

– **注意：** 这些“一系列向量”的个数为超参数，我们可以手动设置。一般情况下，该长度为**训练数据中最长句子的长度**。

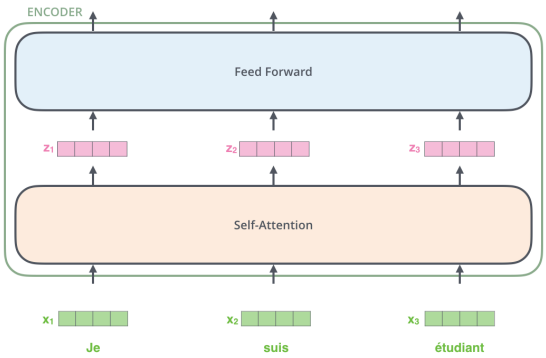


图 79: A list of word embedding flow through two layers of a specific encoder

- Transformer 的一个关键性质：每一个位置的单词（词向量）在 encoder 中其自己独立的路径中流动。因此在 Self-Attention 层中的操作可以并行执行（即可以利用矩阵运算）。

Now Encoding! Tensors 在相邻 encoders 之间的流动

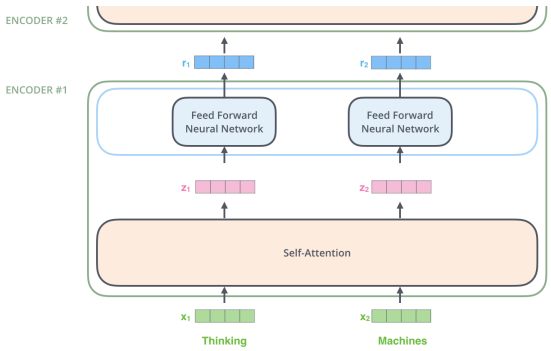


图 80: An encoder receives a list of vectors as input. It processes this list by passing these vectors into a self-attention layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder.

Self-Attention at a High Level

- 考虑以下句子，现在需要对其进行翻译："The animal didn't cross the street because it was too tired." 那么句子中的“it”指代 street 还是 animal？这个问题对人类来说很简单，但是对算法而言是有难度的
- 当 Transformer 模型处理单词“it”的时候，self-attention 将关联“it”和“animal”

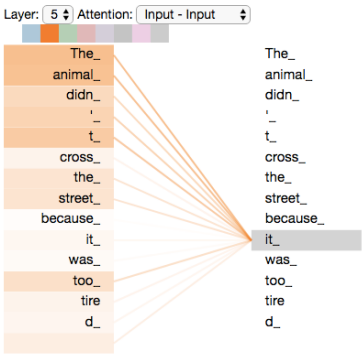


图 81: Self-Attention allows to associate“it”with“animal”

- 当模型处理每一个单词（输入序列（input sequence）中的每个位置（position））时，self-attention 允许它查看输入序列中的其他位置，以寻找可以帮助更好地编码该单词的线索

Self-Attention in Detail 如下先考虑如何利用向量（vectors）计算 self-attention，然后再看实际是如何利用矩阵（matrices）进行计算的。

Self-attention by vector calculation

1. 为 encoder 的每个输入向量 $\mathbf{X}_j$, $j \in \{1, ..., D\}$（以 $\mathbf{X}_1$ 为例）创建三个向量：
   - Query vector: $\mathbf{q}_1 = \mathbf{X}_1\mathbf{W}^Q$
   - Key vector: $\mathbf{k}_1 = \mathbf{X}_1\mathbf{W}^K$
   - Value vector: $\mathbf{v}_1 = \mathbf{X}_1\mathbf{W}^V$

   其中 **Q** 对应中心词，**K** 对应上下文词，**V** 对应上下文词。

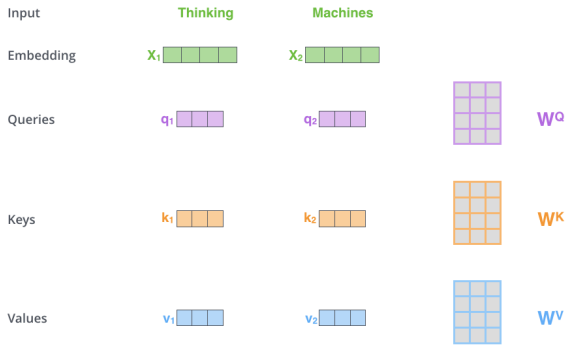


图 82: Create a query vector $\mathbf{q}_j$, a key vector $\mathbf{k}_j$ and a value vector $\mathbf{v}_j$ for each input vector $\mathbf{X}_j$ using matrix multiplication

2. 为每一个位置（position）的词 $\mathbf{X}_j$ 计算 self-attention。则实际为计算该词对应的 query vector $\mathbf{q}_j$ 与**句子中所有 key vectors**的内积：

$$\text{Score}(j, j') = \mathbf{q}_j \cdot \mathbf{k}_{j'}, \quad j' \in \{1, ..., D\}$$

注意：

- 内积的计算是为了描述**中心词特征向量**表示 $\mathbf{q}_j$ 与所有**上下文词特征向量**(包含自己)表示 $\{\mathbf{k}_1, ..., \mathbf{k}_D\}$ 的相关性。如图可知：Thinking 这个词相对于 Machines 词，其和自身的相关性更大
- $\mathbf{k}_{j'}$ 的选取是在 sentence 内的，而不是整个 vocabulary 内

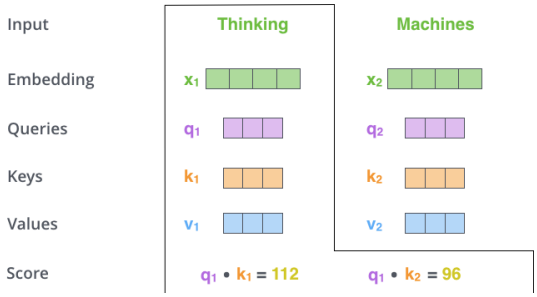


图 83: Calculate self-attention for a specifc word in a sentence

3. $\text{Score}(j, j') \Leftarrow \frac{\text{Score}(j,j')}{\sqrt{d_k}}$, $j' \in \{1, ..., D\}$, d_k 为 key vector 的维度.

4. Softmax operation. 作用是使得 Softmax-score$(j, j') \in (0, 1)$, $j' \in \{1, ..., D\}$. 该 Softmax 分数 Softmax-score(j, j') 表达了每个上下文位置 j' 的单词 $X_{j'}$ 在该中心词位置 j 的**表达程度**

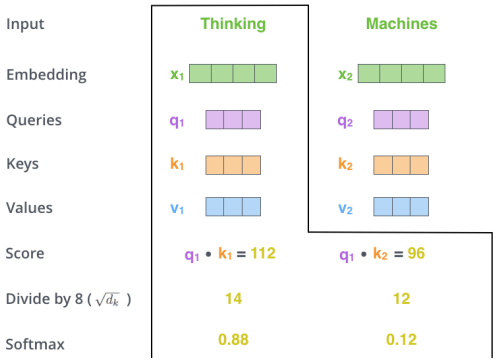


图 84: Softmax score Score(j, j') determines how much each word $X_{j'}$ will be expressed at this position j

5. $\boldsymbol{v}_{j'} \Leftarrow$ Softmax-score$(j, j') \times \boldsymbol{v}_{j'}$, $j' \in \{1, ..., D\}$. 这里的直觉是:

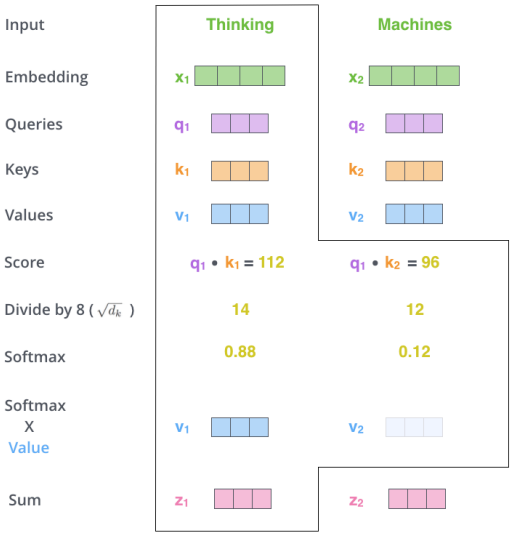


图 85: Multiply each value vector of position j' by the softmax score Softmax-score(j, j') when processing the central word in position j

- 保持想要关注单词的值的完整性
- 忽略不相关的词（e.g. 对该上下文词的 value vector 乘以一个很小的 softmax value 譬如 0.001）

6. Sum up the weighted value vectors: $\boldsymbol{z}_j = \sum_{j'=1}^{D} \boldsymbol{v}_{j'}$. 其中 $\boldsymbol{z}_j$ 表示**句子中第 j 个中心词位置，对所有上下文位置 j' 信息容纳表达后的抽象表示**。

Self-attention by matrix calculation 上述过程是以向量运算的形式描述 self-attention layer 对一个中心词 position j 的计算过程。但是在实际执行过程中为了高效快速（同时执行多个中心词 position 的 self-attention 操作），这些操作是以矩阵运算的方式执行的，如下将以矩阵运算的方式描述上述过程。

1. Calculate the Query, Key, and Value matrices

   - $\because \mathbf{X} \in \mathbb{R}^{D \times E}$ (E is the embedding size (e.g. E = 512)), 注意 $\mathbf{X}$ 矩阵中的每一行对应 input sentence 中的一个词
   - $\because \mathbf{W^Q}, \mathbf{W^K}, \mathbf{W^V} \in \mathbb{R}^{E \times H}$ (H is the dimension of query vectors, key vectors and value vectors (e.g. H = 64))
   - $\therefore \mathbf{Q} = \mathbf{X}\mathbf{W^Q} \in \mathbb{R}^{D \times H}$, $\mathbf{K} = \mathbf{X}\mathbf{W^K} \in \mathbb{R}^{D \times H}$, $\mathbf{V} = \mathbf{X}\mathbf{W^V} \in \mathbb{R}^{D \times H}$

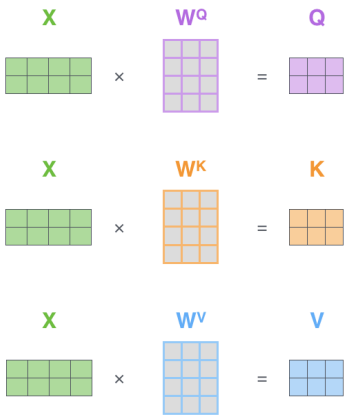


图 86: Calculate $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ in matrix format

2. Calculate the outputs of the self-attention layer. (由于是矩阵运算，所以可以用一步替代向量运算中的 2-6 步)

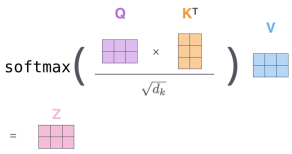


图 87: Calculate the outputs of the self-attention layer

The Beast With Many Heads

- Multi-head attention: 多个 self-attention 组成（论文中总计 8 个）。这样的做法类似与 CNN 中设置多个卷积核，因此可以提供多个表示子空间（multiple representation subspaces），从而提高模型的表达能力。

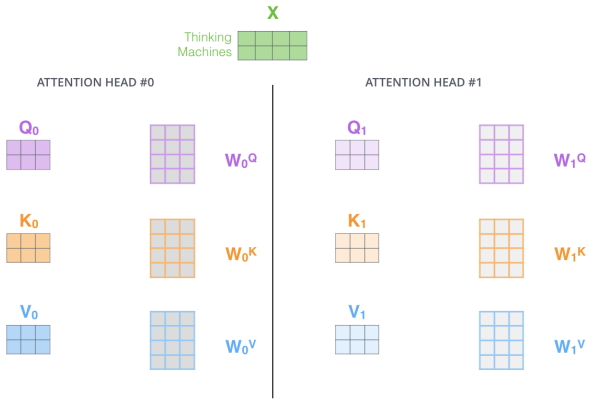


图 88: Multiple $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ correspond to multiple representation subspaces

- 分别计算每个 Attention head：

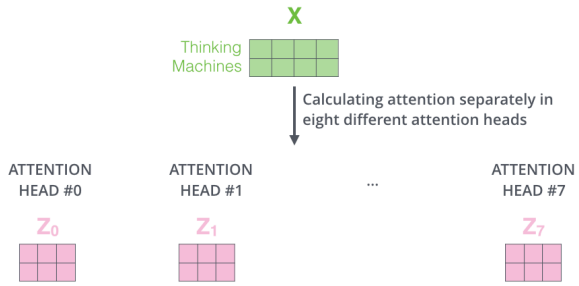


图 89: Calculating each self-attention head separately

- Multi-head attention layer 最后的计算流程如下：

  1. Concatenate all attention heads, get a output matrix $[\mathbf{Z_0}, \mathbf{Z_1}, ..., \mathbf{Z_{n-1}}] \in \mathbb{R}^{D\times(N\times H)}$, N is the number of self-attention
  2. Multiply with a weight matrix $\mathbf{W}^O \in \mathbb{R}^{(N\times H)\times E}$
  3. get the final output of multi-head attention layer: $\mathbf{Z} \in \mathbb{R}^{D\times E}$

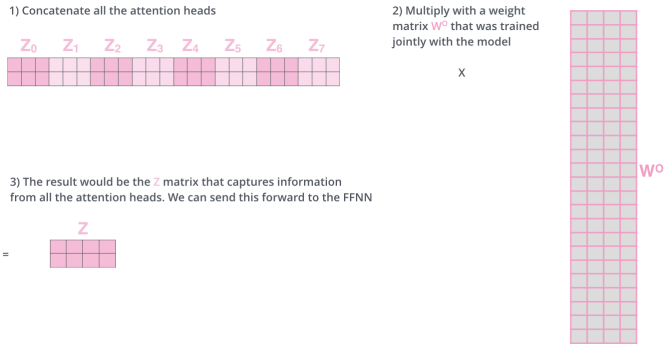


图 90: Concatenate all attention heads, and multiply with a weight matrix $\mathbf{W}^o \in \mathbb{R}^{(n\times E)\times D}$

- Recap of Transformer multi-headed self-attentions:

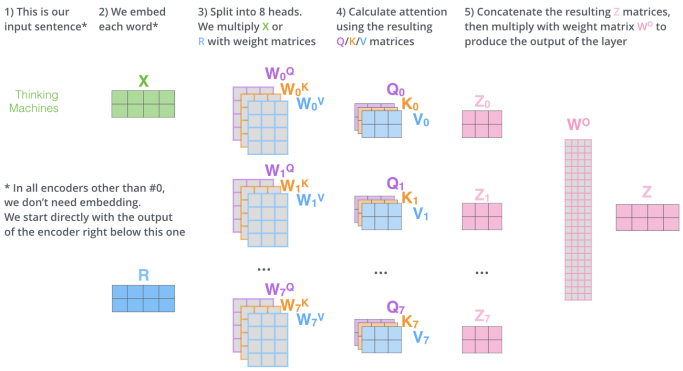


图 91: Transformer multi-headed self-attention recap

Representing The Order of The Sequence Using Positional Encoding 迄今依旧没有考虑的一个问题：输入序列中词的位置信息

18.1.3 References

- The Illustrated Transformer
- The Annotated Transformer
- Blog:The Transformer - Attention is all you need
- Blog of VARUNA JAYASIRI: Transformer in Tensorflow - Attention Is All You Need
- Mac Brennan: Neural Translation Model

18.2 Stabilizing Transformers for Reinforcement Learning（未完成）

19 Generative Adversarial Networks (GANs)

19.1 Vanilla GAN

19.1.1 Introduction

本文选自Generative Adversarial Nets[?]，生成对抗网络 (GANs) 的框架如图所示：

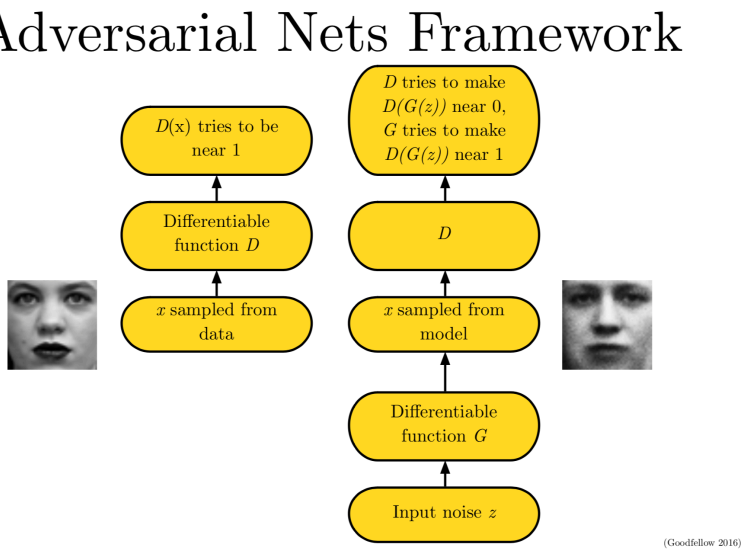


图 92: Adversarial Nets Framework[?]

- $\boldsymbol{z}$: Input noise variables
- $\boldsymbol{x} \in \mathbb{R}^d$: Input data, a high-dimensional vector
- $p_{\boldsymbol{z}}(\boldsymbol{z})$: The defined prior distribution on input noise variables
- $\boldsymbol{\theta_g}$: Training parameters of generator G
- $\boldsymbol{\theta_d}$: Training parameters of discriminator D
- $G(\boldsymbol{z}; \boldsymbol{\theta_g}) \in \mathbb{R}^d$: Data space, a high-dimensional vector (e.g. an image, a sequence of words)
- $D(\boldsymbol{x}; \boldsymbol{\theta_d}) \in (0, 1)$: An single scalar output represents the probability that $\boldsymbol{x}$ came from the real data rather than generator G (formulated by sigmoid function)

19.1.2 Model formulation

Build objective function 优化任务：

- Discriminator D: Maximize the probability of assigning the correct label to both training examples and samples from generator G（即：最大化真实数据被 Discriminator 判别为真的概率，同时生成数据被判别为假的概率）

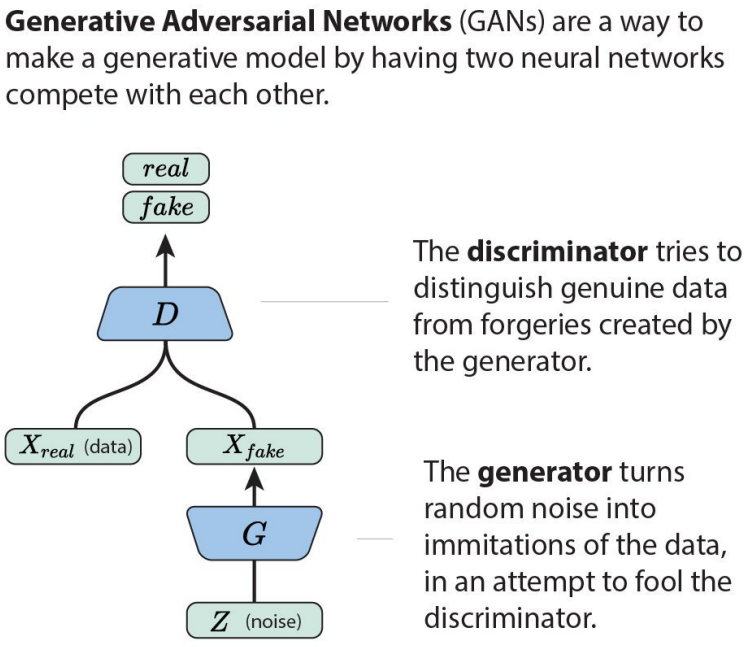
$$\begin{aligned} \max_D \quad & \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[1 - D(G(\boldsymbol{z}))] \\ \Rightarrow \max_D \quad & \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \end{aligned}$$

- Generator G: Maximize the probability that the Discriminator D discriminates the generated data $G(\boldsymbol{z})$ as real data（即：最大化生成数据被判别为真的概率）

$$\begin{aligned} \max_G \quad & \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))] \\ \Rightarrow \min_G \quad & \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[1 - D(G(\boldsymbol{z}))] \\ \Rightarrow \min_G \quad & \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \end{aligned}$$

因此同时训练 Discriminator 和 Generator 的目标函数为：

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$



Optimization 以 SGD 优化算法为例，模型的优化过程如下所示：

Algorithm 21: Minibatch stochastic gradient descent training of generative adversarial nets[**?**]

Input: The number of steps k to apply to the discriminator;
Initial training parameters of discriminator $\boldsymbol{\theta_d}$;
Initial training parameters of generator $\boldsymbol{\theta_g}$;
Global learning rate of discriminator η_d;
Global learning rate of generator η_g.

for *number of training iterations* **do**
  for k *steps* **do**
    Sample minibatch of m noise samples $\{\boldsymbol{z^{(1)}}, \cdots, \boldsymbol{z^{(m)}}\}$ from noise prior $p_g(\boldsymbol{z})$
    Sample minibatch of m examples $\{\boldsymbol{x^{(1)}}, \cdots, \boldsymbol{x^{(m)}}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$
    Update the discriminator by ascending its stochastic gradient:

$$\boldsymbol{g_d} = -\nabla_{\boldsymbol{\theta_d}} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\boldsymbol{x^{(i)}}) + \log \Big(1 - D\big(G(\boldsymbol{z^{(i)}})\big)\Big) \right]$$

$$\boldsymbol{\theta_d} \leftarrow \boldsymbol{\theta_d} - \eta_d \cdot \boldsymbol{g_d}$$

  end
  Sample minibatch of m noise samples $\{\boldsymbol{z^{(1)}}, \cdots, \boldsymbol{z^{(m)}}\}$ from noise prior $p_g(\boldsymbol{z})$
  Update the generator by descending its stochastic gradient:

$$\boldsymbol{g_g} = \nabla_{\boldsymbol{\theta_g}} \frac{1}{m} \sum_{i=1}^{m} \left[\log \Big(1 - D\big(G(\boldsymbol{z^{(i)}})\big)\Big) \right]$$

$$\boldsymbol{\theta_g} \leftarrow \boldsymbol{\theta_g} - \eta_g \cdot \boldsymbol{g_g}$$

end

Note:

- 注意在任意 iteration 中是先训练 discriminator，再训练 generator
- 因为只有尽快提高 discriminator（警察）的鉴别能力，才能带动提高 generator（造假者）的造假水平，因此在任意一个 iteration 中，训练 discriminator 多次，而只训练 generator 一次
- 在任意一个 iteration 中，如果只训练 discriminator 一次，而训练 generator 多次，那实际上 discriminator 的鉴别水平不会有太大提高，这时训练 generator 再多次，顶破天也只是达到抗衡当前 discriminator 的水平，所以这时训练 generator 再多次也无意义

Note about GANs

- **GANs 系列模型的适用场景：学习给定空间样本分布 $\mathcal{Z}$ 到目标空间样本分布 $\mathcal{X}$ 之间的映射函数，即实现了从 $z \in \mathcal{Z}$ 到 $x \in \mathcal{X}$ 的样本转换**
- 在 GAN 的训练中，生成器 Generator 被鼓励产生与真实数据分布 $p_r(\boldsymbol{x})$ 相似的生成数据分布 $p_g(\boldsymbol{x})$，从而达到产生“以假乱真”生成数据的目的

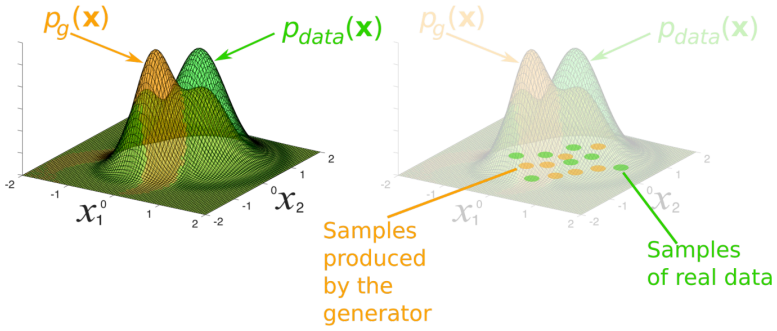


图 93: During GAN training, the generator is encouraged to produce a distribution of samples, $p_g(\boldsymbol{x})$ to match that of real data, $p_{\text{data}}(\boldsymbol{x})$. For an appropriately parametrized and trained GAN, these distributions will be nearly identical.[?]

19.2 DCGAN（未完成）

19.2.1 Introduction

本文选自Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks[?]

19.2.2 Model formulation

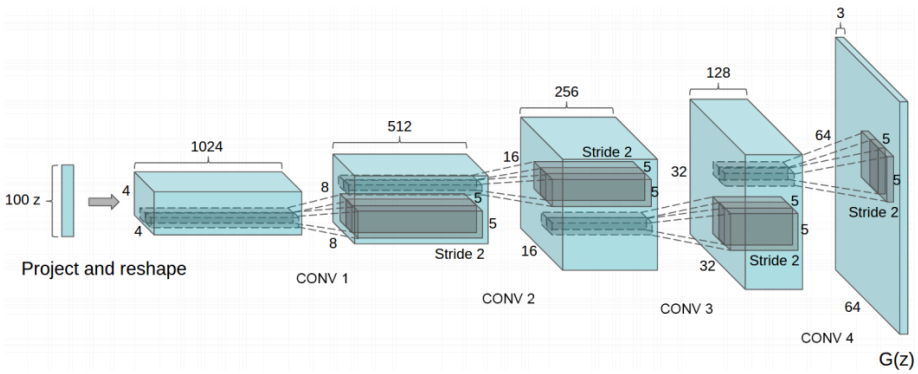


图 94: DCGAN Generator

deconvolution

19.3 Mode Seeking GANs (MSGANs)

19.3.1 Introduction

本文选自Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis[?]，用于解决cGAN 中存在的**mode collapse**的问题。同期与之类似的工作还有Diversitysensitive conditional generative adversarial networks[?]

Mode collapse issue for cGANs 原始一系列基于cGAN[?] 的模型存在一个普遍且严重的问题：mode collapse.

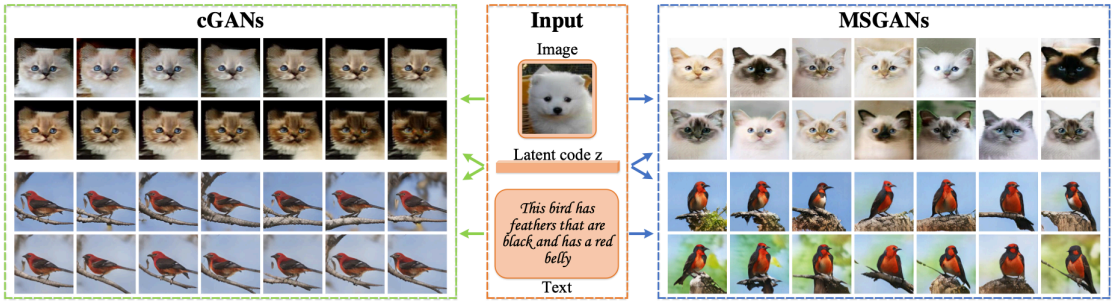


图 95: Mode seeking generative adversarial networks (MSGANs). (Left) Existing conditional generative adversarial networks tend to ignore the input latent code z and generate images of similar modes. (Right) We propose a simple yet effective mode seeking regularization term that can be applied to arbitrary conditional generative adversarial networks in different tasks to alleviate the mode collapse issue and improve the diversity.

原始 cGAN 的目标函数和结构示意图如下：

$$\min_G \max_D V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\big[\log D(\boldsymbol{x}|\boldsymbol{y})\big] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\big[\log(1-D(G(\boldsymbol{z}|\boldsymbol{y})))\big]$$

其中：

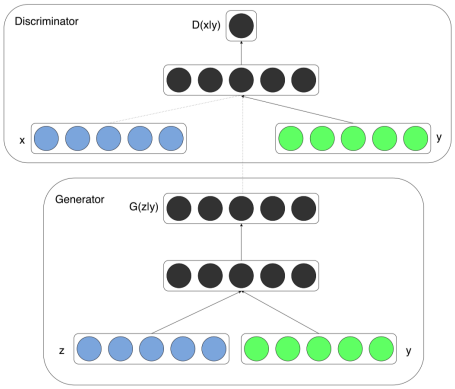


图 96: Conditional adversarial net

- $\boldsymbol{y}$，一般称其为 conditional contexts，其可以包含任何形式的辅助信息（auxiliary information），譬如：类别标签和其他形式数据
- 可在 discriminator 和 generator 中将 $\boldsymbol{y}$ 作为附加输入层（additional input layer）以达到执行条件（perform the conditioning）的效果。即：**先验条件就是附加输入层**

尽管 cGANs 已在各种应用中取得了成功，但是现有方法仍遭存在 mode collapse 的问题。这是因为：

1. 条件向量 $\boldsymbol{y}$ 为输出数据（一般是图片）提供了极强的结构先验信息（structural prior information）
2. 条件向量 $\boldsymbol{y}$ 相对于随机噪声向量 $\boldsymbol{z}$，拥有更高的特征向量维度

因此导致 generator 更倾向于忽略随机噪声向量 $\boldsymbol{z}$，而随机噪声向量 $\boldsymbol{z}$ 正是造就生成图像多样性的原因。因此 generator 忽略随机噪声向量 $\boldsymbol{z}$ 带来的结果就是 generator 只产生相似的图片。

Contribution 因此本文的贡献是：提出了一种 mode seeking regularization 的方法用于缓和（alleviate，并非完全解决）cGANs 中存在的 mode collapse 问题，同时该方法是一种通用的方法，而不必像之前的一些做法需要根据特定的任务、特定的数据去修改网络结构（modifications of the network structure）。

19.3.2 Model formulation

Illustration of motivation 本文设计 mode seeking regularization 的动机是：

- 从模式分布（直观理解）的角度看，虽然真实数据中存在很多的 modes，但是当 mode collapse 问题发生时，生成器只会产生很少种类的 modes.
- 从数据分布(微观分析)的角度看，分别计算输入噪声空间 $\mathcal{Z}$ 内两个点 $\boldsymbol{z}_{\mathrm{a}}$ 和 $\boldsymbol{z}_{\mathrm{b}}$ 之间的距离 $d_{\mathcal{Z}}(\boldsymbol{z}_{\mathrm{a}}, \boldsymbol{z}_{\mathrm{b}})$，以及输出样本空间 $\mathcal{I}$ 内两个点 $\boldsymbol{I}_{\mathrm{a}}$ 和 $\boldsymbol{I}_{\mathrm{b}}$ 之间的距离 $d_{\mathcal{I}}(\boldsymbol{I}_{\mathrm{a}}, \boldsymbol{I}_{\mathrm{b}})$，结果通过数值计算发现，当 mode collapse 问题发生时，随着原始空间 $\mathcal{Z}$ 内两个点之间距离的缩短，目标空间 $\mathcal{I}$ 内距离加倍（不成比例）地缩短。换言之，当 $d_{\mathcal{Z}}(\boldsymbol{z}_{\mathrm{a}}, \boldsymbol{z}_{\mathrm{b}})$ 很大时，$d_{\mathcal{I}}(\boldsymbol{I}_{\mathrm{a}}, \boldsymbol{I}_{\mathrm{b}})$ 会很小；而当 $d_{\mathcal{Z}}(\boldsymbol{z}_{\mathrm{a}}, \boldsymbol{z}_{\mathrm{b}})$ 很小时，$d_{\mathcal{I}}(\boldsymbol{I}_{\mathrm{a}}, \boldsymbol{I}_{\mathrm{b}})$ 会等于 0，因此造成了 mode collapse 的问题。

注意：由上述微观分析可知 L2 距离更容易导致 mode collapse 的问题，因为平方实际上是对差异的放缩。

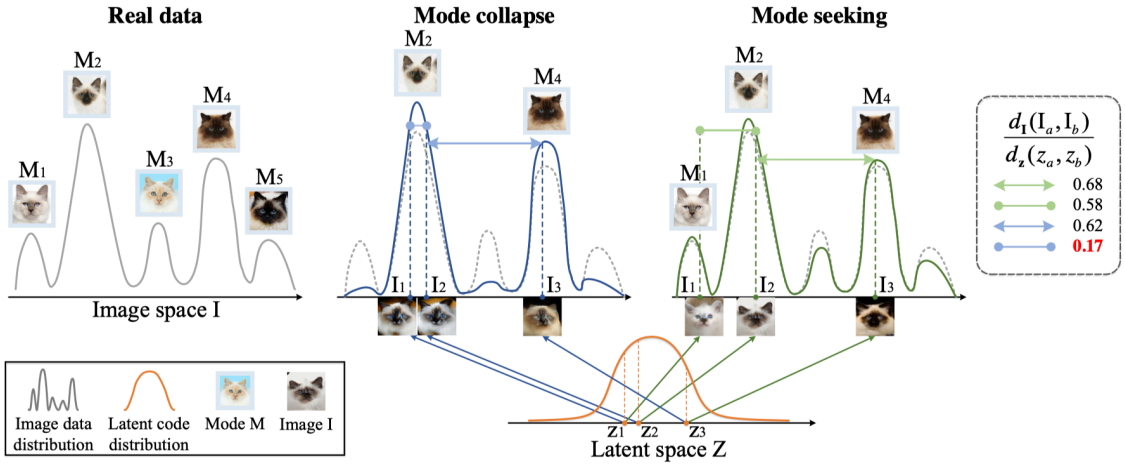


图 97: Real data distribution contains numerous modes. However, when mode collapse occurs, generators only produce samples from a few modes. From the data distribution when mode collapse occurs, we observe that for latent vectors $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$, the distance between their mapped images $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$ will become shorter in a disproportionate rate when the distance between two latent vectors is decreasing. We present on the right the ratio of the distance between images with respect to the distance of the corresponding latent vectors, where we can spot an anomalous case (colored in red) where mode collapse occurs. The observation motivates us to leverage the ratio as the training objective explicitly.

因此也就是说两个空间中的距离比（ratio of the distance）：$\frac{d_{\mathcal{I}}(\boldsymbol{I}_{\mathrm{a}}, \boldsymbol{I}_{\mathrm{b}})}{d_{\mathcal{Z}}(\boldsymbol{z}_{\mathrm{a}}, \boldsymbol{z}_{\mathrm{b}})}$ 很小。

Mode seeking regularization 因此针对 Generator 的目标函数中，需要添加如下正则化项，用于最大化 ratio of the distance，尽量缓解 mode collapse 的问题：

$$\max_{G} \mathcal{V}_{\mathrm{ms}}(\boldsymbol{z_1}, \boldsymbol{z_2}) = \frac{d_{\mathcal{I}}(G(\boldsymbol{c}, \boldsymbol{z_1}), G(\boldsymbol{c}, \boldsymbol{z_2}))}{d_{\mathcal{Z}}(\boldsymbol{z_1}, \boldsymbol{z_2})}$$
$$\Rightarrow \min_{G} \mathcal{L}_{\mathrm{ms}}(\boldsymbol{z_1}, \boldsymbol{z_2}) = -\mathcal{V}_{\mathrm{ms}}(\boldsymbol{z_1}, \boldsymbol{z_2})$$

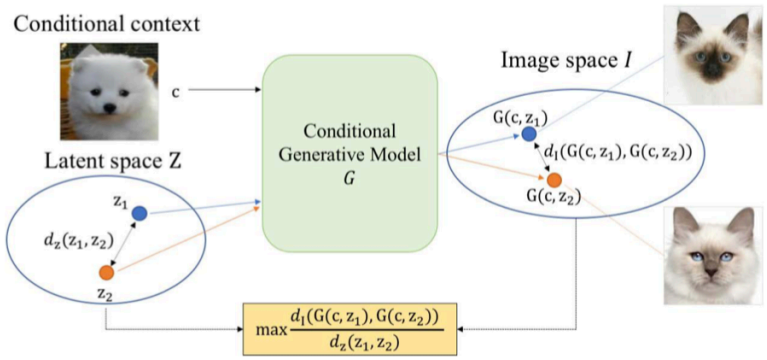
其中：

- $d_*(\cdot)$: distance metric
- $\boldsymbol{c}$: conditional contexts input

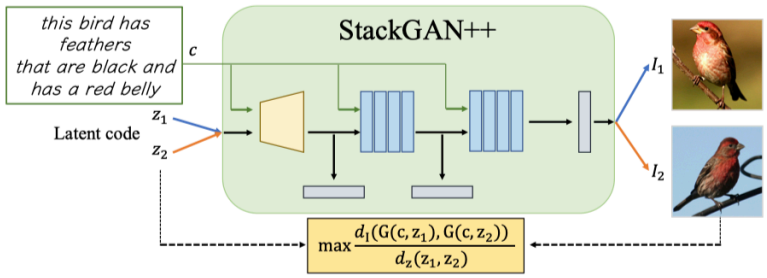
有了此正则化项之后，就增大了 $\mathcal{I}$ 空间内样本的“活动范围”，会有助于生成更多的 mode。譬如图97的例子，对于原始点 $\boldsymbol{z_1}$，不加 mode seeking regularization 时，由于生成分布的活动范围小，因此产生 M_2 模式，而添加 mode seeking regularization 后，得到的对应 $\boldsymbol{I_1}$ 属于 M_1，因此相较于之前更大可能产生更多的 mode。

Objective 以 image-to-image translation 任务中的 cGAN 模型 (来自论文Image-to-Image Translation with Conditional Adversarial Networks)[?] 为例，其目标函数为：

$$\min_G \max_D \mathbb{E}_{\boldsymbol{c},\boldsymbol{y}}\big[\log D(\boldsymbol{c},\boldsymbol{y})\big] + \mathbb{E}_{\boldsymbol{c},\boldsymbol{x}}\big[\log\big(1 - D(\boldsymbol{c}, G(\boldsymbol{c},\boldsymbol{x}))\big)\big] + \mathbb{E}_{\boldsymbol{c},\boldsymbol{y},\boldsymbol{x}}\big[\|\boldsymbol{y} - G(\boldsymbol{c},\boldsymbol{x})\|_1\big]$$



(a) Proposed regularization



(b) Applying proposed regularization on StackGAN++

图 98: Proposed regularization. (a) We propose a regularization term that maximizes the ratio of the distance between generated images with respect to the distance between their corresponding input latent codes. (b) The proposed regularization method can be applied to arbitrary cGANs. Take StackGAN++[?], a model for text-to-image synthesis, as an example, we easily apply the regularization term regardless of the complex tree-like structure of the original model.

