$\mathcal{N} = 8$ Best Practices
For NLP & ML

Daniel Fleischer

Introduction

AMOBEE

- Collection of insights.
- Things we got wrong.
- Problems: sentiment, embeddings.

Keras API

1

AMOBEE

Keras API

AMOBEE

- **Pros:**
  - High-level wrapper for TensorFlow.
  - Abstract, clear notation.
  - Can mix code, interact w/ backend.
  - Official support.

Keras API

AMOBEE

- **Pros:**
  - High-level wrapper for TensorFlow.
  - Abstract, clear notation.
  - Can mix code, interact w/ backend.
  - Official support.
- **Cons:**
  - Buggy.
  - API mismatch.
  - Does not cover everything.

Keras Pros/Cons

AMOBEE

- Prefer Functional API over Sequential.

Keras Pros/Cons

AMOBEE

- Prefer Functional API over Sequential.

```
model = Sequential()
model.add(Dense(5, activation='relu', input_dim=8))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Keras Pros/Cons

AMOBEE

- Prefer Functional API over Sequential.

```
model = Sequential()
model.add(Dense(5, activation='relu', input_dim=8))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

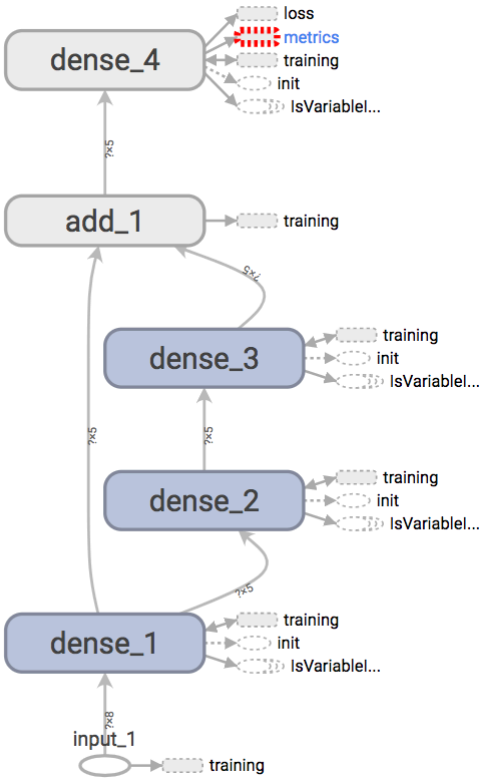
```
input_0 = Input(shape=(8,))

layer_0 = Dense(5, activation='relu', input_dim=8) (input_0)
layer_1 = Dense(5, activation='relu') (layer_0)
layer_2 = Dense(1, activation='sigmoid') (layer_1)

model = Model(inputs=[input_0], outputs=[layer_2])
```

Keras Pros/Cons

AMOBEE

- Example: fast-forward connection.

```
input_0 = Input(shape=(8,))

layer_0 = Dense(5, input_dim=8) (input_0)
layer_1 = Dense(5) (layer_0)
layer_2 = Dense(5) (layer_1)
layer_3 = Add() ([layer_0, layer_2])
layer_4 = Dense(1, activation='sigmoid') (layer_3)

model = Model(inputs=[input_0], outputs=[layer_4])
```



Keras Pros/Cons

AMOBEE

- Treat as beta! Examples:
  - Freezing weights.
  - Shuffle's not shuffling.

Transfer Learning

AMOBEE

- Based on Hierarchy.
- Example: Word Embedding.
- Similarity of Tasks.

- Try it!

Transfer Learning

AMOBEE

- Example:

```
def create_model():
    ...

    old = keras.model.load_model('old_model.hdf5')

    old.trainable = False

    hidden = Model(inputs = old.layers[0].input,
                   outputs = old.get_layer('some_hidden_layer').output)

    layer_n = hidden(input_n)

    ...

    model = Model(inputs=[...], outputs=[...])
    model.compile(...)

    return model
```

Custom Loss

AMOBEE

- Custom Metric → Custom Loss
- Pearson Correlation, Jaccard Similarity.
- Need to be implemented in TF.

