OPEN DATA SCIENCE CONFERENCE

San Francisco | Oct. 28 - Nov. 1, 2019

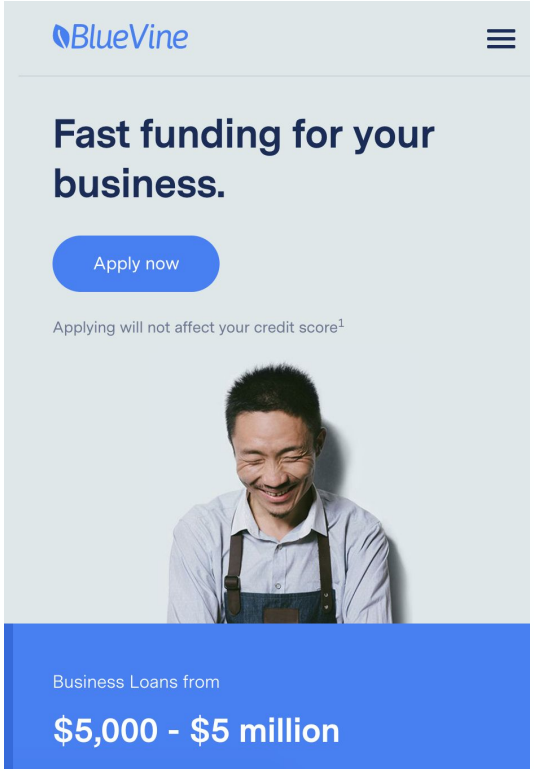
Building an Industry Classifier

Scrapy / BERT / PyTorch / SageMaker

Ido Shlomo - BlueVine

About BlueVine

- Fintech startup up based in Redwood City, CA
- Provides working capital (loans) to small & medium sized businesses
- Over $2 BN funded to date
- Mission of DS team:
  - Automate
  - Scale
  - Improve



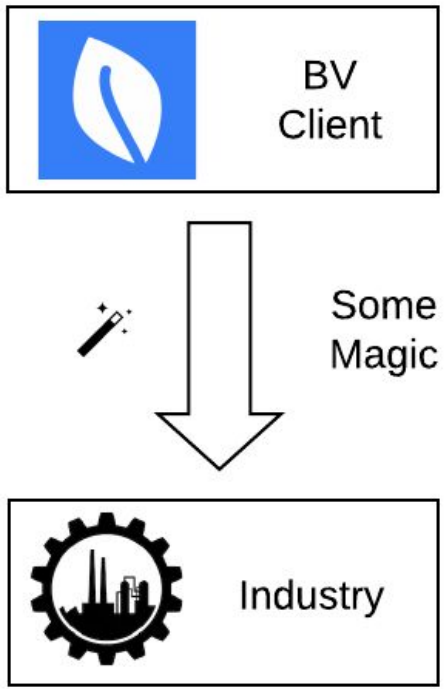
About Me

- Data Science Manager @ BlueVine
- Lead BlueVine's DS team in Redwood City, CA (total of ~20 people across RWC & TLV)
- Team focus:
  - Risk / Fraud
  - Financial Strength
  - Marketing
- Personal interests: Unstructured data and DS Infrastructure.

Slides at: https://github.com/ido-sh/odsc_presentations

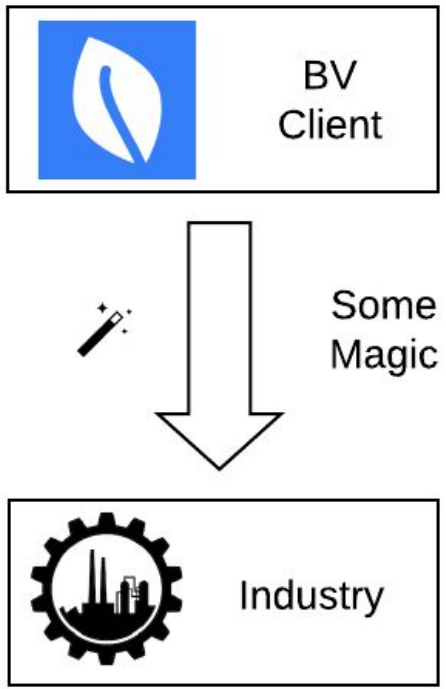
The Point of this Presentation

- Business need: For every client → Predict their Industry
- Industry can be one of 135 categories, e.g:
  - Electrician
  - Marketing
  - Steel manufacturing
- Problem: Traditional sources are pricey, inaccurate and not widely available

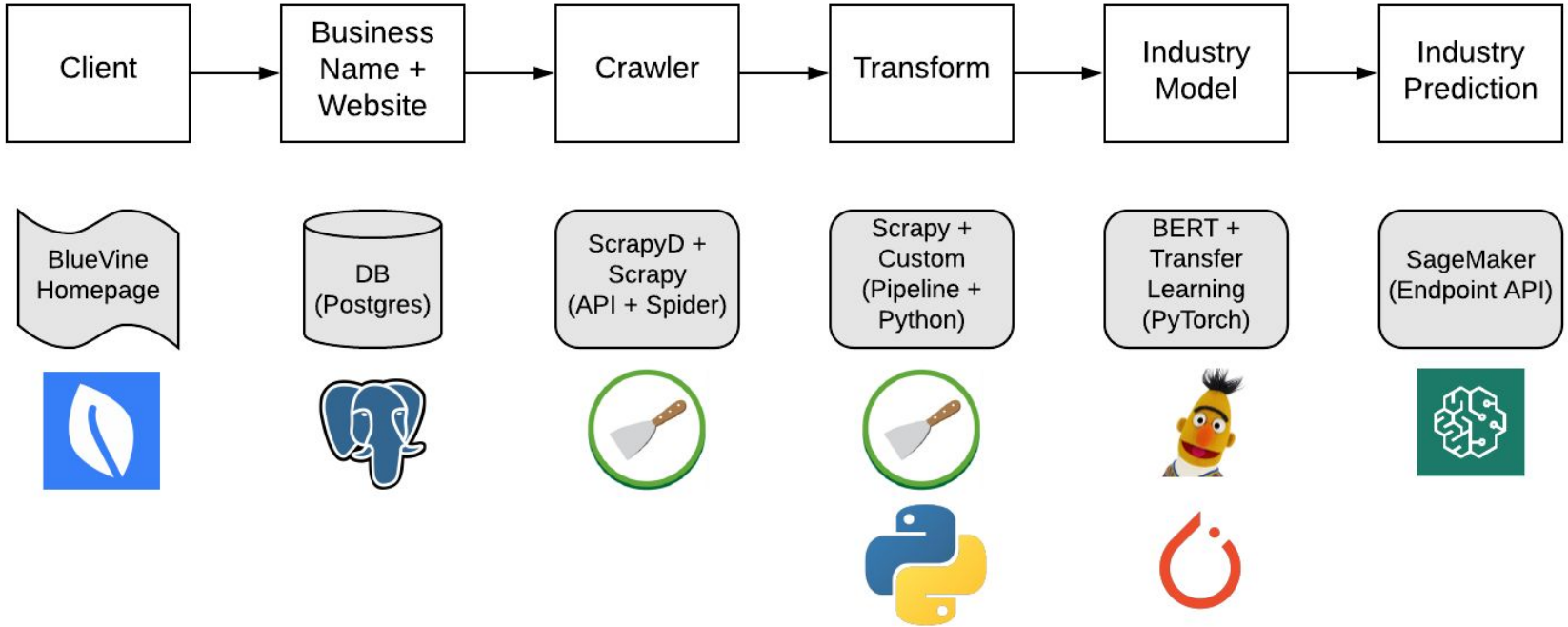


The Point of this Presentation

- Our solution: Infer industry from the business website
- Main Challenges
  - High quality sample
  - Efficient crawling
  - Strong NLP model
  - Scalable & independent deployment
- Main tools: Scrapy D, Scrapy, PyTorch, BERT, Amazon SageMaker



Inference Pipeline



Run example

<end to end notebook>

Presentation Overview

1. Target
   - Gathering websites
   - Tagging the industry

2. Data
   - Building the crawler
   - Crawling at scale

3. Model
   - Design (BERT)
   - Train
   - Deploy

4. Conclusion
   - Summary
   - Key takeaways

Target - Gathering websites

Potential sources:

- Internal BV data

  E.g: Client provided information

  - Not very big
  - Skewed scope

- Directed web searches

  E.g: “car dealerships in the US”

  - High effort
  - Limited scope

- Proprietary Databases

  E.g: Various vendors

  - Big
  - Pricy
  - Outdated

Target - Gathering websites

Problem:

- We have 135 industries
- Want sample to be balanced
- In order to balance, need to know the Industry
- If we knew the Industry, wouldn't need the model...

Target - Gathering websites

Problem:

- We have 135 industries
- Want sample to be balanced
- In order to balance, need to know the Industry
- If we knew the Industry, wouldn't need the model...

Solution:

- Use proxy to "guess" the industry
  - Internal BV Data: User provided
  - Web Searches: Search term
  - Proprietary Data: NAICS codes
- Fingers crossed, hope for the best...

Target - Tagging the Industry

Goals:

- Get quality tags
- Reasonable Timeframe
- Stay within budget

Crowdsourcing challenges:

- Understanding 135 industry categories is hard
- Understanding what a business does is hard
- Tie-breaking is hard

Row Id	Url	Biz Name	Industry
1	a.com	a inc	X
2	b.com	b corp	Y
3	c.bom	c llc	Z
4	d.com	d	X
5	e.com	e	Y
...	...	...	...

Target - Tagging the Industry

Crowdsourcing Strategies:

- Break up the task into smaller parts:
  - One obs requires multiple decisions
  - More tasks → more $$$
- Get multiple judgements for every website:
  - If everybody are bad, result will still be bad
  - More judgements → more $$$
- Train team for complex decision making:
  - Time intensive → more $$$
  - Slow

Target - Tagging the Industry

What we did:

- Set max $$$ Budget
- Set max time per task (in seconds)
- Developed small in-house tagged test set
- Held vendor trials for all strategies
- From vendors within max time cap: chose one with highest accuracy
- PLUG: CloudFactory (they are great) **https://www.cloudfactory.com/**

Target - Tagging the Industry

What we did:

- Set max $$$ Budget
- Set max time per task (in seconds)
- Developed small in-house tagged test set
- Held vendor trials for all strategies
- From vendors within max time cap: chose one with highest accuracy
- PLUG: CloudFactory (they are great) **https://www.cloudfactory.com/**

Outcome:

- Overall time: 3 months
- Overall size: 250K observations (business names + websites)
- NOT balanced, BUT sufficient
- Fingers crossed, hope for the best...