其中符号表示解释为：

- $\boldsymbol{c}$: conditional contexts input(e.g. image of the specific category)
- $\boldsymbol{y}$: target domain input
- $\boldsymbol{x}$: source domain input

因此在此基础上添加 mode seeking regularization，则模型的目标函数变为（未完成）：

$$\min_G \max_D \mathcal{V}(G,D) = \mathbb{E}_{\boldsymbol{c},\boldsymbol{y}}\big[\log D(\boldsymbol{c},\boldsymbol{y})\big] + \mathbb{E}_{\boldsymbol{c},\boldsymbol{x}}\big[\log\left(1 - D(\boldsymbol{c}, G(\boldsymbol{c},\boldsymbol{x}))\right)\big] + \mathbb{E}_{\boldsymbol{c},\boldsymbol{y},\boldsymbol{x}}\big[\|\boldsymbol{y} - G(\boldsymbol{c},\boldsymbol{x})\|_1\big] - \lambda_{\text{ms}}\mathbb{E}_{(\boldsymbol{x_1},\boldsymbol{x_2})}\mathcal{V}_{\text{ms}}(\boldsymbol{x_1},\boldsymbol{x_2})$$

注意其中 mode seeking regularization 中在原始空间任意采样样本对满足：$\boldsymbol{x_1} \neq \boldsymbol{x_2}, \forall \boldsymbol{x_1}, \boldsymbol{x_2} \in p_{\boldsymbol{x}}(\boldsymbol{x})$

Training procedure

Evaluation metrics and experiments

19.4 Wasserstein GAN (WGAN)

19.4.1 Introduction（未完成）

本文选自Wasserstein Generative Adversarial Networks[?]，从 GAN 到 WGAN 中使用的概念演化图如下：

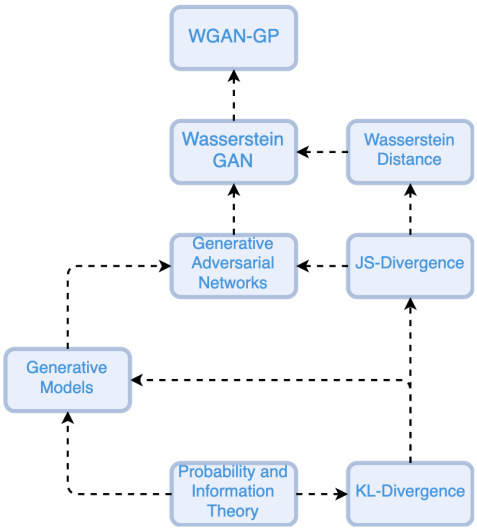


图 99: Concepts used in Wasserstein GAN

The problem of GAN 回顾原始 GAN 模型，模型中存在三种分布：

Symbol	Meaning	Notes
p_z	Data distribution over noise input $\boldsymbol{z}$	Usually, just uniform.
p_g	The generator ' s distribution over data $\boldsymbol{x}$	
p_r	Data distribution over real sample $\boldsymbol{x}$	

一方面对于 discriminator, 希望 D 在真实样本上的决策可以通过优化以下目标函数而变得准确：

$$\max \ \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right]$$

同时在给定假样本 $G(\boldsymbol{z}), \boldsymbol{z}\sim p_z(\boldsymbol{z})$ 的前提下，D 可以通过优化以下目标函数而达到同样对假样本准确决策的目的：

$$\max \ \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\left[\log(1-D(G(\boldsymbol{z})))\right]$$

另一方面，generator 通过优化以下目标函数被训练，以达到增强 D 对假样本产生高判别为真概率的目的：

$$\min \ \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\left[\log(1-D(G(\boldsymbol{z})))\right]$$

将这两个方面进行合并，D 和 G 之间实际上是在进行一个 minmax 博弈，待优化的目标函数如下：

$$\min_G \max_D \ V(D,G)=\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right]+\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\left[\log(1-D(G(\boldsymbol{z})))\right]$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\left[\log\left(1-D(\boldsymbol{x})\right)\right]$$

注意，在梯度下降更新过程中 $\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right]$ 这一部分对 G 没有影响。

而实际上，GAN 的训练效果并不好，因为其在训练过程中存在以下两个重要的问题：

1. **Vanishing gradient**：判别器 D 判别效果越好，对生成器 G 训练中的梯度消失越严重
2. **Mode collapse**：生成器 G 宁愿生成重复样本，也不愿生成多样性样本

针对以上遇到的这个问题，以下进行理论分析。

Analysis of vanishing gradient problem

What is the optimal value of D? 因为实际中发现判别器 D 判别效果越好，对生成器 G 训练中的梯度消失越严重，那么就分析在判别器效果最好时的极端情况，因此首先从理论角度找到 D 最优解。

$$\begin{aligned}
V(D,G) &= \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\left[\log\left(1-D(\boldsymbol{x})\right)\right] \\
&= \int_{\boldsymbol{x}} p_r(\boldsymbol{x})\log\big(D(\boldsymbol{x})\big)d\boldsymbol{x} + \int_{\boldsymbol{x}} p_g(\boldsymbol{x})\log\big(1-D(\boldsymbol{x})\big)d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} \Big(p_r(\boldsymbol{x})\log\big(D(\boldsymbol{x})\big) + p_g(\boldsymbol{x})\log\big(1-D(\boldsymbol{x})\big)\Big)d\boldsymbol{x}
\end{aligned}$$

对该分布中任意某一样本情形（概率分布中的一种情况）$\boldsymbol{x}$ 进行分析，损失函数在该情形上为：

$$V(D,G;\boldsymbol{x}) = p_r(\boldsymbol{x})\log\big(D(\boldsymbol{x})\big) + p_g(\boldsymbol{x})\log\big(1-D(\boldsymbol{x})\big)$$

对其求关于 $D(\boldsymbol{x})$ 的梯度，即：

$$\frac{\partial V(D,G;\boldsymbol{x})}{\partial D(\boldsymbol{x})} = \frac{p_r(\boldsymbol{x})}{D(\boldsymbol{x})} + \frac{p_g(\boldsymbol{x})}{1-D(\boldsymbol{x})}$$

令梯度为 0，可得：

$$\begin{aligned}
& \frac{p_r(\boldsymbol{x})}{D^*(\boldsymbol{x})} + \frac{p_g(\boldsymbol{x})}{1-D^*(\boldsymbol{x})} = 0 \\
\Rightarrow\quad & \frac{p_r(\boldsymbol{x})\big(1-D^*(\boldsymbol{x})\big) + p_g(\boldsymbol{x})D^*(\boldsymbol{x})}{D^*(\boldsymbol{x})\big(1-D^*(\boldsymbol{x})\big)} = 0 \\
\Rightarrow\quad & p_r(\boldsymbol{x})\big(1-D^*(\boldsymbol{x})\big) + p_g(\boldsymbol{x})D^*(\boldsymbol{x}) = 0 \quad (D^*(\boldsymbol{x})\in(0,1)) \\
\Rightarrow\quad & p_r(\boldsymbol{x}) - p_r(\boldsymbol{x})D^*(\boldsymbol{x}) + p_g(\boldsymbol{x})D^*(\boldsymbol{x}) = 0 \\
\Rightarrow\quad & p_r(\boldsymbol{x}) = \big(p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})\big)D^*(\boldsymbol{x}) \\
\Rightarrow\quad & D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})} \in (0,1] \quad (p_r(\boldsymbol{x})\neq 0 \text{ or } p_g(\boldsymbol{x})\neq 0)
\end{aligned}$$

即当判别器达到最优时，其对任意样本 $\boldsymbol{x}$ 的判别输出为 $D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x})+p_g(\boldsymbol{x})}$

What is the loss function when D achieves optimal? 将 $D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x})+p_g(\boldsymbol{x})}$ 带入上述针对任意单个样本的目标函数 $V(D,G;\boldsymbol{x})$ 中，可得在 D 取得最优时，G 的训练目标函数：

$$V(D^*,G;\boldsymbol{x}) = p_r(\boldsymbol{x})\log\big(D^*(\boldsymbol{x})\big) + p_g(\boldsymbol{x})\log\big(1-D^*(\boldsymbol{x})\big)$$

$$
\begin{aligned}
&= p_r(\boldsymbol{x}) \log \left(\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) + p_g(\boldsymbol{x}) \log \left(1 - \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) \\
&= p_r(\boldsymbol{x}) \log \left(\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) + p_g(\boldsymbol{x}) \log \left(\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right) \\
&= p_r(\boldsymbol{x}) \Big(\log \big(p_r(\boldsymbol{x})\big) - \log \big(p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})\big) \Big) + p_g(\boldsymbol{x}) \Big(\log \big(p_g(\boldsymbol{x})\big) - \log \big(p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})\big) \Big) \\
&= p_r(\boldsymbol{x}) \Big(\log \big(p_r(\boldsymbol{x})\big) - \log \big(\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2} \cdot 2\big) \Big) + p_g(\boldsymbol{x}) \Big(\log \big(p_g(\boldsymbol{x})\big) - \log \big(\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2} \cdot 2\big) \Big) \\
&= p_r(\boldsymbol{x}) \Big(\log \big(p_r(\boldsymbol{x})\big) - \log \Big(\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) - \log 2 \Big) + p_g(\boldsymbol{x}) \Big(\log \big(p_g(\boldsymbol{x})\big) - \log \Big(\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) - \log 2 \Big)
\end{aligned}
$$

将两个括号中的常数项 $\log 2$ 提出，可得：

$$
\begin{aligned}
V(D^*, G; \boldsymbol{x}) &= p_r(\boldsymbol{x}) \Big(\log \big(p_r(\boldsymbol{x})\big) - \log \Big(\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) \Big) + p_g(\boldsymbol{x}) \Big(\log \big(p_g(\boldsymbol{x})\big) - \log \Big(\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}\Big) \Big) \\
&\quad - p_r(\boldsymbol{x}) \log 2 - p_g(\boldsymbol{x}) \log 2 \\
&= p_r(\boldsymbol{x}) \log \Big(\frac{p_r(\boldsymbol{x})}{\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) + p_g(\boldsymbol{x}) \log \Big(\frac{p_g(\boldsymbol{x})}{\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) - p_r(\boldsymbol{x}) \log 2 - p_g(\boldsymbol{x}) \log 2
\end{aligned}
$$

因此，当判别器 D 取最优时，GAN 的目标函数为：

$$
\begin{aligned}
V(D^*, G) &= \int_{\boldsymbol{x}} V(D^*, G; \boldsymbol{x}) d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} p_r(\boldsymbol{x}) \log \Big(\frac{p_r(\boldsymbol{x})}{\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) d\boldsymbol{x} + \int_{\boldsymbol{x}} p_g(\boldsymbol{x}) \log \Big(\frac{p_g(\boldsymbol{x})}{\frac{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} \Big) d\boldsymbol{x} - \log 2 \int_{\boldsymbol{x}} p_r(\boldsymbol{x}) d\boldsymbol{x} - \log 2 \int_{\boldsymbol{x}} p_g(\boldsymbol{x}) d\boldsymbol{x} \\
&= D_{\mathrm{KL}} \Big(p_r(\boldsymbol{x}) \,\Big\|\, \frac{p_r(\boldsymbol{x} + p_g(\boldsymbol{x}))}{2} \Big) + D_{\mathrm{KL}} \Big(p_g(\boldsymbol{x}) \,\Big\|\, \frac{p_r(\boldsymbol{x} + p_g(\boldsymbol{x}))}{2} \Big) - 2 \log 2 \\
&= 2 \Bigg(\underbrace{\frac{1}{2} D_{\mathrm{KL}} \Big(p_r(\boldsymbol{x}) \,\Big\|\, \frac{p_r(\boldsymbol{x} + p_g(\boldsymbol{x}))}{2} \Big) + \frac{1}{2} D_{\mathrm{KL}} \Big(p_g(\boldsymbol{x}) \,\Big\|\, \frac{p_r(\boldsymbol{x} + p_g(\boldsymbol{x}))}{2} \Big)}_{\text{Jensen-Shannon Divergence (JSD)}} \Bigg) - 2 \log 2 \\
&= 2 \mathrm{D}_{JS} \Big(p_r(\boldsymbol{x}) \,\|\, p_g(\boldsymbol{x}) \Big) - 2 \log 2
\end{aligned}
$$

What is the loss function when D and G both achieve optimal? 因此，一旦生成器 G 也被训练到理想最佳状况下，那么有 $\mathrm{D}_{JS} \Big(p_r(\boldsymbol{x}) \,\|\, p_g(\boldsymbol{x}) \Big) = 0$，即 $p_r(\boldsymbol{x}) = p_g(\boldsymbol{x})$, $\forall \boldsymbol{x}$，$D^*(\boldsymbol{x}) = \frac{1}{2}$，则有:

$$
\begin{aligned}
V(D^*, G; \boldsymbol{x}) &= p_r(\boldsymbol{x}) \log \big(\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x})} \big) + p_g(\boldsymbol{x}) \log \big(\frac{p_g(\boldsymbol{x})}{p_g(\boldsymbol{x})} \big) - 2 \log 2 \\
&= p_r(\boldsymbol{x}) \log 1 + p_g(\boldsymbol{x}) \log 1 - 2 \log 2 \\
&= -2 \log 2 \quad (\because \log 1 = 0)
\end{aligned}
$$

因此当 D 理想最优时，针对 G 的目标函数变为：

$$
\begin{aligned}
V(D^*, G) &= \int_{\boldsymbol{x}} V(D^*, G; \boldsymbol{x}) d\boldsymbol{x} \\
&= -2 \log 2 \int_{\boldsymbol{x}} d\boldsymbol{x} \\
&= -2 \log 2
\end{aligned}
$$

而即使 G 没有完完全全达到理想最佳状况，只是达到近似最佳状况时，也有:

$$
p_r(\boldsymbol{x}) \approx p_g(\boldsymbol{x})
$$

$$\Rightarrow \quad p_r(\boldsymbol{x}) = p_g(\boldsymbol{x}) + \epsilon, \ (\epsilon \text{ is a very small number})$$

则针对某一样本情形 $\boldsymbol{x}$ 的目标函数可以写为：

$$\begin{aligned}
V(D^{\sim *}, G; \boldsymbol{x}) &= p_r(\boldsymbol{x}) \log \left(\frac{p_r(\boldsymbol{x})}{\frac{p_r(\boldsymbol{x}) + p_r(\boldsymbol{x}) - \epsilon}{2}} \right) + p_g(\boldsymbol{x}) \log \left(\frac{p_g(\boldsymbol{x})}{\frac{p_g(\boldsymbol{x}) + \epsilon + p_g(\boldsymbol{x})}{2}} \right) - 2 \log 2 \\
&= p_r(\boldsymbol{x}) \log \left(\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) - \frac{\epsilon}{2}} \right) + p_g(\boldsymbol{x}) \log \left(\frac{p_g(\boldsymbol{x})}{p_g(\boldsymbol{x}) + \frac{\epsilon}{2}} \right) - 2 \log 2 \\
&= p_r(\boldsymbol{x}) \log \left(\frac{p_r(\boldsymbol{x}) - \frac{\epsilon}{2} + \frac{\epsilon}{2}}{p_r(\boldsymbol{x}) - \frac{\epsilon}{2}} \right) + p_g(\boldsymbol{x}) \log \left(\frac{p_g(\boldsymbol{x}) + \frac{\epsilon}{2} - \frac{\epsilon}{2}}{p_g(\boldsymbol{x}) + \frac{\epsilon}{2}} \right) - 2 \log 2 \\
&= p_r(\boldsymbol{x}) \log \left(1 - \frac{\frac{\epsilon}{2}}{\frac{\epsilon}{2} + p_r(\boldsymbol{x})} \right) + p_g(\boldsymbol{x}) \log \left(1 - \frac{\frac{\epsilon}{2}}{\frac{\epsilon}{2} + p_g(\boldsymbol{x})} \right) - 2 \log 2 \\
&\approx -2 \log 2
\end{aligned}$$

因此当 D 近似理论最优时，针对 G 的目标函数变为：

$$\begin{aligned}
V(D^{\sim *}, G) &= \int_{\boldsymbol{x}} V(D^{\sim *}, G; \boldsymbol{x}) d\boldsymbol{x} \\
&\approx -2 \log 2 \int_{\boldsymbol{x}} d\boldsymbol{x} \\
&= -2 \log 2
\end{aligned}$$

因此此时目标函数关于生成器 G 参数 $\boldsymbol{\theta_g}$ 的梯度为：

$$\nabla_{\boldsymbol{\theta_g}} V(D^{\sim *}, G) = \nabla_{\boldsymbol{\theta_g}} (-2 \log 2) = 0$$

所以结论是：当判别器 D 达到理论最优，同时生成器 G 达到最优或近似最优时，GAN 的目标函数恒等于或近似等于常数 $-2\log 2$，因此无论采用什么样的优化算法，由于梯度恒为 0，所以生成器 G 的参数永远不会得到更新，即梯度消失（Vanishing gradient）。

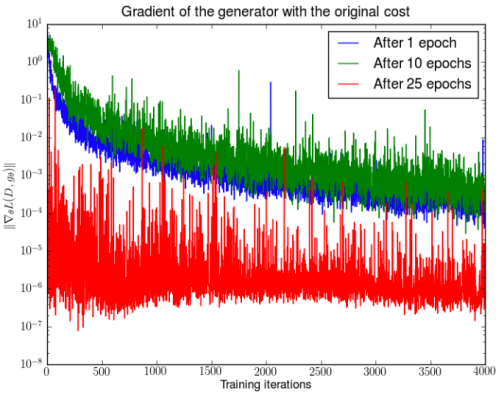


图 100: Gradient value of generator in DCGAN[?]

Analysis of mode collapse problem（未完成）

Improved GAN training（未完成） 以下建议将被用于帮助稳定化 GANs 的训练过程以及提高 GANs 的训练效果，其中：

- 1-5 被实际用于提高 GAN 训练的收敛速度，来自论文：Improve Techniques for Training GANs[?]
- 6-7 被用于解决不相交分布（disjoint distributions）的问题，来自论文：Towards principled methods for training generative adversarial networks[?]

这些方法如下：

- Feature Matching
- Minibatch Discrimination
- Historical Averaging
- One-sided Label Smoothing
- Virtual Batch Normalization(VBN)
- Adding Noises
- Use Better Metric of Distribution Similarity

19.4.2 Wasserstein distance and the Kantorovich-Rubinstein duality（未完成）

19.4.3 Model formulation

Objective WGAN 最终的目标函数为：

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\omega}} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\left[f_{\boldsymbol{\omega}}(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[f_{\boldsymbol{\omega}}(g_{\boldsymbol{\theta}}(\boldsymbol{z}))\right]$$

其中判别器的判别函数 $f_{\boldsymbol{\omega}}(\cdot)\in\mathbb{R}$，而非原始 GAN 中的 $f_{\boldsymbol{\omega}}(\cdot)\in(0,1)$.

Training procedure 训练算法的伪代码如下：

Algorithm 22: Wasserstein GAN[?]

Input: α: the learning rate (defaults to 0.00005);
c: the clipping parameter (defaults to 0.01);
m: the batch size (defaults to 64);
n_{critic}: the number of iterations of the critic per generator iteration (defaults to 5).

Input: $\boldsymbol{\omega}$: initial critic parameters;
$\boldsymbol{\theta}$: initial generator's parameters.

Output: The final parameters of generator $\boldsymbol{\theta}$

while $\boldsymbol{\theta}$ *has not converged* **do**
  for $t=0,...,n_{critic}$ **do**
    Sample $\{\boldsymbol{x}^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from real data.
    Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p_z(\boldsymbol{z})$ a batch of prior samples.
    Compute update direction for critic: $\Delta_{\boldsymbol{\omega}} \leftarrow \nabla_{\boldsymbol{\omega}}\left[\frac{1}{m}\sum_{i=1}^m f_{\boldsymbol{\omega}}(\boldsymbol{x}^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_{\boldsymbol{\omega}}(g_{\boldsymbol{\theta}}(\boldsymbol{z}^{(i)}))\right]$
    Updata critic parameters: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \text{RMSProp}(\boldsymbol{\omega}, \Delta_{\boldsymbol{\omega}}, \alpha)$
    Clipping for critic parameters: $\boldsymbol{\omega} \leftarrow \text{clip}(\boldsymbol{\omega}, -c, c)$
  end
  Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p_z(\boldsymbol{z})$ a batch of prior samples.
  Compute update direction for generator: $\Delta_{\boldsymbol{\theta}} \leftarrow \nabla_{\boldsymbol{\theta}}\frac{1}{m}\sum_{i=1}^m f_{\boldsymbol{\omega}}(g_{\boldsymbol{\theta}}(\boldsymbol{z}^{(i)}))$
  Updata generator's parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \text{RMSProp}(\boldsymbol{\theta}, \Delta_{\boldsymbol{\theta}}, \alpha)$
end

注意：

- critic(discriminator) 的优化为 max，所以更新方向为梯度**正方向**
- generator 优化为 min，所以更新方向为梯度**负方向**，或者直观来看，其目标函数可以转换为:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[f_{\boldsymbol{\omega}}(g_{\boldsymbol{\theta}}(\boldsymbol{z}))\right]$$

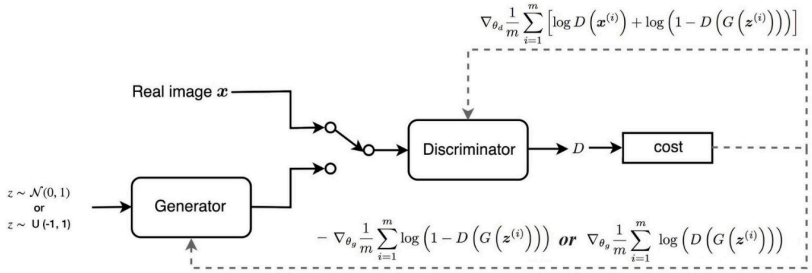
  因此更新方向就是梯度方向，即 $\nabla_{\boldsymbol{\theta}}\frac{1}{m}\sum_{i=1}^m f_{\boldsymbol{\omega}}(g_{\boldsymbol{\theta}}(\boldsymbol{z}^{(i)}))$

Summary WGAN 和 GAN 的区别：

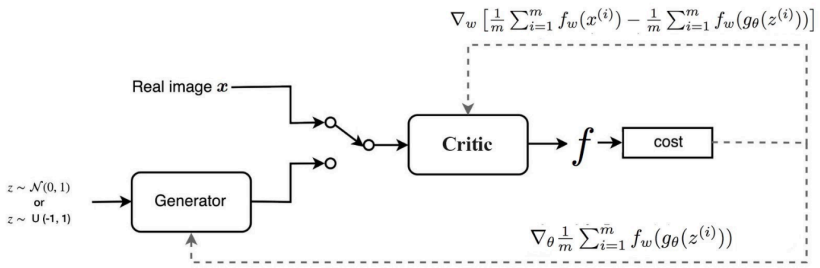
- WGAN 的 Critic(Discriminator) 映射关系为 $f_{\boldsymbol{\omega}}(\cdot):\mathbb{R}^{\mathcal{X}}\to\mathbb{R}$，而 GAN（Vanilla GAN）中为 $f_{\boldsymbol{\omega}}(\cdot):\mathbb{R}^{\mathcal{X}}\to(0,1)$，也就是说 WGAN 最后一层没有 sigmoid 激活函数

- WGAN 的 Critic(Discriminator) 输出函数没有 log 项
- WGAN 的 Critic 在每次更新后都要把参数截断在某个范围，即 weight clipping，这是为了保证上面讲到的 Lipschitz 限制
- Critic 训练得越好，对 Generator 的提升更有利，因此可以放心地多训练 Critic，而 Vanilla GAN 的 Discriminator 训练的越好，对 Generator 的提升更不利
- WGAN 和 GAN 的模型对比图如下：

GAN:



WGAN



19.4.4 Reference

- From GAN to WGAN by Lil'Log
- Wasserstein GAN by James Allingham
- Read-through: Wasserstein GAN
- An intuitive guide to optimal transport, part I: formulating the problem
- GAN—Wasserstein GAN & WGAN-GP by Jonathan Hui

19.5 Wasserstein GAN with Gradient Penalty (WGAN-GP)

19.5.1 Introduction

本文选自Improved Training of Wasserstein GANs[?]

19.5.2 Model formulation

Objective 模型的目标函数如下：

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\omega}} V(\boldsymbol{\theta}, \boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] - \mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\tilde{\boldsymbol{x}})\big] - \lambda\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\big[(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2 - 1)^2\big]$$

其中：

$$\hat{\boldsymbol{x}} = \epsilon\boldsymbol{x} + (1-\epsilon)\tilde{\boldsymbol{x}}$$

Training procedure 训练算法的伪代码如下：

Algorithm 23: WGAN with gradient penalty[?]

Input: λ: the gradient penalty coefficient;
n_{critic}: the number of iterations of the critic per generator iteration (defaults to 5);
m: the batch size (defaults to 64);
α, β_1, β_2: Adam hyperparameters.

Input: $\boldsymbol{\omega}$: initial critic parameters;
$\boldsymbol{\theta}$: initial generator's parameters.

Output: The final parameters of generator $\boldsymbol{\theta}$

while $\boldsymbol{\theta}$ *has not converged* **do**
  for $t = 0, ..., n_{critic}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      Compute the output of generator: $\tilde{\boldsymbol{x}} \leftarrow G_{\boldsymbol{\theta}}(\boldsymbol{z})$
      Construct a noise-convex sample: $\hat{\boldsymbol{x}} \leftarrow \epsilon\boldsymbol{x} + (1-\epsilon)\tilde{\boldsymbol{x}}$
      Compute value function: $V^{(i)} \leftarrow D_{\boldsymbol{\omega}}(\boldsymbol{x}) - D_{\boldsymbol{\omega}}(\tilde{\boldsymbol{x}}) - \lambda(\|\nabla_{\hat{\boldsymbol{x}}}D_{\boldsymbol{\omega}}(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end
    Compute the update direction for critic: $\Delta_{\boldsymbol{\omega}} \leftarrow \nabla_{\boldsymbol{\omega}} \frac{1}{m} \sum_{i=1}^{m} V^{(i)}$
    Update critic parameters: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \text{Adam}(\boldsymbol{\omega}, \Delta_{\boldsymbol{\omega}}, \alpha, \beta_1, \beta_2)$
  end
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  Compute the update direction for generator: $\Delta_{\boldsymbol{\theta}} \leftarrow \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} D_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}^{(i)}))$
  Update generator's parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \text{Adam}(\boldsymbol{\theta}, \Delta_{\boldsymbol{\theta}}, \alpha, \beta_1, \beta_2)$
end

19.6 Wasserstein GAN with Lipschitz Penalty (WGAN-LP)

19.6.1 Introduction

本节选自On the regularization of Wasserstein GANs[?]

19.6.2 Model formulation

Penalizing the violation of the Lipschitz constraint

$$\left(\max\left\{0, \frac{|f(g(\boldsymbol{x})) - f(\boldsymbol{y})|}{\|g(\boldsymbol{x}) - \boldsymbol{y}\|_2} - 1\right\}\right)^2$$

其中 $f(\cdot): \mathbb{R}^{\mathcal{Y}} \to \mathbb{R}$ 为判别器对应的判别函数；$g(\cdot): \mathbb{R}^{\mathcal{X}} \to \mathbb{R}^{\mathcal{Y}}$ 为生成器对应的生成函数。

注意：区分WGAN-LP[?] 和Mode-Seeking GAN[?]

19.7 Wasserstein GAN with Wasserstein Gradient Regularization (WWGAN)

19.7.1 Introduction

本节选自Wasserstein of Wasserstein Loss for Learning Generative Models[?]

19.8 Wasserstein GAN with Consistency Term (CT-GAN)（未完成）

19.8.1 Introduction

本文选自Improving the Improved Training of Wasserstein GANs: A Consistency Term and Its Dual Effect[?]

19.9 Virtual Adversarial Lipschitz Regularization（未完成）

19.9.1 Introduction

本节选自Virtual Adversarial Lipschitz Regularization[?]

19.9.2 model formulation

19.10 Survey and Summary（未完成）

19.10.1 Introduction

本节对一系列的 GAN 进行总结归纳，其中内容选自综述论文：

- Generative Adversarial Networks: A Survey and Taxonomy

本章将从模型构造，目标函数设计以及实际应用场景三个方面介绍不同的 GANs 模型

19.10.2 Architecture-variant GANs

19.10.3 Loss-variant GANs

19.10.4 Applications of GANs

- Fingerprint Producer:
  - Fingerprint Inpainting with Generative Models
  - DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution[?]
  - Finger-GAN: Generating Realistic Fingerprint Images Using Connectivity Imposed GAN[?]
- Semantic image inpainting:
  - Semantic Image Inpainting with Deep Generative Models[?]
- Text to Image Synthesis:
  - Generative Adversarial Text to Image Synthesis[?]

19.10.5 Personal ideas of GANs

About Fingerprints-GANs 图为DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution[?] 该文提供了几条关于指纹合成很关键的**指导思想**：

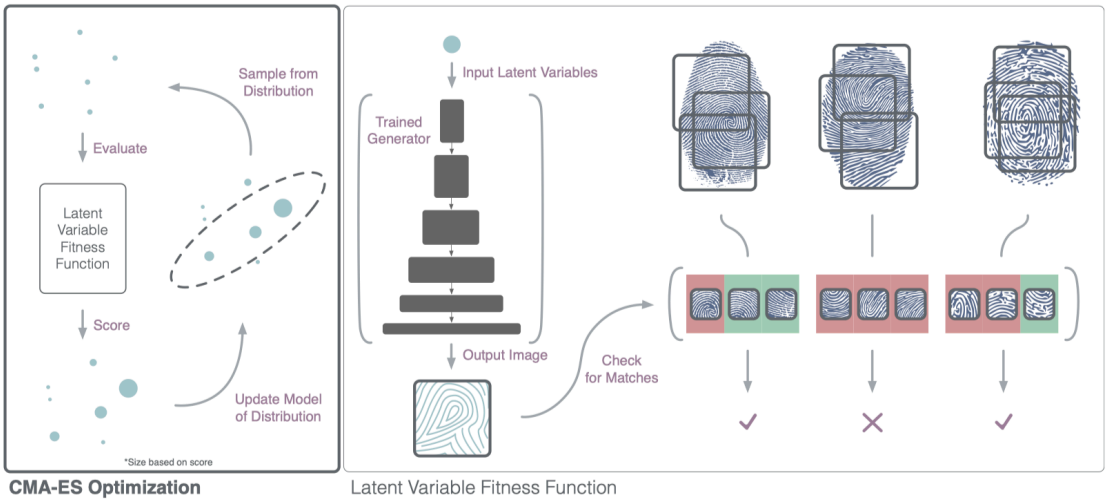


图 101: Latent Variable Evolution with a trained network. On the left is a high level overview of CMA-ES and the box on the right shows how the latent variables are evaluated.

- 在一些实际设备（譬如智能手机、指纹传感器等）的设计中，由于人体工程学的原因，传感器只会获取用户指纹的一部分，而部分指纹又不像全部指纹那样特殊，因此一个部分指纹和另一个部分指纹被错误匹配的概率就大大增强，于是这就有了“可乘之机”
- 使用标准的 GANs 方法（不加额外约束）是远远不够的，因为迄今为止的 GAN 模型都只是识别了 domain 的视觉风格（visual realism），而没有和许多图的许多局部细节进行匹配

20 Variational Autoencoders (VAEs)

20.1 An Introduction to Variational Autoencoders（未完成）

20.1.1 Introduction

本文选自An Introduction to Variational Autoencoders

21 Graph Neural Networks (GNNs)

21.1 Overview of GCNs

In this section, the notations used in this survey are listed for convenience. Besides, some definitions and categorizations about graph neural networks are introduced, and some related surveys are also provided.

The notations used in GCNs show as follows:

表 1: Commonly used notations of graph neural networks

Notations	Description
$\|\cdot\|$	The length of a set.
$*$	Graph convolution operation.
$\odot$	Element-wise product.
$\mathcal{G}$	A graph.
$\mathcal{V}$	The set of nodes in a graph.
v	A node $v \in \mathcal{V}$.
$\mathcal{E}$	The set of edges in a graph.
e_{ij}	An edge $e_{ij} \in \mathcal{E}$.
$\mathcal{N}(v)$	The neighbors of a node v.
$\mathbf{A}$	The graph adjacency matrix.
$\mathbf{A}^{\mathrm{T}}$	The transpose of the matrix $\mathbf{A}$.
$\mathbf{A}^n$	The n^{th} power of $\mathbf{A}$.
$[\mathbf{A}, \mathbf{B}]$	The concatenation of $\mathbf{A}$ and $\mathbf{B}$.
$\mathbf{D}$	The degree matrix corresponding $\mathbf{A}$. $\mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{A}_{ij}$.
n	The number of nodes, $n = \|\mathcal{V}\|$.
m	The number of edges, $m = \|\mathcal{E}\|$.
d	The dimension of a node feature vector.
$\mathbf{X} \in \mathbb{R}^{d \times n}$	The feature matrix of a graph (for theoretical derivation).
$\mathbf{X}^{\mathrm{T}} \in \mathbb{R}^{n \times d}$	The feature matrix of a graph (for programming implementation).
$\mathbf{x}^{(i)} \in \mathbb{R}^d$	The feature vector of the node v_i.
$\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$	The feature vector of a graph in the case $d = j$.

The definition of graph shows as follows:

Definition 21.1 (Graph[?][?])**.** A graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices or nodes $\mathcal{V} = \{v_1, \ldots, v_n\}$, and $\mathcal{E}$ is the set of edges.

Let $v_i \in \mathcal{V}$ to denote a vertice and $e_{ij} = (v_i, v_j) \in \mathcal{E}$ to denote an edge pointing from v_j to v_i.

The neighborhood of a node v is defined as $\mathcal{N}(v) = \{u \in \mathcal{V} | (v, u) \in \mathcal{E}\}$.

The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a symmetric (typically sparse) matrix where $a_{ij} \in \mathbb{R}$ denotes the edge weight between nodes v_i and v_j. A missing edge is represented through $\mathbf{A}_{ij} = 0, \forall (v_i, v_j) \notin \mathcal{E}$. And $\mathbf{A}_{ii} = 0, \forall i \in \{1, \cdots, n\}$.

Also, we define the degree matrix $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ as a diagonal matrix where each entry on the diagonal is equal to the row-sum of the adjacency matrix: $d_i = \mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{A}_{ij}$.

Each node v_i in the graph has a corresponding d-dimensional feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^d$. The entire feature matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ stacks n nodes feature vectors $\left[\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(i)}, \cdots, \mathbf{x}^{(n)}\right]$. Each node belongs to one out of C classes and can be labeled with a C-dimensional one-hot vector $\mathbf{y}^{(i)} \in \{0,1\}^C$. We only know the labels of a subset of all nodes $\mathcal{V}$ and want to predict the unknown labels.

