Part II

Building DL Models

Introduction to TensorFlow

M.Sc. M. Fischer

Prof. Dr.-Ing. B. Yang

Agenda – DL Models

- TensorFlow vs PyTorch
- Keras API
- Build Neural Networks

TensorFlow vs PyTorch

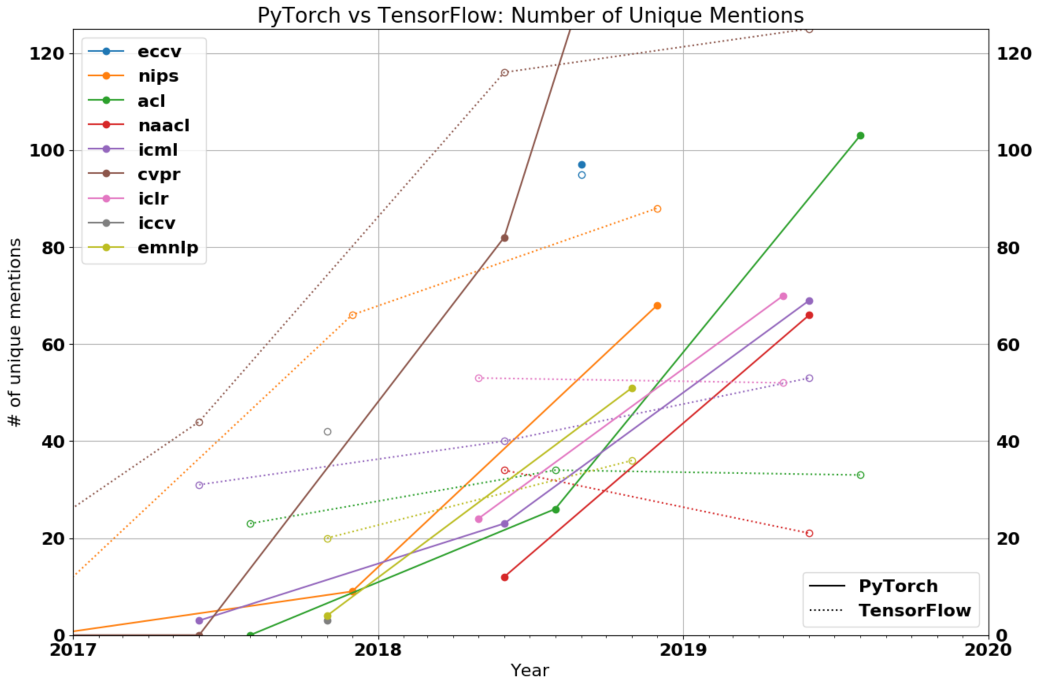
TensorFlow

- Most popular (open-source) DL library
- Large community

  >2000 contributors worldwide, tutorials, examples
- Growing TensorFlow eco system
- Provides excellent combination of low- and high-level API
- TF 2.0: Eager execution

But …

- PyTorch is becoming more popular in research
- Feature sets become more similar



Source: The State of Machine Learning Frameworks in 2019, https://thegradient.pub

TensorFlow vs PyTorch

TensorFlow

- Eager execution / debugging
- AutoGraph
- Keras
- TFRecords
- Dataset API
- Distributed Training

PyTorch

- Very similar to Autograd
- TorchScript
- Ignite / Lightning
- No own data structure / hdf5
- DataLoader
- Distributed Training

→ Community and respective libraries will likely decide your choice

Keras API

Keras makes model building easy

- Keras introduces high level concepts
- Layers / models are python objects
- Readily available activation functions, losses, optimizers
- Deeply integrated within TF 2.0

Keras API

- Keras provides out of the box models supporting
  - Layers
  - Inputs / Outputs
  - Functionality for training / evaluation

- A model can be defined in three different ways
  - Sequential (beginners) – stacking of layers
  - Functional (advanced) – instantiation and connection of layers
  - Custom (experts) – subclassing of Model class

- Part of the TensorFlow API – tf.keras

Keras API

Sequential model

- Easiest model definition
- Provides safeguards - checks if layers are compatible
- Allows for simple layer-by-layer feedforward networks
- Single input, single output

```
num_classes = 10

# Sequential Model
model_sequential = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')
])
```