Data - Building the Crawler

For each URL, we need:

- Home page text cleaned (no HTML)
- Internal page(s) text cleaned (no HTML)
- Page metadata

  E.g: Path, title, click text, etc.
- Ability to choose

  E.g. do I even want this page?
- Raw data as backup

Data - Building the Crawler

For each URL, we need:

- Home page text cleaned (no HTML)
- Internal page(s) text cleaned (no HTML)
- Page metadata

  E.g: Path, title, click text, etc.
- Ability to choose

  E.g. do I even want this page?
- Raw data as backup

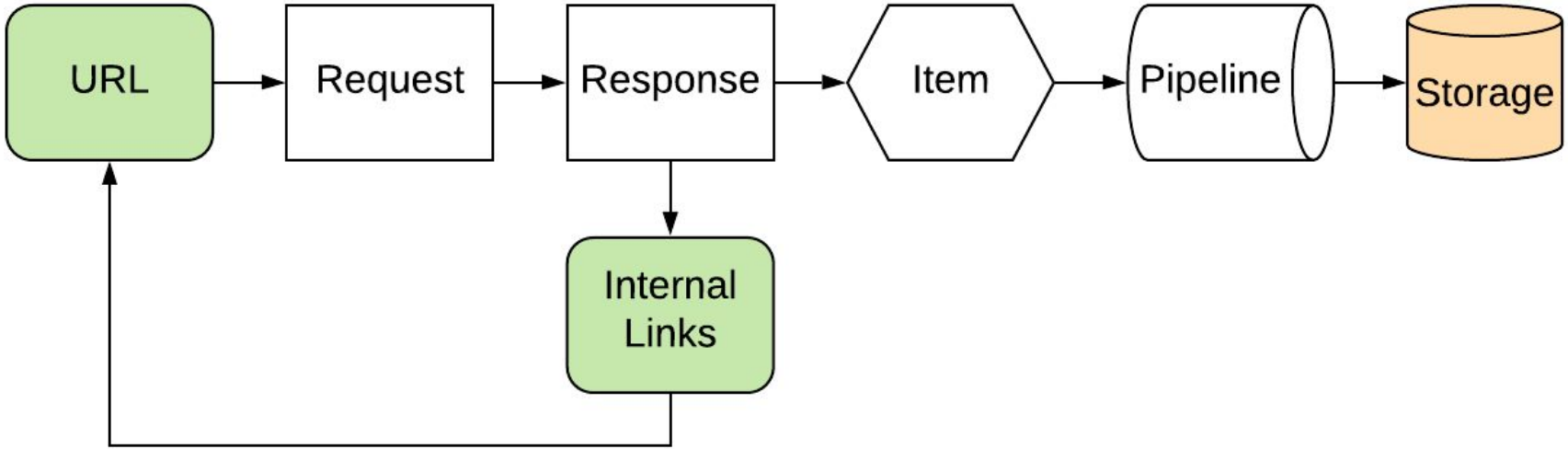
Our tool:

- Scrapy: fast high-level web crawling and web scraping framework

  https://github.com/scrapy/scrapy

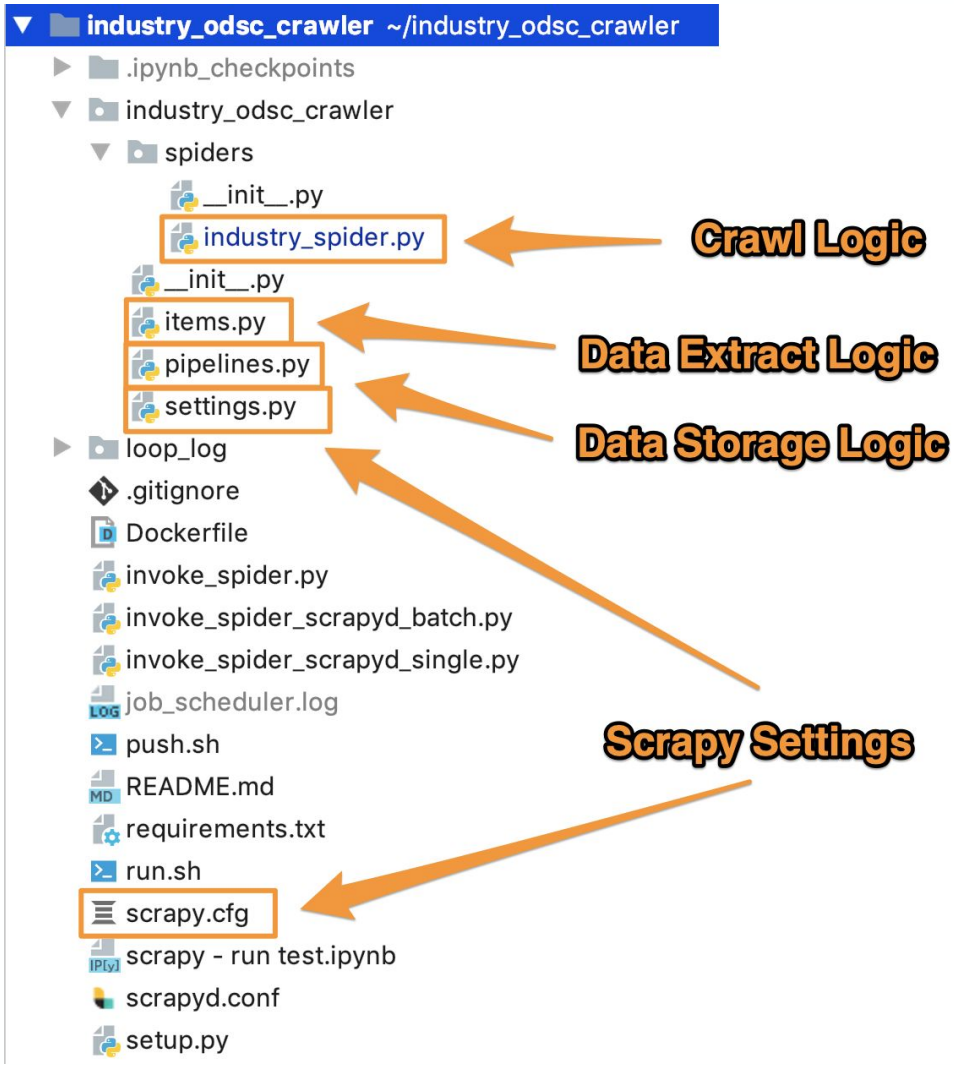
Data - Building the Crawler

Scrapy Flow: Single URL



Data - Building the Crawler

<scrapy - run test notebook>


industry_odsc_crawler ~/industry_odsc_crawler
.ipynb_checkpoints
industry_odsc_crawler
spiders
__init__.py
industry_spider.py
__init__.py
items.py
pipelines.py
settings.py
loop_log
.gitignore
Dockerfile
invoke_spider.py
invoke_spider_scrapyd_batch.py
invoke_spider_scrapyd_single.py
job_scheduler.log
push.sh
README.md
requirements.txt
run.sh
scrapy.cfg
scrapy - run test.ipynb
scrapyd.conf
setup.py
Crawl Logic
Data Extract Logic
Data Storage Logic
Scrapy Settings

```
14  class IndustrySpider(Spider):
15      name = 'industry_v4_spider'
16
17      def __init__(self, ids, urls, save_local=False,
18                   **kwargs):
19          self.start_urls = urls.split(",")
20          self.ids = [int(x) for x in ids.split(",")]
21          self.link_counter = dict(zip(self.ids, [0] * len(self.ids)))
22          self.save_local = save_local == 'True'
23          super().__init__(**kwargs)
24
25      def start_requests(self):
26          for url_id, url in zip(self.ids, self.start_urls):
27              try:
28                  request = Request(url=url, callback=self.parse,
29                                    dont_filter=True,
30                                    errback=self.handle_error)
31              except ValueError:...
36              request.meta.update(url_id=url_id, url_order=1,
37                                  url_rank=1,
38                                  url_click_text=None,
39                                  url_domain=urlparse(url).netloc)
40              yield request
```

URLs to start with

Internal link counter

Make 1st request per URL

```
42      def parse(self, response):
43          web_page = WebPage()
44          web_page['url_id'] = response.meta['url_id']
45          web_page['url_order'] = response.meta['url_order']
46          web_page['input_url'] = response.meta['download_slot']
47          web_page['depth'] = response.meta['depth'] if \
48              'depth' in response.meta.keys() else None
49          web_page['url'] = response.url
50          web_page['path'] = urlparse(response.url).path
51          web_page['url_click_text'] = response.meta['url_click_text']
52          web_page['response_raw'] = response.text
53          web_page['url_rank'] = response.meta['url_rank']
54          web_page['response_text'], web_page['url_title'] = \
55              text_from_html(response.text)
56          web_page['status_code'] = response.status
57          web_page['is_error'] = False
58          self.link_counter[response.meta['url_id']] += 1
59          yield web_page
```

Extract only text

```
61      allowed_domains = {response.meta['url_domain']}
62      if 'redirect_urls' in response.meta.keys() \
63              and isinstance(response.meta['redirect_urls'], list):...
69
70      allowed_domains = {x.replace("www.", "") for x in allowed_domains}
71      links = LinkExtractor(canonicalize=True, unique=True).\
72          extract_links(response)
73      links = [link for link in links if self.is_link_allowed(
74          allowed_domains, link.url)]
75      links = sorted(links, key=self.get_link_rank)
76
77      for link in links:
78          is_below_limit = self.link_counter[response.meta['url_id']] < \
79                           MAX_PAGES_PER_DOMAIN
80          if is_below_limit:...
```

Extract Internal Links

Sort by "priority"

Stay <= X links per domain

```
89      @staticmethod
90      def get_link_rank(link):
91          good_hits = [
92              ('about', 2),
93              ('story', 2),
94              ('what we do', 3),
95              ('capabilities', 3),
96              ('services', 4),
97              ('products', 4),
98              ('amenities', 5),
99              ('overview', 6)
100         ]
101         url_rank = 100
102         text_rank = 100
103         for hit, rank in good_hits:...
106
107         for hit, rank in good_hits:...
110
111         link_rank = min(url_rank, text_rank)
112         return link_rank
```