In summary, some notations of graph constants defined as follows:

- $\mathbf{A} \in \mathbb{R}^{n \times n}$: Adjacency matrix of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
  - $(v_i, v_j) \in \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} \neq 0$
  - $(v_i, v_j) \notin \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} = 0$
  - $\mathbf{A}_{ii} = 0, \forall i \in \{1, \cdots, n\}$
- $\mathbf{I} \in \mathbb{R}^{n \times n}$: Identity matrix containing identity self-loops
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$: Augmented adjacency matrix, namely adjacency matrix with self-loops
  - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij}, \forall i, j \in \{1, \cdots, n\}$
  - $\tilde{\mathbf{A}}_{ii} = \mathbf{A}_{ii} + \mathbf{I}_{ii} = \mathbf{A}_{ii} + 1, \forall i \in \{1, \cdots, n\}$
  - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij} = \mathbf{A}_{ij}, \forall i, j \in \{1, \cdots, n\}, i \neq j$
- $\mathbf{D} = \operatorname{diag}(d_1, \cdots, d_n)$: Degree matrix corresponding $\mathbf{A}$
  - $d_i = \mathbf{D}_{ii} = \sum_{j=1}^n \mathbf{A}_{ij}, \forall i \in \{1, \cdots, n\}$
- $\tilde{\mathbf{D}} = \operatorname{diag}(\tilde{d}_1, \cdots, \tilde{d}_n)$: Augmented degree matrix corresponding $\tilde{\mathbf{A}}$
  - $\tilde{d}_i = \tilde{\mathbf{D}}_{ii} = \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} = \sum_{j=1}^n (\mathbf{A}_{ij} + \mathbf{I}_{ij})$

Comprehension of local smoothing and normalized Laplacian. Here we consider four cases to introduce normalized Laplacian. Summary of four cases show as follows:

表 2: Local smoothing using asymmetric or symmetric normalized edge weight

Normalized Weight	Element Operation	Matrix Operation
$\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}}$	$\overline{\mathbf{x}}^{(i)} = \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \mathbf{x}^{(j)}$	$\overline{\mathbf{X}}^{\mathrm{T}} = \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \mathbf{X}^{\mathrm{T}}$
$\frac{\tilde{\mathbf{A}}_{ij}}{\left(\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}\right)^{\frac{1}{2}} \left(\sum_{l=1}^n \tilde{\mathbf{A}}_{jl}\right)^{\frac{1}{2}}}$	$\overline{\mathbf{x}}^{(i)} = \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} \mathbf{x}^{(j)}$	$\overline{\mathbf{X}}^{\mathrm{T}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^{\mathrm{T}}$

表 3: Laplacian using asymmetric or symmetric normalized edge weight

Normalized Weight	Element Operation	Matrix Operation
$\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}}$	$\mathbf{x}^{(i)} - \overline{\mathbf{x}}^{(i)} = \sum_{j=1}^n \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ij} \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right)$	$\mathbf{X}^{\mathrm{T}} - \overline{\mathbf{X}}^{\mathrm{T}} = \left(\mathbf{I} - \tilde{\mathbf{D}}^{-1} (\mathbf{A} + \mathbf{I})\right) \mathbf{X}^{\mathrm{T}}$
$\frac{\tilde{\mathbf{A}}_{ij}}{\left(\sum_{k=1}^n \tilde{\mathbf{A}}_{ik}\right)^{\frac{1}{2}} \left(\sum_{l=1}^n \tilde{\mathbf{A}}_{jl}\right)^{\frac{1}{2}}}$	$\mathbf{x}^{(i)} - \overline{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^n \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} \mathbf{x}^{(j)}$	$\mathbf{X}^{\mathrm{T}} - \overline{\mathbf{X}}^{\mathrm{T}} = \left(\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \tilde{\mathbf{D}}^{-\frac{1}{2}}\right) \mathbf{X}^{\mathrm{T}}$

Local smoothing with asymmetric normalized argumented adjacency Consider to aggregate neighbors' feature vector:

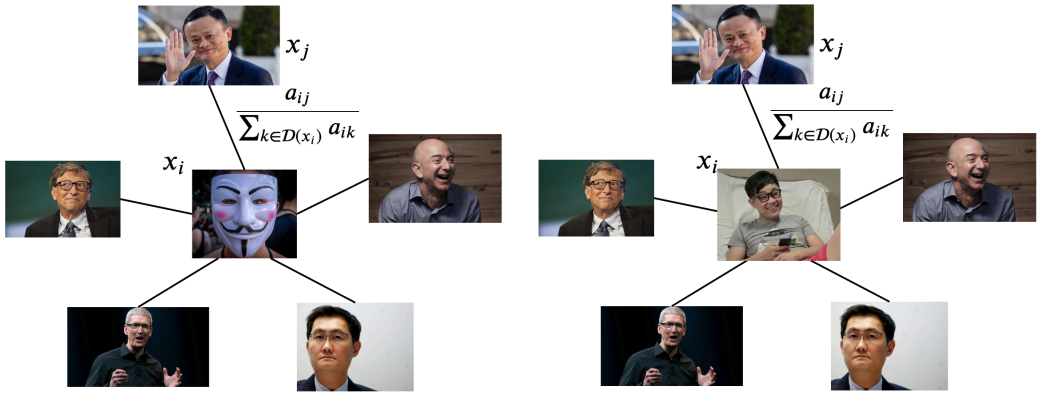


图 102: Social network examples for predicting annual salary informations. Using argumented adjacency matrix $\tilde{\mathbf{A}}$ to replace original adjacency matrix $\mathbf{A}$.

$$
\begin{aligned}
\overline{\mathbf{x}}^{(i)} &= \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^{n} \tilde{\mathbf{A}}_{ik}} \mathbf{x}^{(j)} \quad \left(\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^{n} \tilde{\mathbf{A}}_{ik}} \in \mathbb{R} \text{ is the normalized argumented edge weight between } v_i \text{ and } v_j, \text{ centring with } v_i\right) \\
&= \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}} \mathbf{x}^{(j)} \quad (\tilde{\mathbf{D}} \text{ is the degree matrix corresponding } \tilde{\mathbf{A}}) \\
&= \sum_{j=1}^{n} \tilde{\mathbf{D}}_{ii}^{-1} \tilde{\mathbf{A}}_{ij} \mathbf{x}^{(j)} \\
&= \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij} \mathbf{x}^{(j)} \in \mathbb{R}^d
\end{aligned}
$$

where $\bar{\cdot}$ means an aggregator function. Corresponding matrix operation expression (in programming implementation) as follows:

$$
\begin{aligned}
\overline{\mathbf{X}}^{\mathrm{T}} &= \left[\overline{\mathbf{x}}^{(1)}, \cdots, \overline{\mathbf{x}}^{(i)}, \cdots, \overline{\mathbf{x}}^{(n)}\right]^{\mathrm{T}} \\
&= \begin{bmatrix} (\overline{\mathbf{x}}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(n)})^{\mathrm{T}} \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{1j} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-1} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-1} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{nj} (\mathbf{x}^{(j)})^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-1} \end{bmatrix} \begin{bmatrix} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{1j} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \sum_{j=1}^{n} \tilde{\mathbf{A}}_{nj} (\mathbf{x}^{(j)})^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-1} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}_{11} & \cdots & \tilde{\mathbf{A}}_{1j} & \cdots & \tilde{\mathbf{A}}_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{i1} & \cdots & \tilde{\mathbf{A}}_{ij} & \cdots & \tilde{\mathbf{A}}_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{n1} & \cdots & \tilde{\mathbf{A}}_{nj} & \cdots & \tilde{\mathbf{A}}_{nn} \end{bmatrix} \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}} \end{bmatrix} \\
&= \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \mathbf{X}^{\mathrm{T}}
\end{aligned}
$$

Local smoothing with symmetric normalized argumented adjacency Above we use asymmetric normalized edge weight $\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^{n} \tilde{\mathbf{A}}_{ik}}$, however, for example, some people v_j are friends with everyone, even if v_i and v_j are very close, when these persons v_j are green tea bitches, we need to limit the role of these persons v_j, therefore consider symmetric nomalized edge weight $\frac{\tilde{\mathbf{A}}_{ij}}{(\sum_{k=1}^{n} \tilde{\mathbf{A}}_{ik})^{\frac{1}{2}} (\sum_{l=1}^{n} \tilde{\mathbf{A}}_{jl})^{\frac{1}{2}}}$:

$$
\begin{aligned}
\overline{\mathbf{x}}^{(i)} &= \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\left(\sum_{k=1}^{n} \tilde{\mathbf{A}}_{ik} \right)^{\frac{1}{2}} \left(\sum_{l=1}^{n} \tilde{\mathbf{A}}_{jl} \right)^{\frac{1}{2}}} \mathbf{x}^{(j)} \\
&= \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} \tilde{\mathbf{D}}_{jj}^{\frac{1}{2}}} \mathbf{x}^{(j)} \\
&= \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} \mathbf{x}^{(j)}
\end{aligned}
$$

where $\bar{\cdot}$ means an aggregator function. Corresponding matrix operation expression (in programming implementation) as follows:

$$
\overline{\mathbf{X}}^{\mathrm{T}} = \left[\overline{\mathbf{x}}^{(1)}, \cdots, \overline{\mathbf{x}}^{(i)}, \cdots, \overline{\mathbf{x}}^{(n)} \right]^{\mathrm{T}}
$$

$$
\begin{aligned}
&= \begin{bmatrix} (\overline{\mathbf{x}}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(n)})^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \sum\limits_{j=1}^{n} \tilde{\mathbf{A}}_{1j} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum\limits_{j=1}^{n} \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \sum\limits_{j=1}^{n} \tilde{\mathbf{A}}_{nj} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}_{11} & \cdots & \tilde{\mathbf{A}}_{1j} & \cdots & \tilde{\mathbf{A}}_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{i1} & \cdots & \tilde{\mathbf{A}}_{ij} & \cdots & \tilde{\mathbf{A}}_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{A}}_{n1} & \cdots & \tilde{\mathbf{A}}_{nj} & \cdots & \tilde{\mathbf{A}}_{nn} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}} \end{bmatrix} \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^{\mathrm{T}}
\end{aligned}
$$

Normalization Laplacian with asymmetric argumented adjacency Above we only consider the normalization for adjacency matrix for computing the estimation of feature vector using local smoothing (graph smoothing), if we consider Laplacian matrix for computing the residual between feature vector and its corresponding local smoothing feature vector, we have:

$$
\begin{aligned}
\mathbf{x}^{(i)} - \overline{\mathbf{x}}^{(i)} &= \mathbf{x}^{(i)} - \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\sum\limits_{k=1}^{n} \tilde{\mathbf{A}}_{ik}} \mathbf{x}^{(j)} \\
&= \mathbf{x}^{(i)} \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\sum\limits_{k=1}^{n} \tilde{\mathbf{A}}_{ik}} - \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\sum\limits_{k=1}^{n} \tilde{\mathbf{A}}_{ik}} \mathbf{x}^{(j)} \quad \left(\because \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\sum\limits_{k=1}^{n} \tilde{\mathbf{A}}_{ik}} = 1\right) \\
&= \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\sum\limits_{k=1}^{n} \tilde{\mathbf{A}}_{ik}} \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right) \\
&= \sum_{j=1}^{n} \frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}} \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right)
\end{aligned}
$$

Corresponding matrix operation expression (in programming implementation) as follows:

$$
\begin{aligned}
\mathbf{X}^{\mathrm{T}}-\overline{\mathbf{X}}^{\mathrm{T}} &= \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}} \end{bmatrix} - \begin{bmatrix} (\overline{\mathbf{x}}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(n)})^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}}\tilde{\mathbf{D}}_{11}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{1j} \\ \vdots \\ (\mathbf{x}^{(i)})^{\mathrm{T}}\tilde{\mathbf{D}}_{ii}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{ij} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}}\tilde{\mathbf{D}}_{nn}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{nj} \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{1j}(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{ij}(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{nj}(\mathbf{x}^{(j)})^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}}\tilde{\mathbf{D}}_{11}^{-1}\tilde{\mathbf{D}}_{11} \\ \vdots \\ (\mathbf{x}^{(i)})^{\mathrm{T}}\tilde{\mathbf{D}}_{ii}^{-1}\tilde{\mathbf{D}}_{ii} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}}\tilde{\mathbf{D}}_{nn}^{-1}\tilde{\mathbf{D}}_{nn} \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{1j}(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{ij}(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-1}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{nj}(\mathbf{x}^{(j)})^{\mathrm{T}} \end{bmatrix} \quad (\because \tilde{\mathbf{D}}_{ii}^{-1} = \sum_{j=1}^{n}\tilde{\mathbf{A}}_{ij}) \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-1} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn} \end{bmatrix} \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}} \end{bmatrix} - \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{X}^{\mathrm{T}} \\
&= \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{D}}\mathbf{X}^{\mathrm{T}} - \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{X}^{\mathrm{T}} \\
&= \tilde{\mathbf{D}}^{-1}\left(\tilde{\mathbf{D}}-\tilde{\mathbf{A}}\right)\mathbf{X}^{\mathrm{T}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{L}}\mathbf{X}^{\mathrm{T}}
\end{aligned}
$$

Normalization Laplacian with symmetric argumented adjacency Similarly, consider asymmetric case:

$$
\begin{aligned}
\mathbf{x}^{(i)}-\overline{\mathbf{x}}^{(i)} &= \mathbf{x}^{(i)} - \sum_{j=1}^{n}\frac{\tilde{\mathbf{A}}_{ij}}{\left(\sum_{k=1}^{n}\tilde{\mathbf{A}}_{ik}\right)^{\frac{1}{2}}\left(\sum_{l=1}^{n}\tilde{\mathbf{A}}_{jl}\right)^{\frac{1}{2}}}\mathbf{x}^{(j)} \\
&= \mathbf{x}^{(i)} - \sum_{j=1}^{n}\frac{\tilde{\mathbf{A}}_{ij}}{\tilde{\mathbf{D}}_{ii}^{\frac{1}{2}}\tilde{\mathbf{D}}_{jj}^{\frac{1}{2}}}\mathbf{x}^{(j)} \\
&= \mathbf{x}^{(i)} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}}\sum_{j=1}^{n}\tilde{\mathbf{A}}_{ij}\tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}}\mathbf{x}^{(j)}
\end{aligned}
$$

Corresponding matrix operation expression (in programming implementation) as follows:

$$
\begin{aligned}
\mathbf{X}^{\mathrm{T}}-\overline{\mathbf{X}}^{\mathrm{T}} &= \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}} \end{bmatrix} - \begin{bmatrix} (\overline{\mathbf{x}}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\overline{\mathbf{x}}^{(n)})^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(i)})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}} \end{bmatrix} - \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{1j} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \sum_{j=1}^{n} \tilde{\mathbf{A}}_{nj} \tilde{\mathbf{D}}_{jj}^{-\frac{1}{2}} (\mathbf{x}^{(j)})^{\mathrm{T}} \end{bmatrix} \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{D}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^{\mathrm{T}} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^{\mathrm{T}} \quad (\because \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{D}} \tilde{\mathbf{D}}^{-\frac{1}{2}} = \mathbf{I}) \\
&= \left(\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \right) \mathbf{X}^{\mathrm{T}} \\
&= \left(\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \left(\mathbf{A} + \mathbf{I} \right) \tilde{\mathbf{D}}^{-\frac{1}{2}} \right) \mathbf{X}^{\mathrm{T}}
\end{aligned}
$$

Lemma 1 (Non-negativity of $\mathbf{S}$)**.** The augmented normalized Laplacian matrix $\mathbf{S}$ is a symmetric positive semidefinite matrix.

Proof. For any signal $\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$, it satifies

$$
\begin{aligned}
\because \mathbf{S}\mathbf{x}_j^{(1:n)} &= \left(\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \right) \mathbf{x}_j^{(1:n)} \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} \left(\tilde{\mathbf{D}} - \tilde{\mathbf{A}} \right) \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{x}_j^{(1:n)} \\
\because \tilde{\mathbf{D}}^{-\frac{1}{2}} \left(\tilde{\mathbf{D}} - \tilde{\mathbf{A}} \right) &= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11} - \tilde{\mathbf{A}}_{11} & \cdots & -\tilde{\mathbf{A}}_{1i} & \cdots & -\tilde{\mathbf{A}}_{1n} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ -\tilde{\mathbf{A}}_{i1} & \cdots & \tilde{\mathbf{D}}_{ii} - \tilde{\mathbf{A}}_{ii} & \cdots & -\tilde{\mathbf{A}}_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\tilde{\mathbf{A}}_{n1} & \cdots & -\tilde{\mathbf{A}}_{ni} & \cdots & \tilde{\mathbf{D}}_{nn} - \tilde{\mathbf{A}}_{nn} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{ii} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{nn} \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\because \tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{x}_j^{(1:n)} &= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} \end{bmatrix} \begin{bmatrix} x_j^{(1)} \\ \vdots \\ x_j^{(i)} \\ \vdots \\ x_j^{(n)} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} x_j^{(1)} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} x_j^{(i)} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} x_j^{(n)} \end{bmatrix} \\
\therefore \mathbf{S}\mathbf{x}_j^{(1:n)} &= \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{ii} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \tilde{\mathbf{D}}_{nn}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{nn} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} x_j^{(1)} \\ \vdots \\ \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} x_j^{(i)} \\ \vdots \\ \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} x_j^{(n)} \end{bmatrix} \\
&= \begin{bmatrix} \left(\tilde{\mathbf{D}}_{11}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{11}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{11}\right)\tilde{\mathbf{D}}_{11}^{-\frac{1}{2}} x_j^{(1)} \\ \vdots \\ \left(\tilde{\mathbf{D}}_{ii}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{ii}\right)\tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}} x_j^{(i)} \\ \vdots \\ \left(\tilde{\mathbf{D}}_{nn}^{\frac{1}{2}} - \tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{nn}\right)\tilde{\mathbf{D}}_{nn}^{-\frac{1}{2}} x_j^{(n)} \end{bmatrix} \\
&= \begin{bmatrix} \left(1 - \tilde{\mathbf{D}}_{11}^{-1}\tilde{\mathbf{A}}_{11}\right) x_j^{(1)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{ii}^{-1}\tilde{\mathbf{A}}_{ii}\right) x_j^{(i)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{nn}^{-1}\tilde{\mathbf{A}}_{nn}\right) x_j^{(n)} \end{bmatrix}
\end{aligned}
$$

The quadratic form associated with $\mathbf{S}$ is

$$
\begin{aligned}
(\mathbf{x}_j^{(1:n)})^{\mathrm{T}}\mathbf{S}\mathbf{x}_j^{(1:n)} &= \begin{bmatrix} x_j^{(1)} & \cdots & x_j^{(i)} & \cdots & x_j^{(n)} \end{bmatrix} \begin{bmatrix} \left(1 - \tilde{\mathbf{D}}_{11}^{-1}\tilde{\mathbf{A}}_{11}\right) x_j^{(1)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{ii}^{-1}\tilde{\mathbf{A}}_{ii}\right) x_j^{(i)} \\ \vdots \\ \left(1 - \tilde{\mathbf{D}}_{nn}^{-1}\tilde{\mathbf{A}}_{nn}\right) x_j^{(n)} \end{bmatrix} \\
&= \sum_{i=1}^{n} \left(1 - \tilde{\mathbf{D}}_{ii}^{-1}\tilde{\mathbf{A}}_{ii}\right)\left(x_j^{(i)}\right)^2 \\
\because 1 - \tilde{\mathbf{D}}_{ii}^{-1}\tilde{\mathbf{A}}_{ii} &= 1 - \frac{\tilde{\mathbf{A}}_{ii}}{\tilde{\mathbf{D}}_{ii}} \geq 0, \forall i \in \{1, \cdots, n\} \\
\therefore (\mathbf{x}_j^{(1:n)})^{\mathrm{T}}\mathbf{S}\mathbf{x}_j^{(1:n)} &\geq 0
\end{aligned}
$$

Therefore the augmented normalized Laplacian matrix $\mathbf{S}$ is a symmetric positive semidefinite matrix.

21.1.1 Spectral-based GCNs

Background. Spectral-based methods have a solid mathematical foundation in graph signal processing, They assume graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to be undirected. The symmetric augmented normalized graph Laplacian matrix is a mathematical representation of an undirected graph, defined as

$$
\begin{aligned}
&\mathbf{S} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{L}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} \left(\tilde{\mathbf{D}} - \tilde{\mathbf{A}} \right) \tilde{\mathbf{D}}^{-\frac{1}{2}} \quad (\ \tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}} \text{ is the Laplacian matrix corresponding Argumented Adjacency matrix } \tilde{\mathbf{A}}\ ) \\
&= \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{D}} \tilde{\mathbf{D}}^{-\frac{1}{2}} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \\
&= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \\
&= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \tilde{\mathbf{D}}^{-\frac{1}{2}}
\end{aligned}
$$

where:

- $\mathbf{A} \in \mathbb{R}^{n \times n}$: Adjacency matrix of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
  - $(v_i, v_j) \in \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} \neq 0$
  - $(v_i, v_j) \notin \mathcal{E} \Leftrightarrow \mathbf{A}_{ij} = 0$
  - $\mathbf{A}_{ii} = 0, \forall i \in \{1, \cdots, n\}$
- $\mathbf{I} \in \mathbb{R}^{n \times n}$: Identity matrix containing identity self-loops
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$: Augmented adjacency matrix, namely adjacency matrix with self-loops
  - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij}, \forall i, j \in \{1, \cdots, n\}$
  - $\tilde{\mathbf{A}}_{ii} = \mathbf{A}_{ii} + \mathbf{I}_{ii} = \mathbf{A}_{ii} + 1, \forall i \in \{1, \cdots, n\}$
  - $\tilde{\mathbf{A}}_{ij} = \mathbf{A}_{ij} + \mathbf{I}_{ij} = \mathbf{A}_{ij}, \forall i, j \in \{1, \cdots, n\}, i \neq j$
- $\mathbf{D} = \text{diag}(d_1, \cdots, d_n)$: Degree matrix corresponding $\mathbf{A}$
  - $d_i = \mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{A}_{ij}, \forall i \in \{1, \cdots, n\}$
- $\tilde{\mathbf{D}} = \text{diag}(\tilde{d}_1, \cdots, \tilde{d}_n)$: Augmented degree matrix corresponding $\tilde{\mathbf{A}}$
  - $\tilde{d}_i = \tilde{\mathbf{D}}_{ii} = \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij} = \sum_{j=1}^{n} (\mathbf{A}_{ij} + \mathbf{I}_{ij})$

The normalized graph Laplacian matrix possesses the property of being real symmetric positive semidefinite. With this property, the normalized Laplacian matrix can be factored (eigendecomposition) as:

$$\mathbf{S} = \mathbf{U}\ \mathbf{U}^{\mathrm{T}}$$

where $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_n] \in \mathbb{R}^{n \times n}$ is the matrix of eigenvectors ordered by eigenvalues and $\ = \text{diag}(\lambda_1, \cdots, \lambda_n)$ is the diagonal matrix of eigenvalues (spectrum), $_{ii} = \lambda_i$.

The eigenvectors of the normalized Laplacian matrix form an orthonormal space, in mathematical words $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$. In graph signal processing, a graph signal $\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$ (value of all nodes in a specific feature dimension) is a feature vector of all nodes of a graph, where $x_j^{(i)}$ is the value of j^{th} dimension of the i^{th} node.

The graph Fourier transform to a signal $\mathbf{x}_j^{(1:n)} \in \mathbb{R}^n$ is defined as $\mathfrak{F}(\mathbf{x}_j^{(1:n)}) = \mathbf{U}^{\mathrm{T}}\mathbf{x}_j^{(1:n)}$ and the inverse graph Fourier transform is defined as $\mathfrak{F}^{-1}(\hat{\mathbf{x}}_j^{(1:n)}) = \mathbf{U}\hat{\mathbf{x}}_j^{(1:n)}$, where $\hat{\mathbf{x}}_j^{(1:n)}$ represents the resulted signal from the graph Fourier transform.

The graph Fourier transform projects the input graph signal to the orthonormal space where the basis is formed by eigenvectors of the normalized graph Laplacian $\mathbf{S}$. Elements of the transformed signal $\hat{\mathbf{x}}$ are the coordinates of the graph signal in the new space so that the input signal can be represented as $\mathbf{x} = \sum_{i=1}^{n} \hat{x}_i \mathbf{u}_i$, which is exactly the inverse graph Fourier transform. Now the graph convolution of the input signal $\mathbf{x}$ with a filter $\mathbf{g} \in \mathbb{R}^n$ is defined as

$$\begin{aligned}\mathbf{x} * \mathbf{g} &= \mathfrak{F}^{-1}\left(\mathfrak{F}(\mathbf{x}) \odot \mathfrak{F}(\mathbf{g})\right) \\ &= \mathbf{U}\left(\mathbf{U}^{\mathrm{T}}\mathbf{x} \odot \mathbf{U}^{\mathrm{T}}\mathbf{g}\right)\end{aligned}$$

where $\odot$ denotes the element-wise product and $*$ is a custom notation for graph convolution operation. If we denote a filter as $\mathbf{g}_\theta = \mathrm{diag}(\mathbf{U}^{\mathrm{T}}\mathbf{g}) \in \mathbb{R}^{n \times n}$, then the spectral graph convolution is simplified as

$$\mathbf{x} * \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^{\mathrm{T}}\mathbf{x}$$

Spectral-based GCNs all follow this definition. The key difference lies in the choice of the filter $\mathbf{g}_\theta$.

Spectral Convolutional Neural Network (Spectral CNN).

Reading materials.

- QiMai's Home: Laplacian Smoothing and Graph Convolutional Networks
- Simplifying Graph Convolutional Networks

21.1.2 Spatial-based GCNs

Analogous to the convolutional operation of a conventional CNN on an image, spatial-based methods define graph convolutions based on a node ' s spatial relations. According to paper: Simplifying Graph Convolutional Networks[?], similar to CNNs or MLPs, GCNs learn a new feature representation $\mathbf{h}_i^{(k)}$ for the feature $\mathbf{x}_i$ of each node v_i over multiple layers, which is subsequently used as input into a linear classifier. We denote as follows:

- $\mathbf{H}^{(k-1)} \in \mathbb{R}^{n \times d'}$: input node representations of all nodes (a matrix).
- $\mathbf{H}^{(k)} \in \mathbb{R}^{n \times d''}$: output node representations of all nodes (a matrix).

Naturally, the initial node representations are just the original input features:

$$\mathbf{H}^{(0)} = \mathbf{X}^{\mathrm{T}} \in \mathbb{R}^{n \times d}$$

which serve as input to the first GCN layer. A K-layer GCN is identical to applying a K-layer MLP to the feature vector $\mathbf{x}_i$ of each node in the graph, except that **the hidden representation of each node is averaged with its neighbors at the beginning of each layer**. In each graph convolution layer, node representations are updated in three stages:

1. Feature propagation
2. Feature (Linear) transformation
3. Pointwise nonlinear activation

For the sake of clarity, we describe each step in detail.

Feature propagation Feature propagation is what distinguishes a GCN from an MLP. At the beginning of each layer the features $\mathbf{h}_i$ of each node v_i are averaged with the feature vectors in its local neighborhood (**local smoothing**),

$$\bar{\mathbf{h}}_i^{(k)} \leftarrow \frac{1}{d_i + 1}\mathbf{h}_i^{(k-1)} + \sum_{j=1}^{n} \frac{a_{ij}}{\sqrt{(d_i + 1)\,(d_j + 1)}}\mathbf{h}_j^{(k-1)}$$

More compactly, we can express this update over the entire graph as a simple matrix operation. Let $\mathbf{S} \in \mathbb{R}^{n \times n}$ denote the symmetric normalized adjacency matrix with added self-loops,

$$\mathbf{S} = \mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\mathbf{D}$ is the degree matrix corresponding $\tilde{\mathbf{A}}$. The simultaneous update in above update equation for all nodes becomes a simple sparse matrix multiplication

$$\bar{\mathbf{H}}^{(k)} \leftarrow \mathbf{S}\mathbf{H}^{(k-1)}.$$

Intuitively, **this step smoothes the hidden representations locally along the edges of the graph and ultimately encourages similar predictions among locally connected nodes**.

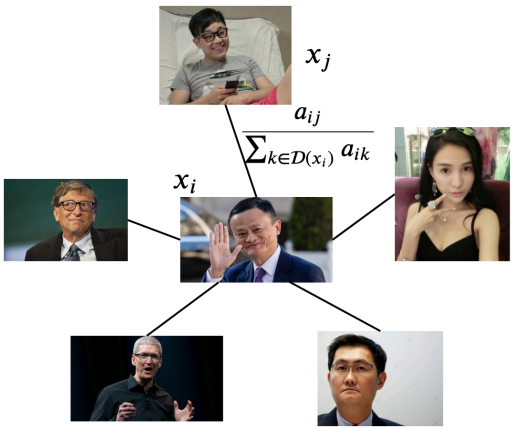


图 103: Social network examples for predicting annual salary informations. Using symmetric nomalized edge weight $\frac{\tilde{\mathbf{A}}_{ij}}{(\sum_{k=1}^{n}\tilde{\mathbf{A}}_{ik})^{\frac{1}{2}}(\sum_{l=1}^{n}\tilde{\mathbf{A}}_{jl})^{\frac{1}{2}}}$ to replace asymmetric normalized edge weight $\frac{\tilde{\mathbf{A}}_{ij}}{\sum_{k=1}^{n}\tilde{\mathbf{A}}_{ik}}$.

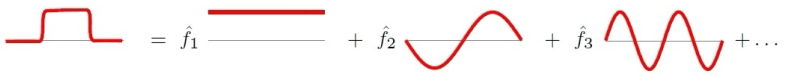


图 104: Fourier transform for a specific function f.

图 105: Schematic layout of a GCN v.s. a SGC. Top row: The GCN transforms the feature vectors repeatedly throughout K layers and then applies a linear classifier on the final representation. Bottom row: the SGC reduces the entire procedure to a simple feature propagation step followed by standard logistic regression. Image source: https://arxiv.org/pdf/1902.07153.pdf.

Proof.

$$
\begin{aligned}
\bar{\mathbf{H}}^{(k)} &= \begin{bmatrix} \bar{\mathbf{h}}_1^{(k)} \\ \vdots \\ \bar{\mathbf{h}}_i^{(k)} \\ \vdots \\ \bar{\mathbf{h}}_n^{(k)} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{d_1+1}\mathbf{h}_1^{(k-1)} + \sum_{j=1}^{n} \frac{a_{1j}}{\sqrt{(d_1+1)(d_j+1)}}\mathbf{h}_j^{(k-1)} \\ \vdots \\ \frac{1}{d_i+1}\mathbf{h}_i^{(k-1)} + \sum_{j=1}^{n} \frac{a_{ij}}{\sqrt{(d_i+1)(d_j+1)}}\mathbf{h}_j^{(k-1)} \\ \vdots \\ \frac{1}{d_n+1}\mathbf{h}_n^{(k-1)} + \sum_{j=1}^{n} \frac{a_{nj}}{\sqrt{(d_n+1)(d_j+1)}}\mathbf{h}_j^{(k-1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{d_1+1}\mathbf{h}_1^{(k-1)} \\ \vdots \\ \frac{1}{d_i+1}\mathbf{h}_i^{(k-1)} \\ \vdots \\ \frac{1}{d_n+1}\mathbf{h}_n^{(k-1)} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{n} \frac{a_{1j}}{\sqrt{(d_1+1)(d_j+1)}}\mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^{n} \frac{a_{ij}}{\sqrt{(d_i+1)(d_j+1)}}\mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^{n} \frac{a_{nj}}{\sqrt{(d_n+1)(d_j+1)}}\mathbf{h}_j^{(k-1)} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{d_1+1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{d_i+1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \frac{1}{d_n+1} \end{bmatrix} \begin{bmatrix} \mathbf{h}_1^{(k-1)} \\ \vdots \\ \mathbf{h}_i^{(k-1)} \\ \vdots \\ \mathbf{h}_n^{(k-1)} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{n} (d_1+1)^{-\frac{1}{2}} a_{1j} (d_j+1)^{-\frac{1}{2}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^{n} (d_i+1)^{-\frac{1}{2}} a_{ij} (d_j+1)^{-\frac{1}{2}} \mathbf{h}_j^{(k-1)} \\ \vdots \\ \sum_{j=1}^{n} (d_n+1)^{-\frac{1}{2}} a_{nj} (d_j+1)^{-\frac{1}{2}} \mathbf{h}_j^{(k-1)} \end{bmatrix} \\
&= \mathbf{D}^{-1}\mathbf{H}^{(k-1)} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(k-1)} \\
&= \left(\mathbf{D}^{-1} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{H}^{(k-1)} \\
&= \left(\mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{-\frac{1}{2}} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{H}^{(k-1)} \quad (\because \mathbf{D} \text{ is a diagonal matrix } \therefore \mathbf{D}^{-1} = \mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{-\frac{1}{2}}) \\
&= \left(\mathbf{D}^{-\frac{1}{2}}\mathbf{I}\mathbf{D}^{-\frac{1}{2}} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{H}^{(k-1)} \\
&= \mathbf{D}^{-\frac{1}{2}}\left(\mathbf{A}+\mathbf{I}\right)\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(k-1)} \\
&= \mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(k-1)} \\
&= \mathbf{S}\mathbf{H}^{(k-1)}
\end{aligned}
$$

Feature transformation and and nonlinear activation After the local smoothing, a GCN layer is identical to a standard MLP. Each layer is associated with a learned weight $^{(k)}$ and the smoothed hidden feature representations are transformed linearly. Finally, a nonlinear activation function such as ReLU is applied pointwise before outputting feature representation $\mathbf{H}^{(k)}$. In summary, the representation updating rule of the k-th layer is:

$$\mathbf{H}^{(k)} \leftarrow \text{ReLU}\left(\bar{\mathbf{H}}^{(k)}\ ^{(k)}\right)$$

The pointwise nonlinear transformation of the k-th layer is followed by the feature propagation of the $(k+1)$-th layer. And $\mathbf{H}^{(K)} \in \mathbb{R}^{n \times d_{\text{embed}}}$ is the final embeddings of all nodes.

21.2 DeepWalk: Online Learning of Social Representations（未完成）

21.2.1 Introduction

本节来自论文：DeepWalk: Online Learning of Social Representations[?]

- 目的：学习图（Graph）中每个节点（Vertex）的社交表达（social representations），其中 social representations 实际上是 latent vector representations

21.2.2 Model formulation

Training procedure DeepWalk 算法伪代码如下：

Algorithm 24: DeepWalk$(G, \omega, d, \gamma, t)$[**?**]

Input: graph $G(V, E)$;
window size ω;
embedding size d;
walks per vertex γ;
walk length t.

Output: matrix of vertex representations $\mathbf{\Phi} \in \mathbb{R}^{|V| \times d}$

Initialization: Sample $\mathbf{\Phi}$ from $\mathcal{U}^{|V| \times d}$
Build a binary tree T from V
for $i = 0$ *to* γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for ***each*** $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = \text{RandomWalk}(G, v_i, t)$
    SkipGram$(\mathbf{\Phi}, \mathcal{W}_{v_i}, \omega)$
  end
end

Algorithm 25: SkipGram$(\mathbf{\Phi}, \mathcal{W}_{v_i}, \omega)$

for ***each*** $v_j \in \mathcal{W}_{v_i}$ **do**
  for do
    for-block
  end
end

21.3 LINE: Large-scale Information Network Embedding（未完成）

21.3.1 Introduction

本节来自论文：LINE: Large-scale Information Network Embedding[?]

21.4 Node2vec: Scalable Feature Learning for Networks（未完成）

21.4.1 Introduction

本节来自论文：Node2vec: Scalable Feature Learning for Networks[?]

21.5 SDNE: Structural Deep Network Embedding（未完成）

21.5.1 Introduction

本节来自论文：Structural Deep Network Embedding[**?**]

21.6 Struc2vec: Learning Node Representations from Structural Identity（未完成）

21.6.1 Introduction

本节来自论文：struc2vec: Learning Node Representations from Structural Identity[?]

21.7 Survey and Summary（未完成）

21.7.1 Introduction

22 Attention Mechanisms

22.1 Self-Attention

22.1.1 Self-attention mechanisms in Computer Vision

在Self-Attention GAN (SAGAN; Zhang et al., 2018)[?] 中

22.1.2 Self-attention mechanisms in Natural Language Processing

在Attention is all you need (Transformer; Vaswani et al., 2017)[?] 中

22.1.3 Reference

- Lil'Log: Attention? Attention![?]

