



**Ido Shlomo**

Senior Data Science Manager  
BlueVine

VIRTUAL CONFERENCE

# Building an Industry Classifier With The Latest Scraping, NLP and Deployment Tools

SEPTEMBER 17-19

# About BlueVine

- Fintech startup up based in Redwood City, CA
- Provides working capital (loans) to small & medium sized businesses
- New Product: Business checking accounts
- Over \$3.5 BN in loans funded to date
- Mission of DS team:
  - Automate
  - Scale
  - Improve

The image shows a screenshot of the BlueVine website. At the top left is the BlueVine logo. To its right is a menu icon consisting of three horizontal lines. The main headline reads "Fast funding for your business." Below it is a blue button with the text "Apply now". A small note below the button states "Applying will not affect your credit score<sup>1</sup>". Below this section is a photograph of a smiling man wearing a light-colored shirt and dark overalls. At the bottom of the page is a blue banner with white text that says "Business Loans from \$5,000 - \$5 million".

# About BlueVine - COVID

- Leading participant the US Govt PPP program
- Automated decisions within minutes
- Over \$4.5 BN in PPP loans approved to date
- Gradual return to regular operations:
  - New risk models / tools
  - New fraud models / tools
  - New financial products

≡

SBA Coronavirus Assistance Program

## Paycheck Protection Program Loans & Forgiveness

BlueVine provided thousands of COVID-19 relief loans to help small businesses pay employees and cover monthly expenses.



# About Me

- Senior 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](https://github.com/ido-sh/odsc_presentations)

# Why do we care about Industry?

- **Compliance:**

We can't work with some industries: Weapons, gambling, other finance companies, etc

- **Portfolio management:**

Limit exposure to specific risky sectors, general balancing, etc.

- **Smarter decision making:**

For key metrics (e.g income, credit score), “normal” and “abnormal” differs across industries.

# Why do we care about Industry?

## **COVID response**

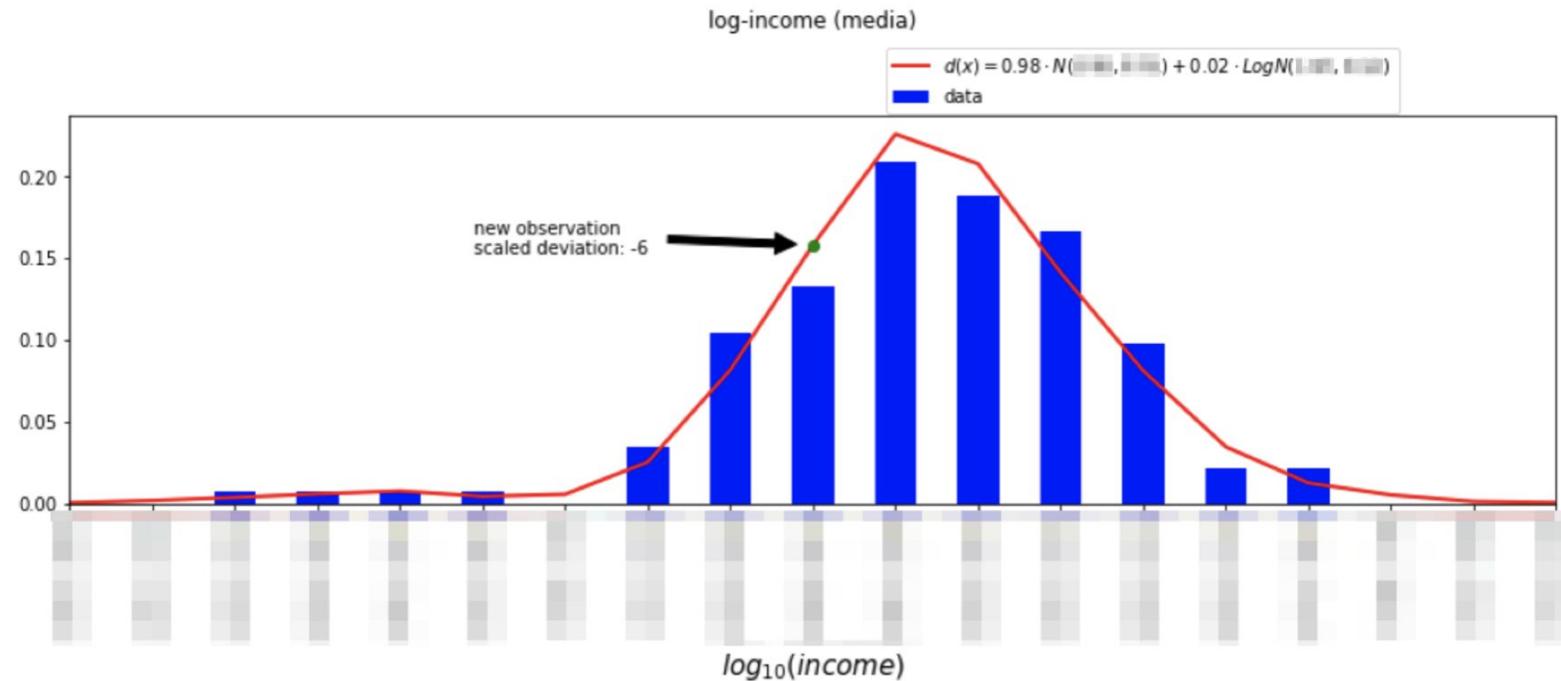
- Different industries were affected in different ways
- Need to track each separately and deal with each separately
- We used a dynamic list of Red / Orange / Green that updates regularly