Custom Loss

AMOBEE

- Pearson:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

Custom Loss

AMOBEE

- Pearson:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

```
x = np.asarray(x)
y = np.asarray(y)

mx = x.mean()
my = y.mean()
xm, ym = x - mx, y - my

r_num = np.add.reduce(xm * ym)
r_den = np.sqrt(_sum_of_squares(xm) * _sum_of_squares(ym))

r = r_num / r_den
r = max(min(r, 1.0), -1.0)
```

scipy.stats.pearsonr

Custom Loss

AMOBEE

- Pearson:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}$$

```
x = np.asarray(x)
y = np.asarray(y)

mx = x.mean()
my = y.mean()
xm, ym = x - mx, y - my

r_num = np.add.reduce(xm * ym)
r_den = np.sqrt(_sum_of_squares(xm) * _sum_of_squares(ym))

r = r_num / r_den
r = max(min(r, 1.0), -1.0)
```

scipy.stats.pearsonr

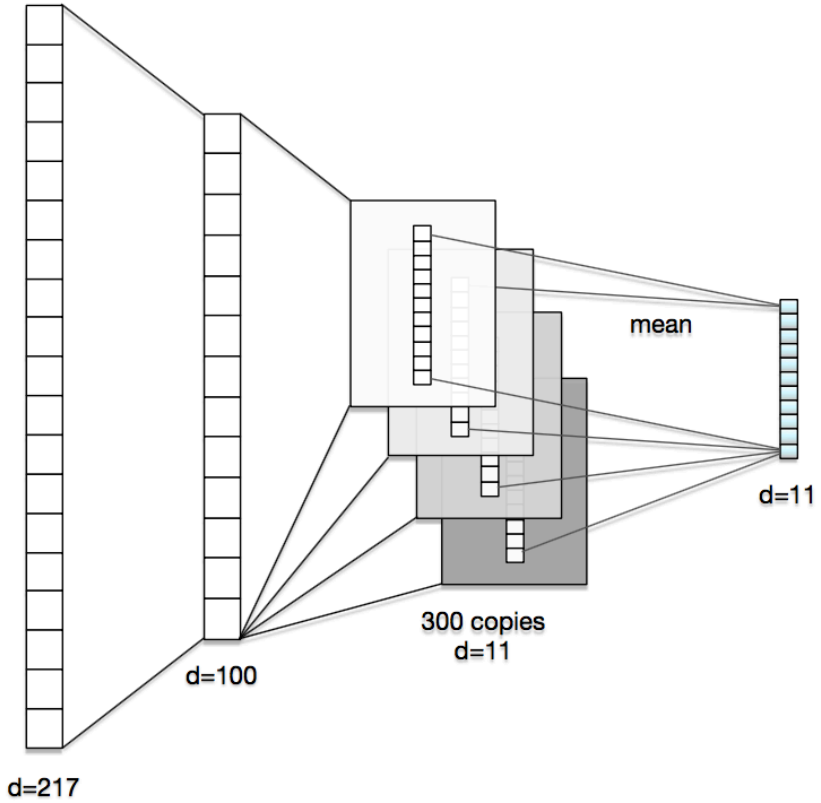
```
mx = K.mean(y_true)
my = K.mean(y_pred)
xm, ym = y_true - mx, y_pred - my

r_num = K.sum(xm * ym)
r_den = K.sqrt(K.sum(xm * xm) * K.sum(ym * ym))

r = r_num / r_den
r = K.maximum(K.minimum(r, 1.0), -1.0)
```

from keras import backend as K

Ensemble Trick

AMOBEE

- Duplicate final layer.
- Soft voting ensemble: mean.
- Better results—sometimes.



Tokenization

AMOBEE

- Text to numbers.
  - Tokens.
  - Dictionary
  - Encoding.

Tokenization

AMOBEE

- Text to numbers.
  - Tokens.
  - Dictionary - non reproducible.
  - Encoding.

Tokenization

AMOBEE

- Text to numbers.
  - Tokens.
  - Dictionary - non reproducible.
  - Encoding.

```
tok = keras.preprocessing.text.Tokenizer()
tok.fit_on_texts(["How do you like them apples?"])
print(tok.word_index)

{'how': 1, 'do': 2, 'you': 3, 'like': 4, 'them': 5, 'apples': 6}
```

Tokenization

AMOBEE

- Embedding example: gensim & keras.

```
# Load embeddings
embed = gensim.models.KeyedVectors.load_word2vec_format(fname)
words = set(embed.vocab.keys())
get_vec = lambda word: embed[word] if word in words else embed["_unk_"]

# Load tokenizer
tokenizer = get_tokenizer(fname)
word_index = tokenizer.word_index

# Build matrix
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_matrix[i] = get_vec(word)
```