23 Network Optimization and Regularization

23.1 Exponential Moving Average (EMA)

令 t 时刻 EMA 一阶动量为 v_t

23.2 Activation Function

23.2.1 Logistic function (Sigmoid function)

Forwrd-propagation

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Back-propagation

$$\begin{aligned}
\sigma'(x) &= -\frac{1}{(1+e^{-x})^2} \cdot \left(e^{-x}\right) \cdot (-1) \\
&= \frac{e^{-x}}{(1+e^{-x})(1+e^{-x})} \\
&= \frac{e^{-x}}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} \\
&= \frac{(1+e^{-x})-1}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} \\
&= \left(1 - \frac{1}{1+e^{-x}}\right) \cdot \frac{1}{1+e^{-x}} \\
&= (1-\sigma(x)) \cdot \sigma(x) \quad (i.e.\ \sigma(x) = \frac{1}{1+e^{-x}})
\end{aligned}$$

Summary

- Logistic 激活函数可视为一个“挤压”函数，其将实数范围内的输入 $x \in \mathbb{R}$ 挤压到 $(0,1)$ 空间；
- 当输入 x 在 0 附近时，sigmoid 型函数近似为线性函数；
- 当输入 $x \to -\infty$ 和 $x \to +\infty$ 时候，输出越接近 0 和 1。

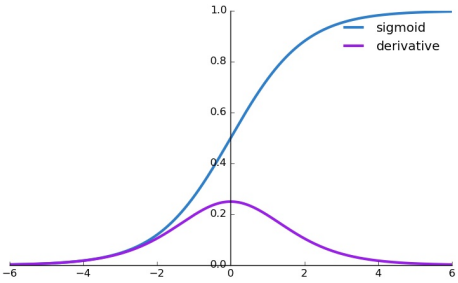


图 106: logistic function and its derivative function. Image source: Ronny Restrepo: Derivative of the Sigmoid function - a worked example

23.2.2 Tanh function

Forwrd-propagation

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},\ x \in \mathbb{R}$$

Proof. tanh 函数和 sigmoid 函数的关系推导如下：

$$\begin{aligned}
\tanh(x) &= \frac{e^x - e^{-x} + (e^x + e^{-x}) - (e^x + e^{-x})}{e^x + e^{-x}} \\
&= \frac{(e^x - e^{-x} + e^x + e^{-x}) - (e^x + e^{-x})}{e^x + e^{-x}} \\
&= \frac{2e^x}{e^x + e^{-x}} - 1 \\
&= \frac{2}{1 + e^{-2x}} - 1 \\
&= 2\sigma(2x) - 1 \quad (i.e.\ \sigma(x) = \frac{1}{1 + e^{-x}})
\end{aligned}$$

因此可以说 tanh 函数可视为经过放大和平移的 sigmoid 函数，其值域为：$\tanh(x) \in (-1, 1)$。

Back-propagation

$$\begin{aligned}
\tanh'(x) &= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\
&= \frac{\left(e^{2x} + 2 + e^{-2x}\right) - \left(e^{2x} - 2 + e^{-2x}\right)}{(e^x + e^{-x})^2} \\
&= \frac{4}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})^2 - (e^x + e^{-x})^2 + 4}{(e^x + e^{-x})^2} \\
&= 1 - \frac{(e^x + e^{-x})^2 - 4}{(e^x + e^{-x})^2} \\
&= 1 - \frac{e^{2x} + 2e^x e^{-x} + e^{-2x} - 4}{(e^x + e^{-x})^2} \\
&= 1 - \frac{e^{2x} + e^{-2x} - 2}{(e^x + e^{-x})^2} \\
&= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
&= 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2 \\
&= 1 - \big(\tanh(x)\big)^2
\end{aligned}$$

Summary Tanh 函数相对于 Logistic 函数作为激活函数的优势是，Tanh 函数的输出是 0 中心（zero-centered）的，而 Logistic 函数的输出恒大于 0。非零中心化的输出会使得其后一层的神经元的输入发生偏置偏移（bias shift），并进一步使得梯度下降的收敛速度变慢。

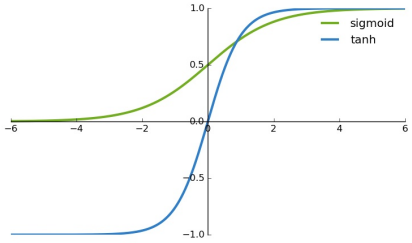


图 108: tanh activation function vs sigmoid activation function. Image source: Ronny Restrepo: Calculating the derivative of Tanh - step by step

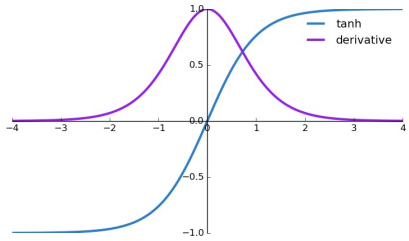


图 109: tanh function and its derivative function. Image source: Ronny Restrepo: Calculating the derivative of Tanh - step by step

23.2.3 ReLU function

论文出处：Deep Sparse Rectifier Neural Networks[?]Deep Learning using Rectified Linear Units (ReLU)[?]。

Forwrd-propagation

$$\begin{aligned}\mathrm{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\ &= \max(0, x)\end{aligned}$$

Back-propagation

$$\mathrm{ReLU}'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Summary

- 优点：
  1. 拥有生物上的解释性（e.g. 单侧抑制、宽兴奋边界）。在生物神经网络中，同时处于兴奋状态的神经元非常稀疏。人脑中在同一时刻大概只有 1% 4% 的神经元处于活跃状态。Sigmoid 型激活函数会导致一个非稀疏的神经网络，而 ReLU 却具有很好的稀疏性，大约 50% 的神经元会处于激活状态。
  2. 在优化方面，相比于 Sigmoid 型函数的两端饱和，ReLU 函数为左饱和函数，且在 $x > 0$ 时导数为 1，因此在一定程度上缓解了神经网络的梯度消失问题，加速梯度下降的收敛速度。
- 缺点：
  1. 输出是非零中心化的，给后一层的神经网络引入偏置偏移，会影响梯度下降的效率
  2. ReLU 神经元在训练时比较容易“死亡”。在训练时，如果参数在一次不恰当的更新后，第一个隐藏层中的某个 ReLU 神经元在所有的训练数据上都不能被激活，那么这个神经元自身参数的梯度永远都会是 0，在以后的训练过程中永远不能被激活。这种现象称为死亡 ReLU 问题（Dying ReLU Problem），并且也有可能会发生在其它隐藏层。因此实际中为了避免上述情况，有几种 ReLU 的变种也会被广泛使用。

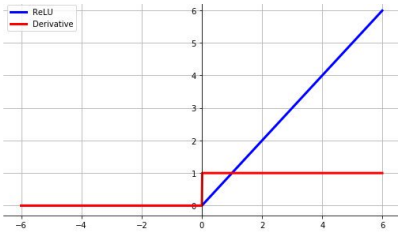


图 109: relu function and its derivative function.

23.2.4 Leaky-ReLU function

论文出处：Empirical Evaluation of Rectified Activations in Convolution Network[?]。

23.2.5 Parametric-ReLU function

论文出处：Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification[?]。

23.3 Network Initialization

23.3.1 Xavier initialization

论文出处：Understanding the difficulty of training deep feedforward neural networks[?]。

Analytics 从正向传播和反向传播两个方向分析单一链路上神经元的均值和方差。

Analytics of forward-propagation 正向传播中，假设第 l 层的某一个(第 j 个)隐层神经元 $z_j^{(l)}$，接收前一层(第 $l-1$ 层)共 $n^{(l-1)}$ 个神经元的输出 $a_i^{(l-1)}, i \in \{1, ..., n^{(l-1)}\}$，对应线路上的权重为 $w_{i,j}^{(l-1,l)}, i \in \{1, ..., n^{(l-1)}\}$，即：

$$z_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l-1,l)} a_i^{(l-1)}$$

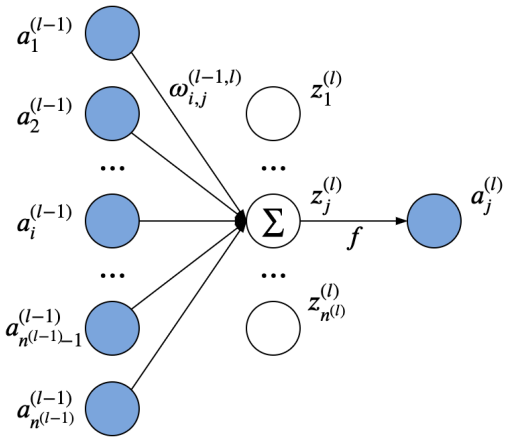


图 110: 正向传播中典型的神经元关系结构

为了避免初始化参数 $\boldsymbol{w}_{:,j}^{(l-1,l)} = \{w_{1,j}^{(l-1,l)}, ..., w_{i,j}^{(l-1,l)}, ..., w_{n^{(l-1)},j}^{(l-1,l)}\}$ 使得激活值 $a_j^{(l)} = f(z_j^{(l)})$ 变得饱和，我们需要**尽量使得 $z_j^{(l)} \in \mathbb{R}$ 处于激活函数的线性区间**，也就是其绝对值 $|z_j^{(l)}|$ 相对较小，此时该神经元的激活函数值满足：

$$a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)}$$

假设在初始化时，参数 $w_{i,j}^{(l-1,l)}$ 和输入 $a_i^{(l-1)}$ 的均值都为 0，并且相互独立，则激活输出 $a_j^{(l)}$ 的均值为：

$$\begin{aligned}
\because \mathbb{E}\left[z_j^{(l)}\right] &= \mathbb{E}\left[\sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right] \\
&= \sum_{i=1}^{n^{(l-1)}} \mathbb{E}\left[w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right] \quad (\because w_{i,j}^{(l-1,l)} a_i^{(l-1)}, \forall i \in \{1, ..., n^{(l-1)}\} \text{ 之间相互独立}) \\
&= \sum_{i=1}^{n^{(l-1)}} \mathbb{E}\left[w_{i,j}^{(l-1,l)}\right] \mathbb{E}\left[a_i^{(l-1)}\right] \quad (\because w_{i,j}^{(l-1,l)} \text{ 和 } a_i^{(l-1)} \text{ 相互独立})
\end{aligned}$$

$$= \sum_{i=1}^{n^{(l-1)}} 0 \times 0 = 0 \quad (\because \mathbb{E}\left[w_{i,j}^{(l-1,l)}\right] = 0, \mathbb{E}\left[a_i^{(l-1)}\right] = 0)$$

$$\therefore \mathbb{E}\left[a_j^{(l)}\right] \approx \mathbb{E}\left[z_j^{(l)}\right] = 0 \quad (\because \text{初始化时} a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)})$$

同时激活输出 $a_j^{(l)}$ 的方差为：

$$\because \operatorname{Var}\left[z_j^{(l)}\right] = \operatorname{Var}\left[\sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right]$$

$$= \sum_{i=1}^{n^{(l-1)}} \operatorname{Var}\left[w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right] \quad (\because w_{i,j}^{(l-1,l)} a_i^{(l-1)}, \forall i \in \{1, ..., n^{(l-1)}\} \text{ 之间相互独立})$$

$$\because \operatorname{Var}\left[w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right] = \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right]\right)^2 \quad (\because \operatorname{Var}[\mathcal{X}] = \mathbb{E}\left[\mathcal{X}^2\right] - (\mathbb{E}[\mathcal{X}])^2)$$

$$= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2 \left(a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right]\right)^2$$

$$= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right] \mathbb{E}\left[\left(a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)}\right] \mathbb{E}\left[a_i^{(l-1)}\right]\right)^2 \quad (\because w_{i,j}^{(l-1,l)} \text{ 和 } a_i^{(l-1)} \text{ 相互独立})$$

$$= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right] \mathbb{E}\left[\left(a_i^{(l-1)}\right)^2\right] - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)}\right]\right)^2 \left(\mathbb{E}\left[a_i^{(l-1)}\right]\right)^2 \quad (\because (ab)^2 = a^2 b^2, a, b \in \mathbb{R})$$

$$= \underbrace{\left(\operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] + \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)}\right]\right)^2\right)}_{\mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right]} \underbrace{\left(\operatorname{Var}\left[a_i^{(l-1)}\right] + \left(\mathbb{E}\left[a_i^{(l-1)}\right]\right)^2\right)}_{\mathbb{E}\left[\left(a_i^{(l-1)}\right)^2\right]}$$

$$- \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)}\right]\right)^2 \left(\mathbb{E}\left[a_i^{(l-1)}\right]\right)^2 \quad (\because \operatorname{Var}[\mathcal{X}] = \mathbb{E}\left[\mathcal{X}^2\right] - (\mathbb{E}[\mathcal{X}])^2 \therefore \mathbb{E}\left[\mathcal{X}^2\right] = \operatorname{Var}[\mathcal{X}] + (\mathbb{E}[\mathcal{X}])^2)$$

$$= \left(\operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] + 0^2\right)\left(\operatorname{Var}\left[a_i^{(l-1)}\right] + 0^2\right) - 0^2 \times 0^2 \quad (\because \mathbb{E}\left[w_{i,j}^{(l-1,l)}\right] = 0, \mathbb{E}\left[a_i^{(l-1)}\right] = 0)$$

$$= \operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] \operatorname{Var}\left[a_i^{(l-1)}\right]$$

$$\therefore \operatorname{Var}\left[z_j^{(l)}\right] = \sum_{i=1}^{n^{(l-1)}} \operatorname{Var}\left[w_{i,j}^{(l-1,l)} a_i^{(l-1)}\right]$$

$$= \sum_{i=1}^{n^{(l-1)}} \left(\operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] \operatorname{Var}\left[a_i^{(l-1)}\right]\right)$$

$$= n^{(l-1)} \operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] \operatorname{Var}\left[a_i^{(l-1)}\right] \quad (\because \text{所有 } w_{i,j}^{(l-1,l)}, \forall i \in \{1, ..., n^{(l-1)}\} \text{ 来自相同分布，同理 } a_i^{(l-1)})$$

$$\therefore \operatorname{Var}\left[a_j^{(l)}\right] \approx \operatorname{Var}\left[z_j^{(l)}\right]$$

$$= n^{(l-1)} \operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] \operatorname{Var}\left[a_i^{(l-1)}\right] \quad (\because \text{初始化时} a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)})$$

也就是说，前向推理过程中，输入信号 $a_i^{(l-1)}$ 的方差经过该**激活输出**神经元后被**放大或缩小**了 $n^{(l-1)}\operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right]$ 倍。为了保障在初始阶段，信号在经过多层网络传递的稳定性，即**信号不被过分放大也不被过分减弱**，需要尽可能地保障经过权重 $w_{i,j}^{l-1,l}$ 加权作用和激活函数 $f(\cdot)$ 激活作用前后的神经元信号 $a_i^{(l-1)}$ 和 $a_j^{(l)}$ 的方差一致，即：

$$\operatorname{Var}\left[a_j^{(l)}\right] = \operatorname{Var}\left[a_i^{(l-1)}\right] \implies n^{(l-1)} \operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] = 1 \implies \operatorname{Var}\left[w_{i,j}^{(l-1,l)}\right] = \frac{1}{n^{(l-1)}}$$

Analytics of back-propagation 同理反向传播过程中，针对 $\mathcal{L} \to a_j^{(l)} \to\to z_j^{(l)} \to a_i^{(l-1)}\ \forall j \in \{1, ..., n^{(l)}\}$ 链路：

$$\begin{aligned}\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}} &= \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \cdot \frac{\mathrm{d}a_j^{(l)}}{\mathrm{d}z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial a_i^{(l-1)}} \right) \quad (\mathcal{D}\text{: mini-batch dataset}) \\ &\approx \sum_{j=1}^{n^{(l)}} \left(\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \cdot 1 \cdot w_{i,j}^{(l-1,l)} \right) \quad (\because \text{初始化时 } a_j^{(l)} = f(z_j^{(l)}) \approx z_j^{(l)}) \\ &= \sum_{j=1}^{n^{(l)}} \left(w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)\end{aligned}$$

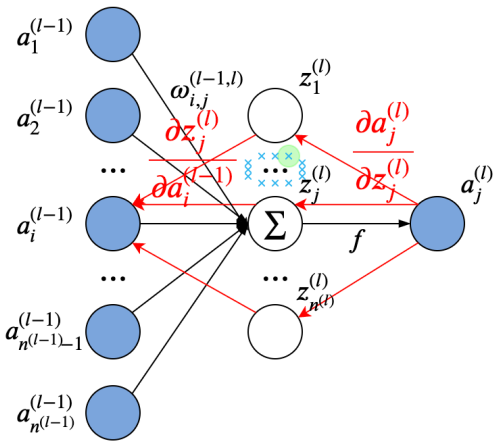


图 111: 反向传播中典型的神经元关系结构

因此在初始化阶段，分析均值：

$$\begin{aligned}\mathbb{E}\left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}}\right] &= \mathbb{E}\left[\sum_{j=1}^{n^{(l)}} \left(w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)\right] \\ &= \sum_{j=1}^{n^{(l)}} \mathbb{E}\left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \quad (\because \text{反向传播多链路之间相互独立}) \\ &= \sum_{j=1}^{n^{(l)}} \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)} \right] \mathbb{E}\left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \right) \quad (\because w_{i,j}^{(l-1,l)} \text{ 和 } \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \text{ 相互独立}) \\ &= n^{(l)} \cdot 0 \cdot \mathbb{E}\left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] = 0 \quad (\because \mathbb{E}\left[w_{i,j}^{(l-1,l)} \right] = 0)\end{aligned}$$

分析方差：

$$\begin{aligned}\mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_i^{(l-1)}}\right] &= \mathrm{Var}\left[\sum_{j=1}^{n^{(l)}} \left(w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right)\right] \\ &= \sum_{j=1}^{n^{(l)}} \mathrm{Var}\left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta; \mathcal{D})}{\partial a_j^{(l)}} \right] \quad (\because \text{反向传播多链路相互独立})\end{aligned}$$

针对反向传播中的单一链路：

$$
\begin{aligned}
\because \ \mathrm{Var}\left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right] &= \mathbb{E}\left[\left(\sum_{j=1}^{n^{(l)}} w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right)^2\right] - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]\right)^2 \\
&= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2 \left(\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right)^2\right] - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]\right)^2 \\
&= \mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right] \mathbb{E}\left[\left(\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right)^2\right] - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)}\right] \mathbb{E}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]\right)^2 \\
&= \underbrace{\left(\mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] + \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)}\right]\right)^2\right)}_{\mathbb{E}\left[\left(w_{i,j}^{(l-1,l)}\right)^2\right]} \underbrace{\left(\mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right] + \left(\mathbb{E}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]\right)^2\right)}_{\mathbb{E}\left[\left(\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right)^2\right]} \\
&\quad - \left(\mathbb{E}\left[w_{i,j}^{(l-1,l)}\right]\right)^2 \left(\mathbb{E}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]\right)^2 \\
&= \left(\mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] + 0\right)\left(\mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right] + 0\right) - 0 \times \left(\mathbb{E}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]\right)^2 \\
&= \mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] \mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right] \\
\therefore \ \mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_i^{(l-1)}}\right] &= \sum_{j=1}^{n^{(l)}} \mathrm{Var}\left[w_{i,j}^{(l-1,l)} \cdot \frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right] \\
&= \sum_{j=1}^{n^{(l)}} \left(\mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] \mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]\right) \\
&= n^{(l)} \mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] \mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right]
\end{aligned}
$$

因此为了保障反向传播信号的稳定性，需要：

$$
\begin{aligned}
\mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_i^{(l-1)}}\right] = \mathrm{Var}\left[\frac{\partial \mathcal{L}(\theta;\mathcal{D})}{\partial a_j^{(l)}}\right] &\Rightarrow n^{(l)} \mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] = 1 \\
&\Rightarrow \mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] = \frac{1}{n^{(l)}}
\end{aligned}
$$

Analytics of both forward-propagation and back-propagation 作为折中，**同时考虑信号在正向传播和反向传播中都不被放大或缩小**，可以设置：

$$
\mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] = \frac{2}{n^{(l-1)} + n^{(l)}}
$$

因此计算出了信号在正向传播和反向传播中都不被放大或缩小时参数理想的方差，而后可以通过高斯分布或均匀分布来随机初始化参数。

- 高斯分布初始化方法：

$$
w_{i,j}^{(l-1,l)} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{(l-1)} + n^{(l)}}}\right)
$$

- 均匀分布初始化方法：

$$w_{i,j}^{(l-1,l)} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n^{(l-1)}+n^{(l)}}}, \sqrt{\frac{6}{n^{(l-1)}+n^{(l)}}}\right)$$

Proof. 因为随机变量 x 在区间 $[a,b]$ 的方差为：

$$\mathrm{Var}(x) = \frac{(b-a)^2}{12}$$

所以，当采用区间为 $[-r,r]$ 的均匀分布来初始化参数 $w_{i,j}^{(l-1,l)}$，并满足 $\mathrm{Var}\left[w_{i,j}^{(l-1,l)}\right] = \frac{2}{n^{(l-1)}+n^{(l)}}$，那么可解得 r 的取值为：

$$\begin{aligned}\frac{4r^2}{12} = \frac{2}{n^{(l-1)}+n^{(l)}} \;&\Rightarrow\; r^2 = \frac{6}{n^{(l-1)}+n^{(l)}} \\ &\Rightarrow\; r = \sqrt{\frac{6}{n^{(l-1)}+n^{(l)}}}\end{aligned}$$

23.3.2 He initialization

论文出处：Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification[?]。该方法和的区别是：

- He initialization在 **ReLU 型**激活函数选择下表现的更好。
- Xavier initialization在 **Sigmoid 型**激活函数的选择下表现的更好。

Analytics 考虑 ReLU 型激活函数，形式如下：

$$a_j^{(l)} = f(z_j^{(l)}) = \begin{cases} z_j^{(l)}, & \text{if } z_j^{(l)} > 0 \\ \alpha_j^{(l)} z_j^{(l)}, & \text{if } z_j^{(l)} \leq 0 \end{cases}$$

其中：

- ReLU[?]: $\alpha_j^{(l)} = 0$, constant.
- LeakyReLU[?]: $\alpha_j^{(l)} > 0$, constant.
- Parametric-ReLU (PReLU)[?]: $\alpha_j^{(l)} \in \mathbb{R}$, variable.

Analytics of forward-propagation

Analytics of back-propagation

23.4 Network Normalization

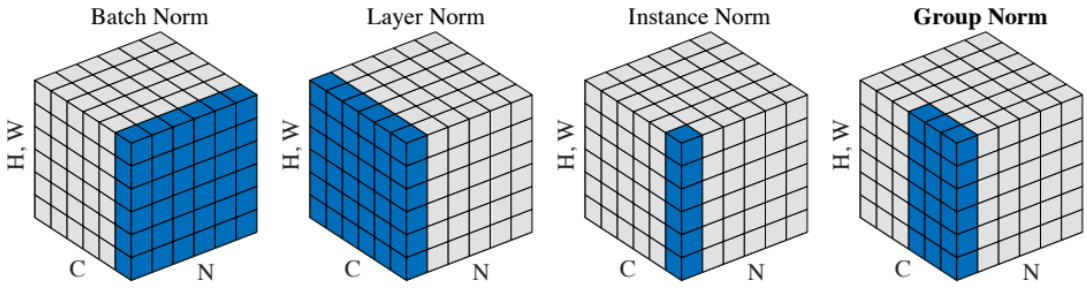


图 112: **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and(H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

23.4.1 Batch Normalization (BN)

选自论文Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[?]

Methodology 伪代码如下：

Algorithm 26: Batch Normalizing Transformation[?]

Input: Value of $\boldsymbol{x} \in \mathbb{R}^d$ over a mini-batch: $\mathcal{D} = \{\boldsymbol{x^{(i)}}\}_{i=1}^m$ (d is the number of units, m is the number of batch samples);
Parameters to be learned: $\boldsymbol{\gamma} \in \mathbb{R}^d$, $\boldsymbol{\beta} \in \mathbb{R}^d$.

Output: $\{\boldsymbol{y^{(i)}} = \mathbf{BN}_{\gamma,\beta}(\boldsymbol{x^{(i)}})\}_{i=1}^m$, $\boldsymbol{y^{(i)}} \in \mathbb{R}^d$

1 Calculate mini-batch mean: $\boldsymbol{\mu_{\mathcal{D}}} \leftarrow \frac{1}{m}\sum_{i=1}^m \boldsymbol{x^{(i)}}$, $\boldsymbol{\mu_{\mathcal{D}}} \in \mathbb{R}^d$

2 Calculate mini-batch variance: $\boldsymbol{\sigma^2_{\mathcal{D}}} \leftarrow \frac{1}{m}\sum_{i=1}^m (\boldsymbol{x^{(i)}} - \boldsymbol{\mu_{\mathbf{B}}})^2$, $\boldsymbol{\sigma^2_{\mathcal{D}}} \in \mathbb{R}^d$

3 Normalization: $\widehat{\boldsymbol{x^{(i)}}} \leftarrow \frac{\boldsymbol{x^{(i)}} - \boldsymbol{\mu_{\mathcal{D}}}}{\sqrt{\boldsymbol{\sigma^2_{\mathcal{D}}} + \epsilon}}$, $\widehat{\boldsymbol{x^{(i)}}} \in \mathbb{R}^d$ (Division and square root applied element-wise)

4 Scale & Shift: $\boldsymbol{y^{(i)}} \leftarrow \boldsymbol{\gamma} \odot \widehat{\boldsymbol{x^{(i)}}} + \boldsymbol{\beta} \equiv \mathbf{BN}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\boldsymbol{x^{(i)}})$, $\boldsymbol{y^{(i)}} \in \mathbb{R}^d$

算法解释

- Batch Normalization 是**针对列**（batch dataset 中每个独立的 unit）进行的操作。
- Batch Normalization 的操作：
  - Batch Normalization 先将 batch 中每一列的数据分布转变为标准正态分布，因为如此操作以后，batch 内所有列的数据分布都是正态分布，所以**学习率的选择就会更加容易**，譬如可以选择更大的学习率。
  - Batch Normalization 然后对每一列加入了再平移参数 β_j 和再缩放参数 γ_j，使得最终第 $j \in \{1, ..., d\}$ 个 unit 的分布为 $\mathcal{N} \sim (\beta_j, \gamma_j^2)$，这样做是为了**保证模型的表达能力不因为规范化而下**

降。但其实质是**防止梯度弥散或梯度爆炸**：

$$\because\ \boldsymbol{y}^{(i)}=\boldsymbol{\gamma}\odot\widehat{\boldsymbol{x}^{(i)}}+\boldsymbol{\beta}\quad\therefore\ \frac{\partial l}{\partial\widehat{\boldsymbol{x}^{(i)}}}=\frac{\partial l}{\partial\boldsymbol{y}^{(i)}}\odot\frac{\partial\boldsymbol{y}^{(i)}}{\partial\widehat{\boldsymbol{x}^{(i)}}}=\frac{\partial l}{\partial\boldsymbol{y}^{(i)}}\odot\boldsymbol{\gamma}$$

也就是说：

1. 即使 $\frac{\partial l}{\partial\boldsymbol{y}^{(i)}}$ 出现了很小的情况（梯度弥散），但也有一定希望通过一个比较大的 $\boldsymbol{\gamma}$ 调整回来；
2. 即使 $\frac{\partial l}{\partial\boldsymbol{y}^{(i)}}$ 出现了很大的情况（梯度爆炸），但也有一定希望通过一个比较小的 $\boldsymbol{\gamma}$ 调整回来。

- Batch Normalization 的理解：
  - 每个列的参数（训练的内容为变量）$\boldsymbol{\mu}_{\mathcal{D}}$ 和 $\boldsymbol{\sigma}^2_{\mathcal{D}}$ 是一个 mini-batch 的一阶统计量和二阶统计量；
  - 这就要求每一个 mini-batch 的统计量是整体统计量的近似估计，或者说每一个 mini-batch 彼此之间，以及和整体数据，都应该是**近似同分布**的；
  - 分布差距较小的 mini-batches 可以看做是为规范化操作和模型训练引入了噪声，可以**增加模型的鲁棒性**；
  - 但如果每个 mini-batch 的**原始分布差别很大**，那么不同 mini-batch 的数据将会进行不一样的数据变换，这就**增加了模型训练的难度**。
- Batch Normalization 的适用场景：
  - **每个 mini-batch 中样本数量 m 比较大，且不同 mini-batch 之间的相同 unit 数据分布比较接近**。在进行训练之前，要做好充分的 shuffle. 否则效果会差很多。
  - 由于 BN 需要在运行过程中统计每个 mini-batch 的一阶统计量和二阶统计量，因此**不适用于动态的网络结构和 RNN 网络**。

Implement TensorFlow 实现如下：

```
import tensorflow as tf
out = tf.layers.batch_normalization(
    inputs,
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer=tf.zeros_initializer(),
    gamma_initializer=tf.ones_initializer(),
    moving_mean_initializer=tf.zeros_initializer(),
    moving_variance_initializer=tf.ones_initializer(),
    beta_regularizer=None,
    gamma_regularizer=None,
    training=False, # training = tf.estimator.ModeKeys.TRAIN
    trainable=True,
    name=None,
)
```

注意易出错忽略的一点是，如果使用该方法定义 Batch Normalization，则需要在训练时写：

```
import tensorflow as tf
loss = ...
if is_training:
   optimizer = tf.train.AdamOptimizer(params.learning_rate)
   global_step = tf.train.get_or_create_global_step()
   if use_batch_norm:
      # Add a dependency to update the moving mean and variance for batch normalization
      with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
         train_op = optimizer.minimize(loss, global_step=global_step)
      else:
         train_op = optimizer.minimize(loss, global_step=global_step)
```

自定义实现如下：

```
# Copyright (C) 2019 Tong Jia. All rights reserved.
import tensorflow as tf


def batch_norm_2d_fn(x, name_or_scope, is_training, epsilon=1e-6, decay=0.999, dtype_val=tf.float32):
    """Definition of batch normalization for a 2D-tensor.

    Reference:
        Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[Ioffe et
                                                                                                    al., 2015]
        (https://arxiv.org/pdf/1502.03167.pdf)

    Parameters
    ----------
    :param x: (tensor) -- input tensor with shape [None, D].
    :param name_or_scope: (str) -- name of variable scope.
    :param is_training: (bool) -- indicate whether in train mode.
    :param epsilon: (Optional, float) -- a small float preventing the denominator from being 0
    :param decay: (Optional, float) -- decay rate of exponential moving average of mean and variance.
    :param dtype_val: (Optional, tf.Dtype) -- data type of value variable or constant.

    Returns
    ----------
    :return: (tensor) -- [None, D] tensor represrnting the mini-batch tensor after batch normalization
                                                                operation.
    """
    with tf.variable_scope(name_or_scope=name_or_scope):
        dim =x.get_shape().as_list()[-1]
        # Define trainable variables
        gamma =tf.get_variable(
            name="gamma",
            shape=[dim],
            dtype=dtype_val,
            initializer=tf.ones_initializer(dtype=dtype_val) # another choice is constant initializer with 0.1
        )
```

```
    beta =tf.get_variable(
        name="beta",
        shape=[dim],
        dtype=dtype_val,
        initializer=tf.zeros_initializer(dtype=dtype_val)
    )

    pop_mean =tf.get_variable(
        name="pop_mean",
        shape=[dim],
        dtype=dtype_val,
        initializer=tf.zeros_initializer(dtype=dtype_val),
        trainable=False
    )
    pop_var =tf.get_variable(
        name="pop_var",
        shape=[dim],
        dtype=dtype_val,
        initializer=tf.ones_initializer(dtype=dtype_val),
        trainable=False
    )
    if is_training: # for train mode
        # compute mini-batch mean and variance along the first dim(sample index dim), both batch_mean and
                                                                batch_var
        # are tensors with shape (D, ).
        batch_mean, batch_var =tf.nn.moments(x=x, axes=[0])
        train_mean_op =tf.assign(ref=pop_mean, value=pop_mean *decay +batch_mean *(1 -decay))
        train_var_op =tf.assign(ref=pop_var, value=pop_var *decay +batch_var *(1 -decay))

        with tf.control_dependencies(control_inputs=[train_mean_op, train_var_op]):
            return tf.nn.batch_normalization(
                x=x,
                mean=batch_mean, variance=batch_var,
                offset=beta, scale=gamma,
                variance_epsilon=epsilon
            )
    else: # for eval and infer mode
        return tf.nn.batch_normalization(
            x=x,
            mean=pop_mean, variance=pop_var,
            offset=beta, scale=gamma,
            variance_epsilon=epsilon
        )
```

Reference

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[?]
- Understanding Batch Normalization[?]
- Implementing Batch Normalization in Tensorflow

23.4.2 Layer Normalization (LN)

论文出处：Layer Normalizing[?]

Methodology 伪代码如下：

Algorithm 27: Layer Normalizing Transformation[?]

Input: Value of $\boldsymbol{x} \in \mathbb{R}^d$ over a mini-batch: $\mathcal{D} = \{\boldsymbol{x^{(i)}}\}_{i=1}^m$ (d is the number of units, m is the number of batch samples); Parameters to be learned: $\gamma \in \mathbb{R}$, $\beta \in \mathbb{R}$.

Output: $\{\boldsymbol{y^{(i)}} = \mathbf{BN}_{\gamma,\beta}(\boldsymbol{x^{(i)}})\}_{i=1}^m$, $\boldsymbol{y^{(i)}} \in \mathbb{R}^d$

1 Calculate per-sample mean: $\mu^{(i)} = \frac{1}{d}\sum_{j=1}^{d} x_j^{(i)}$;

$$\boldsymbol{\mu} \in \mathbb{R}^m \leftarrow (\frac{1}{d}\sum_{j=1}^{d} x_j^{(1)}, ..., \frac{1}{d}\sum_{j=1}^{d} x_j^{(i)}, ..., \frac{1}{d}\sum_{j=1}^{d} x_j^{(m)})$$

2 Calculate per-sample variance: $\sigma^{(i)} = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_j^{(i)} - \mu^{(i)})^2}$;

$$\boldsymbol{\sigma}^2 \in \mathbb{R}^m \leftarrow \Big(\frac{1}{d}\sum_{j=1}^{d}(x_j^{(1)} - \mu^{(1)})^2, ..., \frac{1}{d}\sum_{j=1}^{d}(x_j^{(i)} - \mu^{(i)})^2, ..., \frac{1}{d}\sum_{j=1}^{d}(x_j^{(m)} - \mu^{(m)})^2\Big)$$

3 Normalization: $\widehat{x_j^{(i)}} \leftarrow \frac{x_j^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}$ (Division and square root applied element-wise)

$$\widehat{\boldsymbol{x^{(i)}}} \in \mathbb{R}^d \leftarrow \Big(\frac{x_1^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}, ..., \frac{x_j^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}, ..., \frac{x_d^{(i)} - \mu^{(i)}}{\sqrt{(\sigma^{(i)})^2 + \epsilon}}\Big)$$

4 Scale & Shift, broadcasting γ & β and element-wise multiplication & addition operations:

$$\boldsymbol{y^{(i)}} \leftarrow \gamma \odot \widehat{\boldsymbol{x^{(i)}}} + \beta \equiv \mathbf{LN}_{\gamma,\beta}(\boldsymbol{x^{(i)}}),\ \boldsymbol{y^{(i)}} \in \mathbb{R}^d$$

算法解释

- Layer Normalization 是**针对行**（batch dataset 中每个独立 sample 的所有维度）进行的操作。
- Layer Normalization 的理解：
  - Layer Normalization 对于一整层的神经元训练得到同一个转换——所有的输入都在同一个区间范围内。**如果不同输入特征不属于相似的类别（比如颜色和大小），那么 Layer Normalization 的处理可能会降低模型的表达能力**。
  - Layer Normalization 不需要保存 mini-batch 的均值和方差，节省了额外的存储空间。
- Layer Normalization 的适用场景：
  - **可用于小 mini-batch 场景、动态网络场景和 RNN**。LN 针对单个训练样本进行，不依赖于其他数据，因此可以避免 Batch Normalization 中受 mini-batch 数据分布影响的问题。
  - **适用于计算机图像领域，特别适用于自然语言处理领域。**因为一整层的所有神经元都属于相同类型（NLP 的语义空间）

23.4.3 Instance Normalization (IN)

论文出处：Instance Normalization: The Missing Ingredient for Fast Stylization[?]，主要针对于图像的风格迁移（style transfer），因为**图像各个 channel 的的跨空间（height & width）统计信息（e.g. mean, variance）代表了图像的风格（artistic style）**。
Idea.通过**分别统计一个 channel 的多个局部区域的统计信息**，设计局部自适应的 normalization 方法。

Methodology 假设一个 batch 的图像样本 $x \in \mathbb{R}^{N\times C\times H\times W}$，并记其中的一个元素点(像素点)为 $x_{n,c,h,w}$，其中：

- h and w span spatial dimensions.
- c is the feature channel (color channel if the input is an RGB image).
- n is the index of the image in the batch.

