MakiFlow Documentation

Release 1.3.0

Igor Kilbas, Gribanov Danil, Mukhin Artem

Mar 16, 2020

Contents

1 License 1

2 Contact 3

3 Model code organizing 5

4 Layers 9

CHAPTER 1

License

CHAPTER 2

Contact

If you have any questions, please contact Igor Kilbas via whitemarsstudios@gmail.com.

CHAPTER 3

Model code organizing

Each model consists of the following components:

- main entity
- training modules

The model package is organized as follows:

- **main_modules**
  - main_module1
  - main_module2
- **training_modules**
  - training_module1
  - training_module2
- compile.py - compiles the main modules with the training ones and provides the full model.

3.1 Main modules

Each model has the main class entity that provides the basic functionality for working with the model. It also share the common components that are using by the training modules.

Interface of the main module class:

```
class ModelBasis(MakiModel):
    def __init__(self, ...)
        # Setting up the variables.
        self._training_vars_are_ready = False
        pass

    def _get_model_info(self):
```

(continues on next page)

(continued from previous page)

```
        # This method is required by the MakiModel interface.
        pass

    # COMMON TRAINING FUNCTIONALITY
    def _prepare_training_vars(self):
        # Setting up the variables, losses, etc
        self._training_vars_are_ready = True
        pass

    def _other_methods_for_the_training_modules(self):
        pass
```

3.2 Training modules

Each training module is responsible for training the model using a certain loss function. Therefore, its name reflects the employed training loss: LossNameTrainingModule.

Interface of the training module class:

```
class LossNameTrainingModule(ModelBasis):
    def _prepare_training_vars(self):
        self._lossname_loss_is_built = False
        super()._prepare_training_vars()

    def _build_lossname_loss(self):
        # Code that builds the scalar tensor of the minimized loss.
        lossname_loss = ...
        # This is the method built into the MakiModel.
        # It is used to include the regularization term into the total loss.
        self._final_lossname_loss = self._build_final_loss(lossname_loss)

    def _setup_lossname_inputs(self):
        # Here the necessary placeholder are set up.
        pass

    # This method signature can include other arguments if needed.
    def _minimize_lossname_loss(self, optimizer, global_step):
        if not self._training_vars_are_ready:
            self._prepare_training_vars()

        if not self._lossname_is_built:
            self._setup_lossname_inputs()
            self._build_lossname_loss()
            self._lossname_optimizer = optimizer
            self._lossname_train_op = optimizer.minimize(
                self._final_lossname_loss, var_list=self._trainable_vars, global_
↪step=global_step
            )
            self._session.run(tf.variables_initializer(optimizer.variables()))
            self._lossname_loss_is_built = True
            # This is a common utility for printing info messages
            loss_is_built()

        if self._lossname_optimizer != optimizer:
```

(continues on next page)

(continued from previous page)

```
            # This is a common utility for printing info messages
            new_optimizer_used()
            self._lossname_optimizer = optimizer
            self._lossname_train_op = optimizer.minimize(
                self._final_lossname_loss, var_list=self._trainable_vars, global_
↪step=global_step
            )
            self._session.run(tf.variables_initializer(optimizer.variables()))

    return self._lossname_train_op

    def fit_lossname(self, ..., optimizer, epochs=1, global_step=None):
        assert (optimizer is not None)
        assert (self._session is not None)

        train_op = self._minimize_abs_loss(optimizer, global_step)
        # Training cycle
```

You can copy this code a modify accordingly.

3.3 compile.py

In this file all the modules are assembled into the final model.

```
from .training_modules import Lossname1TrainingModule, Lossname2TrainingModule


class Model(Lossname1TrainingModule, Lossname2TrainingModule):
    pass
```

This model is then used for the one's purposes.

CHAPTER 4

Layers

Contents

- *Layers*
  - *Convolutional layers*
    * *ConvLayer*
    * *UpConvLayer*
    * *DepthWiseConvLayer*
    * *SeparableConvLayer*
    * *AtrousConvLayer*
  - *Normalization layers*
    * *BatchNormLayer*
    * *GroupNormLayer*
    * *NormalizationLayer*
    * *InstanceNormLayer*
  - *Tensor manipulation layers*
  - *Other layers*
    * *BiasLayer*
    * *DenseLayer*
    * *ScaleLayer*

4.1 Convolutional layers

4.1.1 ConvLayer

Parameters

kw [int] Kernel width.

kh [int] Kernel height.

in_f [int] Number of input feature maps. Treat as color channels if this layer is first one.

out_f [int] Number of output feature maps (number of filters).

stride [int] Defines the stride of the convolution.

padding [str] Padding mode for convolution operation. Options: 'SAME', 'VALID' (case sensitive).

activation [tensorflow function] Activation function. Set None if you don't need activation.