Tokenization

AMOBEE

- Embedding example: gensim & keras.

```
# Load embeddings
embed = gensim.models.KeyedVectors.load_word2vec_format(fname)
words = set(embed.vocab.keys())
get_vec = lambda word: embed[word] if word in words else embed["_unk_"]

# Load tokenizer
tokenizer = get_tokenizer(fname)
word_index = tokenizer.word_index

# Build matrix
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_matrix[i] = get_vec(word)
```

```
embed_layer = Embedding(len(word_index) + 1, embedding_dim,
                        weights=[embedding_matrix],
                        input_length=MAX_LENGTH,
                        trainable=trainable)
```

Tokenization

AMOBEE

- Pay attention to tokenization.

```
tok = Tokenizer()
tok.fit_on_texts(["ynet.co.il, facebook.com, mail.google.com"])
print(tok.word_counts)

OrderedDict([('ynet', 1), ('co', 1), ('il', 1),
             ('facebook', 1), ('com', 2), ('mail', 1),
             ('google', 1)])
```

Tokenization

AMOBEE

- Pay attention to tokenization.

```
tok = Tokenizer()
tok.fit_on_texts(["ynet.co.il, facebook.com, mail.google.com"])
print(tok.word_counts)

OrderedDict([('ynet', 1), ('co', 1), ('il', 1),
             ('facebook', 1), ('com', 2), ('mail', 1),
             ('google', 1)])
```

```
tok = Tokenizer(filters=',')
tok.fit_on_texts(["ynet.co.il, facebook.com, mail.google.com"])
print(tok.word_counts)

OrderedDict([('ynet.co.il', 1), ('facebook.com', 1), ('mail.google.com', 1)])
```

Working With GPU

6

AMOBEE

Working With GPU

AMOBEE

- RAM is limited.

Working With GPU

AMOBEE

- RAM is limited.

- Restrict RAM. Helps with multi-user training.

Working With GPU

AMOBEE

- RAM is limited.
- Restrict RAM. Helps with multi-user training.

```
import tensorflow as tf
from keras.backend.tensorflow_backend import set_session

config = tf.ConfigProto()
config.gpu_options.allow_growth=True
config.gpu_options.per_process_gpu_memory_fraction = 0.25

set_session(tf.Session(config=config))
```

Working With GPU

AMOBEE

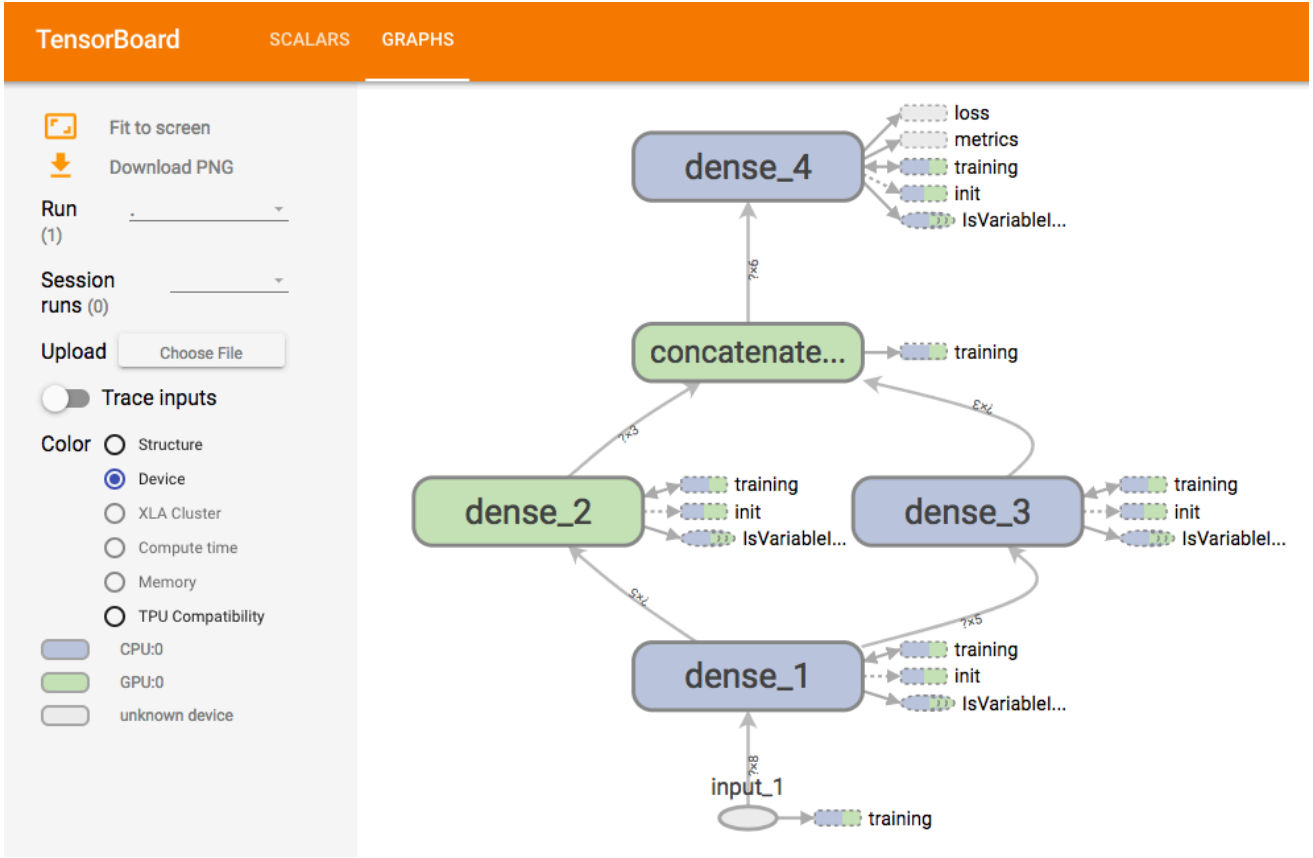
- Change computation location.

```
input_0 = Input(shape=(8,))

with tf.device('/cpu:0'):
    layer_0 = Dense(5, input_dim=8, activation='relu')(input_0)

with tf.device('/gpu:0'):
    layer_1_1 = Dense(3, activation='relu')(layer_0)

with tf.device('/cpu:0'):
    layer_1_2 = Dense(3, activation='relu')(layer_0)

with tf.device('/gpu:0'):
    layer_2 = concatenate([layer_1_1, layer_1_2])

with tf.device('/cpu:0'):
    layer_3 = Dense(1, activation='sigmoid')(layer_2)
```