Algorithm 28: Instance Normalization Transformation for Image Input[?]

Input: Input of a mini-batch dataset: $x \in \mathbb{R}^{N\times C\times H\times W}$;
A small constant ϵ;
learnable affine parameters: $\gamma \in \mathbb{R}$, $\beta \in \mathbb{R}$.

Output: $y_{n,c,h,w} = \mathbf{IN}_{\gamma,\beta}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.

1 Calculate inside per-sample-per-channel mean $\mu_{n,c}\,(\forall n \in N, c \in C)$:

$$\mu_{n,c} \leftarrow \frac{1}{H\times W}\sum_{i=1}^{H}\sum_{j=1}^{W} x_{n,c,i,j}$$

2 Calculate inside per-sample-per-channel variance: $\sigma^2_{n,c}\,(\forall n \in N, c \in C)$:

$$\sigma^2_{n,c} \leftarrow \frac{1}{H\times W}\sum_{i=1}^{H}\sum_{j=1}^{W} (x_{n,c,i,j} - \mu_{n,c})^2$$

3 Normalization for each element inside mini-batch:

$$\widehat{x}_{n,c,h,w} \leftarrow \frac{x_{n,c,h,w} - \mu_{n,c}}{\sqrt{\sigma^2_{n,c} + \epsilon}}$$

4 Scale & Shift:

$$y_{n,c,h,w} \leftarrow \gamma \cdot \widehat{x}_{n,c,h,w} + \beta \equiv \mathbf{IN}_{\gamma,\beta}(x_{n,c,h,w})$$

类似地，Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization[?][?] 论文中为了更好地实现图像的风格迁移而提出了 Adaptive Instance Normalization，即：

$$\text{AdaIN}(x, y) = \sigma_{n,c}(y)\left(\frac{x - \mu_{n,c}(x)}{\sigma_{n,c}(x)} + \mu_{n,c}(y)\right)$$

其中 $x \in \mathbb{R}^{M\times C\times H_{\text{source}}\times W_{\text{source}}}$ 是原始图像输入，$y \in \mathbb{R}^{M\times C\times H_{\text{target}}\times W_{\text{target}}}$ 是风格图像输入。Adaptive Instance Normalization 和 Instance Normalization 唯一的区别是，Adaptive Instance Normalization 中没有训练参数 μ 和 σ，用于 scale 和 shift 的 $\sigma_{n,c}(y)$ 和 $\mu_{n,c}(y)$ 直接来自于 y 的统计信息。

使用 Instance Normalization 的风格迁移如下：

图 113: Artistic style transfer example of Gatys et al. (2016)[?] method.

图 114: Stylization examples using proposed method. First row: style images; second row: original image and its stylized versions.

使用 Adaptive Instance Normalization 的风格迁移如下：

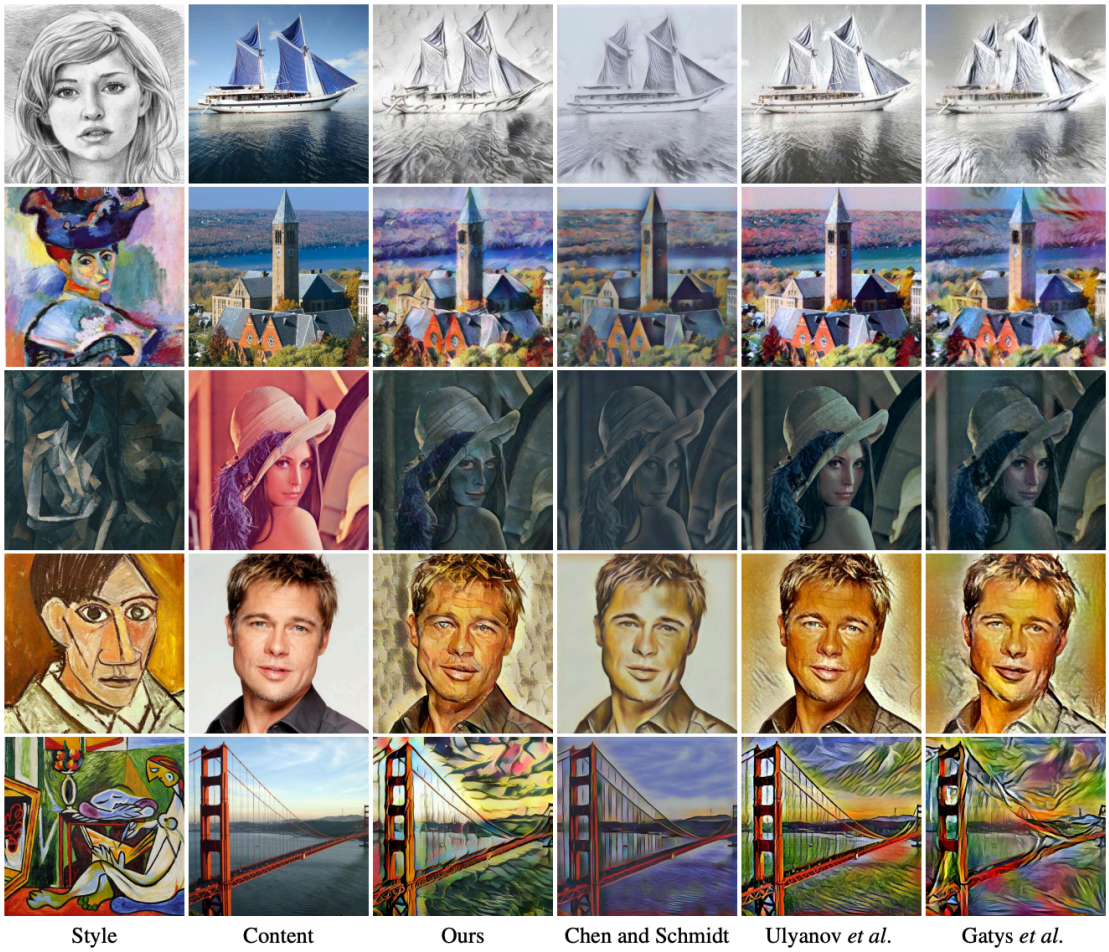
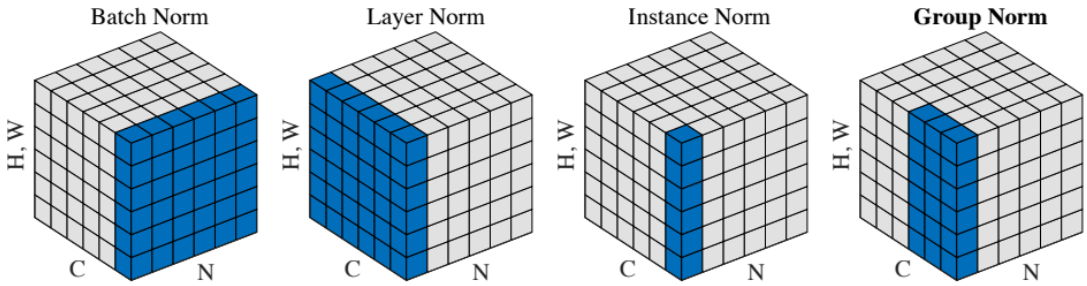


图 115: Example style transfer results. All the tested content and style images are never observed by the network during training.

为了方便对照，针对图像类型数据的 Batch Normalization 和 Layer Normalization 伪代码如下：



Algorithm 29: Batch Normalization Transformation for Image Input[?]

Input: Input of a mini-batch dataset: $x \in \mathbb{R}^{N\times C\times H\times W}$;
A small constant ϵ;
learnable affine parameters: $\gamma \in \mathbb{R}^C$, $\beta \in \mathbb{R}^C$.

Output: $y_{n,c,h,w} = \mathbf{BN}_{\gamma,\beta}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.

1 Calculate inside per-channel mean $\mu_c \, (\forall c \in C)$:

$$\mu_c \leftarrow \frac{1}{N\times H\times W}\sum_{k=1}^{N}\sum_{i=1}^{H}\sum_{j=1}^{W} x_{k,c,i,j}$$

2 Calculate inside per-channel variance: $\sigma_c^2 \, (\forall c \in C)$:

$$\sigma_c^2 \leftarrow \frac{1}{N\times H\times W}\sum_{k=1}^{N}\sum_{i=1}^{H}\sum_{j=1}^{W} (x_{k,c,i,j} - \mu_c)^2$$

3 Normalization for each element inside mini-batch:

$$\widehat{x}_{n,c,h,w} \leftarrow \frac{x_{n,c,h,w} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

4 Scale & Shift:

$$y_{n,c,h,w} \leftarrow \gamma_c \cdot \widehat{x}_{n,c,h,w} + \beta_c \equiv \mathbf{BN}_{\gamma,\beta}(x_{n,c,h,w}), \quad \forall c \in C$$

Algorithm 30: Layer Normalization Transformation for Image Input[?]

Input: Input of a mini-batch dataset: $x \in \mathbb{R}^{N\times C\times H\times W}$;
A small constant ϵ;
learnable affine parameters: $\gamma \in \mathbb{R}$, $\beta \in \mathbb{R}$.

Output: $y_{n,c,h,w} = \mathbf{LN}_{\gamma,\beta}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.

1 Calculate inside per-sample mean $\mu_n \, (\forall n \in N)$:

$$\mu_n \leftarrow \frac{1}{C\times H\times W}\sum_{k=1}^{C}\sum_{i=1}^{H}\sum_{j=1}^{W} x_{n,k,i,j}$$

2 Calculate inside per-sample variance: $\sigma_n^2 \, (\forall n \in N)$:

$$\sigma_n^2 \leftarrow \frac{1}{C\times H\times W}\sum_{k=1}^{C}\sum_{i=1}^{H}\sum_{j=1}^{W} (x_{n,k,i,j} - \mu_n)^2$$

3 Normalization for each element inside mini-batch:

$$\widehat{x}_{n,c,h,w} \leftarrow \frac{x_{n,c,h,w} - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}$$

4 Scale & Shift:

$$y_{n,c,h,w} \leftarrow \gamma \cdot \widehat{x}_{n,c,h,w} + \beta \equiv \mathbf{LN}_{\gamma,\beta}(x_{n,c,h,w})$$

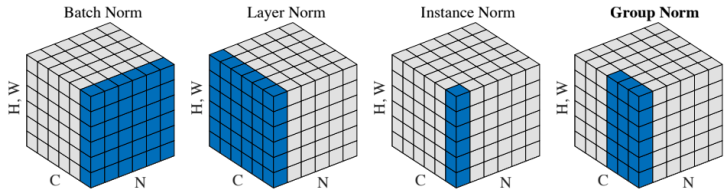
23.4.4 Group Normalization (GN)

论文出处：Group Normalization[?]

23.4.5 Batch-Instance Normalization (BIN)

论文出处：Batch-Instance Normalization for Adaptively Style-Invariant Neural Networks[?]。实际上该方法在 Batch Normalization 和 Instance Normalization 之间做了**动态加权平均**。

Methodolody



Batch Normalization:

$$\mu_c^{(\mathrm{B})} = \frac{1}{N \times H \times W} \sum_{k=1}^{N} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{k,c,i,j}$$

$$(\sigma_c^{(\mathrm{B})})^2 = \frac{1}{N \times H \times W} \sum_{k=1}^{N} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(x_{k,c,i,j} - \mu_c^{(\mathrm{B})} \right)^2$$

$$\widehat{x}_{n,c,h,w}^{(\mathrm{B})} = \frac{x_{n,c,h,w} - \mu_c^{(\mathrm{B})}}{\sqrt{(\sigma_c^{(\mathrm{B})})^2 + \epsilon}}$$

$$y_{n,c,h,w}^{(\mathrm{B})} = \gamma \cdot \widehat{x}_{n,c,h,w}^{(\mathrm{B})} + \beta$$

Instance Normalization:

$$\mu_{n,c}^{(\mathrm{I})} = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{n,c,i,j}$$

$$(\sigma_{n,c}^{(\mathrm{I})})^2 = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(x_{n,c,i,j} - \mu_{n,c}^{(\mathrm{I})} \right)^2$$

$$\widehat{x}_{n,c,h,w}^{(\mathrm{I})} = \frac{x_{n,c,h,w} - \mu_{n,c}^{(\mathrm{I})}}{\sqrt{(\sigma_{n,c}^{(\mathrm{I})})^2 + \epsilon}}$$

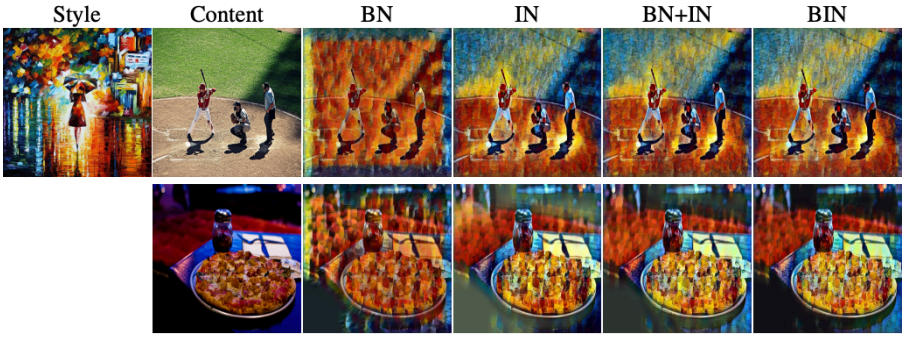
$$y_{n,c,h,w}^{(\mathrm{I})} = \gamma \cdot \widehat{x}_{n,c,h,w}^{(\mathrm{I})} + \beta$$

Batch Normalization 因为求神经元大小的均值和方差时，考虑了样本 N 的维度，因此：**样本之间的样式差异（style difference between examples）得以保留（通过方差）**；而 Instance Normalization 因为独立地对每个样本的每个 channel 统计均值和方差信息，因此其中不包含样本之间的样式差异信息，只包含单独样本内部一个 channel 内的样式差异信息。而我们希望达到的目的是：对每一个 channel，保留重要的样式属性，而干扰的属性被平滑，因此有了 BIN：

$$y_{n,c,h,w}^{(\mathrm{BI})} = \left(\rho_c \cdot \widehat{x}_{n,c,h,w}^{(\mathrm{B})} + (1 - \rho_c) \cdot \widehat{x}_{n,c,h,w}^{(\mathrm{I})} \right) \cdot \gamma_c + \beta_c$$

其中：

- affine transformation parameters: $\gamma = (\gamma_1, ..., \gamma_C) \in \mathbb{R}^C$, $\beta = (\beta_1, ..., \beta_C) \in \mathbb{R}^C$
- learnable parameters: $\rho = (\rho_1, ..., \rho_C) \in \mathbb{R}^C$. 但是一般会限制 $\rho \in [0, 1]^C$: $\rho \leftarrow \mathrm{clip}_{[0,1]}(\rho - \eta \Delta \rho)$



Algorithm 31: Batch-Instance Normalization Transformation for Image Input[**?**]

Input: Input of a mini-batch dataset: $x \in \mathbb{R}^{N\times C\times H\times W}$;
A small constant ϵ;
learnable affine parameters: $\boldsymbol{\gamma} = (\gamma_1, ..., \gamma_C) \in \mathbb{R}^C$, $\boldsymbol{\beta} = (\beta_c, ..., \beta_C) \in \mathbb{R}^C$;
learnable parameters: $\boldsymbol{\rho} = (\rho_1, ..., \rho_C) \in [0, 1]^C$.

Output: $y_{n,c,h,w} = \mathbf{BIN}_{\boldsymbol{\gamma},\boldsymbol{\beta},\boldsymbol{\rho}}(x_{n,c,h,w})$, $y_{n,c,h,w} \in \mathbb{R}$.

1 Calculate inside per-channel (BN) mean μ_c^{B} $(\forall c \in C)$ and inside per-sample-per-channel (IN) mean $\mu_{n,c}^{(\mathrm{I})}$ $(\forall n \in N, c \in C)$:

$$\mu_c^{(\mathrm{B})} \leftarrow \frac{1}{N\times H\times W}\sum_{k=1}^{N}\sum_{i=1}^{H}\sum_{j=1}^{W} x_{k,c,i,j}$$

$$\mu_{n,c}^{(\mathrm{I})} \leftarrow \frac{1}{H\times W}\sum_{i=1}^{H}\sum_{j=1}^{W} x_{n,c,i,j}$$

2 Calculate inside per-channel (BN) variance $(\sigma_c^{\mathrm{B}})^2$ $(\forall c \in C)$ and inside per-sample-per-channel (IN) variance $(\sigma_{n,c}^{(\mathrm{I})})^2$ $(\forall n \in N, c \in C)$:

$$(\sigma_c^{(\mathrm{B})})^2 \leftarrow \frac{1}{N\times H\times W}\sum_{k=1}^{N}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(x_{k,c,i,j} - \mu_c^{(\mathrm{B})}\right)^2$$

$$(\sigma_{n,c}^{(\mathrm{I})})^2 \leftarrow \frac{1}{H\times W}\sum_{i=1}^{H}\sum_{j=1}^{W}\left(x_{n,c,i,j} - \mu_{n,c}^{(\mathrm{I})}\right)^2$$

3 Normalization for each element inside mini-batch:

$$\widehat{x}_{n,c,h,w}^{(\mathrm{B})} \leftarrow \frac{x_{n,c,h,w} - \mu_c^{(\mathrm{B})}}{\sqrt{(\sigma_c^{(\mathrm{B})})^2 + \epsilon}},\ \widehat{x}_{n,c,h,w}^{(\mathrm{I})} \leftarrow \frac{x_{n,c,h,w} - \mu_{n,c}^{(\mathrm{I})}}{\sqrt{(\sigma_{n,c}^{(\mathrm{I})})^2 + \epsilon}}$$

4 Clip the learnable parameter ρ_c $\forall c \in C$:

$$\tilde{\rho}_c \leftarrow \mathrm{clip}_{[0,1]}(\rho_c)$$

5 Scale & Shift within each channel $c \in C$:

$$y_{n,c,h,w}^{(\mathrm{BI})} \leftarrow \left(\tilde{\rho}_c \cdot \widehat{x}_{n,c,h,w}^{(\mathrm{B})} + (1 - \tilde{\rho}_c)\cdot \widehat{x}_{n,c,h,w}^{(\mathrm{I})}\right)\cdot \gamma_c + \beta_c \equiv \mathbf{BIN}_{\boldsymbol{\gamma},\boldsymbol{\beta},\boldsymbol{\rho}}(x_{n,c,h,w})$$

Implement BIN 的 TensorFlow 简单实现如下，注意：该实现方法可作为其他 Normalization 方法实现的参考：

```
import tensorflow as tf

def batch_instance_norm(x, scope='batch_instance_norm'):
    # x is a tensor with shape [None, H, W, C]
    with tf.variable_scope(scope):
        ch = x.shape[-1]
        eps = 1e-5

        batch_mean, batch_sigma = tf.nn.moments(x, axes=[0, 1, 2], keep_dims=True)
        x_batch = (x - batch_mean) / (tf.sqrt(batch_sigma + eps))

        ins_mean, ins_sigma = tf.nn.moments(x, axes=[1, 2], keep_dims=True)
        x_ins = (x - ins_mean) / (tf.sqrt(ins_sigma + eps))

        rho = tf.get_variable(name="rho",
                              shape=[ch],
                              dtype=tf.float32,
                              initializer=tf.constant_initializer(1.0),
                              constraint=lambda x: tf.clip_by_value(x, clip_value_min=0.0,
                                  clip_value_max=1.0))
        gamma = tf.get_variable(name="gamma",
                                shape=[ch],
                                 dtype=tf.float32,
                                 initializer=tf.constant_initializer(1.0))
        beta = tf.get_variable("beta", [ch], initializer=tf.constant_initializer(0.0))

        x_hat = rho * x_batch + (1 - rho) * x_ins
        x_hat = x_hat * gamma + beta

    return x_hat
```

23.4.6 Local Response Normalization (LRN)

论文出处：ImageNet Classification with Deep Convolutional Neural Networks (AlexNet)[?]。LRN 是一种收生物学启发的归一化方法，通常用在基于卷积的图像处理问题上。

Methodology 假设一个卷积层的输出特征映射 $\mathbf{Y} \in \mathbb{R}^{H' \times W' \times C'}$ 为三维张量，其中每个切片矩阵（针对 channel 切片）$\mathbf{Y}^{(c)} \in \mathbb{R}^{H' \times W'}, \forall c \in C'$ 为一个输出 channel 的特征映射。局部响应归一化（LRN）是对临近的 channel 特征映射进行局部归一化，即：

$$\begin{aligned}\hat{Y}_{i,j}^{(c)} &= \frac{Y_{i,j}^{(c)}}{\left(k + \alpha \sum\limits_{l=\max(1,c-\frac{n}{2})}^{\min(C',c+\frac{n}{2})} \left(Y_{i,j}^{(l)}\right)^2\right)^{\beta}} \\ &\equiv \mathrm{LRN}_{n,k,\alpha,\beta}\left(Y_{i,j}^{(c)}\right)\end{aligned}$$

因此 LRN 可以理解为是对邻近的几个 channel 上相同的切片内位置 (i,j) 进行平滑放大作用。LRN 的所有**运算都是 element-wise**的。

Summery

- 局部响应归一化（LRN）[?] 和层归一化（LN）[?] 的异同：
  - 相同点：
    1. 都是对单独样本内的特定神经元 $Y_{i,j}^{(c)}$ 进行归一化。
    2. 对 $Y_{i,j}^{(c)}$ 进行归一化时考虑了多个 channels。
  - 相异点：
    1. **LRN 用于激活函数作用之后**（具体来说：卷积层激活函数之后，汇聚层之前），LN 用于激活函数作用之前。
    2. LN 计算均值，LRN 没有计算均值。
    3. LN 考虑了全部 channels，LRN 只考虑邻近部分 channels。
- 局部响应归一化（LRN）和生物神经元中的**侧抑制**（lateral inhibition）现象比较类似，即**活跃神经元对相邻神经元具有抑制作用**。当使用 ReLU 作为激活函数时，神经元的活性值是没有限制的，局部响应归一化可以起到平衡和约束作用。如果一个神经元的活性值 $Y_{i,j}^{(c)}$ 非常大，那么和它邻近的神经元（相邻近 channels 上相同位置 $Y_{i,j}^{(l)}, \in \{\max(1, c-\frac{n}{2}), \min(C', c+\frac{n}{2})\}$）就近似地归一化为 0，从而起到抑制作用，增强模型的泛化能力。
- 局部响应归一化（LRN）和最大汇聚（Max Pooling）之间的异同：
  - 相同点：
    1. 都有对邻近神经元的测抑制作用。
  - 相异点：
    1. LRN 针对几个邻近 channels 上同一个切片内位置 (i,j) 的神经元进行抑制。
    2. Max Pooling 针对同一个 channel 上的邻近切片位置的神经元进行抑制。

23.4.7 Gradient Normalization (Gradient Clipping)

Methodology 梯度裁剪（Gradient Clipping）是一种有效的防止**梯度爆炸**（Gradient Exploding）的方法

Algorithm 32: Norm clipping the gradients whenever they explode[**?**]

Compute gradient: $\boldsymbol{g} \leftarrow \frac{\partial J}{\partial \boldsymbol{\theta}}$
if $\|\boldsymbol{g}\| \geq threshold$ **then**
  $\boldsymbol{g} \leftarrow \frac{\text{threshold}}{\|\boldsymbol{g}\|}\boldsymbol{g}$
end

Implement TensorFlow 实现如下：

```
import tensorflow as tf
...
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
grads_and_vars = optimizer.compute_gradients(loss=loss)
for i, (g, v) in enumerate(grads_and_vars):
  if g is not None:
    grads_and_vars[i] = (tf.clip_by_norm(t=g, clip_norm=5), v) # Set threshold = 5
train_op = optimizer.apply_gradients(
  grads_and_vars=grads_and_vars,
  global_step=tf.train.get_global_step()
)
...
```

注意：除此之外还有很多 Gradient Clipping 的方法。

23.5 Network Regularization

23.5.1 Label Smoothing Regularization (LSR)

论文出处：Regularizing Neural Networks by Penalizing Confident Output Distributions[?]。该方法用于缓解分类问题中 label 不够 soft 容易导致过拟合的问题，从而提高神经网络的泛化性。

Methodology 将真实概率改造为：

$$p_k = \begin{cases} 1-\epsilon & \text{if real} \\ \frac{\epsilon}{K-1} & \text{otherwise } (K \text{ is the total number of classes}) \end{cases}$$

譬如，对于一个五分类问题，原始 label 的概率分布为 $\boldsymbol{t} = (0, 0, 1, 0, 0)$，而经过 LSR，选择 $\epsilon = 0.01$，则平滑后的 label 概率分布为 $\boldsymbol{y} = (0.01, 0.01, 0.96, 0.01, 0.01)$
Note:

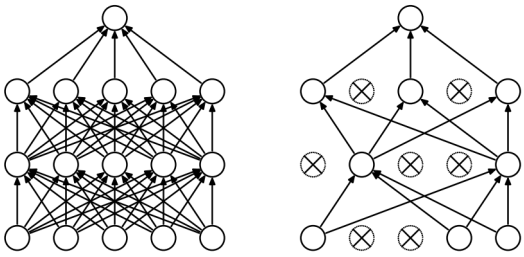
- 一个自己的 idea：可将概率改造为概率的概率分布（e.g. Beta 分布）

Reference

- 知乎：Label Smoothing Regularization（LSR）原理是什么？

23.6 Dropout

23.6.1 Dropout of DNN



(a) Standard Neural Net (b) After applying dropout.

图 116: Neural network with dropout[?]: subsampling of hidden units

Forward propagation of dropout in DNN 多输入-单输出神经元之间的标准网络（Standard Network）和 Dropout 网络（Dropout Network）之间区别如下：

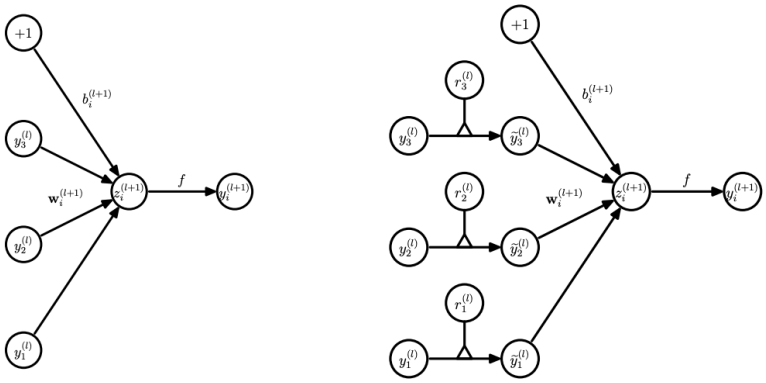


图 117: Comparison of the basic operations of a standard and dropout network. Left figure is standard network and right is dropout network. **Dropout operates before matrix multiplication**.

- 训练阶段：

  标准网络的计算公式：

  $$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \cdot \boldsymbol{y}^{(l)} + b_i^{(l+1)} \quad \text{(vector-wise)}$$
  $$y_i^{(l+1)} = f(z_i^{(l+1)})$$

  采用 Dropout 的网络计算公式：

  $$r_j^{(l)} \sim \text{Bernoulli}(p)$$
  $$\tilde{\boldsymbol{y}}^{(l)} = \boldsymbol{r}^l \odot \boldsymbol{y}^l \quad \text{(element-wise)}$$
  $$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \cdot \tilde{\boldsymbol{y}}^{(l)} + b_i^{(l+1)}$$
  $$y_i^{(l+1)} = f(z_i^{(l+1)})$$

- 测试阶段：在任意 l 层神经元 y_j^l 与 $l+1$ 层神经元 z_i^{l+1} 的连接中：**使用 dropout 进行训练时**，y_j^l 以概率 p 处于激活状态，并且 $y_j^l \to y_j^l$ 的链路上权重为 $w_{i,j}^{l+1}$，即相当于该权值大小为 $w_{i,j}^{l+1}$ 的链路以概

率 $p \in (0,1)$ 存在，因此训练阶段该点对点（unit-to-unit）链路上的期望权重为：

$$\mathbb{E}_{\text{train}}\left[w_{i,j}^{l+1}\right] = p \times 1 \times w_{i,j}^{l+1} + (1-p) \times 0 \times w_{i,j}^{l+1} = p \times w_{i,j}^{l+1}$$

而在测试阶段，因为 y_j^l 以 100% 概率处于激活状态，即该权值大小为 $w_{i,j}^{l+1}$ 的链路以 100% 概率存在，因此测试阶段该点对点（unit-to-unit）链路上的期望权重为：

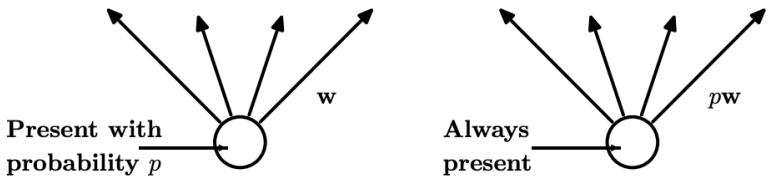
$$\mathbb{E}_{\text{test}}\left[w_{i,j}^{l+1}\right] = 1 \times 1 \times w_{i,j}^{l+1} + (1-0) \times 0 \times w_{i,j}^{l+1} = w_{i,j}^{l+1}$$

所以导致：

$$\mathbb{E}_{\text{test}}\left[w_{i,j}^{l+1}\right] > \mathbb{E}_{\text{train}}\left[w_{i,j}^{l+1}\right] \quad (\textbf{错误原因！})$$

因此为了保障在训练阶段和测试阶段该链路的期望权重相同，需要在测试阶段为每个权重做：

$$\boldsymbol{w}_{\text{test}}^{(l+1)} = p \odot \boldsymbol{w}^{(l+1)}$$



(a) At training time (b) At test time

图 118: **Left**: A unit at training phase that is present with probability p and is connected to units in the next layer with weights w. **Right**: At test phase, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Model implement of dropout in DNN TensorFlow 代码实现如下：

```
import tensorflow as tf
out = tf.nn.dropout(
   x,
   keep_prob, # 1 - keep_prob = dropout_rate
   noise_shape=None,
   seed=None,
   name=None
)
```

Reference

- Dropout: A Simple Way to Prevent Neural Networks from Overfitting[?]

23.6.2 Dropout of CNN

为了捕获局部特征的准确性，建议不要在 CNN（i.e. convolutional layer & pooling layer）中使用 Dropout。

Reference

- Towards Dropout Training for Convolutional Neural Networks[**?**]

23.6.3 Dropout of RNN

Forward propagation of dropout in RNN 只丢弃当前时刻输出的参数，不丢弃输入参数，时间轴上的参数。

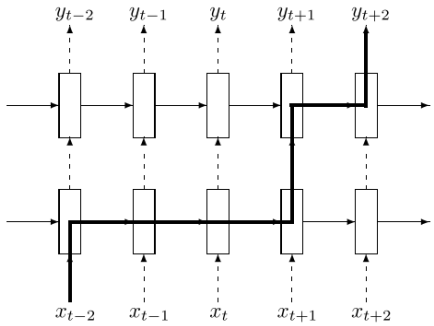


图 119: Dropout in RNN[?]

Model implement of dropout in RNN TensorFlow 代码实现如下：

```
import tensorflow as tf
def rnn_cell(mode):
  # Define such a function to make a basic cell(RNN, LSTM, GRU) with dropout wrapper in model
      function.
  cell = tf.nn.rnn_cell.LSTMCell(
    num_units=num_hidden_units,
    use_peepholes=False,
    initializer=tf.orthogonal_initializer(dtype=dtype),
    forget_bias=1.0,
    state_is_tuple=True
  )
  if mode == tf.estimator.ModeKeys.TRAIN:
    cell = tf.nn.rnn_cell.DropoutWrapper(
      cell=cell,
      input_keep_prob=1.0,
      output_keep_prob=1 - dropout_rate, # Key point
      state_keep_prob=1.0,
      dtype=dtype
  )
  return cell
```

23.7 Learning Rate Schedules

23.7.1 Learning Rate Decay

学习率衰减（learning rate decay），也称为学习率退火（learning rate annealing）策略是极其有效的优化函数训练 trick，其主要作用如下：

- 对于非自适应学习率优化算法（e.g. SGD, Momentum, Nesterov），其可以调节学习率，使得在训练前期使用较大的学习率保障充分学习、快速收敛；而在训练后期使用较小学习率避免来回震荡，从而无法收敛
- 对于自适应学习率优化算法（e.g. AdaGrad, RMSProp, Adam），其（譬如余弦衰减策略）可以增加优化过程的随机性，从而有助于跳出鞍点（**因此并不是自适应优化算法就没有学习率衰减的需求**）

不失一般性，此处衰减方式设置为按照迭代次数 t 进行衰减，假设初始学习率为 η_0，在第 t 次迭代时的学习率为 η_t，则常见的学习率衰减策略有以下几种：

衰减类型	公式表示
分段常数衰减（Piecewise Constant Decay）	$T_1, ..., T_m$ 次迭代后将 η_t 衰减为 η_0 的 $\beta_1, ..., \beta_m$ 倍 $(\beta_m < 1)$
逆时衰减（Inverse Time Decay）	$\eta_t = \eta_0 \frac{1}{1+\beta \times t}$，其中 β 为衰减率
指数衰减（Exponential Decay）	$\eta_t = \eta_0 \beta^t$，其中 $\beta \in (0, 1)$ 为衰减率
自然指数衰减（Natural Exponential Decay）	$\eta_t = \eta_0 e^{-\beta \times t}$，其中 β 为衰减率
余弦衰减（Cosine Decay）	$\eta_t = \frac{1}{2}\eta_0 \left(1 + \cos(\frac{t\pi}{T})\right)$，其中 T 为总的迭代次数

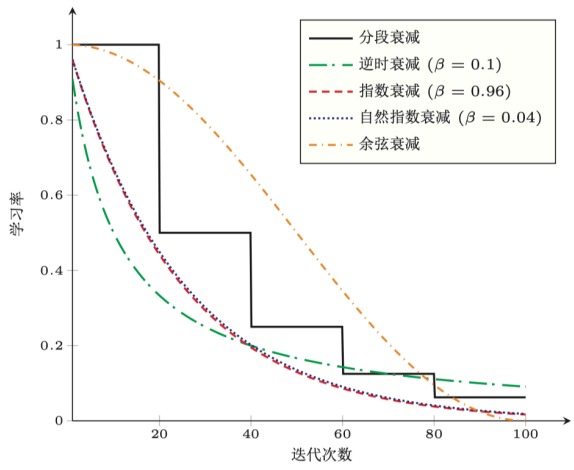


图 120: 不同学习率衰减方法的比较，假设初始学习率 $\eta_0 = 1$。

23.7.2 Learning Rate Warmup

初始训练阶段关于学习率大小选择存在的问题：

- 在 mini-batch 梯度下降算法中，在批数据量（batch size）$|\mathcal{B}|$ 比较大的时候，通常需要较大的学习率 η。但是由于此时模型参数由随机初始化得到，导致梯度 $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{B}; f_{\boldsymbol{\theta}})$ 往往也较大，加之此时学习率 η 也很大，那么会使得**训练非常不稳定**。
- 因此为了在**初始训练阶段**提高训练的稳定性，在最初几轮迭代时，采用较小的学习率，等梯度下降到一定程度（模型参数相对稳定）后再恢复到初始的相对较大学习率。这种方法被称为**学习率预热** (learning rate warmup)。

学习率预热主要分为两大类方法：

1. **Constant Warmup** (瞬间变化): (e.g. ResNet 图像分类 [?] 中先用 0.01 的学习率训练直到训练误差低于 80%(正确率大于 20%)，然后使用 0.1 的学习率进行训练)
2. **Gradual Warmup** (平滑过度): 18 年 Facebook 针对上述 Constant warmup 进行了改进 [?]
   - 因为从一个很小的学习率一下子变为一个较大的学习率可能会导致训练训练误差突然增大
   - 所以 gradual warmup 即：从最开始的小学习率开始，每个 iteration 增大一点，直到最初设置的比较大的学习率

   预热过程中，每次更新的学习率为：

$$\eta'_t = \frac{t}{T'}\eta_0, \quad 1 \le t \le T'$$

   其中 T' 为**预热阶段**的总迭代次数，η_0 为**手动设置的较大初始学习率**，η'_t 随着迭代次数 t' 的增加逐步增大；当预热过程结束，再选择一种学习率衰减方法来逐渐降低学习率。

整个训练阶段中学习率大小的变化模式原则应是：

- **前期到中期：越来越大**
- **中起到后期：越来越小**

23.7.3 Periodic Learning Rate Adjustment

Triangular Cyclic Learning Rate

Cosine Decay with Warm Restarts

23.7.4 Linear Scaling Learning Rate

Linear scaling learning rate 是在论文 [?] 中针对比较大的 batch size 而提出的一种方法

- 凸优化问题中，随着批量的增加，收敛速度会降低
- 神经网络也有类似的实证结果。随着 batch size 的增大，处理相同数据量的速度会越来越快，但是达到相同精度所需要的 epoch 数量越来越多
- 使用相同的 epoch 时，大 batch size 训练的模型与小 batch size 训练的模型相比，验证准确率会减小
- 上面提到的 gradual warmup 是解决此问题的方法之一，此外 linear scaling learning rate 也是一种有效的方法
- 在 mini-batch SGD 训练时，梯度下降的值是随机的，因为每一个 batch 的数据是随机选择的。增大 batch size 不会改变梯度的期望，但是会降低它的方差。也就是说，**因为大 batch size 会降低梯度中的噪声，所以我们可以增大学习率来加快收敛**
- 具体做法：e.g. ResNet 原论文 [?] 中 batch size 为 256 时选择的学习率是 0.1，当我们把 batch size 变为一个较大的数 b 时，学习率应该变为 $\eta = 0.1 \times \frac{b}{256}$

23.8 Summary Tricks of Training Neural Networks

23.8.1 Tricks for DNN

23.8.2 Tricks for CNN

根据 Bag of Tricks to Train Convolutional Neural Networks for Image Classification[?] 一文，训练 CNN 做图像分类任务时有如下 Tricks：

23.8.3 Tricks for RNN

Part IV

Reinforcement Learning Model and Theory

24 Reinforcement Learning Basic

24.1 Markov Decision Processes (MDPs)

24.1.1 Infinite-horizon discounted MDPs

In general, Reinforcement Learning(RL) problems can be formulated as Markov Decision Process(MDP) models (**environment models**), and it can be abstracted by a five-tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$[**?**] :

- **A finite set of states**: $\mathcal{S}$
  - Only consider **discrete** and **finite** state spaces here.
- **A finite set of actions**: $\mathcal{A}$
  - Only consider **discrete** and **finite** action spaces here.
- **Transition function**: $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{P}$
  - $\mathcal{P}(\boldsymbol{s}_{t+1} = \boldsymbol{s}' \mid \boldsymbol{s}_t = \boldsymbol{s}, a_t = a)$ is the probability of transitioning into state $\boldsymbol{s}'$ upon taking action a in state $\boldsymbol{s}$, and we always use $\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, a_t)$ to represent transition probability.
- **Reward function**: $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$
  - we always set reward in range of $[0, r_{\max}]$, where $r_{\max} > 0$ is a constant.
  - $r_t = r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1})$ means an **immediate** reward associated with taking action a_t in state $\boldsymbol{s}_t$ at time-step t, and transitioning into state $\boldsymbol{s}_{t+1}$.
  - $r(\boldsymbol{s}_t = \boldsymbol{s}, a_t = a, \boldsymbol{s}_{t+1} = \boldsymbol{s}')$ is the immediate reward of transitioning into state $\boldsymbol{s}'$ upon taking action a in state $\boldsymbol{s}$.
  - $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, **not** $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, namely r_t belongs to $(\boldsymbol{s}_t \to a_t \to \boldsymbol{s}_{t+1})$, not $(\boldsymbol{s}_t \to a_t)$
- **Discount factor**: $\gamma \in (0, 1)$
  - Lower value of γ place more emphasis on immediate reward.