Keras API

Functional Model

- Flexible model, e.g. allows for residual / skip connections
- Provides shape checking between layers
- Multiple input, outputs can be defined
- Suitable most of the time

```
# Functional Model
inputs = keras.layers.Input(shape=(28, 28))  # special input layer

# layer classes create a layer instance that is callable on a tensor, and returns a tensor
x = keras.layers.Flatten()(inputs)
x = keras.layers.Dense(128, activation='relu')(x)
predictions = keras.layers.Dense(num_classes, activation='softmax')(x)

model_functional = keras.Model(inputs=inputs, outputs=predictions)  # pass input and outputs to keras model
```

Keras API

Custom Model

- Unrestricted freedom in model creation
- No shape check at compile time – harder to debug
- Allows overwriting parent functions

```python
# Custom Model
class CustomModel(keras.Model):

    def __init__(self, num_classes=10):
        super().__init__(name='custom_model')
        self.flatten_1 = keras.layers.Flatten()
        self.dense_1 = keras.layers.Dense(128, activation='relu')
        self.dense_2 = keras.layers.Dense(num_classes, activation='softmax')

    def call(self, inputs):
        # Define your forward pass here.
        x = self.flatten_1(inputs)
        x = self.dense_1(x)
        predictions = self.dense_2(x)

        return predictions
```

Keras API

- Keras does provide out-of-the-box optimizers, metrics and losses
- Can be used during model compilation

```
# Compile Model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- Compiled model provides training / evaluation functions
  - Fit
  - Evaluate
  - Predict

Keras API

Layers

- Takes care of *tf.Variable* creation and tracks them
- Library of popular layers available:

  Dense, Pooling, Convolutions, Activations, Dropout, Batchnorm

- Recurrent layers for sequence handling

  LSTM, GRU, RNN, Convolutional Variants, RNN Cells

- Sequential and Functional *keras.Model* consist of *keras.layers*

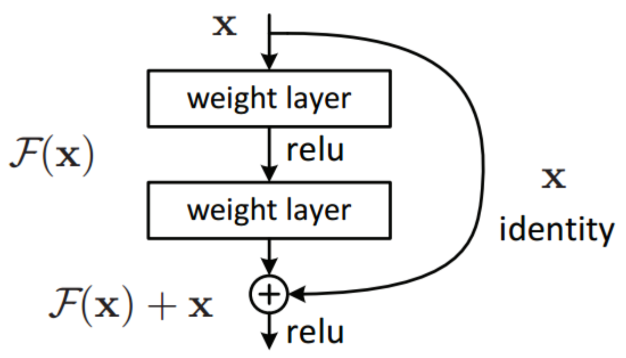
Keras API

Custom layers

- Created via lambda layer, e.g. to wrap TF functionality
- Subclass of *keras.layers.Layer*

(Custom) models

- Models can themselves incorporate Models
- *keras.Model* can be subclassed
- Can be used to build
  - Block of layers, e.g. ResNet blocks
  - Part of an architecture, e.g. Encoder-Decoder
  - A whole architecture

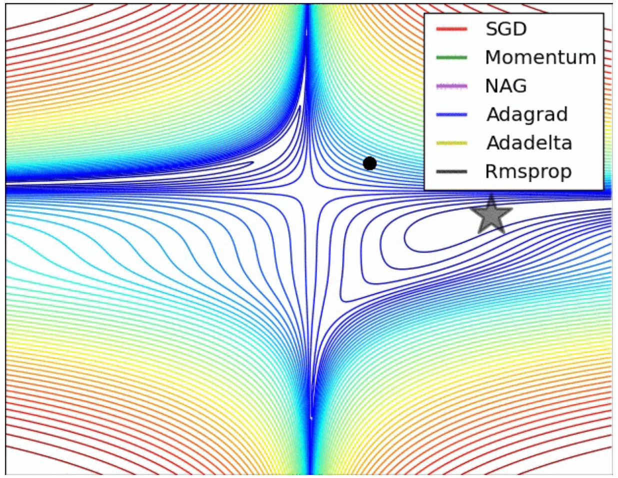


Residual Block

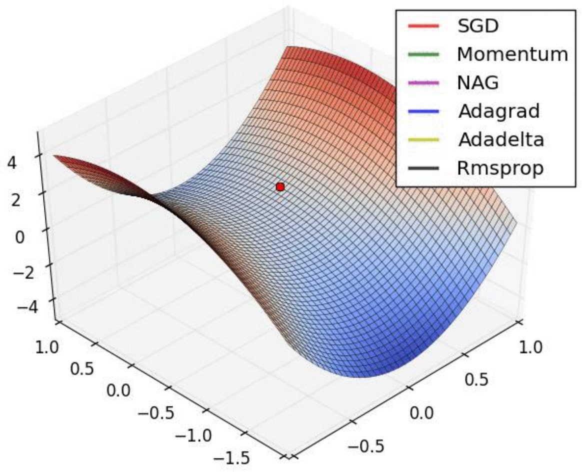
Keras API

Optimizers

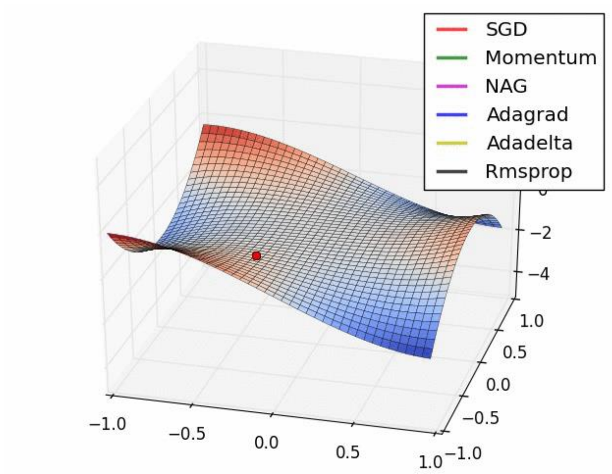
- *tf.keras.optimizers* provides popular optimizers
- Each optimizer has its own benefits and drawbacks
- Optimizers can use schedulers to define their behavior during training



Beale’s function



Long Valley



Saddle Point

Alec Radford, https://imgur.com/a/Hqolp

Keras API

Object-oriented metrics

- Each object keeps track of its internal metrics
  - *Metric.update_state()* – accumulates statistics
  - *Metric.result()* – computes and returns current metric value
  - *Metric.reset_states()* – resets all state variable
- Standard metrics are available

  Precision, Recall, Accuracy, Mean Squared Error, …
- Custom metrics
  - Using *keras.metrics.Mean*
  - Implement subclass of *keras.metrics.Metric*