Working With GPU

AMOBEE

- Change computation location.



Working With GPU

AMOBEE

- Break into sub-models; train-predict multiple times.
  - Can’t train everything together.

Feature Ranking / Selection

7

AMOBEE

Feature Ranking / Selection

AMOBEE

- Importance-based Ranking.

Feature Ranking / Selection

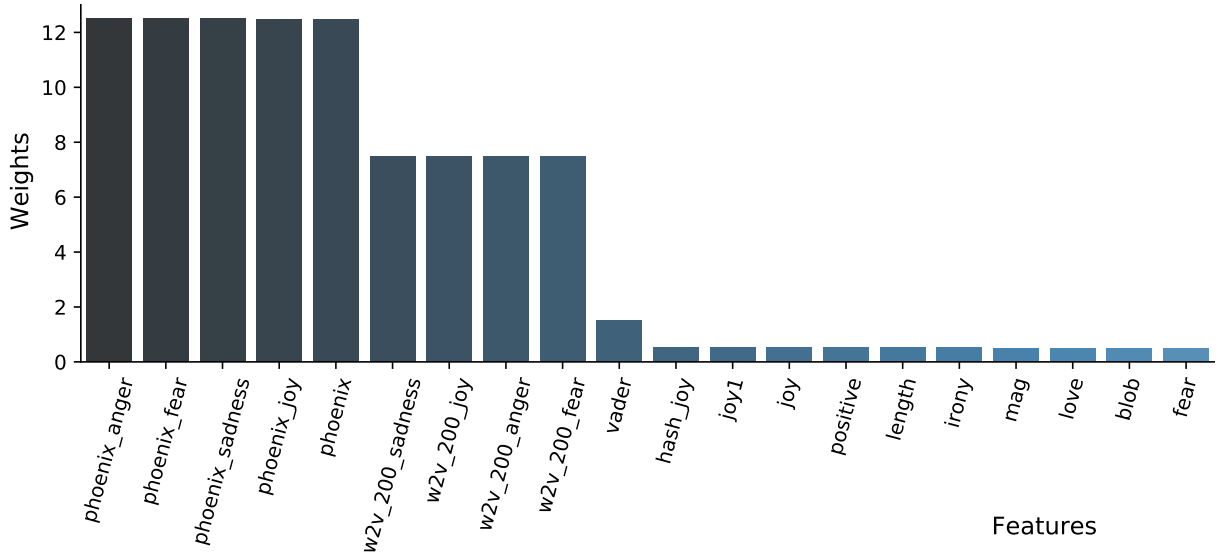
AMOBEE

- Importance-based Ranking.
- Naive guess: LR weights. Sounds good, doesn’t work.