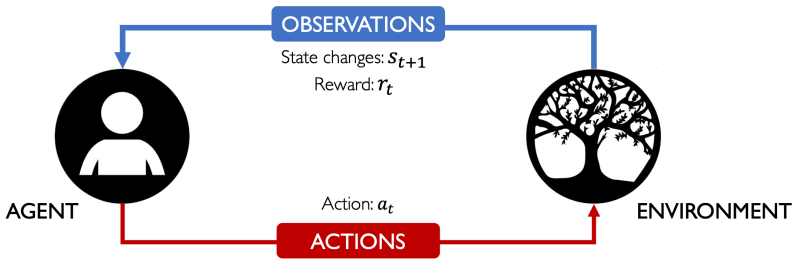


图 121: [**?**]The perception-action-learning loop. At time t, the agent receives state $\boldsymbol{s}_t$ from the environment. The agent uses its policy to choose an action a_t . Once the action is executed, the environment transitions a step, providing the next state $\boldsymbol{s}_{t+1}$ as well as feedback in the form of a reward r_t. The agent uses knowledge of state transitions, of the form $(\boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}_{t+1})$, in order to learn and improve its policy.

24.1.2 Interaction protocol

In a given MDP $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, the agent interacts with the environment according to the following protocol:

1. the agent starts at some state $\boldsymbol{s}_0$;
2. at each time step $t = 0, 1, ...$, the agent takes an action $a_t \in \mathcal{A}$, then obtains the immediate reward r_t, and observes the next state $\boldsymbol{s}_{t+1}$ sampled from $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot\,|\boldsymbol{s}_t, a_t)$

The interaction record

$$\tau = (\boldsymbol{s}_0^\tau, a_0^\tau, r_0^\tau, \boldsymbol{s}_1^\tau, a_1^\tau, r_1^\tau, \boldsymbol{s}_2^\tau, ..., \boldsymbol{s}_T^\tau)$$

is called a **trajectory** of length T.

In some situations, it is necessary to specify how the initial state $\boldsymbol{s}_0$ is generated. We consider $\boldsymbol{s}_0$ is sampled from an initial distribution $\mu : \mathcal{S} \to \mathbb{P}$: $\boldsymbol{s}_0 \sim \mu(\cdot)$. When μ is of importance to the discussion, we include it as part of the MDP definition, and write $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mu)$

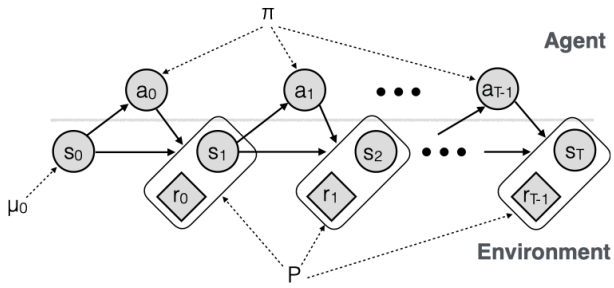


图 122: Illustrate the trajectory of the agent interacts with environment in MDP[?]

24.1.3 Policy and value

Definition 24.1 (**deterministic policy**). A deterministic policy π is a mapping from state space to action space:

$$\pi : \mathcal{S} \to \mathcal{A}$$

which the agent chooses an action a_t adaptively based on the current state $\boldsymbol{s}_t$:

$$a_t = \pi(\boldsymbol{s}_t)$$

Definition 24.2 (**stochastic policy**). A stochastic policy π is a mapping from state space to a probability distribution over actions:

$$\pi : \mathcal{S} \to \mathbb{P}^{|\mathcal{A}|}$$

and with a slight abuse of notation we write:

$$a_t \sim \pi(\cdot\,|\boldsymbol{s}_t)$$

Definition 24.3 (**return**). The return(rewards-to-go) R_t is the total discounted reward from time-step t.

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Definition 24.4 (**objective of agent**). The objective of the agent is to **choose a policy π to maximize the expected discounted sum of rewards, or expected return**:

$$\max_{\pi} \mathbb{E}_{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, ...} \left[R_0 = \sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, \mathcal{P}, \mu \right]$$

The expectation is with respect to the randomness of the observation trajectory $\tau = (\boldsymbol{s}_0, a_0, r_0, \boldsymbol{s}_1, a_1, r_1, \boldsymbol{s}_2, ...)$, that is:

- the randomness of initial state: $\boldsymbol{s}_0 \sim \mu(\cdot)$
- the randomness in state transitions: $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot\,|\boldsymbol{s}_t, a_t),\ \forall t \in \mathbb{N}_0$
- the stochasticity of policy π: $a_t \sim \pi(\cdot\,|\boldsymbol{s}_t),\ \forall t \in \mathbb{N}_0$

Lemma 2. Since $r_t \in [0, r_{\max}]$, we have:

$$0 \leq \sum_{t=0}^{\infty} \gamma^t r_t \leq \sum_{t=0}^{\infty} \gamma^t r_{\max} = \frac{r_{\max}}{1-\gamma}$$

Hence, the discounted sum of future rewards (or the discounted return) along any actual trajectory is always bounded in range $[0, \frac{r_{\max}}{1-\gamma}]$, and so is its expectation of any form.

24.1.4 Bellman equations

Definition 24.5 (**state value function**)**.** The state value function $V^\pi : \mathcal{S} \to \mathbb{R}$ of a MDP is defined as follow:

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_{a_t, \boldsymbol{s}_{t+1}, a_{t+1}, \dots}\left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, \boldsymbol{s}_t = \boldsymbol{s}\,; \pi, \mathcal{P}\right]$$

which is the expected value of return obtained by following policy π and transition function $\mathcal{P}$, starting at state $\boldsymbol{s}$, and:

- $a_{t+k} \sim \pi(\cdot\,|\boldsymbol{s}_{t+k}),\ \forall k \in \mathbb{N}_0$
- $\boldsymbol{s}_{t+1+k} \sim \mathcal{P}(\cdot\,|\boldsymbol{s}_{t+k}, a_{t+k}),\ \forall k \in \mathbb{N}_0$

Definition 24.6 (**state-action value function**)**.** The state-action value function(quality function) $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of a MDP is defined as follow:

$$Q^\pi(\boldsymbol{s}, a) = \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \boldsymbol{s}_{t+2}, \dots}\left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, \boldsymbol{s}_t = \boldsymbol{s}, a_t = a\,; \pi, \mathcal{P}\right]$$

which is the expected value of return obtained by following policy π and transition function $\mathcal{P}$, starting at state $\boldsymbol{s}$ and taking action a immediately, and:

- $\boldsymbol{s}_{t+1+k} \sim \mathcal{P}(\cdot\,|\boldsymbol{s}_{t+k}, a_{t+k}),\ \forall k \in \mathbb{N}_0$
- $a_{t+k} \sim \pi(\cdot\,|\boldsymbol{s}_{t+k}),\ \forall k \in \mathbb{N}_0$

Theorem 3 (**interconnection between state value function V^π and state-action value function Q^π**)**.** It directly follows from the definitions that state value function and state-action value function are deeply interconnected:

$$Q^\pi(\boldsymbol{s}_t, a_t) = \mathbb{E}_{\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot\,|\boldsymbol{s}_t, a_t)}\left[r_t + \gamma V^\pi(\boldsymbol{s}_{t+1})\right]$$
$$V^\pi(\boldsymbol{s}_t) = \mathbb{E}_{a_t \sim \pi(\cdot\,|\boldsymbol{s}_t)} Q^\pi(\boldsymbol{s}_t, a_t)$$

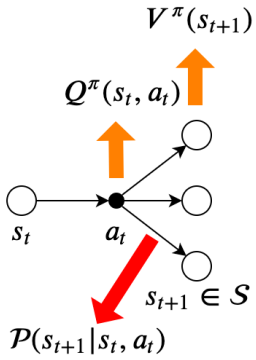


图 123: Illustrate the formula derivation from state-action value function Q^π to state value function V^π

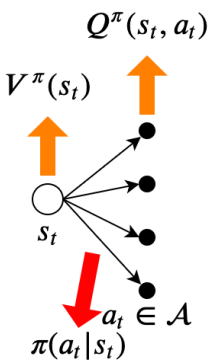


图 124: Illustrate the formula derivation from state value function V^π to state-action value function Q^π

Proof. Firstly we proof the equation: $Q^\pi(\boldsymbol{s}_t, a_t) = \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}) + \gamma V^\pi(\boldsymbol{s}_{t+1})\right]$.

$$
\begin{aligned}
Q^\pi(\boldsymbol{s}_t, a_t) &= \mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[r_t + \gamma\left(r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + \ldots\right)\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[r_t + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] \text{ (pull out the immediate reward alone)} \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[r_t\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] + \mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[\gamma\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t\Big|\boldsymbol{s}_t, a_t\,;\mathcal{P}\right] + \mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[\gamma\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1})\right] + \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[\mathbb{E}_{a_{t+1},\boldsymbol{s}_{t+2},a_{t+2},\ldots}\left[\gamma\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\Big|\boldsymbol{s}_{t+1}\,;\pi,\mathcal{P}\right]\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1})\right] + \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[\gamma\underbrace{\mathbb{E}_{a_{t+1},\boldsymbol{s}_{t+2},a_{t+2},\ldots}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\Big|\boldsymbol{s}_{t+1}\,;\pi,\mathcal{P}\right]}_{\text{definition of } V^\pi(\boldsymbol{s}_{t+1})}\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t + \gamma V^\pi(\boldsymbol{s}_{t+1})\right]
\end{aligned}
$$

Similarly, we proof the equation: $V^\pi(\boldsymbol{s}_t) = \mathbb{E}_{a_t\sim\pi(\cdot\,|\boldsymbol{s}_t)}Q^\pi(\boldsymbol{s}_t, a_t)$ as follow:

$$
\begin{aligned}
V^\pi(\boldsymbol{s}_t) &= \mathbb{E}_{a_t,\boldsymbol{s}_{t+1},a_{t+1},\ldots}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\,\Big|\boldsymbol{s}_t\,;\pi,\mathcal{P}\right] \\
&= \mathbb{E}_{a_t\sim\pi(\cdot\,|\boldsymbol{s}_t)}\left[\underbrace{\mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\boldsymbol{s}_{t+2},\ldots}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\,\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right]}_{\text{definition of } Q^\pi(\boldsymbol{s}_t,\,a_t)}\right] \\
&= \mathbb{E}_{a_t\sim\pi(\cdot\,|\boldsymbol{s}_t)}Q^\pi(\boldsymbol{s}_t, a_t)
\end{aligned}
$$

Theorem 4 (**Bellman equation for the state value function**)**.** It directly follows the definitions that the Bellman equation of state value function can be formulated as follow:

$$
V^\pi(\boldsymbol{s}_t) = \sum_{a_t\in\mathcal{A}}\pi(a_t\,|\boldsymbol{s}_t)\sum_{\boldsymbol{s}_{t+1}\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_{t+1}\,|\boldsymbol{s}_t, a_t)\Big(r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}) + \gamma V^\pi(\boldsymbol{s}_{t+1})\Big)
$$

Proof. Based on the interconnection derivation from state value function to state-action value function: $V^\pi(\boldsymbol{s}_t) = \mathbb{E}_{a_t\sim\pi(\cdot\,|\boldsymbol{s}_t)}Q^\pi(\boldsymbol{s}_t, a_t)$ and interconnection derivation from state-action value function to state value function $Q^\pi(\boldsymbol{s}_t, a_t) = \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t + \gamma V^\pi(\boldsymbol{s}_{t+1})\right]$, we can make a derivation as follow:

$$
V^\pi(\boldsymbol{s}_t) = \mathbb{E}_{a_t\sim\pi(\cdot\,|\boldsymbol{s}_t)}Q^\pi(\boldsymbol{s}_t, a_t)
$$

$$
\begin{aligned}
&= \mathbb{E}_{a_t\sim\pi(\cdot\,|\boldsymbol{s}_t)}\mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t+\gamma V^\pi(\boldsymbol{s}_{t+1})\right] \\
&= \sum_{a_t\in\mathcal{A}}\pi(a_t\,|\boldsymbol{s}_t)\left(\sum_{\boldsymbol{s}_{t+1}\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_{t+1}\,|\boldsymbol{s}_t,a_t)\Big(r_t+\gamma V^\pi(\boldsymbol{s}_{t+1})\Big)\right) \\
&= \sum_{a_t\in\mathcal{A}}\pi(a_t\,|\boldsymbol{s}_t)\sum_{\boldsymbol{s}_{t+1}\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_{t+1}\,|\boldsymbol{s}_t,a_t)\Big(r(\boldsymbol{s}_t,a_t,\boldsymbol{s}_{t+1})+\gamma V^\pi(\boldsymbol{s}_{t+1})\Big)
\end{aligned}
$$

Theorem 5 (**Bellman equation for the state-action value function**)**.** It directly follows the definitions that the Bellman equation of state-action value function can be formulated as follow:

$$
Q^\pi(\boldsymbol{s}_t,a_t)=\sum_{\boldsymbol{s}_{t+1}\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_{t+1}\,|\boldsymbol{s}_t,a_t)\left(r(\boldsymbol{s}_t,a_t,\boldsymbol{s}_{t+1})+\gamma\sum_{a_{t+1}\in\mathcal{A}}\pi(a_{t+1}\,|\boldsymbol{s}_{t+1})Q^\pi(\boldsymbol{s}_{t+1},a_{t+1})\right)
$$

Proof. Based on the interconnection derivation from state-action value function to state value function $Q^\pi(\boldsymbol{s}_t,a_t)=\mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t+\gamma V^\pi(\boldsymbol{s}_{t+1})\right]$ and interconnection derivation from state value function to state-action value function: $V^\pi(\boldsymbol{s}_t)=\mathbb{E}_{a_t\sim\pi(\cdot\,|\boldsymbol{s}_t)}Q^\pi(\boldsymbol{s}_t,a_t)$, we can make a derivation as follow:

$$
\begin{aligned}
Q^\pi(\boldsymbol{s}_t,a_t) &= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\Big[r_t+\gamma V^\pi(\boldsymbol{s}_{t+1})\,\Big|\,\boldsymbol{s}_t,a_t\,;\pi,\mathcal{P}\Big] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\Big[r_t+\gamma\mathbb{E}_{a_{t+1}\sim\pi(\cdot\,|\boldsymbol{s}_{t+1})}Q^\pi(\boldsymbol{s}_{t+1},a_{t+1})\,\Big|\,\boldsymbol{s}_t,a_t\,;\pi,\mathcal{P}\Big] \\
&= \sum_{\boldsymbol{s}_{t+1}\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_{t+1}\,|\boldsymbol{s}_t,a_t)\left(r_t+\gamma\sum_{a_{t+1}\in\mathcal{A}}\pi(a_{t+1}\,|\boldsymbol{s}_{t+1})Q^\pi(\boldsymbol{s}_{t+1},a_{t+1})\right) \\
&= \sum_{\boldsymbol{s}_{t+1}\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_{t+1}\,|\boldsymbol{s}_t,a_t)\left(r(\boldsymbol{s}_t,a_t,\boldsymbol{s}_{t+1})+\gamma\sum_{a_{t+1}\in\mathcal{A}}\pi(a_{t+1}\,|\boldsymbol{s}_{t+1})Q^\pi(\boldsymbol{s}_{t+1},a_{t+1})\right)
\end{aligned}
$$

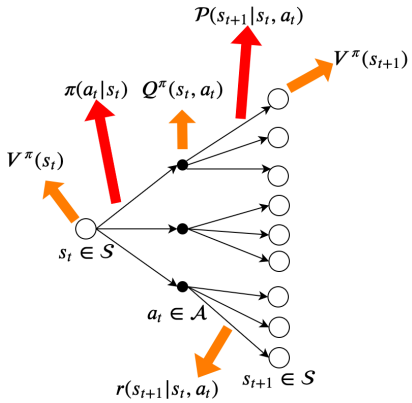


图 125: Illustrate the formula derivation of Bellman equation for state value function V^π

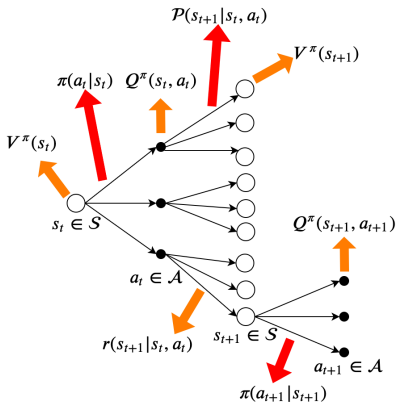


图 126: Illustrate the formula derivation of Bellman equation for state-action value function Q^π

24.1.5 Bellman optimality equations

Definition 24.7 (**optimal state value function**). The optimal state value function $V^*(\boldsymbol{s})$ is the maximum state value function over all policies:

$$V^*(\boldsymbol{s}) = \max_{\pi \in \Pi} V^\pi(\boldsymbol{s}),\ \forall \boldsymbol{s} \in \mathcal{S}$$

Definition 24.8 (**optimal state-action value function**). The optimal state-action value function(optimal quality function) $Q^*(\boldsymbol{s}, a)$ is the maximum state-action value function over all policies:

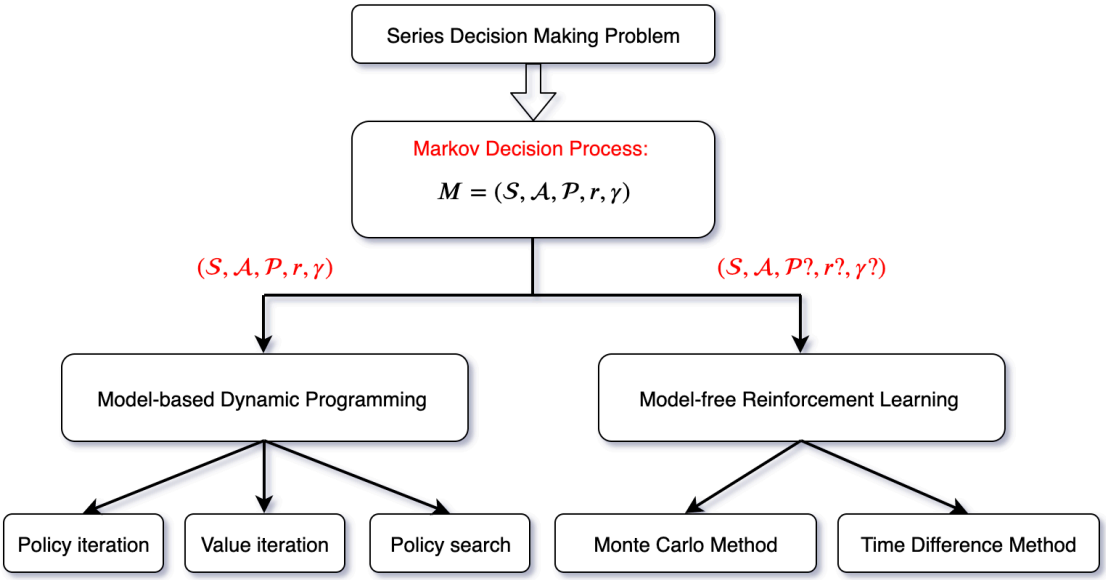
$$Q^*(\boldsymbol{s}, a) = \max_{\pi \in \Pi} Q^\pi(\boldsymbol{s}, a),\ \forall \boldsymbol{s} \in \mathcal{S}, a \in \mathcal{A}$$

Theorem 6 (**interconnection between optimal state value function V^* and optimal state-action value function Q^***). V^* and Q^* satisfy the following set of Bellman optimality equations:

$$\begin{aligned} V^*(\boldsymbol{s}) &= \max_{a \in \mathcal{A}} Q^*(\boldsymbol{s}, a) \\ Q^*(\boldsymbol{s}, a) &= \mathbb{E}_{\boldsymbol{s}' \sim \mathcal{P}(\cdot \mid \boldsymbol{s}, a)} \left[r(\boldsymbol{s}, a, \boldsymbol{s}') + \gamma V^*(\boldsymbol{s}') \right] \end{aligned}$$

Theorem 7 (**Bellman equation for the optimal state value function**). c

24.1.6 Summary



24.1.7 Reference

- CS 598 Statistical Reinforcement Learning (S19)
- Understanding RL: The Bellman Equations
- Lil'Log: A (Long) Peek into Reinforcement Learning

24.2 Semi-Markov Decision Processes (Semi-MDPs)（未完成）

24.3 Partially Observable Markov Decision Processes (POMDPs)

24.3.1 Definition

In general, Partially Observable Markov Decision Processes(POMDPs) can be defined by a siven-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \Omega, \mathcal{O}, \gamma >$, where:

24.3.2 Reference

24.4 Dynamic Programming (DP) in MDPs

从贝尔曼方程可知，如果马尔科夫决策过程 $\mathcal{M}=(\mathcal{S},\mathcal{A},\mathcal{P},r,\gamma)$ 的状态转移概率 $\mathcal{P}(s'|s,a)$ 和即时奖励 $r(s,a,s')$ 已知，那么可以直接通过贝尔曼方程来迭代计算其值函数。这种模型已知的强化学习算法也称为**基于模型的强化学习**（Model-based Reinforcement Learning）算法，这里的**模型**就是指马尔可夫决策过程。

在已知模型时，可以通过动态规划（Dynamic Programming, DP）的方法来计算。常用的方法主要有策略迭代（Policy Iteration）算法和值迭代（Value Iteration）算法。

24.4.1 Value iteration

Algorithm 33: Value Iteration Algorithm

Input: MDPs five-tuple: $\mathcal{M}=(\mathcal{S},\mathcal{A},\mathcal{P},r,\gamma)$;
A small threshold: $\epsilon>0$.

Output: Policy $\pi(\cdot):\mathcal{S}\to\mathcal{A}$.

Initialization: $\forall s\in\mathcal{S}$, $V(s)=0$;

repeat

  Update value function by Bellman equation:

$$\forall s\in\mathcal{S},\ V(s)\leftarrow\max_{a\in\mathcal{A}}\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\Big[r(s,a,s')+\gamma V(s')\Big]$$

until $\forall s\in\mathcal{S}, V(s)$ *converges*;

Compute $Q(s,a)$ based on following equation:

$$Q^{\pi}(s,a)=\mathbb{E}_{s'\sim\mathcal{P}(\cdot\,|s,a)}\Big[r(s,a,s')+\gamma V^{\pi}(s')\Big]$$

Get policy $\pi(\cdot)$ based on $Q(s,a)$:

$$\forall s\in\mathcal{S},\ \pi(s)=\arg\max_{a\in\mathcal{A}}Q(s,a)$$

24.4.2 Policy iteration

Algorithm 34: Policy Iteration Algorithm

Input: MDPs five-tuple: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$;
A small threshold: $\epsilon > 0$.

Output: Policy $\pi(\cdot) : \mathcal{S} \to \mathcal{A}$.

Initialization: $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$, $\pi(a|s) = \frac{1}{|\mathcal{A}|}$;

repeat

  // Policy Evaluation

  repeat

    Compute $V(s)$ by Bellman equation:

$$\forall s \in \mathcal{S},\ V(s) \leftarrow \mathbb{E}_{a \sim \pi(\cdot\,|s)} \mathbb{E}_{s' \sim \mathcal{P}(\cdot\,|s,a)} \Big[r(s,a,s') + \gamma V^{\pi}(s') \Big]$$

  until $\forall s \in \mathcal{S}$, $V^{\pi}(s)$ *converges*;

  // Policy Improvement

  Compute Q-function based on following equation:

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot\,|s,a)} \Big[r(s,a,s') + \gamma V^{\pi}(s') \Big]$$

  Get policy $\pi(\cdot)$ based on $Q(s,a)$:

$$\forall s \in \mathcal{S},\ \pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$$

until $\forall s \in \mathcal{S}$, $\pi(\cdot|s)$ *converges*;

24.4.3 Reference

- Value Iteration - Artificial Intelligence: Foundations of Computational Agents

24.5 Applications of Reinforcement Learning Algorithms

24.5.1 Survey

- Reinforcement Learning Applications

24.5.2 Finance

- Deep Reinforcement Learning for Foreign Exchange Trading

24.5.3 Healthcare

- Reinforcement Learning in Healthcare: A Survey

24.5.4 Robotics

24.5.5 Transportation

- Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control[?]

24.5.6 Recommender Systems

- Deep Reinforcement Learning for List-wise Recommendations[?]

24.6 Frameworks of Reinforcement Learning Algorithms

24.6.1 Commerical

- TF-Agents: A library for Reinforcement Learning in TensorFlow[?]
- OpenAI Baselines: high-quality implementations of reinforcement learning algorithms[?]
- Tensorforce: a TensorFlow library for applied reinforcement learning[?]
- RLlib: Scalable Reinforcement Learning[?]
- Dopamine: a research framework for fast prototyping of reinforcement learning algorithms[?]

24.6.2 Educational

- PyTorch Implementations of Policy Gradient Methods
- Vel (PyTorch)[?]
- RL-Adventure-2 (PyTorch)
- TensorFlow-Model: efficient-hrl
- OpenAI-Spinningup: an educational resource to help anyone learn deep reinforcement learning.
- TRFL: TensorFlow Reinforcement Learning

24.7 Summary of Reinforcement Learning Algorithms

24.7.1 Model-free RL

Algorithm	Policy type	Policy update	State space	Action space
VPG	Policy-based	On-policy	Continuous state space	Discrete action space
DQN	Value-based	Off-policy	Continuous state space	Discrete action space

25 Policy-Based Deep Reinforcement Learning Algorithms

25.1 Vanilla Policy Gradient (VPG)

25.1.1 Introduction

本节选自Policy Gradient Methods for Reinforcement Learning with Function Approximation[?]

Notation

Symbol	Meaning
$\boldsymbol{s} \in \mathcal{S}$	State.
$a \in \mathcal{A}$	Action.
$\mathcal{P}(\boldsymbol{s}'\|\boldsymbol{s}, a)$	Transition probability of getting to the next state $\boldsymbol{s}'$ from the current state $\boldsymbol{s}$ with action a.
$r(\boldsymbol{s}, a, \boldsymbol{s}')$	Reward.
γ	Discounted factor; penalty to uncertainty of future rewards; $\gamma \in (0, 1)$.
$(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}, r_t, d)$	Transition tuple at time step t of a tarjectory.
$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$	Return, or discounted sum of future rewards.
$\pi(\cdot\|\boldsymbol{s})$	Stochastic policy (agent behavior strategy): $\mathcal{S} \to \mathbb{P}^{\|\mathcal{A}\|}$. $\pi(\cdot\|\boldsymbol{s}; \boldsymbol{\theta})$ or $\pi_{\boldsymbol{\theta}}(\cdot\|\boldsymbol{s})$ is a policy parameterized by $\boldsymbol{\theta}$.
$\mu(\boldsymbol{s})$	Deterministic policy: $\mathcal{S} \to \mathcal{A}$, we can also label this as $\pi(\boldsymbol{s})$, but using a different letter gives better distinction so that we can easily tell when the policy is stochastic or deterministic without further explanation. Either π or μ is what a RL algorithm aims to learn.
$V(\boldsymbol{s})$	State value function measures the expected return under state $\boldsymbol{s}$
$V^\pi(\boldsymbol{s})$	The state value under state $\boldsymbol{s}$ when we follow a policy π. $V^\pi(\boldsymbol{s}) = \mathbb{E}_{a_t, \boldsymbol{s}_{t+1}, \dots}\left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle\|\, \boldsymbol{s}_t = \boldsymbol{s}\,; \pi, \mathcal{P}\right]$
$Q(\boldsymbol{s}, a)$	State-action value function measures the expected return of a pair of state and action $(\boldsymbol{s}, a)$.
$Q^\pi(\boldsymbol{s}, a)$	The value of (state, action) pair when we follow a policy π. $Q^\pi(\boldsymbol{s}, a) = \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \boldsymbol{s}_{t+2}, \dots}\left[R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle\|\, \boldsymbol{s}_t = \boldsymbol{s}, a_t = a\,; \pi, \mathcal{P}\right]$
$A^\pi(\boldsymbol{s}, a)$	Advantage function, $A^\pi(\boldsymbol{s}, a) = Q^\pi(\boldsymbol{s}, a) - V^\pi(\boldsymbol{s})$. it can be considered as another version of Q-value with lower variance by taking the state-value off as the baseline.

25.1.2 Model formulation

Objective The goal of the reinforcement learning is to choose a policy π to maximize the expected return:

$$\max_{\pi_{\boldsymbol{\theta}}} \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\ldots}\left[R_0=\sum_{t=0}^{\infty}\gamma^t r_t\middle|\pi_{\boldsymbol{\theta}},\mathcal{P},\mu\right]$$
$$\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[R(\tau)=\sum_{t=0}^{T(\tau)-1}\gamma^t r_t^\tau\right]$$

The expectation is with respect to the randomness of the observation trajectory $\tau=(\boldsymbol{s}_0^\tau,a_0^\tau,r_0^\tau,\boldsymbol{s}_1^\tau,a_1^\tau,r_1^\tau,\boldsymbol{s}_2^\tau,...,\boldsymbol{s}_{T(\tau)}^\tau)$. Assume all trajectories are i.i.d, so we can rewrite the objective as follow:

$$\max_{\boldsymbol{\theta}}\int_{\tau\in\mathcal{T}} R(\tau)\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)d\tau$$

Here $\mathcal{T}$ is the trajectory space. And for each trajectory τ, the probability density function of trajectory $\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)$ depends on there parameter components:

- $\boldsymbol{s}_0^\tau\sim\mu(\cdot),\ \forall\tau\in\mathcal{T}$
- $a_t^\tau\sim\pi(\cdot\,|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}),\ \forall\tau\in\mathcal{T},\forall\, t\in\mathbb{N}_0$
- $\boldsymbol{s}_{t+1}^\tau\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t^\tau,a_t^\tau),\ \forall\tau\in\mathcal{T},\forall\, t\in\mathbb{N}_0$

Considering the Markov property on state-action path $(\boldsymbol{s}_0^\tau,a_0^\tau,\boldsymbol{s}_1^\tau,a_1^\tau,\boldsymbol{s}_2^\tau,...,\boldsymbol{s}_{T(\tau)}^\tau)$ with length $T(\tau)+1$, we can rewrite the probability density function of trajectory $\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)$ as follow:

$$\begin{aligned}\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)&=\mu(\boldsymbol{s}_0^\tau)\cdot\pi(a_0^\tau\,|\boldsymbol{s}_0^\tau;\boldsymbol{\theta})\cdot\mathcal{P}(\boldsymbol{s}_1^\tau\,|\boldsymbol{s}_0^\tau,a_0^\tau)\cdot\pi(a_1^\tau\,|\boldsymbol{s}_1^\tau;\boldsymbol{\theta})\cdot\mathcal{P}(\boldsymbol{s}_2^\tau\,|\boldsymbol{s}_1^\tau,a_1^\tau)\cdot\ldots\cdot\mathcal{P}(\boldsymbol{s}_{T(\tau)}^\tau\,|\boldsymbol{s}_{T(\tau)-1}^\tau,a_{T(\tau)-1}^\tau)\\&=\mu(\boldsymbol{s}_0^\tau)\cdot\prod_{t=0}^{T(\tau)-1}\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})\cdot\prod_{t=0}^{T(\tau)-1}\mathcal{P}(\boldsymbol{s}_{t+1}^\tau\,|\boldsymbol{s}_t^\tau,a_t^\tau)\end{aligned}$$

Replace $\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)$ in the original objective function with this formula, we can rewrite the objective as follow:

$$\begin{aligned}&\max_{\boldsymbol{\theta}}\int_{\tau\in\mathcal{T}}R(\tau)\left(\mu(\boldsymbol{s}_0^\tau)\cdot\prod_{t=0}^{T(\tau)-1}\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})\cdot\prod_{t=0}^{T(\tau)-1}\mathcal{P}(\boldsymbol{s}_{t+1}^\tau\,|\boldsymbol{s}_t^\tau,a_t^\tau)\right)d\tau\\
\Rightarrow&\max_{\boldsymbol{\theta}}\int_{\tau\in\mathcal{T}}R(\tau)\log\left(\mu(\boldsymbol{s}_0^\tau)\cdot\prod_{t=0}^{T(\tau)-1}\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})\cdot\prod_{t=0}^{T(\tau)-1}\mathcal{P}(\boldsymbol{s}_{t+1}^\tau\,|\boldsymbol{s}_t^\tau,a_t^\tau)\right)d\tau\\
\Rightarrow&\max_{\boldsymbol{\theta}}\int_{\tau\in\mathcal{T}}R(\tau)\left(\log\mu(\boldsymbol{s}_0^\tau)+\log\prod_{t=0}^{T(\tau)-1}\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})+\log\prod_{t=0}^{T(\tau)-1}\mathcal{P}(\boldsymbol{s}_{t+1}^\tau\,|\boldsymbol{s}_t^\tau,a_t^\tau)\right)d\tau\\
\Rightarrow&\max_{\boldsymbol{\theta}}\int_{\tau\in\mathcal{T}}R(\tau)\left(\log\mu(\boldsymbol{s}_0^\tau)+\sum_{t=0}^{T(\tau)-1}\log\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})+\sum_{t=0}^{T(\tau)-1}\log\mathcal{P}(\boldsymbol{s}_{t+1}^\tau\,|\boldsymbol{s}_t^\tau,a_t^\tau)\right)d\tau\\
\Rightarrow&\max_{\boldsymbol{\theta}}\underbrace{\cancel{\int_{\tau\in\mathcal{T}}R(\tau)\log\mu(\boldsymbol{s}_0^\tau)d\tau}}_{\text{constant}}+\int_{\tau\in\mathcal{T}}R(\tau)\sum_{t=0}^{T(\tau)-1}\log\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})d\tau+\underbrace{\cancel{\int_{\tau\in\mathcal{T}}R(\tau)\sum_{t=0}^{T(\tau)-1}\log\mathcal{P}(\boldsymbol{s}_{t+1}^\tau\,|\boldsymbol{s}_t^\tau,a_t^\tau)d\tau}}_{\text{constant}}\end{aligned}$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \int_{\tau \in \mathcal{T}} R(\tau) \sum_{t=0}^{T(\tau)-1} \log \pi(a_t^\tau | \boldsymbol{s}_t^\tau; \boldsymbol{\theta}) d\tau$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} R(\tau^{(i)}) \sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | \boldsymbol{s}_t^i; \boldsymbol{\theta}) \quad \text{(sampling } N \text{ trajectories from } \tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu))$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \left(\sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | \boldsymbol{s}_t^i; \boldsymbol{\theta}) \right) \left(\sum_{t=0}^{T(\tau^{(i)})-1} r_t^i \right)$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \left(\sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i | \boldsymbol{s}_t^i; \boldsymbol{\theta}) \right) \left(\underbrace{\sum_{t=0}^{T(\tau^{(i)})-1} r(\boldsymbol{s}_t^i, a_t^i, \boldsymbol{s}_{t+1}^i)}_{R(\tau^{(i)})} \right)$$

So if we note objective as follow:

$$J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \log \pi(a_t^i | \boldsymbol{s}_t^i; \boldsymbol{\theta})$$

we can get the gradient of parameters:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \nabla_{\boldsymbol{\theta}} \log \pi(a_t^i | \boldsymbol{s}_t^i; \boldsymbol{\theta})$$

we can update parameters in each policy iteration:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

where η is learning rate.