W [numpy array] Filter's weights. This value is used for the filter initialization with pretrained filters.

b [numpy array] Bias' weights. This value is used for the bias initialization with pretrained bias.

use_bias [bool] Add bias to the output tensor.

name [str] Name of this layer.

4.1.2 UpConvLayer

Parameters

kw [int] Kernel width.

kh [int] Kernel height.

in_f [int] Number of input feature maps. Treat as color channels if this layer is first one.

out_f [int] Number of output feature maps (number of filters).

size [tuple] Tuple of two ints - factors of the size of the output feature map. Example: feature map with spatial dimension (n, m) will produce output feature map of size (a*n, b*m) after performing up-convolution with *size* (a, b).

padding [str] Padding mode for convolution operation. Options: 'SAME', 'VALID' (case sensitive).

activation [tensorflow function] Activation function. Set None if you don't need activation.

W [numpy array] Filter's weights. This value is used for the filter initialization with pretrained filters.

b [numpy array] Bias' weights. This value is used for the bias initialization with pretrained bias.

use_bias [bool] Add bias to the output tensor.

Important: Shape is different from normal convolution since it's required by transposed convolution. Output feature maps go before input ones.

4.1.3 DepthWiseConvLayer

Parameters

kw [int] Kernel width.

kh [int] Kernel height.

in_f [int] Number of input feature maps. Treat as color channels if this layer is first one.

multiplier [int] Number of output feature maps equals *in_f‘*‘multiplier*.

stride [int] Defines the stride of the convolution.

padding [str] Padding mode for convolution operation. Options: ‘SAME’, ‘VALID’ (case sensitive).

activation [tensorflow function] Activation function. Set None if you don’t need activation.

W [numpy array] Filter’s weights. This value is used for the filter initialization with pretrained filters.

use_bias [bool] Add bias to the output tensor.

name [str] Name of this layer.

4.1.4 SeparableConvLayer

Parameters

kw [int] Kernel width.

kh [int] Kernel height.

in_f [int] Number of the input feature maps. Treat as color channels if this layer is first one.

out_f [int] Number of the output feature maps after pointwise convolution, i.e. it is depth of the final output tensor.

multiplier [int] Number of output feature maps after depthwise convolution equals *in_f‘*‘multiplier*.

stride [int] Defines the stride of the convolution.

padding [str] Padding mode for convolution operation. Options: ‘SAME’, ‘VALID’ (case sensitive).

activation [tensorflow function] Activation function. Set None if you don’t need activation.

W_dw [numpy array] Filter’s weights. This value is used for the filter initialization.

use_bias [bool] Add bias to the output tensor.

name [str] Name of this layer.

4.1.5 AtrousConvLayer

Parameters

kw [int] Kernel width.

kh [int] Kernel height.

in_f [int] Number of input feature maps. Treat as color channels if this layer is first one.

out_f [int] Number of output feature maps (number of filters).

rate [int] A positive int. The stride with which we sample input values across the height and width dimensions

stride [int] Defines the stride of the convolution.

padding [str] Padding mode for convolution operation. Options: 'SAME', 'VALID' (case sensitive).

activation [tensorflow function] Activation function. Set None if you don't need activation.

W [numpy array] Filter's weights. This value is used for the filter initialization with pretrained filters.

b [numpy array] Bias' weights. This value is used for the bias initialization with pretrained bias.

use_bias [bool] Add bias to the output tensor.

name [str] Name of this layer.

4.2 Normalization layers

4.2.1 BatchNormLayer

Batch Normalization Procedure: X_normed = (X - mean) / variance X_final = X*gamma + beta

gamma and beta are defined by the NN, e.g. they are trainable.

Parameters

D [int] Number of tensors to be normalized.

decay [float] Decay (momentum) for the moving mean and the moving variance.

eps [float] A small float number to avoid dividing by 0.

use_gamma [bool] Use gamma in batchnorm or not.

use_beta [bool] Use beta in batchnorm or not.

name [str] Name of this layer.

mean [float] Batch mean value. Used for initialization mean with pretrained value.

var [float] Batch variance value. Used for initialization variance with pretrained value.

gamma [float] Batchnorm gamma value. Used for initialization gamma with pretrained value.

beta [float] Batchnorm beta value. Used for initialization beta with pretrained value.