Feature Ranking / Selection

AMOBEE

- Importance-based Ranking.
- Naive guess: LR weights. Sounds good, doesn't work.



Feature Ranking / Selection

AMOBEE

- 3 Families: Wrappers, Filters, all-the-rest.
- Pratt (1987) - Covariance Decomposition.

Feature Ranking / Selection

AMOBEE

- 3 Families: Wrappers, Filters, all-the-rest.
- Pratt (1987) - Covariance Decomposition.

$$d_i = \frac{\beta_i \, \rho_i}{R^2}$$

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

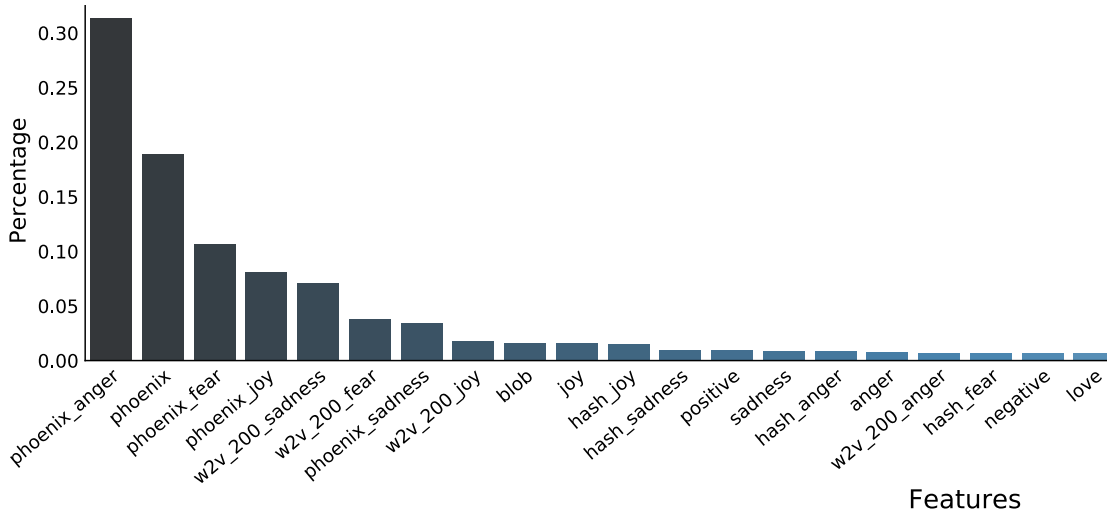
Feature Ranking / Selection

AMOBEE

- 3 Families: Wrappers, Filters, all-the-rest.
- Pratt (1987) - Covariance Decomposition.

$$d_i = \frac{\beta_i \, \rho_i}{R^2}$$

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$



Visualization

AMOBEE

- TensorBoard, Keras
- Used for:
  - Understand
  - Debug
  - Optimize

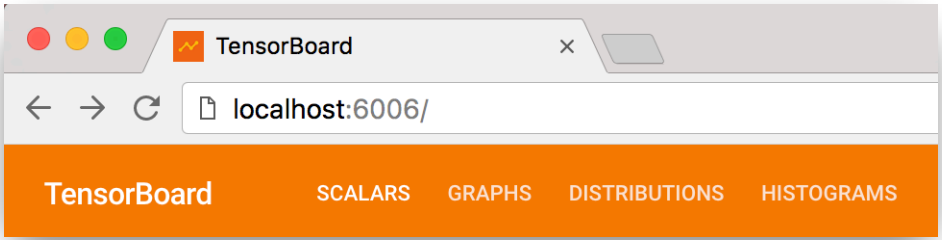
TensorBoard

AMOBEE

- How to use:

```
tensorboard = keras.callbacks.TensorBoard(log_dir='./Graph')
model.fit(..., callbacks=[tensorboard,...])
```