Here we know a stochastic policy π is the function representing a mapping from state to probability distribution over actions:

$$\pi(\cdot \,| \boldsymbol{s}; \boldsymbol{\theta}) : \mathcal{S} \rightarrow \mathbb{P}^{|\mathcal{A}|}$$

then we consider that each action a is sampled from its corresponding known action probability distribution $\boldsymbol{p}$ (e.g. $a = 2$ is sampled from its corresponding action probability distribution $\boldsymbol{p} = (0.02, 0.01, 0.92, 0.05)$):

$$a \in \mathcal{A} \sim \boldsymbol{p} \in \mathbb{P}^{|\mathcal{A}|}$$

Then we can convert **maximizing log likelihood** objective to **minimizing KL-divergence** as follow:

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \log \pi(a_t^i | \boldsymbol{s}_t^i; \boldsymbol{\theta})$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) D_{\mathrm{KL}} \left(\boldsymbol{p}_t^i \,\|\, \pi(\cdot \,| \boldsymbol{s}_t^i; \boldsymbol{\theta}) \right)$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \frac{p_{t,k}^i}{\pi_{t,k}^i(\boldsymbol{\theta})} \quad (\pi_{t,k}^i(\boldsymbol{\theta}) = \pi_k(\cdot \,| \boldsymbol{s}_t^i; \boldsymbol{\theta}))$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \left(\log p_{t,k}^i - \log \pi_{t,k}^i(\boldsymbol{\theta})\right)$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \underbrace{\cancel{\frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log p_{t,k}^i}}_{\text{constant}} - \frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \pi_{t,k}^i(\boldsymbol{\theta})$$

$$\Rightarrow \min_{\boldsymbol{\theta}} -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \pi_{t,k}^i(\boldsymbol{\theta})$$

So we get the final loss function of MCPG:

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} R(\tau^{(i)}) \sum_{k=0}^{|\mathcal{A}|-1} p_{t,k}^i \log \pi_{t,k}^i(\boldsymbol{\theta})$$

namely a weighted cross-entropy.

problem of policy gradient:

- high variance
- hard to choose learning rate

solution:

- variance reduction: **policy at timestep t'(after) cannot affect reward at time t(before) when** $(t < t')$, therefore we can replace objective as:

$$J(\boldsymbol{\theta}) \approx -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i|\boldsymbol{s}_t^i;\boldsymbol{\theta}) \underbrace{\left(\sum_{t'=t}^{T(\tau^{(i)})-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}^i, a_{t'}^i, \boldsymbol{s}_{t'+1}^i)\right)}_{\text{reward-to-go}}$$

  Therefore, **the cumulative discounted reward occur after corresponding state-action pair**

- baselines:

$$\min_{\boldsymbol{\theta}} -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i|\boldsymbol{s}_t^i;\boldsymbol{\theta}) R(\tau^{(i)})$$

$$\Rightarrow \min_{\boldsymbol{\theta}} -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1} \log \pi(a_t^i|\boldsymbol{s}_t^i;\boldsymbol{\theta}) \left(R(\tau^{(i)}) - b\right)$$

  where,

$$b = \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[R(\tau)\right] \approx \frac{1}{N}\sum_{i=0}^{N-1} R(\tau^{(i)})$$

  1. subtracting a baseline is unbiased in expectation
  2. average reward is not the best baseline, but it ' s pretty good

Training procedure Pseudocode of Policy Gradient with Reward-to-Go is as follow:

Algorithm 35: Policy Gradient with Reward-to-Go

Sample a set of trajectories $\{\tau^{(i)}\}_{i=1}^{N}$ under policy $\pi(\cdot\,|\boldsymbol{s};\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$

Compute gradients wrt objective:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_{\boldsymbol{\theta}}\log\pi(a_t^i|\boldsymbol{s}_t^i;\boldsymbol{\theta})\left(\underbrace{\sum_{t'=t}^{T}\gamma^{t'-t}r(\boldsymbol{s}_{t'}^i,a_{t'}^i,\boldsymbol{s}_{t'+1}^i)}_{\text{reward-to-go }\hat{Q}(\boldsymbol{s}_t^i,a_t^i)}\right)\right)$$

Update parameters of policy:

$$\boldsymbol{\theta}\leftarrow\boldsymbol{\theta}+\alpha\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$$

25.1.3 Forms of the policy gradient summary

1. **objective of vanilla policy gradient**:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)R(\tau)\right]$$
$$\approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)R(\tau)$$

   where $\forall\tau\in\mathcal{D}$ is sampled following distribution $\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)$.

2. **objective of policy gradient with Monte Carlo baseline**:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(R(\tau)-b\right)\right]$$
$$\approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(R(\tau)-b\right)$$

   where the baseline(global baseline) b can be estimated by Monte Carlo sampling:

$$b = \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}R(\tau) \approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}R(\tau)$$

   where $\forall\tau\in\mathcal{D}$ is sampled following distribution $\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)$.

3. **objective of policy gradient with reward-to-go**:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\underbrace{\sum_{t'=t}^{T(\tau)-1}\gamma^{t'-t}r(\boldsymbol{s}_{t'}^\tau,a_{t'}^\tau,\boldsymbol{s}_{t'+1}^\tau)}_{\text{reward-to-go: }\hat{Q}(\boldsymbol{s}_t^\tau,a_t^\tau)}\right)\right]$$
$$\approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\sum_{t'=t}^{T(\tau)-1}\gamma^{t'-t}r(\boldsymbol{s}_{t'}^\tau,a_{t'}^\tau,\boldsymbol{s}_{t'+1}^\tau)\right)$$

   where $\forall\tau\in\mathcal{D}$ is sampled following distribution $\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)$.

4. **objective of policy gradient with both reward-to-go and Monte Carlo baseline**:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\sum_{t'=t}^{T(\tau)-1}\gamma^{t'-t}r(\boldsymbol{s}_{t'}^\tau,a_{t'}^\tau,\boldsymbol{s}_{t'+1}^\tau)-b(\boldsymbol{s}_t^\tau)\right)\right]$$
$$\approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\sum_{t'=t}^{T(\tau)-1}\gamma^{t'-t}r(\boldsymbol{s}_{t'}^\tau,a_{t'}^\tau,\boldsymbol{s}_{t'+1}^\tau)-b(\boldsymbol{s}_t^\tau)\right)$$

   where the baseline(baseline of state) $b(\boldsymbol{s}_t^\tau)$ can be estimated by Monte Carlo sampling:

$$b(\boldsymbol{s}) = \hat{V}^\pi(\boldsymbol{s}) \approx V^\pi(\boldsymbol{s})$$

$$
\begin{aligned}
&= \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[R(\tau) = \sum_{t=0}^{T(\tau)-1}\gamma^t r(\boldsymbol{s}_t^\tau, a_t^\tau, \boldsymbol{s}_{t+1}^\tau)\,\middle|\,\boldsymbol{s}_0^\tau = \boldsymbol{s}\right] \\
&\approx \frac{1}{|\mathcal{B}|}\sum_{\tau\in\mathcal{B}}\left[R(\tau)\,\middle|\,\boldsymbol{s}_0^\tau = \boldsymbol{s}\right] \\
&\approx \frac{1}{|\mathcal{B}|}\sum_{\tau\in\mathcal{B}}\left[\sum_{t=0}^{T(\tau)-1}\gamma^t r(\boldsymbol{s}_t^\tau, a_t^\tau, \boldsymbol{s}_{t+1}^\tau)\,\middle|\,\boldsymbol{s}_0^\tau = \boldsymbol{s}\right]
\end{aligned}
$$

where $\forall\tau\in\mathcal{B}$ is sampled following distribution $\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)$. **it requires us to reset the simulator**, namely in order to computing baseline of state $\boldsymbol{s}_t^\tau$, we need to **resample $|\mathcal{B}|$ tarjectories starting from specific state $\boldsymbol{s}$**

5. **objective of policy gradient with on policy action-state value function**:

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^\tau, a_t^\tau)\right] \\
&\approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\hat{Q}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^\tau, a_t^\tau;\boldsymbol{\phi})
\end{aligned}
$$

where $Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, a)$ is the true underlying value function, and $\hat{Q}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, a;\boldsymbol{\phi})$ is the approximate value function with parameters $\boldsymbol{\phi}$.

6. **objective of policy gradient with advantage function(indeed value function)**:

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\sum_{t'=t}^{T(\tau)-1}\gamma^{t'-t}r(\boldsymbol{s}_{t'}^\tau, a_{t'}^\tau, \boldsymbol{s}_{t'+1}^\tau) - b(\boldsymbol{s}_t^\tau)\right)\right] \\
&\approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\sum_{t'=t}^{T(\tau)-1}\gamma^{t'-t}r(\boldsymbol{s}_{t'}^\tau, a_{t'}^\tau, \boldsymbol{s}_{t'+1}^\tau) - b(\boldsymbol{s}_t^\tau)\right) \\
&= \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\hat{Q}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^\tau, a_t^\tau) - \hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^\tau)\right) \\
&= \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\hat{A}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^\tau, a_t^\tau) \\
&\approx \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T(\tau)-1}\log\pi_{\boldsymbol{\theta}}(a_t^\tau|\boldsymbol{s}_t^\tau)\left(\underbrace{r(\boldsymbol{s}_t^\tau, a_t^\tau, \boldsymbol{s}_{t+1}^\tau) + \gamma\hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_{t+1}^\tau)}_{\hat{Q}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^\tau, a_t^\tau)} - \hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^\tau)\right)
\end{aligned}
$$

So **only need to fit value function $V^\pi(\boldsymbol{s})$!**

Difference between advantage function and reward-to-go minus Monte Carlo baseline:

- advantage function:

$$\hat{A}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, a) = r(\boldsymbol{s}, a, \boldsymbol{s}') + \gamma\hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}') - \hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s})$$

indeed we estimate the value function $V^{\pi_{\boldsymbol{\theta}}}(\cdot)$, **it's biased estimation, but the better this estimate, the lower the variance**.

- reward-to-go minus Monte Carlo baseline:

$$\sum_{t'=t}^{T-1} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, a_{t'}, \boldsymbol{s}_{t'+1}) - b(\boldsymbol{s}_t)$$

it's unbiased, but high variance single-sample estimate.

So until now, the problem is how to fit value function(advantage value function) ?

(a) Monte Carlo value function

$$\begin{aligned} V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}) &\approx \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(\boldsymbol{s}_t^\tau, a_t^\tau, \boldsymbol{s}_{t+1}^\tau) \,\middle|\, \boldsymbol{s}_0^\tau = \boldsymbol{s}\,; \pi_{\boldsymbol{\theta}}, \mathcal{P} \right] \\ &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau\in\mathcal{D}} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(\boldsymbol{s}_t^\tau, a_t^\tau, \boldsymbol{s}_{t+1}^\tau) \,\middle|\, \boldsymbol{s}_0^\tau = \boldsymbol{s} \right] \end{aligned}$$

where $\forall\tau \in \mathcal{D}$ is sampled following distribution $\tau \sim \rho(\tau; \pi_{\boldsymbol{\theta}}, \mathcal{P}, \mu)$.

(b) Monte Carlo with value function approximation

- Monte Carlo(MC) target of value function:

$$\begin{aligned} y(\boldsymbol{s}) &= V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}) \\ &= \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)} \left[\sum_{t=0}^{T(\tau)-1} \gamma^t r(\boldsymbol{s}_t^\tau, a_t^\tau, \boldsymbol{s}_{t+1}^\tau) \,\middle|\, \boldsymbol{s}_0^\tau = \boldsymbol{s}\,; \pi_{\boldsymbol{\theta}}, \mathcal{P} \right] \\ &\approx \sum_{t=0}^{T-1} \gamma^t r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}) \Big|_{\boldsymbol{s}_0=\boldsymbol{s}} \quad \text{(directly sample once in current trajectory)} \end{aligned}$$

for time feasibility, sampling only one trajectory starting from state $\boldsymbol{s}$, if we can improve sampling efficiency later, we can use expectation of state function belongs to multiple trajectories starting from state $\boldsymbol{s}$ as Monte Carlo target.

– **problems to be discussed**

i. why not use $\frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=0}^{T}\gamma^t r(\boldsymbol{s}_t^i, a_t^i, \boldsymbol{s}_{t+1}^i)\Big|_{\boldsymbol{s}_0^i=\boldsymbol{s}}\right]$ as Monte Carlo target of $\boldsymbol{s}$ in value function approximation task, but only use one observation ?

ii. is there any other method for policy evaluation ?

then we get training dataset with MC target:

$$\mathcal{D} = \left\{ \boldsymbol{s}, \underbrace{\sum_{t=0}^{T} \gamma^t r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}) \Big|_{\boldsymbol{s}_0=\boldsymbol{s},\, a_t\sim\pi(\cdot\,|\,\boldsymbol{s}_t),\, \boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\,\boldsymbol{s}_t,a_t)}}_{\text{MC target } y} \right\}_{\boldsymbol{s}\in\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}$$

- Fit value function by regression task on mean squared error loss:

$$\boldsymbol{\phi} = \arg\min_{\boldsymbol{\phi}} l(\boldsymbol{\phi}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{s},y)\in\mathcal{D}} \|\hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s};\boldsymbol{\phi}) - y\|^2$$

(c) Temporal Difference with value function approximation (**used in Actor-Critic**)

- Temporal Difference(TD) target of value function:

$$
\begin{aligned}
y(\boldsymbol{s}) &= V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}) \\
&= \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\gamma^t r(\boldsymbol{s}_t^\tau,a_t^\tau,\boldsymbol{s}_{t+1}^\tau)\,\middle|\,\boldsymbol{s}_0^\tau=\boldsymbol{s};\pi_{\boldsymbol{\theta}},\mathcal{P}\right] \\
&= \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[r(\boldsymbol{s}_0^\tau,a_0^\tau,\boldsymbol{s}_1^\tau)+\sum_{t=1}^{T(\tau)-1}\gamma^t r(\boldsymbol{s}_t^\tau,a_t^\tau,\boldsymbol{s}_{t+1}^\tau)\,\middle|\,\boldsymbol{s}_0^\tau=\boldsymbol{s};\pi_{\boldsymbol{\theta}},\mathcal{P}\right] \\
&= \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[r(\boldsymbol{s}_0^\tau,a_0^\tau,\boldsymbol{s}_1^\tau)+\gamma\sum_{k=0}^{T(\tau)-2}\gamma^k r(\boldsymbol{s}_{1+k}^\tau,a_{1+k}^\tau,\boldsymbol{s}_{1+k+1}^\tau)\,\middle|\,\boldsymbol{s}_0^\tau=\boldsymbol{s};\pi_{\boldsymbol{\theta}},\mathcal{P}\right] \\
&= \mathbb{E}_{a_0\sim\pi_{\boldsymbol{\theta}}(\cdot\,|\,\boldsymbol{s}_0)}\mathbb{E}_{\boldsymbol{s}_1\sim\mathcal{P}(\cdot\,|\,\boldsymbol{s}_0,a_0)}\left[r(\boldsymbol{s}_0,a_0,\boldsymbol{s}_1)+\gamma V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_1)\,\middle|\,\boldsymbol{s}_0=\boldsymbol{s}\right] \\
&\approx r(\boldsymbol{s},a,\boldsymbol{s}')+\gamma\hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}') \quad \text{(directly sample once in current trajectory)} \\
&\approx r(\boldsymbol{s},a,\boldsymbol{s}')+\gamma\hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}';\boldsymbol{\phi}) \quad \text{(directly use previous fitted value function)}
\end{aligned}
$$

then we get training dataset with TD target:

$$
\mathcal{D}=\left\{\boldsymbol{s},\underbrace{r(\boldsymbol{s},a,\boldsymbol{s}')+\gamma\hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}';\boldsymbol{\phi})}_{\text{TD target } y}\right\}_{\boldsymbol{s}\in\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu),\,\boldsymbol{s}'\sim\mathcal{P}(\cdot\,|\,\boldsymbol{s},a)}
$$

- Fit value function by regression task on mean squared error loss:

$$
\boldsymbol{\phi}=\arg\min_{\boldsymbol{\phi}} l(\boldsymbol{\phi})=\frac{1}{|\mathcal{D}|}\sum_{(\boldsymbol{s},y)\in\mathcal{D}}\|\hat{V}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s};\boldsymbol{\phi})-y\|^2
$$

25.1.4 Reference

- Lil'Log: Policy Gradient Algorithms
- Part 3: Intro to Policy Optimization - Spinningup.OpenAI
- Paper: High-Dimensional Continuous Control Using Generalized Advantage Estimation[**?**]
- CS231n-Lecture 14: Reinforcement Learning
- Deep Reinforcement Learning in TensorFlow

25.2 Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes

25.2.1 Introduction

本节选自论文Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes[?]

25.2.2 Reference

- Slides in berkeley: Optimality and Approximation with Policy Gradient Methods in Markov Decision Processes

25.3 Trust Region Policy Optimization (TRPO)

25.3.1 Introduction

本节选自Trust Region Policy Optimization[?]，以及John Schulman的博士论文Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs[?]

Preliminaries Consider an infinite-horizon discounted Markov decision process (MDP) defined by the tuple: $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mu)$, where:

- $\mathcal{S}$: A finite set of states.
- $\mathcal{A}$: A finite set of actions.
- $\mathcal{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{P}$, transition probability distribution.
- r: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, immediate reward function.
- $\gamma \in (0, 1)$: discount factor.
- μ: $\mathcal{S} \to \mathbb{P}$: distribution of the initial state $\boldsymbol{s}_0$.

Let π denote a stochastic policy:

$$\pi : \mathcal{S} \to \mathbb{P}^{|\mathcal{A}|}$$

and let $\eta(\pi)$ denote the expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, \mathcal{P}, \mu \right]$$

where:

- $\boldsymbol{s}_0 \sim \mu(\cdot)$
- $a_t \sim \pi(\cdot\,|\boldsymbol{s}_t; \boldsymbol{\theta}),\ \forall t \in \mathbb{N}_0$
- $\boldsymbol{s}_{t+1}^{\tau} \sim \mathcal{P}(\cdot\,|\boldsymbol{s}_t^{\tau}, a_t^{\tau}),\ \forall t \in \mathbb{N}_0$

Besides, use the following standard definitions of the state-action value function Q^{π}, the state value funcion V^{π} and the state-action advantage function A^{π}:

$$Q^{\pi}(\boldsymbol{s}_t, a_t) = \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| \boldsymbol{s}_t, a_t\,; \pi, \mathcal{P} \right]$$

$$V^{\pi}(\boldsymbol{s}_t) = \mathbb{E}_{a_t, \boldsymbol{s}_{t+1}, \dots} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| \boldsymbol{s}_t\,; \pi, \mathcal{P} \right]$$

$$A^{\pi}(\boldsymbol{s}_t, a_t) = Q^{\pi}(\boldsymbol{s}_t, a_t) - V^{\pi}(\boldsymbol{s}_t)$$

where:

- $a_t \sim \pi(\cdot|\boldsymbol{s}_t),\ \forall t \in \mathbb{N}_0$
- $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot|\boldsymbol{s}_t, a_t),\ \forall t \in \mathbb{N}_0$

25.3.2 Model formulation

Step-1: Rewrite $\eta(\pi)$ as an increment We hope that after each update iteration of the policy π, the expected return $\eta(\pi)$ can be monotonically increased. i.e., when the policy iteration is in the following update process:

$$\pi \to \tilde{\pi}$$

the expected return $\eta(\pi)$ can be monotonically increased:

$$\eta(\tilde{\pi}) \geq \eta(\pi)$$

Slightly rewrite $\eta(\pi)$, it will be written as an incremental as follow:

$$\eta(\tilde{\pi}) = \eta(\pi) + \Delta\eta(\pi, \tilde{\pi})$$

therefore, make a prove that the expected return is monotonically increasing is equivalent to make a proof of increment $\Delta\eta(\pi, \tilde{\pi})$ is nonnegative(positive):

$$\Delta\eta(\pi, \tilde{\pi}) \geq 0$$

Lemma 3. Given two policies π, $\tilde{\pi}$, there exists:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(\boldsymbol{s}_t, a_t) \Big| \tilde{\pi}, \mathcal{P}, \mu\right]$$

This expectation is taken over any trajectories $\tau = (\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, a_1, \boldsymbol{s}_2, \ldots)$, and the notation $\mathbb{E}_{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1 \ldots}[\cdot \,|\tilde{\pi}, \mathcal{P}, \mu]$ indicates that actions are sampled from $\tilde{\pi}$ to generate τ.

Proof. First note that:

$$\begin{aligned}
A^{\pi}(\boldsymbol{s}_t, a_t) &= Q^{\pi}(\boldsymbol{s}_t, a_t) - V^{\pi}(\boldsymbol{s}_t) \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] - V^{\pi}(\boldsymbol{s}_t) \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] - V^{\pi}(\boldsymbol{s}_t) \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[r_t \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] + \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[\gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] - V^{\pi}(\boldsymbol{s}_t) \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[r_t \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] + \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[\gamma(r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + \ldots) \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] - V^{\pi}(\boldsymbol{s}_t) \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[r_t \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] - V^{\pi}(\boldsymbol{s}_t)
\end{aligned}$$

For the first term, because the immediate reward r_t only depends on local path of a trajectory: $(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1})$, which only belongs to the state transition of environment. And also the start point of state transition here is $(\boldsymbol{s}_t, a_t)$, so here we only consider the transition probability $\mathcal{P}$, we can rewrite it as follow:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, \ldots}\left[r_t \Big| \boldsymbol{s}_t, a_t \,; \pi, \mathcal{P}\right] &= \mathbb{E}_{\boldsymbol{s}_{t+1}}\left[r_t \Big| \boldsymbol{s}_t, a_t \,; \mathcal{P}\right] \\
&= \mathbb{E}_{\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot \,|\boldsymbol{s}_t, a_t)}\left[r(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1})\right]
\end{aligned}$$

For the second term, the sum of discounted return $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...$ depends on the subsequence of a trajectory: $(\boldsymbol{s}_{t+1}, a_{t+1}, \boldsymbol{s}_{t+2}, ...)$. But the start point here is $(\boldsymbol{s}_t, a_t)$, so we must convert the start point from $(\boldsymbol{s}_t, a_t)$ to $\boldsymbol{s}_{t+1}$, then represent the sum of discounted return by a V^π function:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{s}_{t+1}, a_{t+1}, ...}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\Big|\boldsymbol{s}_t, a_t\,;\pi,\mathcal{P}\right] &= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[\mathbb{E}_{a_{t+1},\boldsymbol{s}_{t+2},a_{t+2},...}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\Big|\boldsymbol{s}_{t+1}\,;\pi,\mathcal{P}\right]\right] \\ &= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[V^\pi(\boldsymbol{s}_{t+1})\right]\end{aligned}$$

So we can rewrite $A^\pi(\boldsymbol{s}_t, a_t)$ as follow:

$$\begin{aligned}A^\pi(\boldsymbol{s}_t, a_t) &= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t\right] + \gamma\mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}V^\pi(\boldsymbol{s}_{t+1}) - V^\pi(\boldsymbol{s}_t) \\ &= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t\right] + \gamma\mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}V^\pi(\boldsymbol{s}_{t+1}) - \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[V^\pi(\boldsymbol{s}_t)\right] \text{ (expect of constant function)} \\ &= \mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t + \gamma V^\pi(\boldsymbol{s}_{t+1}) - V^\pi(\boldsymbol{s}_t)\right]\end{aligned}$$

So based on the above equation of $A^\pi(\boldsymbol{s}_t, a_t)$, we can rewrite $\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(\boldsymbol{s}_t, a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right]$ as follow:

$$\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(\boldsymbol{s}_t, a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right] = \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t,a_t)}\left[r_t + \gamma V^\pi(\boldsymbol{s}_{t+1}) - V^\pi(\boldsymbol{s}_t)\Big|\tilde{\pi},\mathcal{P},\mu\right]\Big|\tilde{\pi},\mathcal{P},\mu\right]$$

Because $(\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, ...) \sim < \mu, \pi, \mathcal{P} >$ contains $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot\,|\boldsymbol{s}_t, a_t)$, we can rewrite above equation as follow:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(\boldsymbol{s}_t, a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right] &= \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t\left(r_t + \gamma V^\pi(\boldsymbol{s}_{t+1}) - V^\pi(\boldsymbol{s}_t)\right)\Big|\tilde{\pi},\mathcal{P},\mu\right] \\ &= \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\left(\gamma^t r_t + \gamma^{t+1}V^\pi(\boldsymbol{s}_{t+1}) - \gamma^t V^\pi(\boldsymbol{s}_t)\right)\Big|\tilde{\pi},\mathcal{P},\mu\right] \\ &= \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t r_t\Big|\tilde{\pi},\mathcal{P},\mu\right] + \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^{t+1}V^\pi(\boldsymbol{s}_{t+1})\Big|\tilde{\pi},\mathcal{P},\mu\right] \\ &\quad - \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t V^\pi(\boldsymbol{s}_t)\Big|\tilde{\pi},\mathcal{P},\mu\right] \\ &= \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t r_t\Big|\tilde{\pi},\mathcal{P},\mu\right] + \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=1}^{\infty}\gamma^t V^\pi(\boldsymbol{s}_t) - \sum_{t=0}^{\infty}\gamma^t V^\pi(\boldsymbol{s}_t)\Big|\tilde{\pi},\mathcal{P},\mu\right] \\ &= \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t r_t\Big|\tilde{\pi},\mathcal{P},\mu\right] - \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t V^\pi(\boldsymbol{s}_t) - \sum_{t=1}^{\infty}\gamma^t V^\pi(\boldsymbol{s}_t)\Big|\tilde{\pi},\mathcal{P},\mu\right] \\ &= \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t r_t\Big|\tilde{\pi},\mathcal{P},\mu\right] - \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[V^\pi(\boldsymbol{s}_0)\Big|\tilde{\pi},\mathcal{P},\mu\right]\end{aligned}$$

Consider the first term, based on the definition of expected discounted reward, we can rewrite it as follow:

$$\eta(\tilde{\pi}) = \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[\sum_{t=0}^{\infty}\gamma^t r_t\Big|\tilde{\pi},\mathcal{P},\mu\right]$$

Consider the second term, $V^\pi(\boldsymbol{s}_0)$ is based on policy π, so it's irrelevant in terms of policy $\tilde{\pi}$, and $V^\pi(\boldsymbol{s}_0)$ is a constant in terms of $\mathbb{E}_{a_0,\boldsymbol{s}_1,...}\left[\cdot\Big|\tilde{\pi},\mathcal{P},\mu\right]$, according to the definition of state value function, we get:

$$\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,...}\left[V^\pi(\boldsymbol{s}_0)\Big|\tilde{\pi},\mathcal{P},\mu\right] = \mathbb{E}_{\boldsymbol{s}_0\sim\mu(\cdot)}\left[\mathbb{E}_{a_0,\boldsymbol{s}_1,a_2,...}\left[V^\pi(\boldsymbol{s}_0)\Big|\tilde{\pi},\mathcal{P},\mu\right]\Big|\mu\right]$$

$$
\begin{aligned}
&= \mathbb{E}_{\boldsymbol{s}_0\sim\mu(\cdot)}\left[V^{\pi}(\boldsymbol{s}_0)\Big|\,\mu\right] \quad (V^{\pi}(\boldsymbol{s}_0) \text{ is a constant in terms of } \mathbb{E}_{a_0,\boldsymbol{s}_1,\dots}\left[\cdot\,\Big|\tilde{\pi},\mathcal{P},\mu\right])\\
&= \mathbb{E}_{\boldsymbol{s}_0\sim\mu(\cdot)}\left[\mathbb{E}_{a_0,\boldsymbol{s}_1,a_2,\dots}\left[\sum_{t=0}^{\infty}\gamma^t r_t\Big|\boldsymbol{s}_0\,;\pi,\mathcal{P}\right]\Big|\pi,\mathcal{P},\mu\right]\\
&= \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\dots}\left[\sum_{t=0}^{\infty}\gamma^t r_t\Big|\pi,\mathcal{P},\mu\right]\\
&= \eta(\pi)
\end{aligned}
$$

so far we have made a proof of the second term of $\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\dots}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right]$, so we have made a proof of:

$$
\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\dots}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right] = \eta(\tilde{\pi}) - \eta(\pi)
$$

Step-2: Rewrite $\eta(\pi)$ by writing $\boldsymbol{s}$ and a explicitly In the above step, we successfully write $\eta(\pi)$ as incremental format. That is, during the policy iteration, we only need to consider whether the increment $\Delta\eta(\pi,\tilde{\pi})$ is positive or not, and the policy update is for positive increment, otherwise the policy will not be updated.

But the expression of above incremental (i.e., $\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\dots}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right]$) does not give too much information explicitly, so consider to express the state $\boldsymbol{s}$ and the action a explicitly, we rewrite the above incremental as follow:

$$
\begin{aligned}
&\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\dots}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right]\\
=&\mathbb{E}_{\boldsymbol{s}_0\sim\mu(\cdot)}\left[\mathbb{E}_{a_0,\boldsymbol{s}_1,a_1,\dots}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]\Big|\mu\right]\\
=&\mathbb{E}_{\boldsymbol{s}_0\sim\mu(\cdot)}\left[\mathbb{E}_{a_0\sim\tilde{\pi}(\cdot\,|\boldsymbol{s}_0)}\left[\mathbb{E}_{\boldsymbol{s}_1,a_1,\boldsymbol{s}_2,\dots}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]\Big|\tilde{\pi}\right]\Big|\mu\right]\\
=&\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]\\
=&\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)\mathbb{E}_{\boldsymbol{s}_1,a_1,\boldsymbol{s}_2,\dots}\left[A^{\pi}(\boldsymbol{s}_0,a_0)+\sum_{t=1}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]\\
=&\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)\mathbb{E}_{\boldsymbol{s}_1,a_1,\boldsymbol{s}_2,\dots}\left[A^{\pi}(\boldsymbol{s}_0,a_0)\Big|\tilde{\pi},\mathcal{P}\right]+\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)\mathbb{E}_{\boldsymbol{s}_1,a_1,\boldsymbol{s}_2,\dots}\left[\sum_{t=1}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]\\
=&\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)A^{\pi}(\boldsymbol{s}_0,a_0)+\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)\mathbb{E}_{\boldsymbol{s}_1,a_1,\boldsymbol{s}_2,\dots}\left[\sum_{t=1}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]
\end{aligned}
$$

Similarly, we get:

$$
\begin{aligned}
&\mathbb{E}_{\boldsymbol{s}_1,a_1,\boldsymbol{s}_2,\dots}\left[\sum_{t=1}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]\\
=&\sum_{\boldsymbol{s}_1\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_1|\boldsymbol{s}_0,a_0)\sum_{a_1\in\mathcal{A}}\tilde{\pi}(a_1|\boldsymbol{s}_1)\gamma A^{\pi}(\boldsymbol{s}_1,a_1)+\sum_{\boldsymbol{s}_1\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_1|\boldsymbol{s}_0,a_0)\sum_{a_1\in\mathcal{A}}\tilde{\pi}(a_1|\boldsymbol{s}_1)\mathbb{E}_{\boldsymbol{s}_2,a_2,\boldsymbol{s}_3,\dots}\left[\sum_{t=2}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P}\right]
\end{aligned}
$$

Namely, when $t \geq 1$, we achieve an iterative formula as follow:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{s}_t,a_t,\boldsymbol{s}_t,\ldots}\left[\sum_{k=t}^{\infty}\gamma^k A^{\pi}(\boldsymbol{s}_k,a_k)\Big|\tilde{\pi},\mathcal{P}\right] \\
&=\sum_{\boldsymbol{s}_t\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_t|\boldsymbol{s}_{T(t-1)},a_{t-1})\sum_{a_t\in\mathcal{A}}\tilde{\pi}(a_t|\boldsymbol{s}_t)\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t) \\
&+\sum_{\boldsymbol{s}_t\in\mathcal{S}}\mathcal{P}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1},a_{t-1})\sum_{a_t\in\mathcal{A}}\tilde{\pi}(a_t|\boldsymbol{s}_t)\mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\ldots}\left[\sum_{k=t+1}^{\infty}\gamma^k A^{\pi}(\boldsymbol{s}_k,a_k)\Big|\tilde{\pi},\mathcal{P}\right]
\end{aligned}$$

and when $t=0$, we achieve:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\ldots}\left[\sum_{k=0}^{\infty}\gamma^k A^{\pi}(\boldsymbol{s}_k,a_k)\Big|\tilde{\pi},\mathcal{P},\mu\right] \\
&=\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)\gamma A^{\pi}(\boldsymbol{s}_0,a_0)+\sum_{\boldsymbol{s}_0\in\mathcal{S}}\mu(\boldsymbol{s}_0)\sum_{a_0\in\mathcal{A}}\tilde{\pi}(a_0|\boldsymbol{s}_0)\mathbb{E}_{\boldsymbol{s}_1,a_1,\boldsymbol{s}_2,\ldots}\left[\sum_{k=1}^{\infty}\gamma^k A^{\pi}(\boldsymbol{s}_k,a_k)\Big|\tilde{\pi},\mathcal{P}\right]
\end{aligned}$$

When $t\to\infty$, $\gamma^t\to 0$ so we can get:

$$\mathbb{E}_{\boldsymbol{s}_{t+1},a_{t+1},\ldots}\left[\sum_{k=t+1}^{\infty}\gamma^k A^{\pi}(\boldsymbol{s}_k,a_k)\Big|\tilde{\pi},\mathcal{P}\right]\longrightarrow 0$$

So let expand the iteration formula, we can get:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\ldots}\left[\sum_{k=0}^{\infty}\gamma^k A^{\pi}(\boldsymbol{s}_k,a_k)\Big|\tilde{\pi},\mathcal{P},\mu\right]&=\sum_{t=0}^{\infty}\left(\sum_{\boldsymbol{s}\in\mathcal{S}}P(\boldsymbol{s}_t=\boldsymbol{s}\,;\tilde{\pi},\mathcal{P},\mu)\sum_{a\in\mathcal{A}}\tilde{\pi}(a|\boldsymbol{s})\gamma^t A^{\pi}(\boldsymbol{s},a)\right) \\
&=\sum_{t=0}^{\infty}\sum_{\boldsymbol{s}\in\mathcal{S}}\gamma^t P(\boldsymbol{s}_t=\boldsymbol{s}\,;\tilde{\pi},\mathcal{P},\mu)\sum_{a\in\mathcal{A}}\tilde{\pi}(a|\boldsymbol{s})A^{\pi}(\boldsymbol{s},a)
\end{aligned}$$

Let $\rho(\cdot\,;\pi,\mathcal{P},\mu)$ be the (unnormalized) discounted visitation probabilities:

$$\rho(\boldsymbol{s}\,;\pi,\mathcal{P},\mu)=P(\boldsymbol{s}_0=\boldsymbol{s}\,;\pi,\mu)+\gamma P(\boldsymbol{s}_1=\boldsymbol{s}\,;\pi,\mathcal{P})+\gamma^2 P(\boldsymbol{s}_2=\boldsymbol{s}\,;\pi,\mathcal{P})+\ldots=\sum_{t=0}^{\infty}\gamma^t P(\boldsymbol{s}_t=\boldsymbol{s}\,;\pi,\mathcal{P},\mu)$$

So we can push above formulation as follow:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\ldots}\left[\sum_{k=0}^{\infty}\gamma^k A^{\pi}(\boldsymbol{s}_k,a_k)\Big|\tilde{\pi},\mathcal{P},\mu\right]&=\sum_{t=0}^{\infty}\sum_{\boldsymbol{s}\in\mathcal{S}}\gamma^t P(\boldsymbol{s}_t=\boldsymbol{s}\,;\tilde{\pi},\mathcal{P},\mu)\sum_{a\in\mathcal{A}}\tilde{\pi}(a|\boldsymbol{s})A^{\pi}(\boldsymbol{s},a) \\
&=\sum_{\boldsymbol{s}\in\mathcal{S}}\left(\sum_{t=0}^{\infty}\gamma^t P(\boldsymbol{s}_t=\boldsymbol{s}\,;\tilde{\pi},\mathcal{P},\mu)\right)\sum_{a\in\mathcal{A}}\tilde{\pi}(a|\boldsymbol{s})A^{\pi}(\boldsymbol{s},a) \\
&=\sum_{\boldsymbol{s}\in\mathcal{S}}\rho(\boldsymbol{s}\,;\tilde{\pi},\mathcal{P},\mu)\sum_{a\in\mathcal{A}}\tilde{\pi}(a|\boldsymbol{s})A^{\pi}(\boldsymbol{s},a)
\end{aligned}$$

So we get:

$$\begin{aligned}
\eta(\tilde{\pi})-\eta(\pi)&=\mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(\boldsymbol{s}_t,a_t)\Big|\tilde{\pi},\mathcal{P},\mu\right] \\
&=\sum_{\boldsymbol{s}\in\mathcal{S}}\rho(\boldsymbol{s}\,;\tilde{\pi},\mathcal{P},\mu)\sum_{a\in\mathcal{A}}\tilde{\pi}(a|\boldsymbol{s})A^{\pi}(\boldsymbol{s},a)
\end{aligned}$$

$$= \sum_{\boldsymbol{s} \in \mathcal{S}} \rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu) \mathbb{E}_{a \sim \tilde{\pi}(\cdot\,|\boldsymbol{s})} \left[A^{\pi}(\boldsymbol{s}, a) \right]$$

This equation implies that for any policy update: $\pi \rightarrow \tilde{\pi}$, a nonnegative expected advantage at every state $\boldsymbol{s}$ (i.e., $\sum_{a \in \mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s}, a) \geq 0$) is guaranteed to increase the policy η performance.

