Prize recipient documentation

Power Laws: Cold Start Energy Forecasting

Guillermo Barbadillo Villanueva a.k.a. “ironbar”
guillermobarbadillo@gmail.com

III. Model documentation and write-up

1. Who are you (mini-bio) and what do you do professionally?

I finished a master in Electronic Engineering 5 years ago. Since June 2014 I have been studying and practicing Artificial Intelligence on my own. First on my free time and 3 years ago I started working also on AI. My expertise is on applying deep learning to computer vision, but I have also worked with structured data and text.
I love doing Data Science challenges and with this is the 3rd time I have been luckily end on a winning position.

2. High level summary of your approach: what did you do and why?

I have prepared a presentation describing my work during the challenge. It should be attached to this mail.

3. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

Taylor Made Neural Network

The following code allows to build arbitrary architectures. It allowed to build the taylor-made neural network used in the challenge. More information and schema on the presentation attached.

```
def create_model(x, conf):
    """
    Given the dictionary with the input to the model a keras
    model is build. It is mandatory to have past_consumption as input.

    The structure will be the following:
        1. mlp for encoding each input except past_consumption
        2. merge of encodings
        3. mlp on top of encodings
        4. predict weights for prediction
        5. weighted average of past consumption
    """
    input_layers = _create_input_layers(x)
```

```
    encodings = _get_encodings(input_layers, conf['encoding'])
    merged_encodings = _merge_encodings(encodings)
    weights_features = _add_layers(merged_encodings, conf['weights'])
    weights = _get_weights_for_prediction(weights_features, x,
conf['repeat_weights'])
    output = Multiply()([weights, input_layers['past_consumption']])
    output = Lambda(lambda x: K.sum(x, axis=2))(output)

    model = Model(list(input_layers.values()), output)
    return model

def _create_input_layers(x):
    input_layers = {}
    for key in x:
        input_layers[key] = Input(shape=(x[key].shape[1:]), name=key)
    return input_layers

def _get_encodings(input_layers, encoding_conf):
    encodings = []
    for key in input_layers:
        if key == 'past_consumption':
            continue
        if key in encoding_conf:
            encodings.append(_add_layers(input_layers[key], encoding_conf[key]))
        else:
            encodings.append(input_layers[key])
    return encodings

def _merge_encodings(encodings):
    if len(encodings) > 1:
        return Concatenate()(encodings)
    elif len(encodings) == 1:
        return encodings[0]
    else:
        raise Exception('No encoding found')
```

```python
STR_TO_LAYER = {
    'Dense': Dense,
    'LSTM': LSTM,
    'Dropout': Dropout,
    'BatchNormalization': BatchNormalization,
}

def _add_layers(output, params):
    for layer_conf in params:
        layer_conf = layer_conf.copy()
        layer = STR_TO_LAYER[layer_conf.pop('layer')]
        output = layer(**layer_conf)(output)
    return output

def _get_weights_for_prediction(weights_features, x, repeat_weights):
    if repeat_weights:
        weights = Dense(x['past_consumption'].shape[2], activation='relu',
                        name='weights')(weights_features)
        weights = RepeatVector(x['past_consumption'].shape[1])(weights)
    else:
        weights = Dense(np.prod(x['past_consumption'].shape[1:]),
                        activation='relu', name='weights')(weights_features)
        weights = Reshape(x['past_consumption'].shape[1:])(weights)

    return weights
```

Train manager

I developed a simple but powerful class that allowed to train multiple models in parallel on my 2 gpu computer. I trained up to 8 models in parallel.

```python
class TrainManager(object):
    """
    Class for making easier to run different trains on parallel
    """
```

```
def __init__(self, n_workers):
    self._pool = None
    self._submits = []
    self._create_pool(n_workers)

def _create_pool(self, n_workers):
    """
    Creates the pool of workers and sets the session distributing the load
    between the two gpus giving preference to gpu 1
    """
    pool = ProcessPoolExecutor(max_workers=n_workers)
    gpus = [str(1-i%2) for i in range(n_workers)]
    submits = [pool.submit(_set_session, gpu) for gpu in gpus]
    self._submits += submits
    while not all([submit.done() for submit in submits]):
        time.sleep(1)
    self._pool = pool

def submit(self, func, *args, **kwargs):
    """ Submits a single job to the pool """
    submit = self._pool.submit(func, *args, **kwargs)
    self._submits.append(submit)
    return submit

def get_remaining_submits(self):
    self._submits = [submit for submit in self._submits if not submit.done()]
    return len(self._submits)
```