Prioritize informative pages

Default value (lower is better

```python
class ConvertToDataFramePipeline:
    def __init__(self):
        self.item_list = []

    def process_item(self, item, spider):
        self.item_list += [item]

    def close_spider(self, spider):
        data_clean_json, fn_clean = combine_items(
            self.item_list, spider, is_raw=False)

        data_raw_json, fn_raw = combine_items(
            self.item_list, spider, is_raw=True)

        if spider.save_local:...
        else:
            fp = 'test_crawls/{}'.format(fn_clean)
            fp_raw = 'test_crawls/{}'.format(fn_raw)
            bucket_name = 'industry-odsc-19'
            s3 = boto3.resource('s3')
            bucket = s3.Bucket(bucket_name)
            logger.info("saving to s3: {}/{}".format(bucket_name, fp))
            data_clean_json_obj = dump_json_to_object(data_clean_json)
            bucket.put_object(Body=data_clean_json_obj, Key=fp)
            logger.info("saving to s3: {}/{}".format(bucket_name, fp_raw))
            data_raw_json_obj = dump_json_to_object(data_raw_json)
            bucket.put_object(Body=data_raw_json_obj, Key=fp_raw)
```

Collect all items from spider

Process "clean" data for model

Keep "raw" as backup

Save in S3

Data - Crawling at Scale

For our list of 250K URLs, we need to:

- Run batches, in parallel
- Persistent process that’s always up
- Some logging / UI mechanism to follow progress

Data - Crawling at Scale

For our list of 250K URLs, we need to:

- Run batches, in parallel
- Persistent process that's always up
- Some logging / UI mechanism to follow progress

Our tool:

- ScrapyD: a service for running Scrapy spiders
  https://github.com/scrapy/scrapyd
- Deploy Scrapy projects and control their spiders using an HTTP JSON API

```
[scrapyd]
eggs_dir    = eggs
logs_dir    = logs
items_dir   =
jobs_to_keep = 500
dbs_dir     = dbs
max_proc    = 0
max_proc_per_cpu = 4
finished_to_keep = 100
poll_interval = 5.0
bind_address = 0.0.0.0
http_port   = 6800
debug       = off
runner      = scrapyd.runner
application = scrapyd.app.application
launcher    = scrapyd.launcher.Launcher
webroot     = scrapyd.website.Root

[services]
schedule.json     = scrapyd.webservice.Schedule
cancel.json       = scrapyd.webservice.Cancel
addversion.json   = scrapyd.webservice.AddVersion
listprojects.json = scrapyd.webservice.ListProjects
listversions.json = scrapyd.webservice.ListVersions
listspiders.json  = scrapyd.webservice.ListSpiders
delproject.json   = scrapyd.webservice.DeleteProject
delversion.json   = scrapyd.webservice.DeleteVersion
listjobs.json     = scrapyd.webservice.ListJobs
daemonstatus.json = scrapyd.webservice.DaemonStatus
```

```
1  #!/usr/bin/env bash
2  set -m
3  scrapyd > /dev/null 2>&1 &
4  sleep 5
5  scrapyd-deploy
6  fg %1
```

Start Server

Deploy Project
(Spider)

```
19  from scrapyd_api import ScrapydAPI
20
21  def chunks(l, n):
22      """Yield successive n-sized chunks from l."""
23      for i in range(0, len(l), n):
24          yield l[i:i + n]
25
26
27  scrapyd_host = 'http://localhost:6800'
28  project_name = 'industry_odsc_crawler'
29  spider_name = 'industry_v4_spider'
30  scrapyd = ScrapydAPI(scrapyd_host)
31
32  data_fn = 'big_url_list.csv'
33  data_fp = os.path.join(os.path.dirname(__file__),
34                         'data', data_fn)
35  data_df = pd.read_csv(data_fp)
36
37  url_id_chunks = chunks(data_df['url_id'], 100)
38  url_id_chunks = list(enumerate(url_id_chunks))
39  total_num_chunks = len(url_id_chunks)
```

Import ScrapyD API constructor

Specify ScrapyD host and Scrapy spider

Load big list of URLs

Split into chunks of 100

```
41  max_running = 10

43  max_pending = 5

45  sleep_interval = 15

47  while len(url_id_chunks) > 0:
48      server_status = scrapyd.list_jobs(project_name)
49
50      num_pending = len(server_status['pending'])
51      num_running = len(server_status['running'])
52
53      if (num_pending > max_pending) or (num_running > max_running):...
62
63      i, url_id_chunk = url_id_chunks.pop(0)
64      urls = data_df.loc[data_df['url_id'].isin(url_id_chunk), 'url'].tolist()
65
66      ids = ','.join([str(x) for x in url_id_chunk])
67      urls = ','.join(urls)
68
69      res = scrapyd.schedule(project_name, spider_name,
70                             ids=ids, urls=urls)
71      logger.info("added job {} (chunk num: {} / {})".format(
72          res, i, total_num_chunks))
```

Max # concurrent batches

Max # batches in queue

Server polling rate

Check which jobs, running, pending, etc

If bounds exceeded, wait

If slot open, schedule a batch

Model - Design (BERT)

BERT (grossly simplified):

- Can turn string of text into a vector of length 768 (or 1024)
- Max 512 tokens allowed
- Open source pre-trained models available for transfer learning (cased, uncased, base, large)
- For PyTorch: **https://github.com/huggingface/transformers**

Model - Design (BERT)

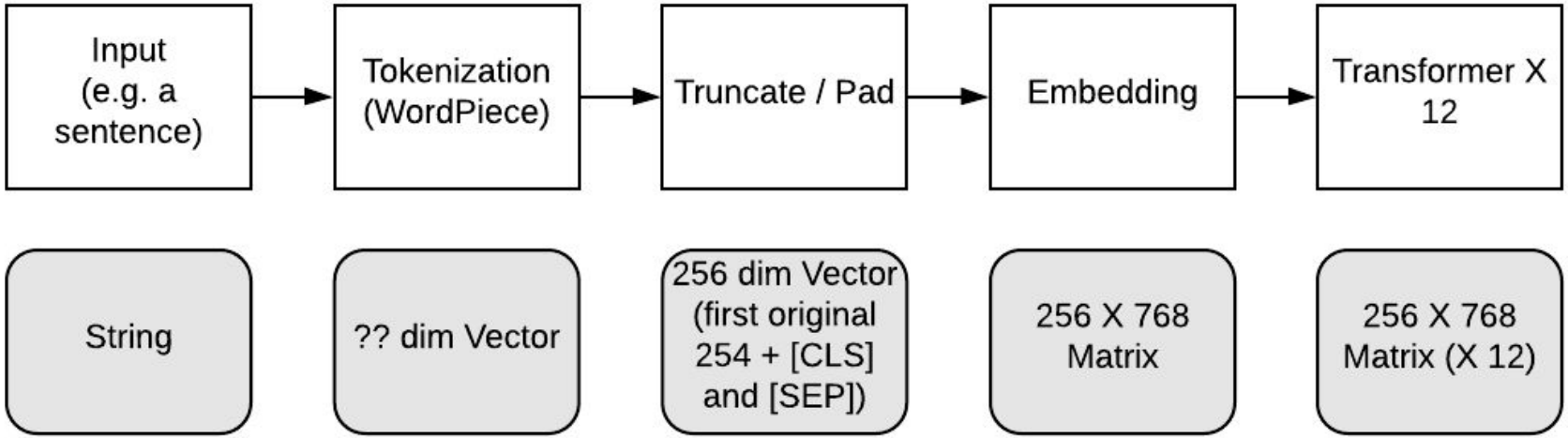
BERT (grossly simplified):

- Can turn string of text into a vector of length 768 (or 1024)
- Max 512 tokens allowed
- Open source pre-trained models available for transfer learning (cased, uncased, base, large)
- For PyTorch: **https://github.com/huggingface/transformers**

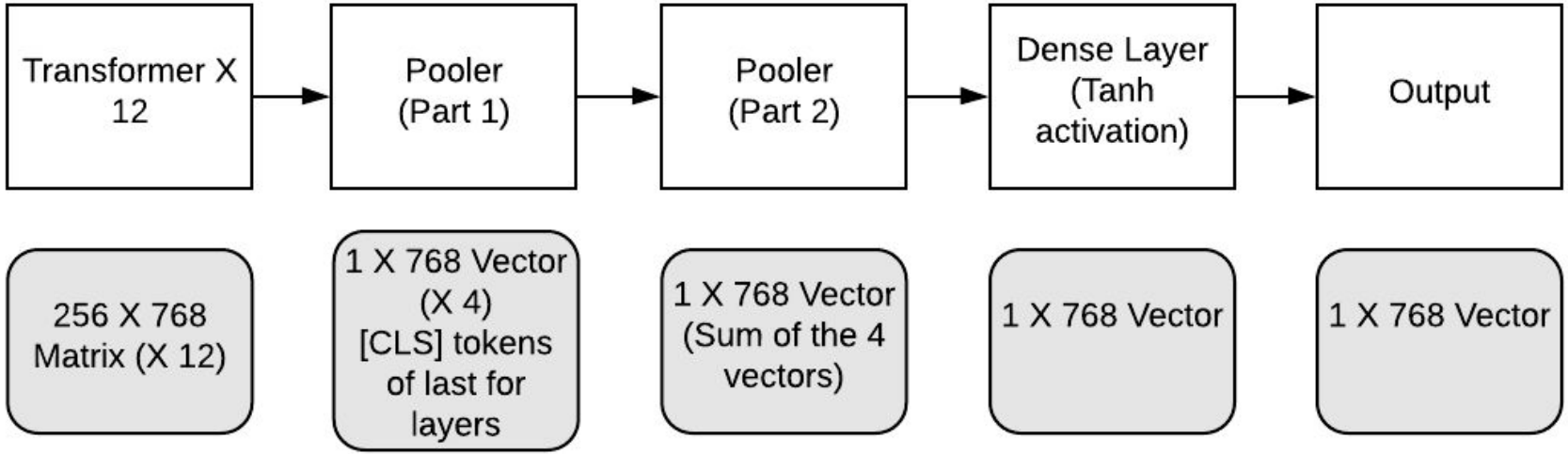
A bit more context:

- Uses WordPiece tokenization to split words into tokens
- Uses Transformers to perform “attention”, i.e: consider same token in different contexts
- Bi directional
- Achieves SOTA performance on benchmark NLP tasks

Model - Design (BERT)



Model - Design (BERT)



Model - Design

The basic idea:

- A single observation is a pair of:

  Business name & Website
- Organize data for every obs into a clear set of inputs
- Use BERT to convert all free text into numeric vectors
- Wrap this into a single model to classify our 135 industries