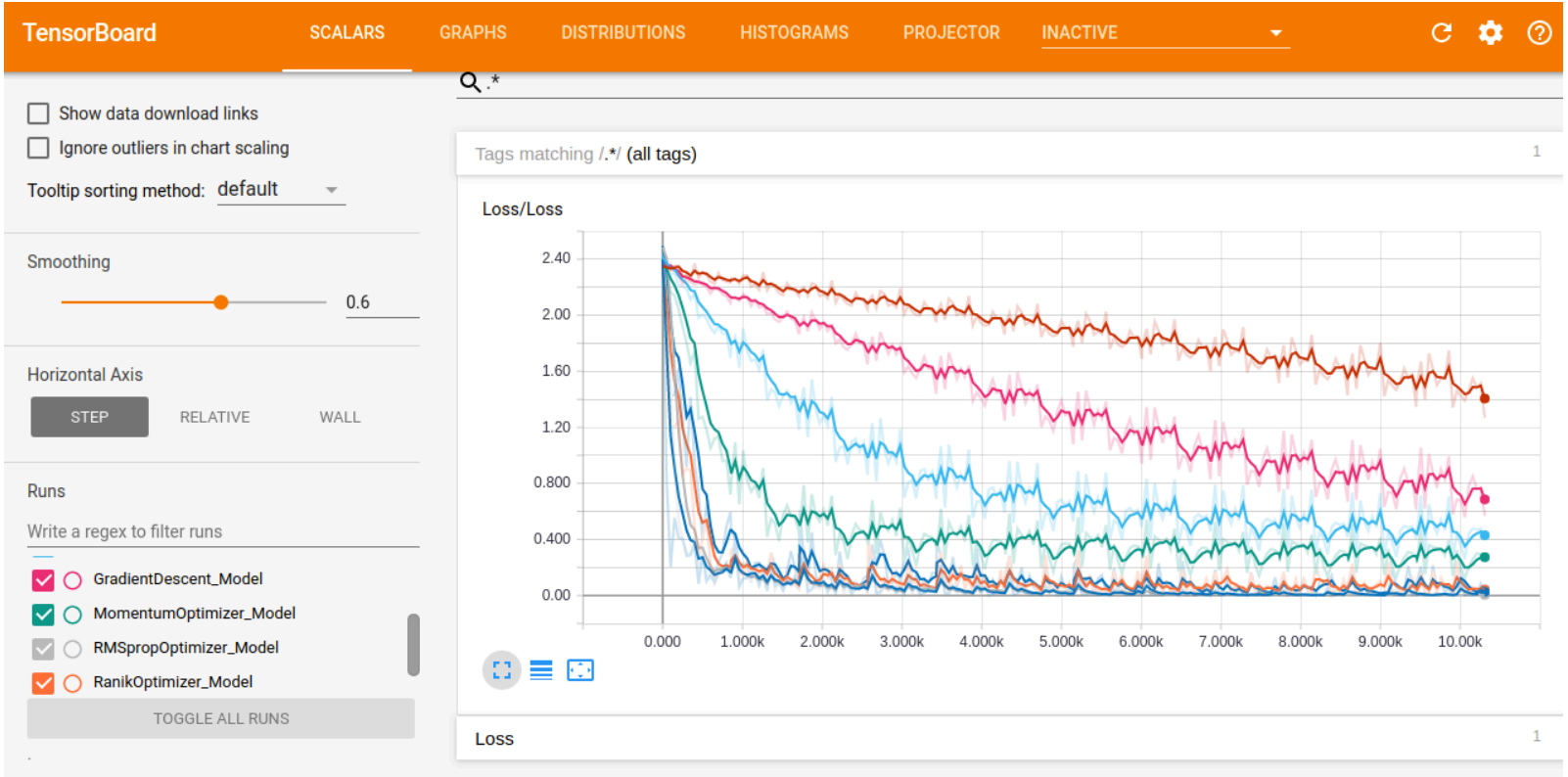
```
$ tensorboard --logdir ./Graph
```



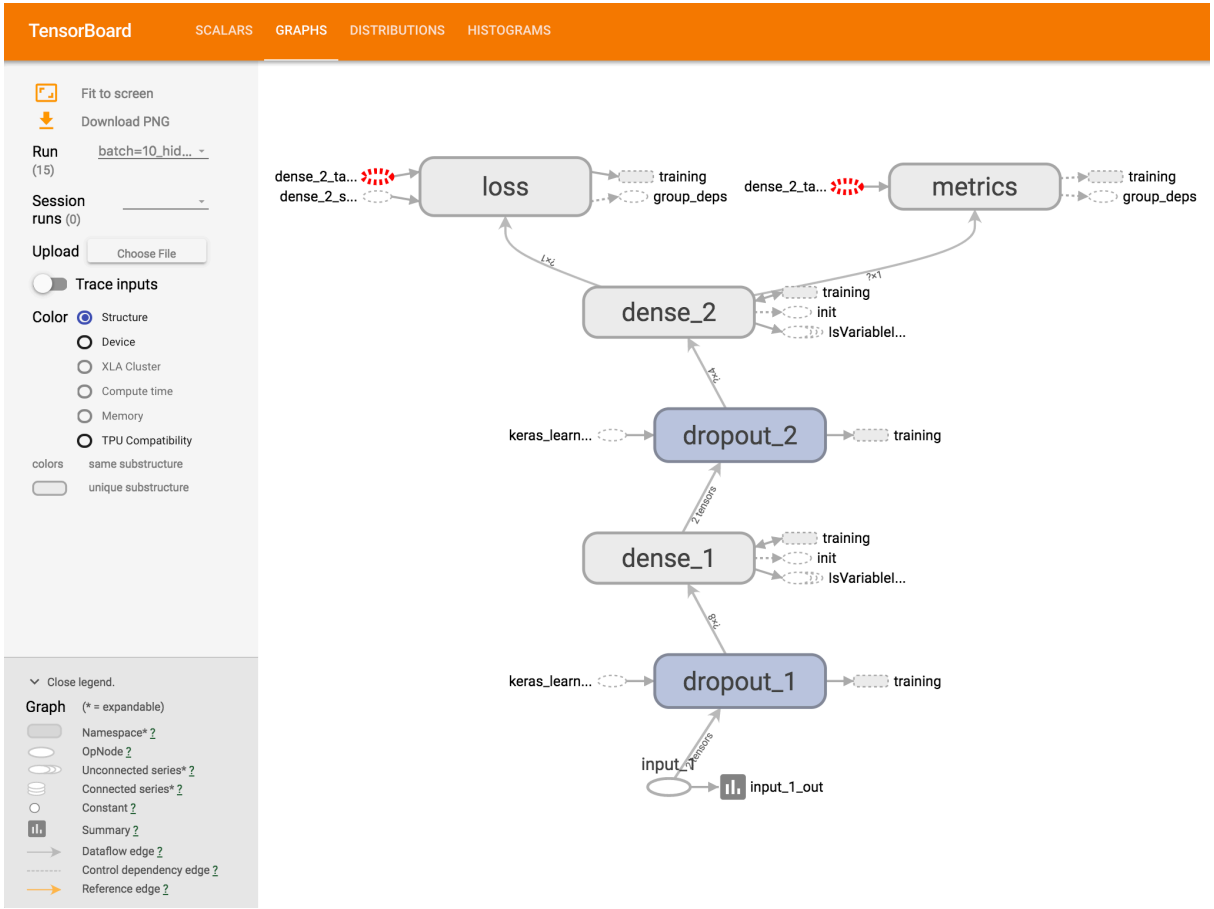
TensorBoard

AMOBEE

- Graph plotting, sanity checks: loss, accuracy, weights.



TensorBoard



Keras Visualization

AMOBEE

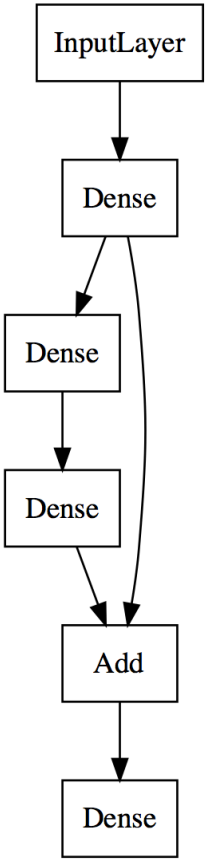
- TB writing can slow the training, overkill.
- Simpler alternative: keras.