```
metric_loss = tf.keras.metrics.Mean("loss")

for idx_epoch in range(n_epochs):
    # average loss across one epoch
    for batch in dataset:
        y_pred = model(batch)
        loss_value = loss(y_true, y_pred)
        metric_loss.update(loss_value)  # add current loss

    metric_loss.result()  # => average loss
    metric_loss.reset_states()  # reset for next epoch
```

Keras API

Callbacks

- Compiled models have a wrapper around their training loops – *model.fit()*
- A callback is a set of functions applied at certain stages of the training procedure
    - *on_epoch_end*
    - *on_batch_begin*
    - *on_batch_end*
- Mostly used to gather internal statistics of the Keras model
- List of callbacks is passed to model.fit()
    - *ModelCheckpoint*
    - *TensorBoard*
    - *LearningRateScheduler*
    - *LambdaCallback*

Notebook – Keras API

Train and evaluate your first Neural Network

- Classification of Fashion MNIST
- Using high-level Keras API
- Two networks: MLP, CNN + dense tail

→ see intro_nb_keras.ipynb

Gradient Tape

Custom training loop

- Independent of Keras API
- Tape context keeps track of all used operations in forward pass
- Enables automatic differentiation in eager mode
- Allows for gradient manipulation; e.g. clipping, regularization
- Does work with Keras model object – no compilation needed

```
def train_step(input, y_true):
    with tf.GradientTape() as tape:
        y_pred = model(input, training=True)
        loss_value = loss(y_true, y_pred)

        # tf.keras.Model keeps track of your trainable variables
        gradients = tape.gradient(loss_value, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

Keras vs TensorFlow

Remarks

- Both offer rich and intuitive APIs
- Keras and TF complement each other
- Keras API has been designed as a **framework agnostic wrapper**
- Drawbacks:
    - Both APIs are in active development
    - Some unfinished / some deprecated elements
    - Keras and TF are competing with respect to certain functionalities
    - Mix of both APIs is common

Notebook – Image Segmentation

Combining Models

- UNet based architecture
- Pretrained MobileNetV2 Encoder
- Using Keras API

→ see intro_nb_segmentation.ipynb