Model - Design

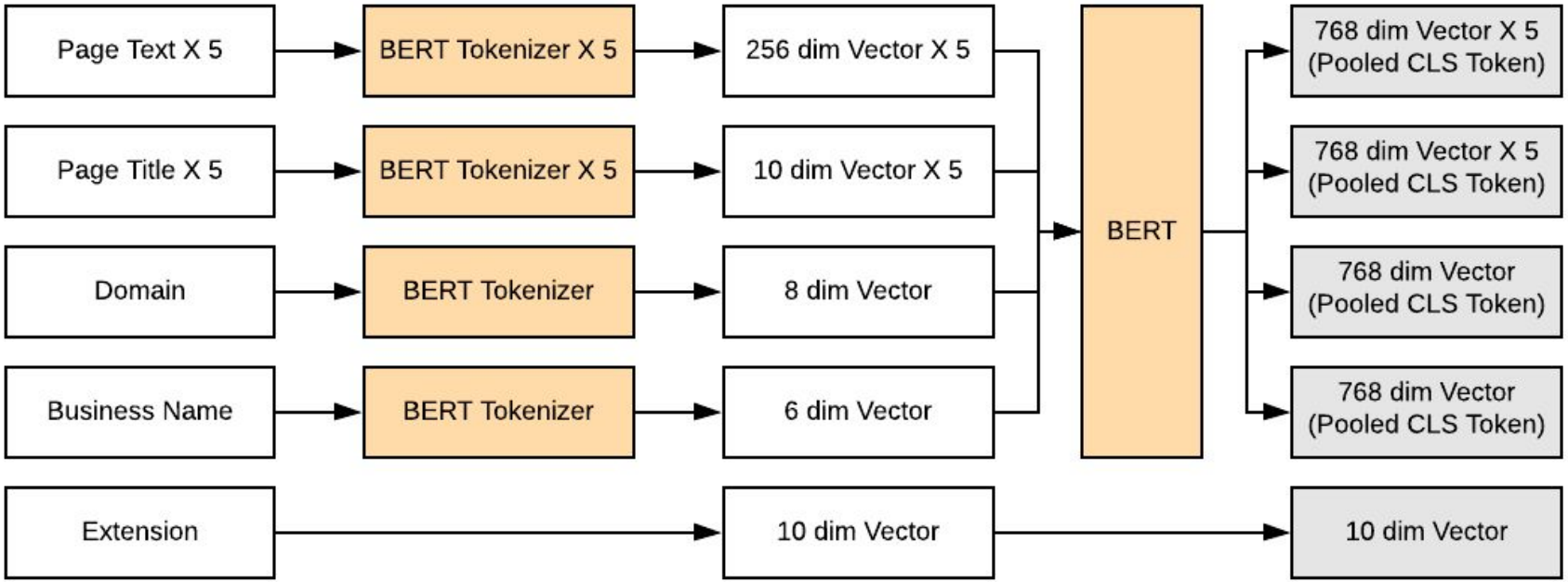
The basic idea:

- A single observation is a pair of: Business name & Website
- Organize data for every obs into a clear set of inputs
- Use BERT to convert all free text into numeric vectors
- Wrap this into a single model to classify our 135 industries

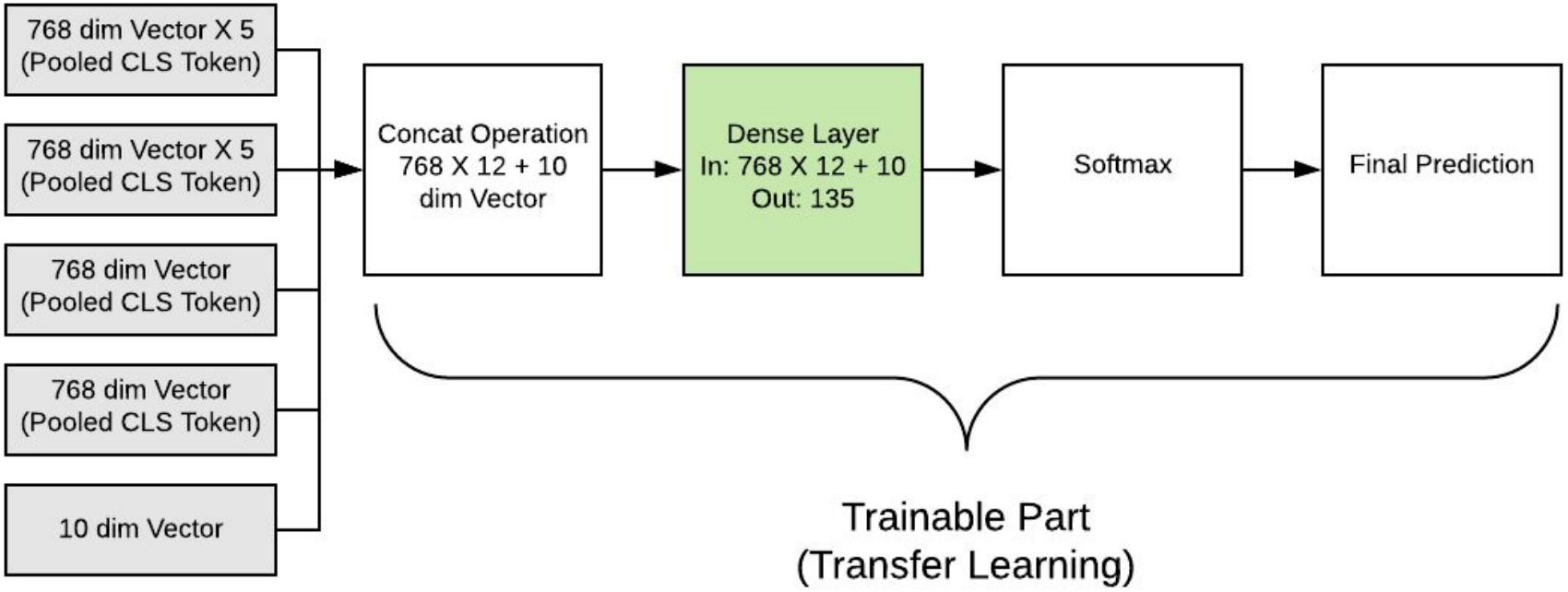
Implementation (moderate complexity)

- Use first 5 links per domain
- Free text:
  - Page Text
  - Page Title
  - Website Domain
- Metadata: Domain extension
- Other: Business Name

Model - Design



Model - Design



Model - Train

Tool:

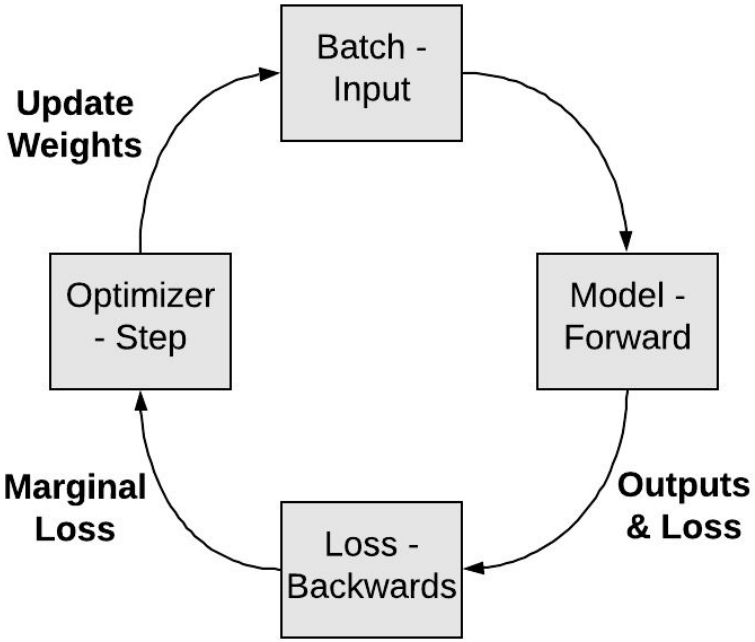
- PyTorch

  https://github.com/pytorch/pytorch
- Basically NumPy + GPUs
- Autograd
- Relatively easy to debug

Model - Train

NN / PyTorch Train Cycle:

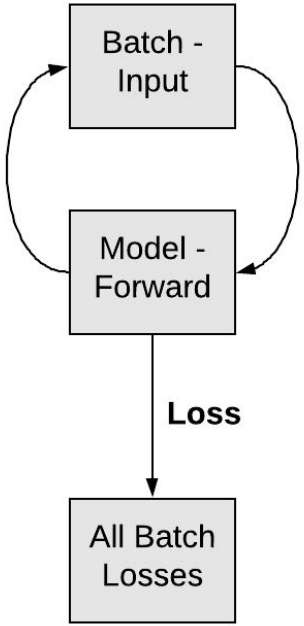
1. Select input batch (e.g. size 16)
2. Calc current outputs (forward)
3. Calc loss: Outputs VS labels
4. Calc marginal loss (backwards)
5. Update weights (step)
6. Again! Until batches / epoch done.



Model - Train

NN / PyTorch Validation Cycle:

1. Select input batch (e.g. size 16)
2. Calc current outputs (forward)
3. Calc loss: Outputs VS labels
4. Collect loss for all batches
5. Again! Until batches / epoch done
6. Report avg loss



Model - Train

Organizing the dataset:

- Convert all text elements into numeric vectors
- Convert everything to tensors
- Put everything together such that:
    - One “item” has all inputs per obs
    - Can be loaded in batches (GPU memory limits)

```
12  def make_dataset(input_df, bert_tokenizer):
13      """transform crawl data input into model readable tensor dataset..."""
20      # split input data into separate pages
21      num_pages = 5
22      data_pages = split_data_into_pages(input_df.copy(), num_pages)
23
24      # generate dummies for top 10 url extensions
25      top_extensions = ['biz', 'co', 'com', 'edu', 'gov',
26                        'info', 'net', 'org', 'site', 'us']
27      url_ext_mat = gen_url_extension_dummies(data_pages[0],
28                                              top_extensions)
29
30      # extract domain from url
31      data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(
32          extract_domain_from_url)
```

DataFrame per web page data

```
34  # tokenize & truncate all input fields
35  max_len = 256
36
37  page_text_tokenized = tokenize_and_truncate_sequence(
38      data_pages,
39      'response_text',
40      max_len,
41      bert_tokenizer)
42
43  page_text_masks = gen_sequence_mask(page_text_tokenized)
```

Max num of tokens

Field to tokenize

1 if token, 0 otherwise

```
64      # combine into tensor dataset
65      data_set = combine_inputs_into_tensor_dataset(
66          page_text_tokenized, page_text_masks, page_titles_tokenized,
67          page_title_masks, domain_tokenized, domain_masks, biz_name_tokenized,
68          biz_name_masks, url_ext_mat)
69
70      urls = data_pages[0]['url'].tolist()
71      return data_set, urls
```

Big collection of tensors, ordered!

```
1  import torch
2  from torch import nn
3  from torch.utils.data import DataLoader, RandomSampler, SequentialSampler
4  from tqdm import trange
5  import numpy as np
6
7  from pytorch_transformers import BertTokenizer, AdamW
8
9  from industry_sagemaker_service.bert_custom_classifier import CustomBertClassifier
10 from industry_sagemaker_service.utils import make_dataset
```