Keras Visualization

AMOBEE

- TB writing can slow the training, overkill.
- Simpler alternative: keras.

```
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```

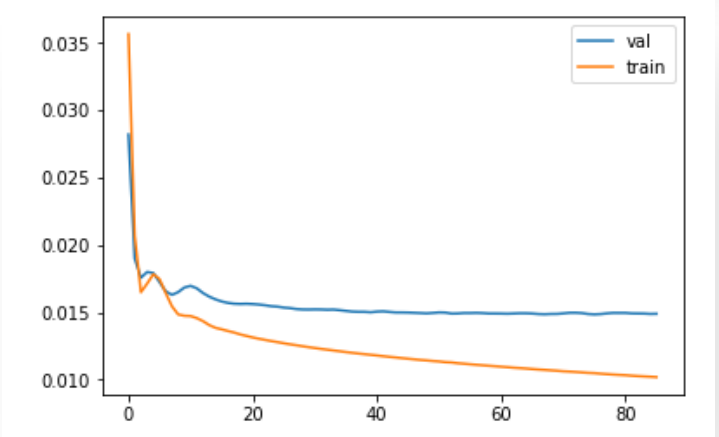


Keras Visualization

AMOBEE

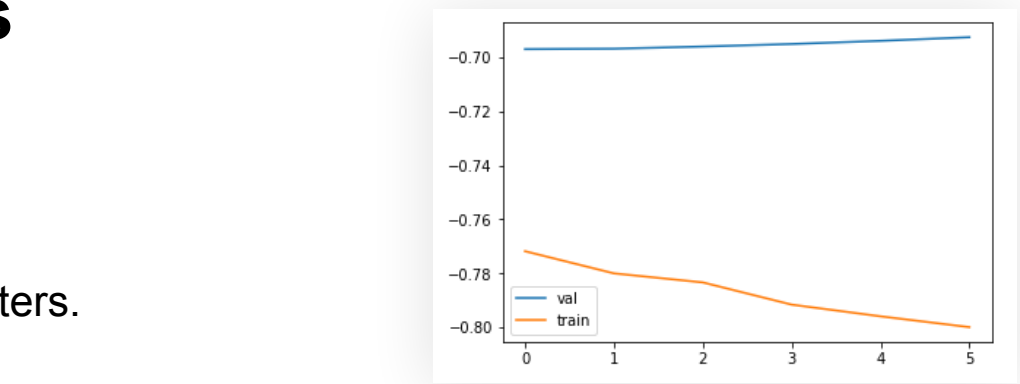
- Keras history callback.
- Epoch-based.

```
import matplotlib.pyplot as plt

...

history = model.fit(x_train, y_train, ...)

plt.plot(history.history['val_loss'])
plt.plot(history.history['loss'])
plt.legend(['val','train'])
plt.show()
```

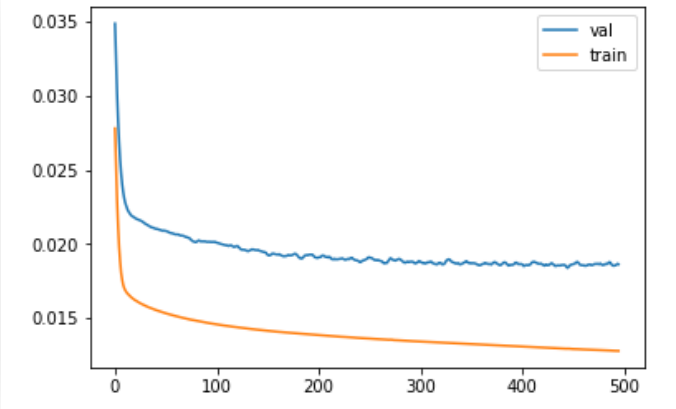


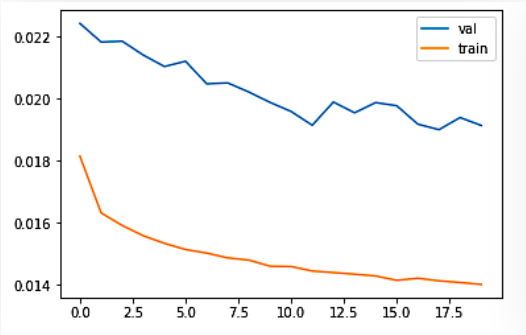
Learning Curves

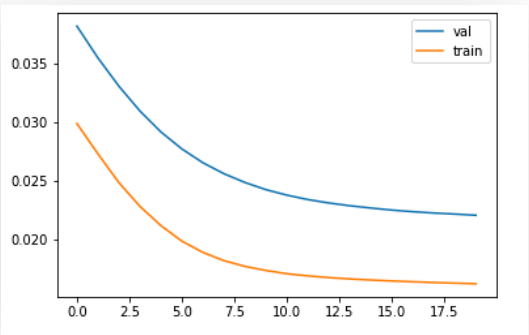
- What can we learn?
  - Compare models.
  - Optimize model parameters.
  - Improve convergence.



Overfitting







Noisy - batch size

Summarize:

AMOBEE

- Input pipeline - “garbage in, garbage out”.
- Transfer Learning - Combine models.
- Visualize your network.

Thank You!

carbon

danielfleischer