ModelCheckpoint in RAM

This callback allows to save the weights of the model on RAM instead of saving them to disk. This is very helpful when the epoch is very short (in the order of seconds) and saving to disk becomes a relevant time.

```
class ModelCheckpointRAM(Callback):
  """Save the model after every epoch.
  `filepath` can contain named formatting options,
  which will be filled the value of `epoch` and
  keys in `logs` (passed in `on_epoch_end`).
  For example: if `filepath` is `weights.{epoch:02d}{val_loss:.2f}.hdf5`,
  then the model checkpoints will be saved with the epoch number and
  the validation loss in the filename.
  # Arguments
      monitor: quantity to monitor.
    verbose: verbosity mode, 0 or 1.
    save_best_only: if `save_best_only=True`,
      the latest best model according to
      the quantity monitored will not be overwritten.
    mode: one of {auto, min, max}.
      If `save_best_only=True`, the decision
      to overwrite the current save file is made
      based on either the maximization or the
      minimization of the monitored quantity. For `val_acc`,
      this should be `max`, for `val_loss` this should
      be `min`, etc. In `auto` mode, the direction is
      automatically inferred from the name of the monitored quantity.
    save_weights_only: if True, then only the model's weights will be
      saved (`model.save_weights(filepath)`), else the full model
      is saved (`model.save(filepath)`).
    period: Interval (number of epochs) between checkpoints.
  """

  def __init__(self, monitor='val_loss', verbose=0,
           mode='auto', period=1, **kwargs):
    self.monitor = monitor
    self.verbose = verbose
    self.period = period
    self.epochs_since_last_save = 0
    self.weights = None
```

```
        if mode not in ['auto', 'min', 'max']:
            warnings.warn('ModelCheckpoint mode %s is unknown, '
                          'fallback to auto mode.' % (mode),
                          RuntimeWarning)
            mode = 'auto'

        if mode == 'min':
            self.monitor_op = np.less
            self.best = np.Inf
        elif mode == 'max':
            self.monitor_op = np.greater
            self.best = -np.Inf
        else:
            if 'acc' in self.monitor or self.monitor.startswith('fmeasure'):
                self.monitor_op = np.greater
                self.best = np.Inf
            else:
                self.monitor_op = np.less
                self.best = np.Inf

    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        self.epochs_since_last_save += 1
        if self.epochs_since_last_save >= self.period:
            self.epochs_since_last_save = 0
            current = logs.get(self.monitor)
            if current is None:
                warnings.warn('Can save best model only with %s available, '
                              'skipping.' % (self.monitor), RuntimeWarning)
            else:
                if self.monitor_op(current, self.best):
                    if self.verbose > 0:
                        print('Epoch %05d: %s improved from %0.5f to %0.5f,'
                              ' saving model'
                              % (epoch, self.monitor, self.best, current))
                    self.best = current
```

```
            self.weights = self.model.get_weights()
        else:
            if self.verbose > 0:
                print('Epoch %05d: %s did not improve' %
                      (epoch, self.monitor))
```

4. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

At the start of the challenge I tried using simple linear regression and although the results were much better than the LSTM baseline they were not good enough to enter in the final solution.

I also tried to train another architecture that used both LSTM and vanilla neural networks but the results were worse so I did not use it. I called this architecture the "Frankenstein".

5. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

I used the typical tools for visualization in python such as matplotlib and seaborn.

6. How did you evaluate performance of the model other than the provided metric, if at all?

I only used the metric provided on the challenge. The only difference is that I computed the metric for hourly, daily and weekly independently instead of mixing them together. That allowed to compare the performance on the different time windows.

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

I have seen that the Taylor-made NN are quite challenging to train. When using the same parameters as the final solution there should not be any problem. However increasing the batch size from 8 to 64 may not allow the train to converge.

8. **Do you have any useful charts, graphs, or visualizations from the process?**

Yes, in the presentation attached there are many useful visualizations.

9. **If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?**

I would like to continue developing the “Frankenstein” model architecture that mixes LSTM and NN. I did not have enough time to make it work but I believe it could produce better results. My believe is based on the fact that LSTM are better for creating representation of time series while other data such as metadata are better encoded by NN.