Loader for BERT Tokenizer

Import the make_dataset method

```
12  bert_model_name = 'bert-base-cased'
13  bert_tokenizer = BertTokenizer.from_pretrained(bert_model_name,
14                                                 do_lower_case=False)
15
16  train_raw_data_df = ''  # train crawl data + business names
17  train_data = make_dataset(train_raw_data_df, bert_tokenizer)
18
19  test_raw_data_df = ''  # test crawl data + business names
20  test_data = make_dataset(test_raw_data_df, bert_tokenizer)
21
22  batch_size = 16
23
24  train_sampler = RandomSampler(train_data)
25  train_dataloader = DataLoader(train_data, sampler=train_sampler,
26                                batch_size=batch_size)
27
28  test_sampler = SequentialSampler(test_data)
29  test_dataloader = DataLoader(test_data, sampler=test_sampler,
30                               batch_size=batch_size)
```

Load pretrained BERT Tokenizer

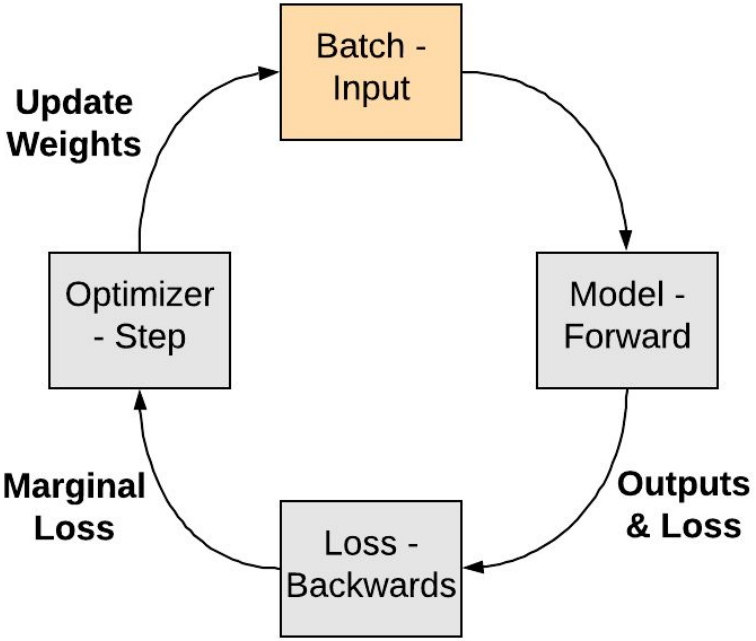
Make Train and Test datasets

Make Train and Test dataloaders

Model - Train

Batch - Input:

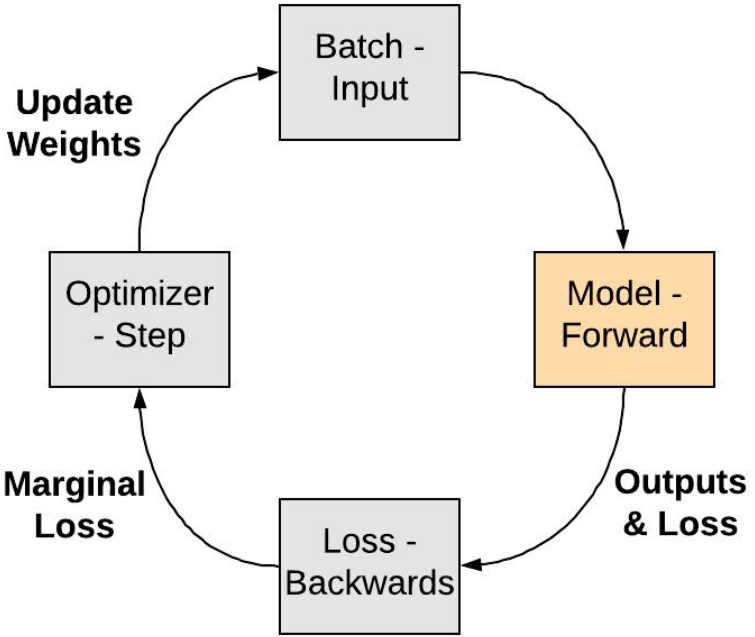
- Create Train Dataloader
- Extract Batch



Model - Train

Forward method:

- Define the graph of the model:
  - Layers
  - Transformations
  - Combinations
- Is invoked with inputs, calculates outputs
- Using labels and a loss metric, calculates loss



```python
from pytorch_transformers import BertModel, BertPreTrainedModel
from torch.nn import CrossEntropyLoss, MSELoss
from torch import nn
import torch


class CustomBertClassifier(BertPreTrainedModel):
    def __init__(self, config):
        super(CustomBertClassifier, self).__init__(config)
        self.num_labels = config.num_labels
        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size * 12 + 10,
                                    self.config.num_labels)

        self.init_weights()
```

Pre-Trained BERT

Trainable Linear Layer

```
18    def forward(self,
19                sentence_ids, sentence_masks,
20                title_ids, title_masks,
21                domain_ids, domain_masks,
22                biz_name_ids, biz_name_masks,
23                url_exts,
24                token_type_ids=None,
25                position_ids=None, head_mask=None,
26                labels=None):
27
28        # domain input
29        domain_outputs = self.bert(
30            domain_ids,
31            attention_mask=domain_masks,
32            token_type_ids=token_type_ids,
33            position_ids=position_ids,
34            head_mask=head_mask)
35
36        domain_pooled_output = domain_outputs[1]
```

All inputs: Tokens + masks & dummies

Domain Tokens + masks fed into BERT

Domain Embedding Vector Obtained

```
48      # page text inputs
49      page_text_outputs = []
50      for page_text_id, page_text_mask in \
51              zip(page_text_ids, page_text_masks):
52          page_text_output_tmp = self.bert(
53              page_text_id,
54              attention_mask=page_text_mask,
55              token_type_ids=token_type_ids,
56              position_ids=position_ids,
57              head_mask=head_mask)
58
59          page_text_pooled_output_tmp = page_text_output_tmp[1]
60          page_text_outputs += [page_text_pooled_output_tmp]
```

Iterating over all 5 page texts

Page texts Tokens + masks fed into BERT

Page Text embeddings Obtained

```
76
77      # combine all outputs
78      outputs_all = page_text_outputs + page_title_outputs + \
79                    [domain_pooled_output, biz_name_pooled_output, url_exts]
80      pooled_output = torch.cat(outputs_all, dim=1)
81      pooled_output = self.dropout(pooled_output)
82
83      # apply classifier
        logits = self.classifier(pooled_output)
```

All vectors concatenated into a big vector

Big vector sent through the trainable Linear layer

```
85          # add hidden states and attention if they are here
86          outputs = (logits,)
87
88          if labels is not None:
89              if self.num_labels == 1:
90                  #  We are doing regression
91                  loss_fct = MSELoss()
92                  loss = loss_fct(logits.view(-1), labels.view(-1))
93              else:
94                  loss_fct = CrossEntropyLoss()
95                  loss = loss_fct(logits.view(-1, self.num_labels),
96                                  labels.view(-1))
97
98              outputs = (loss,) + outputs
99          return outputs
```

In Train mode:
Loss = Batch output VS Labels

Return forward result
In Train mode: also return loss

```
33  from industry_sagemaker_service.bert_custom_classifier \
34      import CustomBertClassifier
35
36  custom_bert_mdl = CustomBertClassifier.from_pretrained(bert_model_name,
37                                                         num_labels=135)
38
39  if torch.cuda.device_count() > 1:
40      print("Let's use", torch.cuda.device_count(), "GPUs!")
41      custom_bert_mdl = nn.DataParallel(custom_bert_mdl)
42
43  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
44  custom_bert_mdl.to(device)
```

Import and init the custom classifier

Setup for multiple GPUs

```
train_loss_set = []
epochs = 3

for _ in trange(epochs, desc="Epoch"):

    custom_bert_mdl.train()

    tr_loss, tr_accuracy = 0, 0
    nb_tr_examples, nb_tr_steps = 0, 0

    for step, batch in enumerate(train_dataloader):

        # Add batch to GPU
        batch = tuple(t.to(device) for t in batch)

        # Unpack the inputs from our dataloader
        page_text_ids1, page_text_ids2, page_text_ids3, \
        page_text_ids4, page_text_ids5, \
            page_text_masks1, page_text_masks2, page_text_masks3, \
            page_text_masks4, page_text_masks5, \
            page_title_ids1, page_title_ids2, page_title_ids3, \
            page_title_ids4, page_title_ids5, \
            page_title_masks1, page_title_masks2, page_title_masks3, \
            page_title_masks4, page_title_masks5, \
            domain_ids, domain_masks, biz_name_ids, biz_name_masks, \
            url_exts, batch_labels = batch
```

Allow model to update its weights

Iterate over all batches in train dataloader

Unpack all inputs from batch

Loss & Outputs calculated

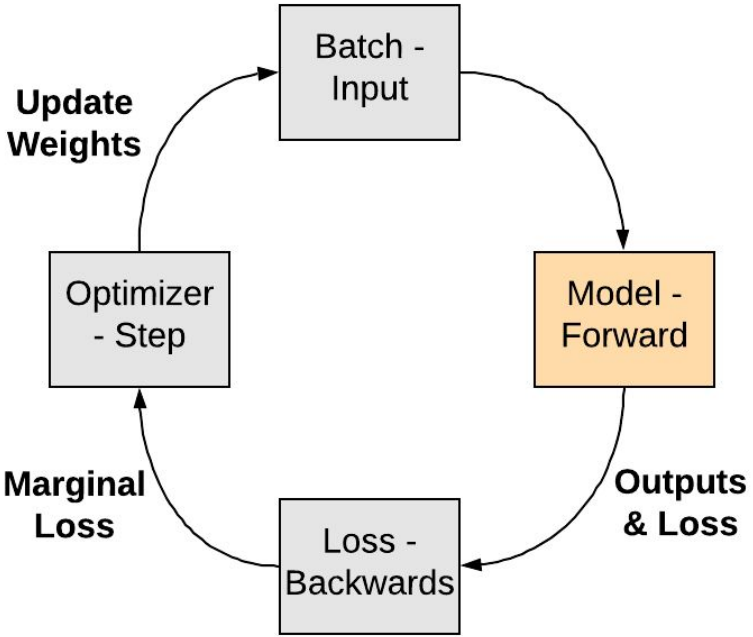
All inputs sent to forward method

```
96    # Clear out the gradients (by default they accumulate)
97    optimizer.zero_grad()
98
99    # Forward pass
100   loss, logits = custom_bert_mdl(
101       sentence_ids=(page_text_ids1, page_text_ids2, page_text_ids3,
102                     page_text_ids4, page_text_ids5),
103       sentence_masks=(page_text_masks1, page_text_masks2, page_text_masks3,
104                       page_text_masks4, page_text_masks5),
105       title_ids=(page_title_ids1, page_title_ids2, page_title_ids3,
106                  page_title_ids4, page_title_ids5),
107       title_masks=(page_title_masks1, page_title_masks2, page_title_masks3,
108                    page_title_masks4, page_title_masks5),
109       domain_ids=domain_ids,
110       domain_masks=domain_masks,
111       biz_name_ids=biz_name_ids,
112       biz_name_masks=biz_name_masks,
113       url_exts=url_exts,
114       token_type_ids=None,
115       labels=batch_labels)
```