4.2.2 GroupNormLayer

GroupNormLayer Procedure: X_normed = (X - mean) / variance X_final = X*gamma + beta

There X (as original) have shape [N, H, W, C], but in this operation it will be [N, H, W, G, C // G]. GroupNormLayer normalized input on N and C // G axis. gamma and beta are learned using gradient descent.

Parameters

D [int] Number of tensors to be normalized.

decay [float] Decay (momentum) for the moving mean and the moving variance.

eps [float] A small float number to avoid dividing by 0.

G [int] The number of groups that normalized. NOTICE! The number D must be divisible by G without remainder

use_gamma [bool] Use gamma in batchnorm or not.

use_beta [bool] Use beta in batchnorm or not.

name [str] Name of this layer.

mean [float] Batch mean value. Used for initialization mean with pretrained value.

var [float] Batch variance value. Used for initialization variance with pretrained value.

gamma [float] Batchnorm gamma value. Used for initialization gamma with pretrained value.

beta : float

4.2.3 NormalizationLayer

NormalizationLayer Procedure: X_normed = (X - mean) / variance X_final = X*gamma + beta

There X have shape [N, H, W, C]. NormalizationLayer normqlized input on N axis gamma and beta are learned using gradient descent.

Parameters

D [int] Number of tensors to be normalized.

decay [float] Decay (momentum) for the moving mean and the moving variance.

eps [float] A small float number to avoid dividing by 0.

use_gamma [bool] Use gamma in batchnorm or not.

use_beta [bool] Use beta in batchnorm or not.

name [str] Name of this layer.

mean [float] Batch mean value. Used for initialization mean with pretrained value.

var [float] Batch variance value. Used for initialization variance with pretrained value.

gamma [float] Batchnorm gamma value. Used for initialization gamma with pretrained value.

beta [float] Batchnorm beta value. Used for initialization beta with pretrained value.

4.2.4 InstanceNormLayer

InstanceNormLayer Procedure: X_normed = (X - mean) / variance X_final = X*gamma + beta

There X have shape [N, H, W, C]. InstanceNormLayer normalized input on N and C axis gamma and beta are learned using gradient descent.

Parameters

D [int] Number of tensors to be normalized.

decay [float] Decay (momentum) for the moving mean and the moving variance.

eps [float] A small float number to avoid dividing by 0.

use_gamma [bool] Use gamma in batchnorm or not.

use_beta [bool] Use beta in batchnorm or not.

name [str] Name of this layer.

mean [float] Batch mean value. Used for initialization mean with pretrained value.

var [float] Batch variance value. Used for initialization variance with pretrained value.

gamma [float] Batchnorm gamma value. Used for initialization gamma with pretrained value.

beta [float] Batchnorm beta value. Used for initialization beta with pretrained value.

4.3 Tensor manipulation layers

ReshapeLayer is used to changes size from some input_shape to new_shape (include batch_size and color dimension).

Parameters

new_shape [list] Shape of output object.

name [str] Name of this layer.

MulByAlphaLayer is used to multiply input MakiTensor by *alpha*.

Parameters

alpha [int] The constant to multiply by.

name [str] Name of this layer.

SumLayer is used add input MakiTensors together.

Parameters

name [str] Name of this layer.

Concatenates input MakiTensors along certain *axis*.

Parameters

axis [int] Dimension along which to concatenate.

name [str] Name of this layer.

Adds rows and columns of zeros at the top, bottom, left and right side of an image tensor.

Parameters

padding [list] List the number of additional rows and columns in the appropriate directions. For example like [[top,bottom], [left,right]]

name [str] Name of this layer.

Performs global maxpooling. NOTICE! The output tensor will be flattened, i.e. will have a shape of [batch size, num features].

4.4 Other layers

4.4.1 BiasLayer

BiasLayer adds a bias vector of dimension D to a tensor.

Parameters

D [int] Dimension of bias vector.

name [str] Name of this layer.

4.4.2 DenseLayer

Parameters

in_d [int] Dimensionality of the input vector. Example: 500.

out_d [int] Dimensionality of the output vector. Example: 100.

activation [TensorFlow function] Activation function. Set to None if you don’t need activation.

W [numpy ndarray] Used for initialization the weight matrix.

b [numpy ndarray] Used for initialisation the bias vector.

use_bias [bool] Add bias to the output tensor.

name [str] Name of this layer.

4.4.3 ScaleLayer

ScaleLayer is used to multiply input MakiTensor on *init_value*, which is trainable variable.

Parameters

init_value [int] The initial value which need to multiply by input.

name [str] Name of this layer.