This implies the classic result that the update performed by exact policy iteration, which uses the deterministic policy: $\tilde{\pi}(\boldsymbol{s}) = \arg\max_{a} A^{\pi}(\boldsymbol{s}, a)$, improves the policy if there is at least one state-action pair $(\boldsymbol{s}, a)$ with a positive advantage value (i.e., $A^{\pi}(\boldsymbol{s}, a) > 0$), and nonzero state visitation probability (i.e., $\rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu) \neq 0$), otherwise the algorithm has converged to the optimal policy.

However, in the approximate setting, it will typically be unavoidable that there will be some state $\boldsymbol{s}$ for which the expected advantage is negative (i.e., $\sum_{a \in \mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s}, a) < 0$) due to estimation and approximation error. Therefore, based on the sampling states $\boldsymbol{s} \in \mathcal{D}_{\mathcal{S}}$ under policy $\tilde{\pi}$, we may get the result:

$$\eta(\tilde{\pi}) - \eta(\pi) = \frac{1}{|\mathcal{D}_{\mathcal{S}}|} \sum_{\boldsymbol{s} \in \mathcal{D}_{\mathcal{S}}} \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s}, a) < 0$$

$\mathcal{D}_{\mathcal{S}}$ is a set of sampled states under policy $\tilde{\pi}$, and $|\mathcal{D}_{\mathcal{S}}|$ is the length of the set $\mathcal{D}_{\mathcal{S}}$.

Step-3: Remove the dependency of discounted visitation probabilities under the new policy $\tilde{\pi}$

The expression derived in the previous step:

$$\eta(\tilde{\pi}) - \eta(\pi) = \sum_{\boldsymbol{s} \in \mathcal{S}} \rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s}, a)$$

therefore during a policy iteration procedure from π to $\tilde{\pi}$, what we need to do is, **under the new policy $\tilde{\pi}$**:

- for all possible states $\boldsymbol{s}$;
- for all possible actions a that could be taken under the state $\boldsymbol{s}$.

compute the value of expected advantage $A^{\pi}(\boldsymbol{s}, a)$ **under the old policy π**. If we operate in this way, we should consider the time efficiency of sampling, which contains :

- For $\sum_{a \in \mathcal{A}} \tilde{\pi}(a\,|\boldsymbol{s})$: given the state $\boldsymbol{s}$, it's time-saving to sample all actions $\boldsymbol{a} \in \mathcal{A}$ under stochastic policy $\tilde{\pi}$ because of the relatively small size of the finite set of actions (i.e., $|\mathcal{A}|$)
- For $\sum_{\boldsymbol{s} \in \mathcal{S}} \rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu)$: it's rather a time-consuming process not only to sample all $\boldsymbol{s} \in \mathcal{S}$ because of the huge size of finite state space $|\mathcal{S}|$, but also to sample a specific state $\boldsymbol{s}$ in different time-step t, in order to estimate the discounted visitation probability: $\rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu) = \sum_{t=0}^{\infty} \gamma^t P(\boldsymbol{s}_t = \boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu)$.

that is, the complex dependency of the the visitation probability $\rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu)$ under policy $\tilde{\pi}$ makes $\eta(\tilde{\pi}) - \eta(\pi) = \sum_{\boldsymbol{s} \in \mathcal{S}} \rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s}, a)$ difficult to optimize directly. Instead, consider ignoring changes in discount visitation probability of state due to policy updates $\pi \Rightarrow \tilde{\pi}$, introduce a local approximation to η as follow:

$$L^{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_{\boldsymbol{s} \in \mathcal{S}} \rho(\boldsymbol{s}\,; \pi, \mathcal{P}, \mu) \sum_{a \in \mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s}, a)$$

namely, $L^{\pi}(\tilde{\pi})$ **uses the visitation probability $\rho(\boldsymbol{s}\,; \pi, \mathcal{P}, \mu)$ under old policy π** in incremental, rather than use the visitation probability $\rho(\boldsymbol{s}\,; \tilde{\pi}, \mathcal{P}, \mu)$ under new policy $\tilde{\pi}$, just like $\eta(\tilde{\pi})$.

So according to whether the states are sampled under the old policy π or the new policy $\tilde{\pi}$, here we have two versions of incremental in policy iteration from old policy π to new policy $\tilde{\pi}$, i.e., $\Delta(\pi, \tilde{\pi})$:

- the states are sampled under the old policy π:

$$L^{\pi}(\tilde{\pi}) = \eta(\pi) + \sum_{\boldsymbol{s}\in\mathcal{S}} \rho(\boldsymbol{s}\,;\pi,\mathcal{P},\mu) \sum_{a\in\mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s},a)$$

- the states are sampled under the new policy $\tilde{\pi}$:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{\boldsymbol{s}\in\mathcal{S}} \rho(\boldsymbol{s}\,;\tilde{\pi},\mathcal{P},\mu) \sum_{a\in\mathcal{A}} \tilde{\pi}(a|\boldsymbol{s}) A^{\pi}(\boldsymbol{s},a)$$

But **what is the the difference between $L^{\pi}(\tilde{\pi})$ and $\eta(\tilde{\pi})$, i.e., $|\eta(\tilde{\pi}) - L^{\pi}(\tilde{\pi})|$?**

Step-4: Bound the difference between $L^{\pi}(\tilde{\pi})$ and $\eta(\tilde{\pi})$（未完成） To bound the difference between $L^{\pi}(\tilde{\pi})$ and $\eta(\tilde{\pi})$, we will bound the difference arising from each timestep t. To do this, we first need to introduce a measure of how much π and $\tilde{\pi}$ agree. Specifically, we ' ll couple the policies, so that they define a joint distribution over pairs of actions.

Definition 25.1. $(\pi,\tilde{\pi})$ is an α-coupled policy pair if it defines a joint distribution $(a,\tilde{a})|\boldsymbol{s}$, such that $P(a\neq\tilde{a}|\boldsymbol{s})\leq\alpha$ for all $\boldsymbol{s}$. π and $\tilde{\pi}$ will denote the marginal distributions of a and $\tilde{a}$, respectively.

In words, this means that at each state $\boldsymbol{s}$, $(\pi,\tilde{\pi})$ gives us a pair of actions $(a,\tilde{a})$, and these actions differ with probability $\mathbb{P}\leq\alpha$.

Definition 25.2.

25.3.3 Reference

- Jonathan Hui: RL—Trust Region Policy Optimization (TRPO) Explained

25.4 Proximal Policy Optimization (PPO)

25.4.1 Introduction

本节选自Proximal Policy Optimization Algorithms[?] 以及Emergence of Locomotion Behaviours in Rich Environments[?]

Importance Sampling Importance sampling (IS) refers to a collection of Monte Carlo methods where a mathematical expectation of function $f(\boldsymbol{x})$ with respect to a target distribution $p(\boldsymbol{x})$, is approximated by a weighted average of random draws from another distribution $q(\boldsymbol{x})$.

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right] &= \int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \\ &= \int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x} \ (\forall \boldsymbol{x}\in\mathcal{X}, q(\boldsymbol{x})\neq 0) \\ &= \mathbb{E}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]\end{aligned}$$

Analysis Importance Sampling by mathematical expectation and mathematical variance:

- expectation:
  - theoretical expectation: same

$$\mathbb{E}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right] = \mathbb{E}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]$$

  - practical expectation: **different**

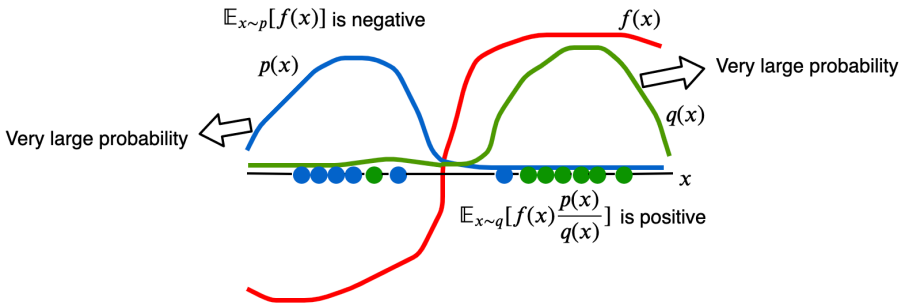


图 127: Importance sampling for a simple practical case, where $f(\boldsymbol{x})$ is a cost function, both $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ are two different probability density functions in terms of variable $\boldsymbol{x}$.

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right] &= \int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \\ &\approx \frac{1}{N}\sum_{i=0}^{N-1} f(\boldsymbol{x}^{(i)}) \quad \text{(sampling } N \text{ samples from } \boldsymbol{x}\sim p\text{)} \\ \mathbb{E}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] &= \int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x}\end{aligned}$$

$$\approx \frac{1}{N}\sum_{i=0}^{N-1}\left(f(\boldsymbol{x}^{(i)})\frac{p(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}\right) \quad \text{(sampling } N \text{ samples from } \boldsymbol{x}\sim q\text{)}$$

When sampling from the p-probability distribution, the samples are likely to come from the left part of the graph. So it's very likely to happen:

$$\mathbb{E}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{i=0}^{N-1} f(\boldsymbol{x}^{(i)}) < 0$$

and when sampling from the q-probability distribution, the samples are likely to come from the left part of the graph. So it's very likely to happen:

$$\mathbb{E}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] \approx \frac{1}{N}\sum_{i=0}^{N-1}\left(f(\boldsymbol{x}^{(i)})\frac{p(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}\right) > 0$$

even though $\frac{p(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}$ is a very small positive number. Therefore it's very likely to happen in practical that:

$$\mathbb{E}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right] \neq \mathbb{E}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]$$

when the two probability distributions p and q are comparatively different.

- variance:
  - theoretical variance: **different**

$$\begin{aligned}
\mathrm{Var}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right] &= \mathbb{E}_{\boldsymbol{x}\sim p}\left[\left(f(\boldsymbol{x})\right)^2\right] - \left(\mathbb{E}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right]\right)^2 \quad \text{(based on the definition of variance)} \\
&= \int_{\boldsymbol{x}\in\mathcal{X}} \left(f(\boldsymbol{x})\right)^2 p(\boldsymbol{x})d\boldsymbol{x} - \left(\int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}\right)^2 \\
\mathrm{Var}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] &= \mathbb{E}_{\boldsymbol{x}\sim q}\left[\left(f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)^2\right] - \left(\mathbb{E}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]\right)^2 \\
&= \int_{\boldsymbol{x}\in\mathcal{X}} \left(f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)^2 q(\boldsymbol{x})d\boldsymbol{x} - \left(\int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x}\right)^2 \\
&= \int_{\boldsymbol{x}\in\mathcal{X}} \left(f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)^2 q(\boldsymbol{x})d\boldsymbol{x} - \left(\int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}\right)^2 \\
&= \int_{\boldsymbol{x}\in\mathcal{X}} \left(f(\boldsymbol{x})\right)^2 \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}p(\boldsymbol{x})d\boldsymbol{x} - \left(\int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}\right)^2
\end{aligned}$$

if p and q follow any two different distributions, for any $\boldsymbol{x}\in\mathcal{X}$, there always exist:

$$\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \neq 1$$

therefore:

$$\mathrm{Var}_{\boldsymbol{x}\sim p}\left[f(\boldsymbol{x})\right] \neq \mathrm{Var}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]$$

  - practical variance: **different**

Summary: **When using the importance sampling for sampling, if the two probability distributions p and q are very different, the practical effect will be poor. Therefore, in order to ensure the relatively practical good results when using Importance Sampling, it is necessary to limit the two probability distributions p and q will not be much different.**

Trust Region Methods Policy gradient estimates can have high variance and algorithms can be sensitive to the settings of their hyperparameters. Several approaches have been proposed to make policy gradient algorithms more robust. One effective measure is to employ a trust region constraint that restricts the amount by which any update is allowed to change the policy. A popular algorithm that makes use of this idea is trust region policy optimization(TRPO)[**?**]. In every iteration given current parameters $\boldsymbol{\theta}_{\text{old}}$, TRPO collects a (relatively large) batch of data and optimizes the surrogate objective:

$$J_{\text{TRPO}}(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\rho(\tau;\mu,\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P})}\left[\sum_{t=0}^{T(\tau)-1}\gamma^t\frac{\pi_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|\boldsymbol{s}_t)}A^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\boldsymbol{s}_t,a_t)\right]$$

subject to a constraint on how much the policy is allowed to change, expressed in terms of the Kullback-Leibler divergence(KL-divergence):

$$\mathbb{E}_{\boldsymbol{s}\sim\pi_{\boldsymbol{\theta}_{\text{old}}}}\left[D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\boldsymbol{s})\,\middle\|\,\pi_{\boldsymbol{\theta}}(\cdot\,|\boldsymbol{s})\right)\right]\leq\delta$$

where A^{π} is the advantage function given as:

$$A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t,a_t) = Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t,a_t) - V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t)$$

25.4.2 Model formulation

Objective The goal of the reinforcement learning is to choose a policy π with parameters $\boldsymbol{\theta}$ to maximize the expected return, and here we transform the on-policy policy gradient optimization into off-policy by importance sampling, the derivation process of objective function is as follow:

$$\max_{\pi} \mathbb{E}_{\boldsymbol{s}_0,a_0,\boldsymbol{s}_1,\ldots}\left[R_0=\sum_{t=0}^{\infty}\gamma^t r_t\Big|\pi,\mathcal{P},\mu\right]$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}\left[R(\tau)=\sum_{t=0}^{T(\tau)-1}\gamma^t r_t^{\tau}\right] \quad (T(\tau)\text{ is the length of trajectory }\tau)$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \int_{\tau\in\mathcal{T}} R(\tau)\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)d\tau \quad \text{(definition of mathematical expectation)}$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \int_{\tau\in\mathcal{T}} R(\tau)\frac{\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}{\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)d\tau \quad \text{(definition of importance sampling, } \forall\tau\in\mathcal{T},\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)\neq 0)$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\frac{\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}{\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\right] \quad \text{(definition of mathematical expectation)}$$

Here $\mathcal{T}$ is the trajectory space, for each trajectory τ, the probability density function of trajectory $\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)$ depends on there parameter components:

- $\boldsymbol{s}_0^{\tau}\sim\mu(\cdot),\ \forall\tau\in\mathcal{T}$
- $a_t^{\tau}\sim\pi(\cdot\,|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta}_{\text{old}}),\ \forall\tau\in\mathcal{T},\forall\, t\in\mathbb{N}_0$
- $\boldsymbol{s}_{t+1}^{\tau}\sim\mathcal{P}(\cdot\,|\boldsymbol{s}_t^{\tau},a_t^{\tau}),\ \forall\tau\in\mathcal{T},\forall\, t\in\mathbb{N}_0$

Considering the Markov property on state-action path $(\boldsymbol{s}_0^{\tau},a_0^{\tau},\boldsymbol{s}_1^{\tau},a_1^{\tau},\boldsymbol{s}_2^{\tau},...,\boldsymbol{s}_{T(\tau)}^{\tau})$ with length $T(\tau)+1$, which sampled from $\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)$, we can rewrite the probability density function of trajectory $\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)$ as follow:

$$\begin{aligned}\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)&=\mu(\boldsymbol{s}_0^{\tau})\cdot\pi(a_0^{\tau}\,|\boldsymbol{s}_0^{\tau};\boldsymbol{\theta}_{\text{old}})\cdot\mathcal{P}(\boldsymbol{s}_1^{\tau}\,|\boldsymbol{s}_0^{\tau},a_0^{\tau})\cdot\pi(a_1^{\tau}\,|\boldsymbol{s}_1^{\tau};\boldsymbol{\theta}_{\text{old}})\cdot\mathcal{P}(\boldsymbol{s}_2^{\tau}\,|\boldsymbol{s}_1^{\tau},a_1^{\tau})\cdot\ldots\cdot\mathcal{P}(\boldsymbol{s}_{T(\tau)}^{\tau}\,|\boldsymbol{s}_{T(\tau)-1}^{\tau},a_{T(\tau)-1}^{\tau})\\&=\mu(\boldsymbol{s}_0^{\tau})\cdot\prod_{t=0}^{T(\tau)-1}\pi(a_t^{\tau}|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta}_{\text{old}})\cdot\prod_{t=0}^{T(\tau)-1}\mathcal{P}(\boldsymbol{s}_{t+1}^{\tau}\,|\boldsymbol{s}_t^{\tau},a_t^{\tau})\end{aligned}$$

we can rewrite the objective as follow:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\frac{\rho(\tau;\pi_{\boldsymbol{\theta}},\mathcal{P},\mu)}{\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\right]$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\frac{\cancel{\mu(\boldsymbol{s}_0^{\tau})}\cdot\prod_{t=0}^{T(\tau)-1}\pi(a_t^{\tau}|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta})\cdot\cancel{\prod_{t=0}^{T(\tau)-1}\mathcal{P}(\boldsymbol{s}_{t+1}^{\tau}|\boldsymbol{s}_t^{\tau},a_t^{\tau})}}{\cancel{\mu(\boldsymbol{s}_0^{\tau})}\cdot\prod_{t=0}^{T(\tau)-1}\pi(a_t^{\tau}|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta}_{\text{old}})\cdot\cancel{\prod_{t=0}^{T(\tau)-1}\mathcal{P}(\boldsymbol{s}_{t+1}^{\tau}|\boldsymbol{s}_t^{\tau},a_t^{\tau})}}\right]$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\frac{\prod_{t=0}^{T(\tau)-1}\pi(a_t^{\tau}|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta})}{\prod_{t=0}^{T(\tau)-1}\pi(a_t^{\tau}|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta}_{\text{old}})}\right]$$

$$\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\log\left(\frac{\prod_{t=0}^{T(\tau)-1}\pi(a_t^{\tau}|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta})}{\prod_{t=0}^{T(\tau)-1}\pi(a_t^{\tau}|\boldsymbol{s}_t^{\tau};\boldsymbol{\theta}_{\text{old}})}\right)\right]$$

$$
\begin{aligned}
&\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\left(\log\left(\prod_{t=0}^{T(\tau)-1}\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})\right)-\log\left(\prod_{t=0}^{T(\tau)-1}\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})\right)\right)\right]\\
&\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\left(\sum_{t=0}^{T(\tau)-1}\log\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})-\sum_{t=0}^{T(\tau)-1}\log\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})\right)\right]\\
&\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\log\Big(\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})-\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})\Big)\right]\\
&\Rightarrow \max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\log\frac{\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})}{\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})}\right]
\end{aligned}
$$

Besides, consider the constraint that:

$$
\mathbb{E}_{\boldsymbol{s}\sim\pi_{\boldsymbol{\theta}_{\text{old}}}}\left[D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\boldsymbol{s})\,\middle\|\,\pi_{\boldsymbol{\theta}}(\cdot\,|\boldsymbol{s})\right)\right]\leq\delta
$$

using a penalty instead of a constraint, we can convert the constrainted problem(TRPO):

$$
\begin{aligned}
\max_{\boldsymbol{\theta}}\quad & \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\log\frac{\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})}{\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})}\right]\\
\text{s.t.}\quad & \mathbb{E}_{\boldsymbol{s}\sim\rho(\boldsymbol{s};\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\boldsymbol{s})\,\middle\|\,\pi_{\boldsymbol{\theta}}(\cdot\,|\boldsymbol{s})\right)\right]\leq\delta
\end{aligned}
$$

into an unconstrainted problem(PPO):

$$
\max_{\boldsymbol{\theta}}\ J_{\text{PPO}}(\boldsymbol{\theta})=\mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\log\frac{\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})}{\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})}\right]-\beta\mathbb{E}_{\boldsymbol{s}\sim\rho(\boldsymbol{s};\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\boldsymbol{s})\,\middle\|\,\pi_{\boldsymbol{\theta}}(\cdot\,|\boldsymbol{s})\right)\right]
$$

where the hyperparameter $\beta>0$ is a penalty coefficient.

Convert maximize log-likelihood to minimize kl-divergence for stochastic policy Here we consider the stochastic policy case, each action a is sampled from its corresponding known action probability distribution $\boldsymbol{p}$ (e.g. $a=2$ is sampled from its corresponding action probability distribution $\boldsymbol{p}=(0.02,0.01,0.92,0.05)$):

$$
a\in\mathcal{A}\sim\boldsymbol{p}\in\mathbb{P}^{|\mathcal{A}|}
$$

therefore we can convert the **first term** of above objective with maximizing log-likelihood to minimizing kl-divergence:

$$
\begin{aligned}
&\max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\Big(\log\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})-\log\pi(a_t^\tau|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})\Big)\right]\\
&\Rightarrow \min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\Big(D_{\text{KL}}\big(\boldsymbol{p}_t^\tau\|\pi(\cdot\,|\boldsymbol{s}_t^\tau;\boldsymbol{\theta})\big)-D_{\text{KL}}\big(\boldsymbol{p}_t^\tau\|\pi(\cdot\,|\boldsymbol{s}_t^\tau;\boldsymbol{\theta}_{\text{old}})\big)\Big)\right]\\
&\Rightarrow \min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\left(\sum_{k\in\mathcal{A}}p_{t,k}^\tau\log\frac{p_{t,k}^\tau}{\pi_{t,k}^\tau(\boldsymbol{\theta})}-\sum_{k\in\mathcal{A}}p_{t,k}^\tau\log\frac{p_{t,k}^\tau}{\pi_{t,k}^\tau(\boldsymbol{\theta}_{\text{old}})}\right)\right]
\end{aligned}
$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}} p_{t,k}^{\tau}\log\frac{\frac{\cancel{p_{t,k}^{\tau}}}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}}{\frac{\cancel{p_{t,k}^{\tau}}}{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}}\right]$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}} p_{t,k}^{\tau}\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}\right]$$

also we can convert the second term of objective as follow in order to matching the first term:

$$\begin{aligned}
&\max_{\boldsymbol{\theta}} -\beta\mathbb{E}_{\boldsymbol{s}\sim\rho(\boldsymbol{s};\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\boldsymbol{s})\,\middle\|\,\pi_{\boldsymbol{\theta}}(\cdot\,|\boldsymbol{s})\right)\right]\ (\beta>0)\\
\Rightarrow &\min_{\boldsymbol{\theta}} \beta\mathbb{E}_{\boldsymbol{s}\sim\rho(\boldsymbol{s};\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\boldsymbol{s})\,\middle\|\,\pi_{\boldsymbol{\theta}}(\cdot\,|\boldsymbol{s})\right)\right]\\
\Rightarrow &\min_{\boldsymbol{\theta}} \beta\mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1} D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot\,|\boldsymbol{s}_t^{\tau})\,\middle\|\,\pi_{\boldsymbol{\theta}}(\cdot\,|\boldsymbol{s}_t^{\tau})\right)\right]\\
\Rightarrow &\min_{\boldsymbol{\theta}} \beta\mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}}\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}\right]
\end{aligned}$$

Therefore combine two terms of object, we can rewrite objective as follow:

$$\begin{aligned}
&\min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}} p_{t,k}^{\tau}\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}\right]+\beta\mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}}\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}\right]\\
\Rightarrow &\min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[R(\tau)\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}} p_{t,k}^{\tau}\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}+\beta\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}}\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}\right]\\
\Rightarrow &\min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}}\left(p_{t,k}^{\tau}\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}R(\tau)+\beta\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}\right)\right]\\
\Rightarrow &\min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}\left[\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}}\left(\underbrace{-p_{t,k}^{\tau}\log\pi_{t,k}^{\tau}(\boldsymbol{\theta})}_{\text{cross-entropy}}R(\tau)+\beta\underbrace{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}}_{\text{kl-divergence}}\right)\right]\\
\Rightarrow &\min_{\boldsymbol{\theta}} \int_{\tau\in\mathcal{T}}\left(\sum_{t=0}^{T(\tau)-1}\sum_{k\in\mathcal{A}}\left(-p_{t,k}^{\tau}\log\pi_{t,k}^{\tau}(\boldsymbol{\theta})R(\tau)+\beta\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau}(\boldsymbol{\theta})}\right)\right)\underbrace{\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)}_{\text{constant}}d\tau\\
\Rightarrow &\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1}\sum_{k\in\mathcal{A}}\left(-p_{t,k}^{\tau^{(i)}}\log\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})R(\tau^{(i)})+\beta\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})}\right)
\end{aligned}$$

where N is the number of trajectories sampled under $\rho(\tau;\pi_{\boldsymbol{\theta}_{\text{old}}},\mathcal{P},\mu)$.

Variance reduction by reward-to-go and introducing baseline

$$\begin{aligned}
&\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1}\sum_{k\in\mathcal{A}}\left(-p_{t,k}^{\tau^{(i)}}\log\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})R(\tau^{(i)})+\beta\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})}\right)\\
\Rightarrow &\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T(\tau^{(i)})-1}\sum_{k\in\mathcal{A}}\left(-p_{t,k}^{\tau^{(i)}}\log\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})\left(\sum_{t'=t}^{T(\tau^{(i)})}\gamma^{t'-t}r_{t'}^{\tau^{(i)}}\right)+\beta\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})\log\frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})}\right)
\end{aligned}$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T(\tau^{(i)})-1} \sum_{k \in \mathcal{A}} \left(-p_{t,k}^{\tau^{(i)}} \log \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}) \left(\sum_{t'=t}^{T(\tau^{(i)})} \gamma^{t'-t} r_{t'}^{\tau^{(i)}} - b \right) + \beta \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \right)$$

Tricks

- **Adaptive penalty**: Dynamic adjustment of penalty coefficient, i.e.,:

$$\frac{1}{N \times T} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \left(\sum_{k \in \mathcal{A}} \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \right) > \alpha_{\text{high}} \cdot D_{\text{KL-target}} \quad \Longrightarrow \quad \beta \leftarrow 2 \times \beta$$

$$\frac{1}{N \times T} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \left(\sum_{k \in \mathcal{A}} \pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}}) \log \frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \right) < \alpha_{\text{low}} \cdot D_{\text{KL-target}} \quad \Longrightarrow \quad \beta \leftarrow \frac{\beta}{2}$$

  where $\alpha_{\text{high}} > 1$, $\alpha_{\text{low}} \in (0, 1)$ and $D_{\text{KL-target}}$ are all hyperparameters.

- **Clipping operation**:

$$\frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})} \Rightarrow \text{clip} \left(\frac{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta}_{\text{old}})}{\pi_{t,k}^{\tau^{(i)}}(\boldsymbol{\theta})}; 1-\epsilon, 1+\epsilon \right)$$

Algorithm 36: Proximal Policy Optimization[**?**]

Input: N: the number of trajectories collected in each policy iteration;
M: the number of batch update in each policy iteration;
T: the length of sampled trajectory.

Input: $\boldsymbol{\theta}_0$: initial policy parameters;
$\boldsymbol{\psi}_0$: initial value function parameters.

Output: the final stochastic policy parameters: $\boldsymbol{\theta}$.

for $k = 0, 1, 2, ...$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau\}$ by running policy $\pi_{\boldsymbol{\theta}_k}$ in the environment.

Compute reward-to-go:

$$\hat{R}_t^\tau = \sum_{t'=t}^{T} \gamma^{t'-t} r(\boldsymbol{s}_{t'}^\tau, a_{t'}^\tau, \boldsymbol{s}_{t'+1}^\tau),\ \forall \tau \in \mathcal{D}_k, t \in \{0, ..., T\}$$

Compute advantage estimates, $\hat{A}_t^\tau$ (using any method of advantage estimation) based on the current value function $V^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}; \boldsymbol{\psi}_k)$, for example:

$$\hat{A}^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}_t^\tau, a_t^\tau) = r(\boldsymbol{s}_t^\tau, a_t^\tau, \boldsymbol{s}_{t+1}^\tau) + \gamma V^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}_{t+1}^\tau; \boldsymbol{\psi}_k) - V^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}_t^\tau; \boldsymbol{\psi}_k),\ \ \forall \tau \in \mathcal{D}_k, t \in \{0, ..., T\}$$

Update the policy by maximizing the PPO-Clip objective:

$$\boldsymbol{\theta}_{k+1} = \arg\max_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_{\boldsymbol{\theta}}(a_t^\tau | \boldsymbol{s}_t^\tau)}{\pi_{\boldsymbol{\theta}_k}(a_t^\tau | \boldsymbol{s}_t^\tau)} \hat{A}^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}_t^\tau, a_t^\tau),\ g(\epsilon, \hat{A}^{\pi_{\boldsymbol{\theta}_k}}(\boldsymbol{s}_t^\tau, a_t^\tau)) \right)$$

Fit value function by regression on mean-squared error:

$$\boldsymbol{\psi}_{k+1} = \arg\min_{\boldsymbol{\psi}} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_{\boldsymbol{\psi}}(\boldsymbol{s}_t^\tau) - \hat{R}_t^\tau \right)^2$$

end

- As the iterative step k proceed:
  - the more accurate the estimation of the value function $V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}; \boldsymbol{\psi})$,
  - the more accurate the estimation of the advantage function $\hat{A}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, a) = r + \gamma V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}'; \boldsymbol{\psi}) - V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}; \boldsymbol{\psi})$,
  - the more accurate updates of stochastic policy $\pi_{\boldsymbol{\theta}}$.
- Training supervised label $\hat{R}_t^\tau$ corresponding to $V(\boldsymbol{s}; \boldsymbol{\psi})$ comes from Monte Carlo sampling.

Problem: How we get PPO-Clip objective in above pseudo-code?

25.4.3 Reference

- Jonathan Hui: RL—Importance Sampling
- Proximal Policy Optimization Algorithms - PPO

25.5 Asynchronous Methods for Deep Reinforcement Learning (A3C)

25.5.1 Introduction

本节选自论文Asynchronous Methods for Deep Reinforcement Learning[?]

25.6 Sample Efficient Actor-Critic with Experience Replay (ACER)

25.6.1 Introduction

本节选自论文Sample Efficient Actor-Critic with Experience Replay[?]

25.7 Deterministic Policy Gradient Algorithms (DPG)

25.7.1 Introduction

本节选自Deterministic Policy Gradient Algorithms[?]

25.8 Continuous Control with Deep Reinforcement Learning (DDPG)

25.8.1 Introduction

本节选自论文Continuous control with deep reinforcement learning[?]

Quick Facts

- DDPG is an off-policy algorithm.
- DDPG can only be used for environments with continuous action spaces.
- DDPG can be thought of as being deep Q-learning for continuous action spaces.

25.8.2 Model formulation

The Q-learning side of DDPG

- Bellman equation in theory:

$$Q^{\pi}(\boldsymbol{s}, a) = \mathbb{E}_{\boldsymbol{s}' \sim \mathcal{P}(\cdot\,|\boldsymbol{s},a)}\Big[r(\boldsymbol{s}, a, \boldsymbol{s}') + \gamma \mathbb{E}_{a' \sim \pi(\cdot\,|\boldsymbol{s}')}\big[Q^{\pi}(\boldsymbol{s}', a')\big]\Big]$$

- Bellman equation in practical as target in discrete action space:

$$y = r(\boldsymbol{s}, a, \boldsymbol{s}') + \gamma \max_{a' \in \mathcal{A}} Q(\boldsymbol{s}', a'; \boldsymbol{\theta}')$$

  using max operation instead expectation.

- Bellman equation in practical as target in continuous action space:

$$y = r(\boldsymbol{s}, a, \boldsymbol{s}') + \gamma Q(\boldsymbol{s}', \mu(\boldsymbol{s}'; \boldsymbol{\theta}); \boldsymbol{\phi})$$

  using policy directly.

Note: 因为在 discrete action space 环境下使用 max 操作代替期望，本身就是不准确的，因此在 continuous action space 环境下，直接使用策略也是完全没有问题的，毕竟大家都是不准确的，都认为近似等于期望

Algorithm 37: Deep Deterministic Policy Gradient(DDPG)[?]

Input: θ: initial policy parameters;
ϕ: initial Q-function parameters;
$\mathcal{D}$: empty replay buffer.

Output: final deterministic policy $\mu_\theta : \mathcal{S} \rightarrow \mathcal{A}$

Set target parameters equal to primary parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$

repeat

  Observe state s and select an action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{low}}, a_{\text{high}})$, where $\epsilon \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ for action exploration

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, s', r, d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state

  if *it's time to update* **then**

    for *however many updates* **do**

      Randomly sample a batch of transitions $\mathcal{B} = \{(s, a, s', r, d)\}$ from $\mathcal{D}$

      Compute Q-targets for all transitions in $\mathcal{B}$:

$$y(s', r, d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using:

$$\nabla_\phi \frac{1}{|\mathcal{B}|} \sum_{(s,a,s',r,d) \in \mathcal{B}} (Q_\phi(s, a) - y(s', r, d))^2$$

      Update policy by one step of gradient ascent using:

$$\nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with:

$$\phi_{\text{targ}} \leftarrow \tau\phi + (1 - \tau)\phi_{\text{targ}}$$
$$\theta_{\text{targ}} \leftarrow \tau\theta + (1 - \tau)\theta_{\text{targ}}$$

    end

  end

until *convergence*;

Note:

- 随着学习学习次数越来越多，Q 函数的学习越来越准确，即：作为 critic，其越来越能够准确地对 state-action 对儿进行打分评判
- 在 policy 的学习过程中，我们假定当前 Q_θ 是公正准确的，那么在当前 state s 下，我们要做的是调