Model - Train

Forward method:

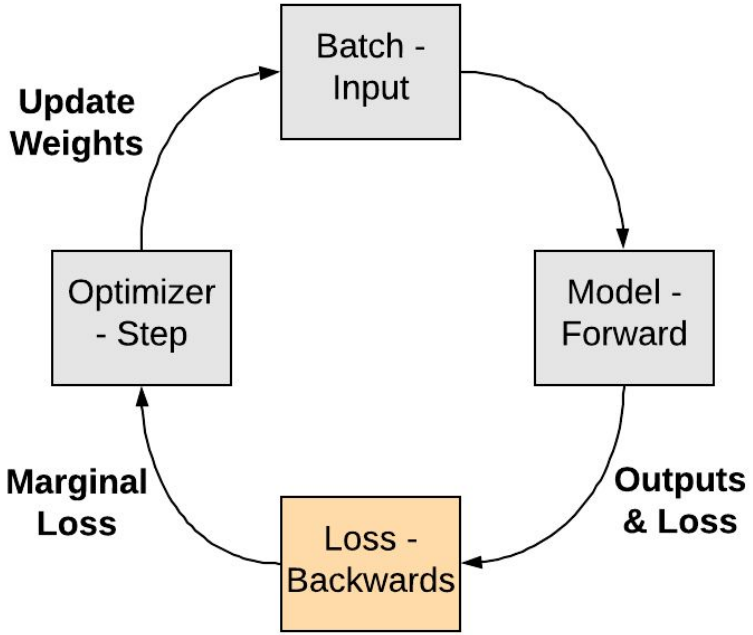
- Define the graph of the model:
  - Layers
  - Transformations
  - Combinations
- Is invoked with inputs, calculates outputs
- Using labels and a loss metric, calculates loss



Model - Train

Loss metric:

- Compares outputs with labels, calculates actual loss
- “backward” method computes the marginal loss w.r.t all the trainable parameters (i.e the loss gradient). Updates “grad” attribute of tensors.
- E.g: Cross Entropy Loss for multi-class problems



```
96      # Clear out the gradients (by default they accumulate)
97      optimizer.zero_grad()
98
99      # Forward pass
100     loss, logits = custom_bert_mdl(
101         sentence_ids=(...),
103         sentence_masks=(...),
105         title_ids=(...),
107         title_masks=(...),
109         domain_ids=domain_ids,
110         domain_masks=domain_masks,
111         biz_name_ids=biz_name_ids,
112         biz_name_masks=biz_name_masks,
113         url_exts=url_exts,
114         token_type_ids=None,
115         labels=batch_labels)
116
117
118     train_loss_set.append(loss.mean().item())
119
120     loss.mean().backward()
```

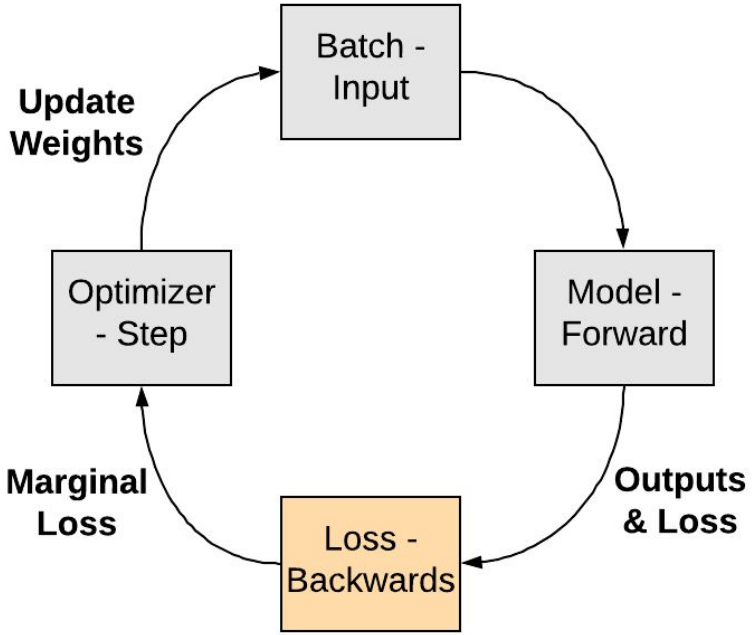
Aggregate loss from multiple GPUs

Calculate Marginal Loss

Model - Train

Loss metric:

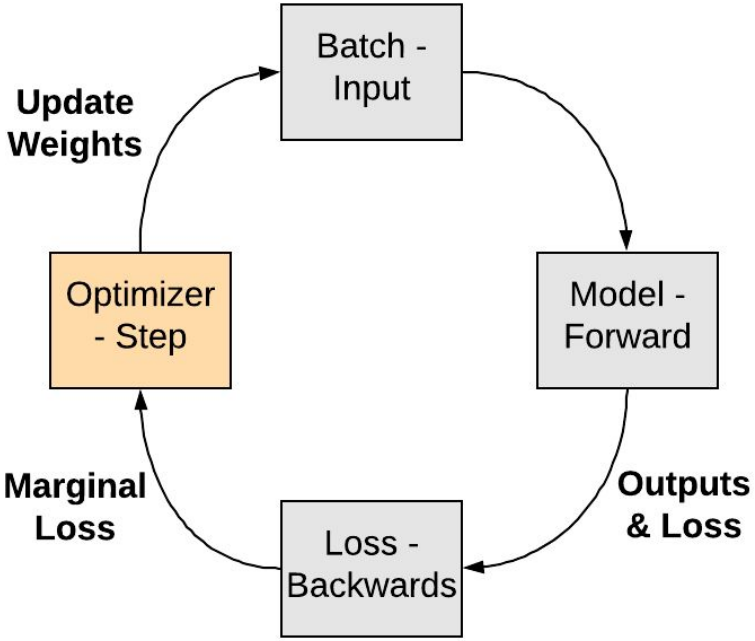
- Compares outputs with labels, calculates actual loss
- “backward” method computes the marginal loss w.r.t all the trainable parameters (i.e the loss gradient). Updates “grad” attribute of tensors.
- E.g: Cross Entropy Loss for multi-class problems



Model - Train

Optimizer:

- Specifies which gradient-based optimization method to use for gradient descent
- “step” method updates all relevant weights (i.e. tensor values) according to the marginal loss (stored in the grad attribute of the tensors)
- E.g: AdamW Optimizer



```
param_optimizer = list(custom_bert_mdl.named_parameters())
no_decay = ['bias', 'gamma', 'beta']
optimizer_grouped_parameters = [
    {'params': [p for n, p in param_optimizer if
                not any(nd in n for nd in no_decay)],
     'weight_decay_rate': 0.01},
    {'params': [p for n, p in param_optimizer if
                any(nd in n for nd in no_decay)],
     'weight_decay_rate': 0.0}
]

optimizer = AdamW(optimizer_grouped_parameters, lr=2e-5)
```

```
96      # Clear out the gradients (by default they accumulate)
97      optimizer.zero_grad()
98
99      # Forward pass
100     loss, logits = custom_bert_mdl(
101         sentence_ids=(...),
103         sentence_masks=(...),
105         title_ids=(...),
107         title_masks=(...),
109         domain_ids=domain_ids,
110         domain_masks=domain_masks,
111         biz_name_ids=biz_name_ids,
112         biz_name_masks=biz_name_masks,
113         url_exts=url_exts,
114         token_type_ids=None,
115         labels=batch_labels)
116
117
118     train_loss_set.append(loss.mean().item())
119
120     loss.mean().backward()
121
122     optimizer.step()
```

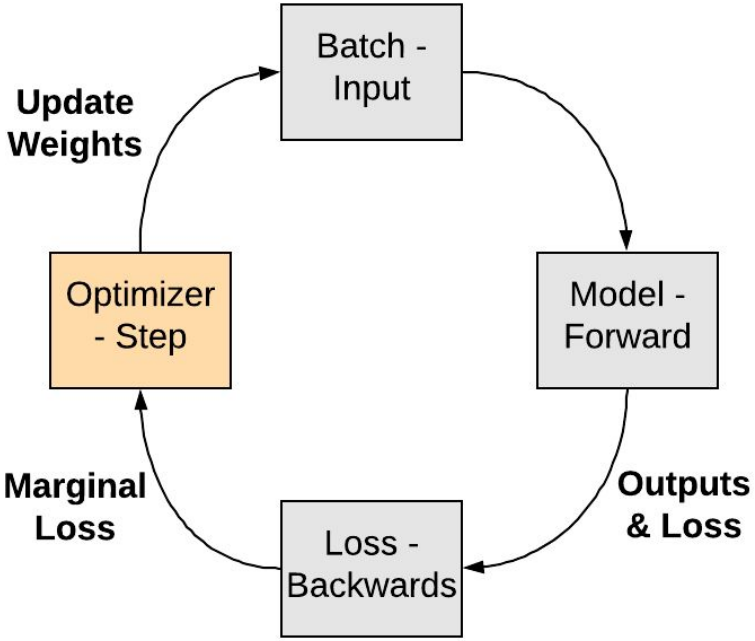
Clear gradients from previous batches

Update weights based on marginal loss

Model - Train

Optimizer:

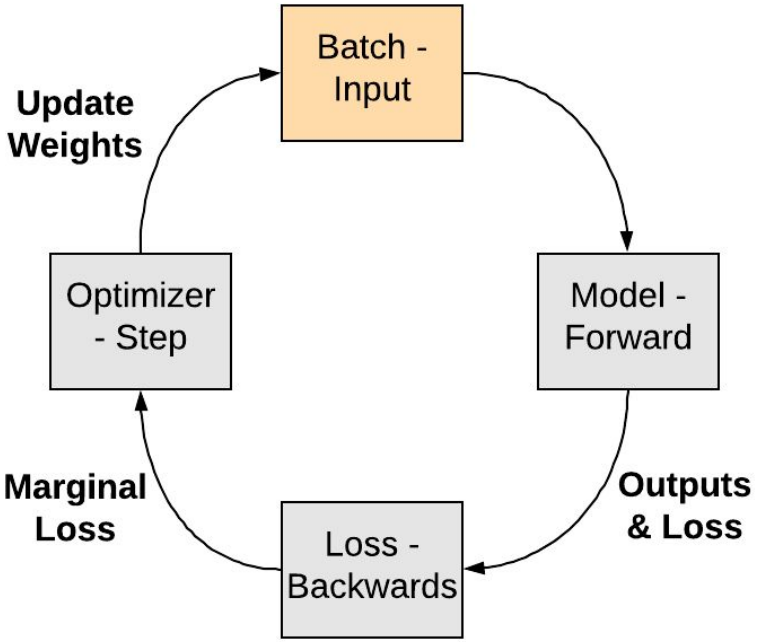
- Specifies which gradient-based optimization method to use for gradient descent
- “step” method updates all relevant weights (i.e. tensor values) according to the marginal loss (stored in the grad attribute of the tensors)
- E.g: AdamW Optimizer



Model - Train

Batch - Input:

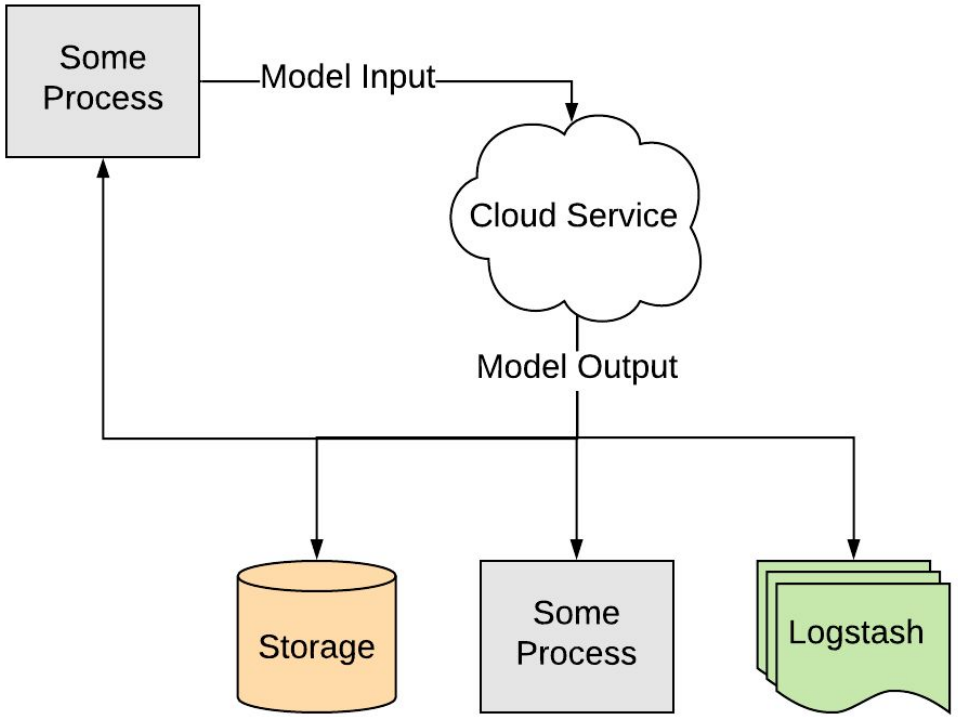
- Next Batch



Model - Deploy

Goal:

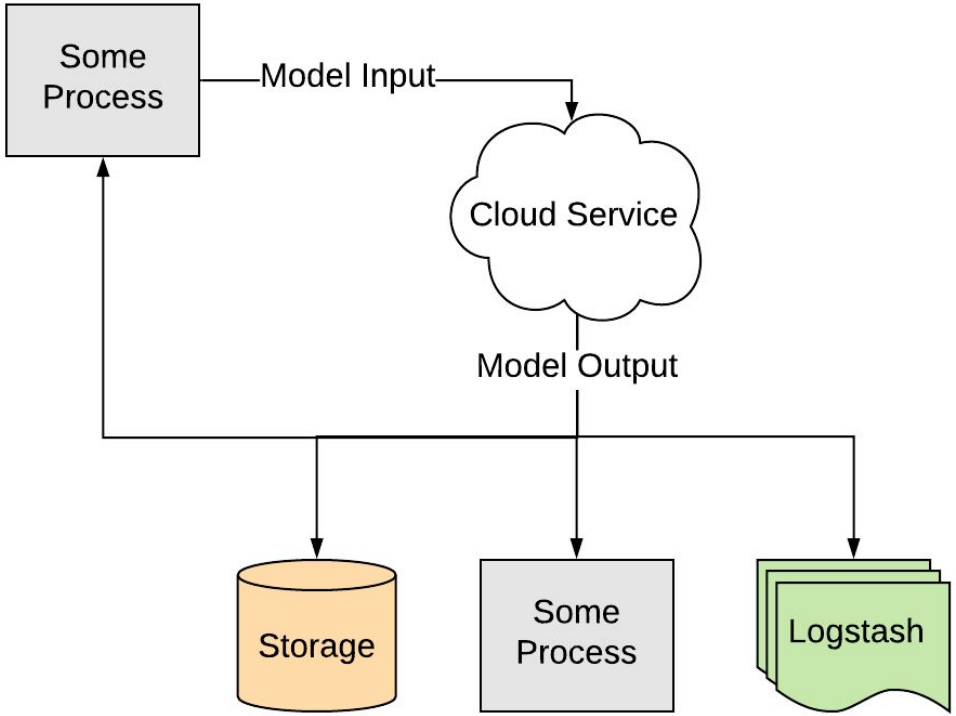
- Model hosted in the cloud
- Always up (persistence)
- Communicates via API
- Can scale



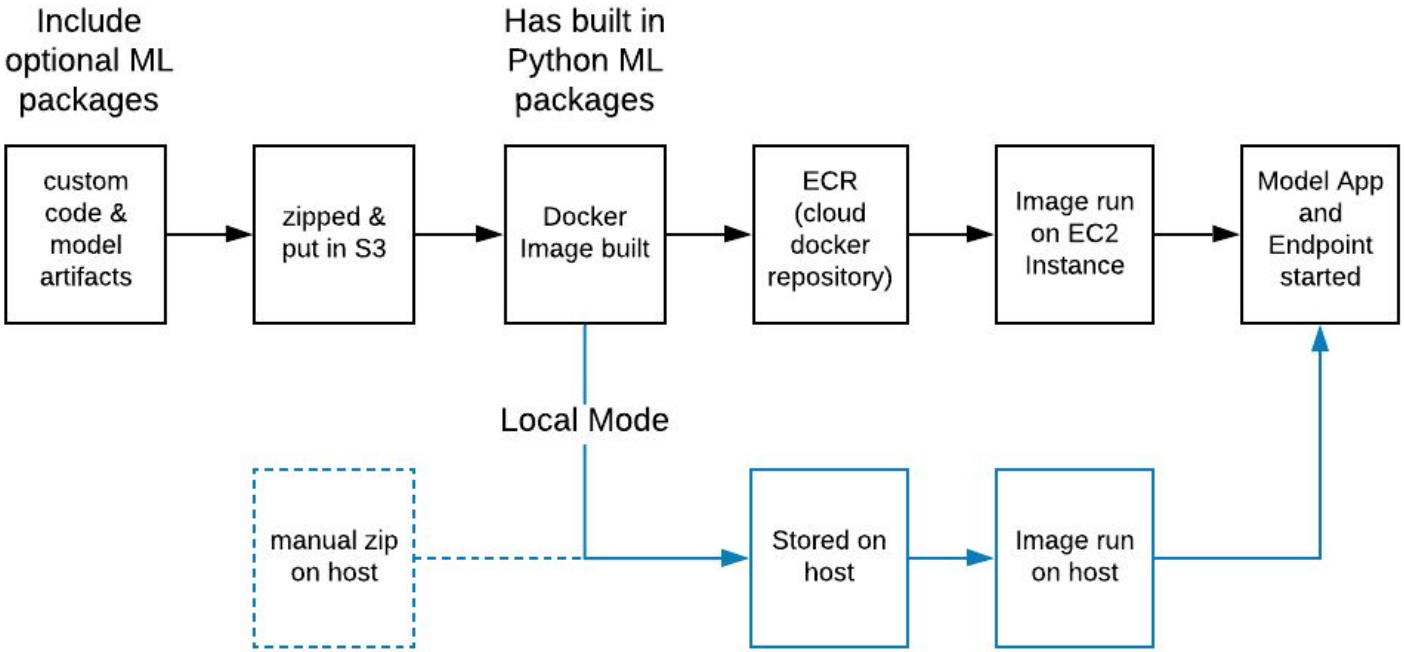
Model - Deploy

Tool:

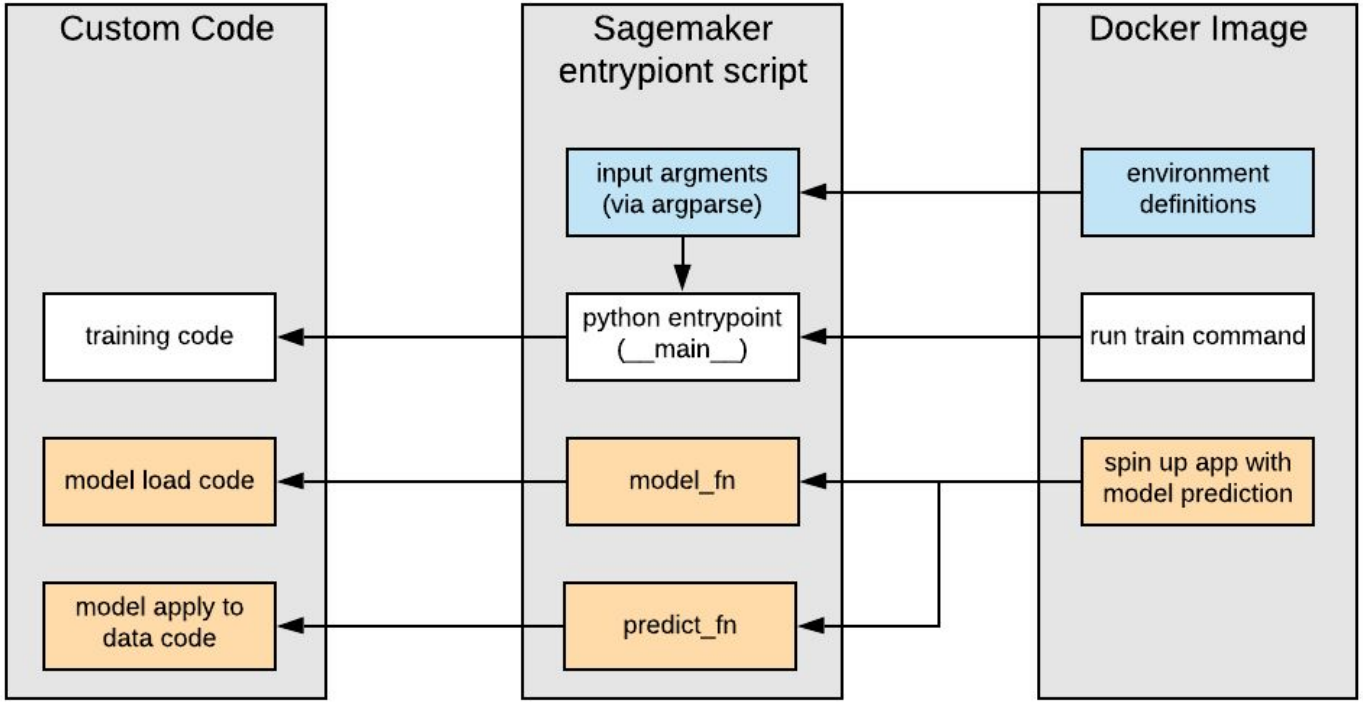
- Amazon SageMaker https://aws.amazon.com/sagemaker/
- All the pros of docker deployment without writing any docker code
- Supports four modes: Tag, Explore, Train and Deploy
- Auto Scaling



Model - Deploy (SageMaker)



Model - Deploy (SageMaker)



```
def model_fn(model_dir):
    """
    initialize resources during service startup
    """
    # load model
    from industry_sagemaker_service.bert_custom_classifier \
        import CustomBertClassifier
    bert_mdl = CustomBertClassifier.from_pretrained(
        os.path.join(model_dir, 'data_files', 'model_e'))

    # set up for GPU if exists
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    if device.type == 'cuda':
        bert_mdl.cuda()

    # load tokenizer
    bert_tokenizer = BertTokenizer.from_pretrained(
        'bert-base-cased', do_lower_case=False)

    # load industry map, int --> industry name
    industry_map_df = pd.read_csv(os.path.join(
        model_dir, 'data_files', 'industry_map.csv'))
    industry_map_dict = industry_map_df.\
        set_index('industry_int').to_dict()['industry_tag']

    return bert_mdl, bert_tokenizer, industry_map_dict
```

Load trained model

Load pertained tokenizer

Return all artifacts

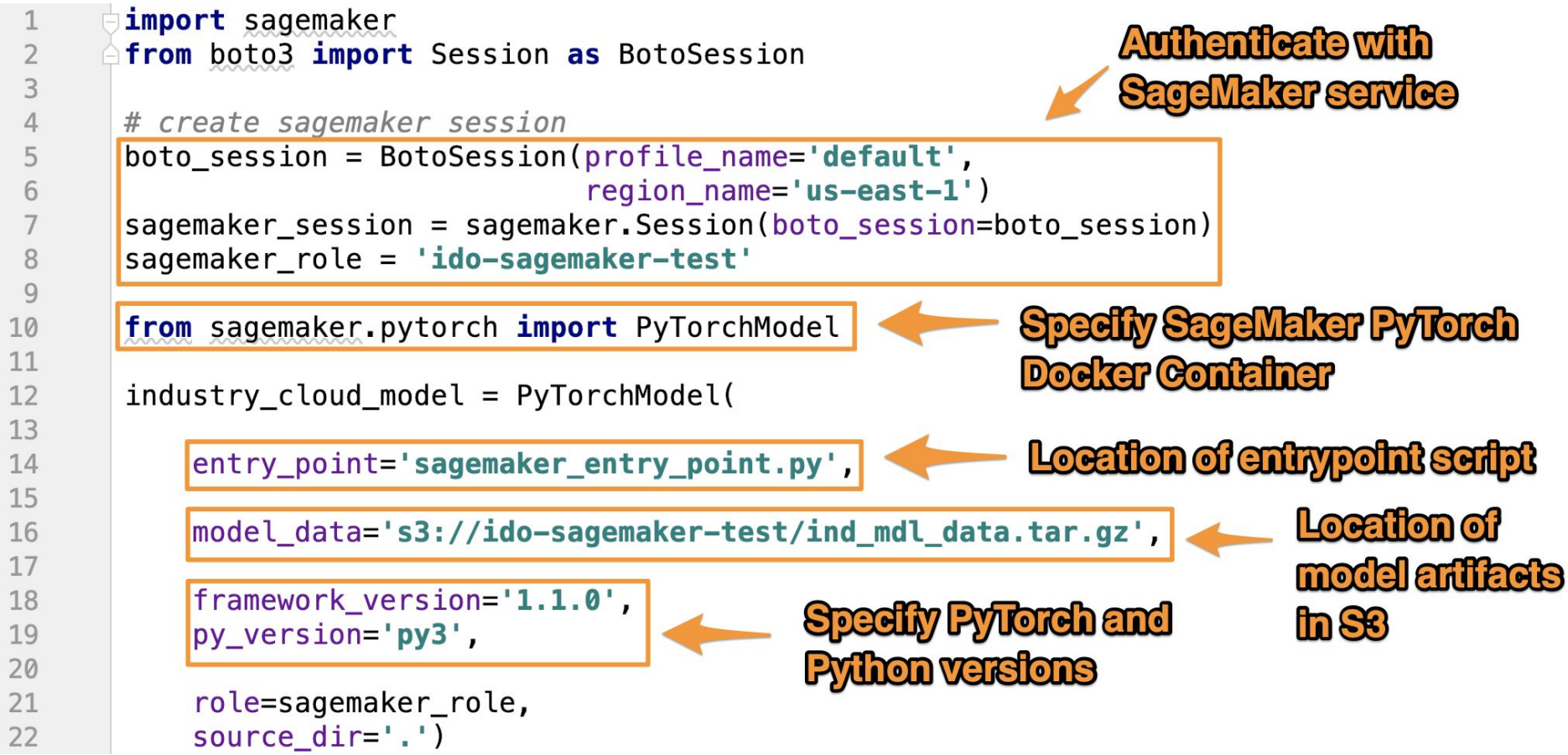
```python
def predict_fn(crawl_df, model_fn_input):
    """
    apply model to input, return inference
    """
    bert_mdl, bert_tokenizer, industry_map_dict = model_fn_input

    from industry_sagemaker_service.utils import make_dataset, predict_industry
    data_set, urls = make_dataset(crawl_df, bert_tokenizer)
    data_sampler = SequentialSampler(data_set)
    data_loader = DataLoader(data_set, sampler=data_sampler, batch_size=8)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    preds = []
    probas = []
    for step, batch in enumerate(data_loader):...

    preds_as_str = [[industry_map_dict[y] for y in x] for x in preds]
    probas_rounded = [[np.round(y, 4) for y in x] for x in probas]
    output = [{k: v for (k, v) in zip(x, y)} for (x, y) in
              zip(preds_as_str, probas_rounded)]
    output_sorted = [sorted(d.items(), key=operator.itemgetter(1),
                            reverse=True) for d in output]

    assert len(urls) == len(output_sorted)
    output_with_urls = {x: y for (x, y) in zip(urls, output_sorted)}

    return json.dumps(output_with_urls)
```

Get preloaded model artifacts

Turn input into batch dataloader

Inference loop by batch

Sort preds in descending order


import sagemaker
from boto3 import Session as BotoSession
# create sagemaker session
boto_session = BotoSession(profile_name='default',
region_name='us-east-1')
sagemaker_session = sagemaker.Session(boto_session=boto_session)
sagemaker_role = 'ido-sagemaker-test'
from sagemaker.pytorch import PyTorchModel
industry_cloud_model = PyTorchModel(
entry_point='sagemaker_entry_point.py',
model_data='s3://ido-sagemaker-test/ind_mdl_data.tar.gz',
framework_version='1.1.0',
py_version='py3',
role=sagemaker_role,
source_dir='.')
Authenticate with SageMaker service
Specify SageMaker PyTorch Docker Container
Location of entrypoint script
Location of model artifacts in S3
Specify PyTorch and Python versions

```
24  industry_cloud_predictor = industry_cloud_model.deploy(
25      initial_instance_count=1,
26      instance_type='ml.p2.xlarge')
27
28
29  from sagemaker.predictor import json_serializer
30
31  # define input format as JSON
32  industry_cloud_predictor.serializer = json_serializer
33  industry_cloud_predictor.content_type = 'application/json'
34
35  input_json = ''  # load some input data
36
37  preds = industry_cloud_predictor.predict(input_json)
```

Call the deploy method on the model

Specify EC2 Instance (GPU)

Call the predict method on the predictor

Amazon SageMaker > Endpoints

Endpoints

Update endpoint | Actions | Create endpoint

Search endpoints

1

Our Endpoint

Name	ARN	Creation time	Status	Last updated
sagemaker-pytorch-2019-10-30-03-50-29-502	[redacted]	Oct 30, 2019 03:50 UTC	InService	Oct 30, 2019 04:00 UTC
sagemaker-scikit-learn-2019-10-08-21-54-08-741	[redacted]	Oct 08, 2019 21:54 UTC	InService	Oct 25, 2019 19:34 UTC

Conclusion - Summary

What we did:

- Built a 250K sized sample of websites
  - Search strategies
  - Balancing strategies
- Tagged that sample
  - Crowdsourcing considerations & strategies
- Crawled all of the websites in it
  - Scrapy for small scale
  - ScrapyD for large scale

Conclusion - Summary

What we did:

- Built a 250K sized sample of websites
  - Search strategies
  - Balancing strategies
- Tagged that sample
  - Crowdsourcing considerations & strategies
- Crawled all of the websites in it
  - Scrapy for small scale
  - ScrapyD for large scale
- Built an NLP industry classification model
  - Transfer learning using BERT
  - Dealt with 512 length limit
- Trained that model with multiple GPUs
  - Pretrained BERT models with PyTorch
- Deployed that model using SageMaker
  - Docker deployment, no docker code

Conclusion - Key Takeaways

- Crowdsourcing has to be done carefully to produce real value
  - Vendor trials
  - Enforce QA
- You can build a very efficient crawler using Scrapy / ScrapyD
- You can build a very powerful NLP model using pretrained BERT
  - Use available wrappers (e.g HuggingFace for PyTorch)
- You can train / deploy models on very powerful machines (e.g. 16 GPUs) using SageMaker
  - No docker knowledge required
- Hope for the best

Questions? + Thanks!

ido.shlomo@bluevine.com