# Why do we care about Industry?

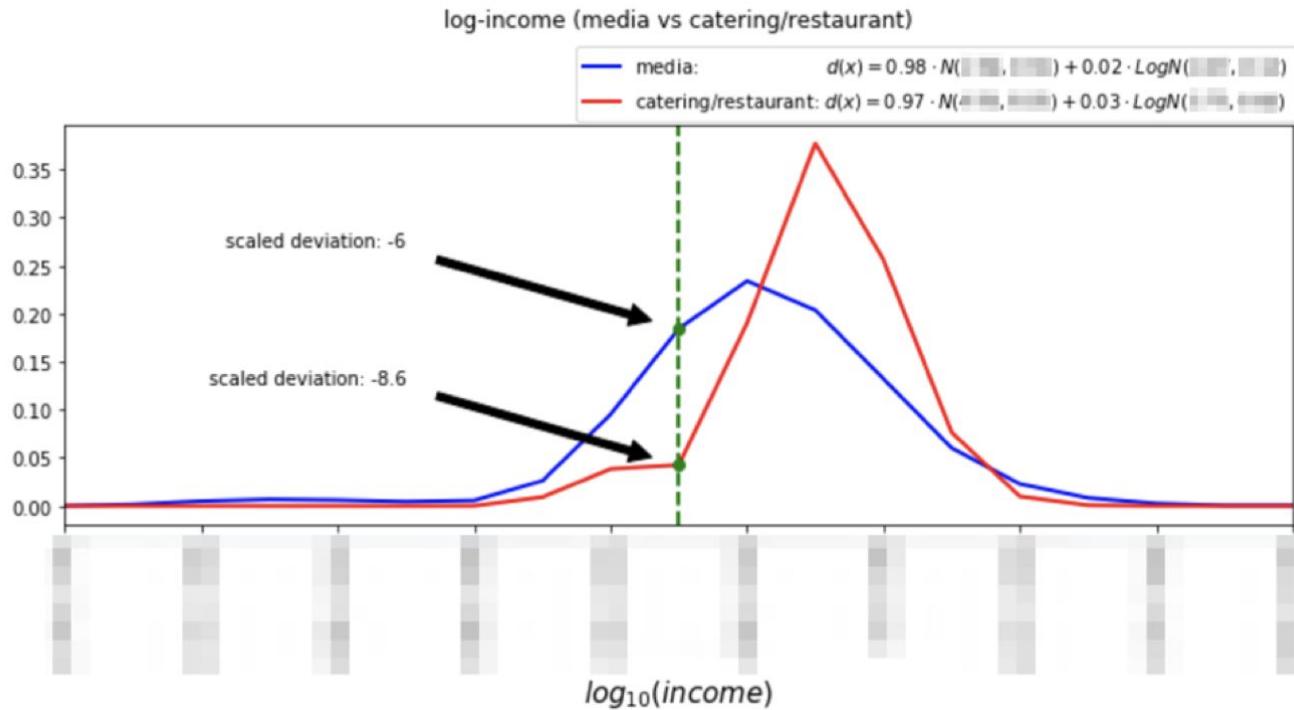
## Covid Industry Groups (examples)

- Red:
  - Restaurants
  - Barber Shop & Beauty Salons
- Orange:
  - Property Management Services
  - Photography Services
- Green:
  - Hardware Stores
  - Pharmacies

# Why do we care about Industry?

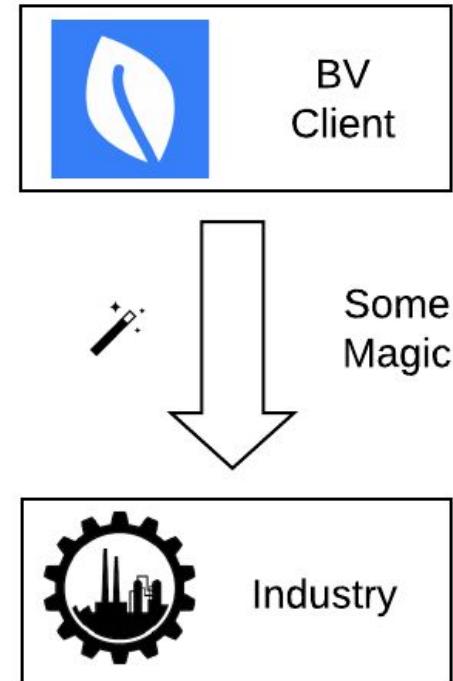


# Why do we care about Industry?



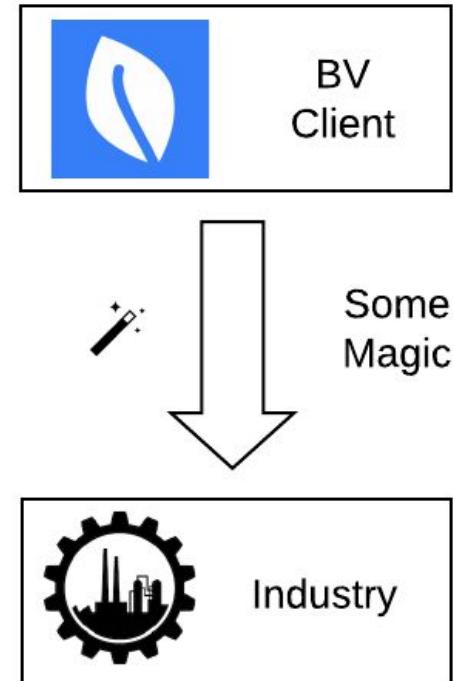
# The Point of this Presentation

- 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

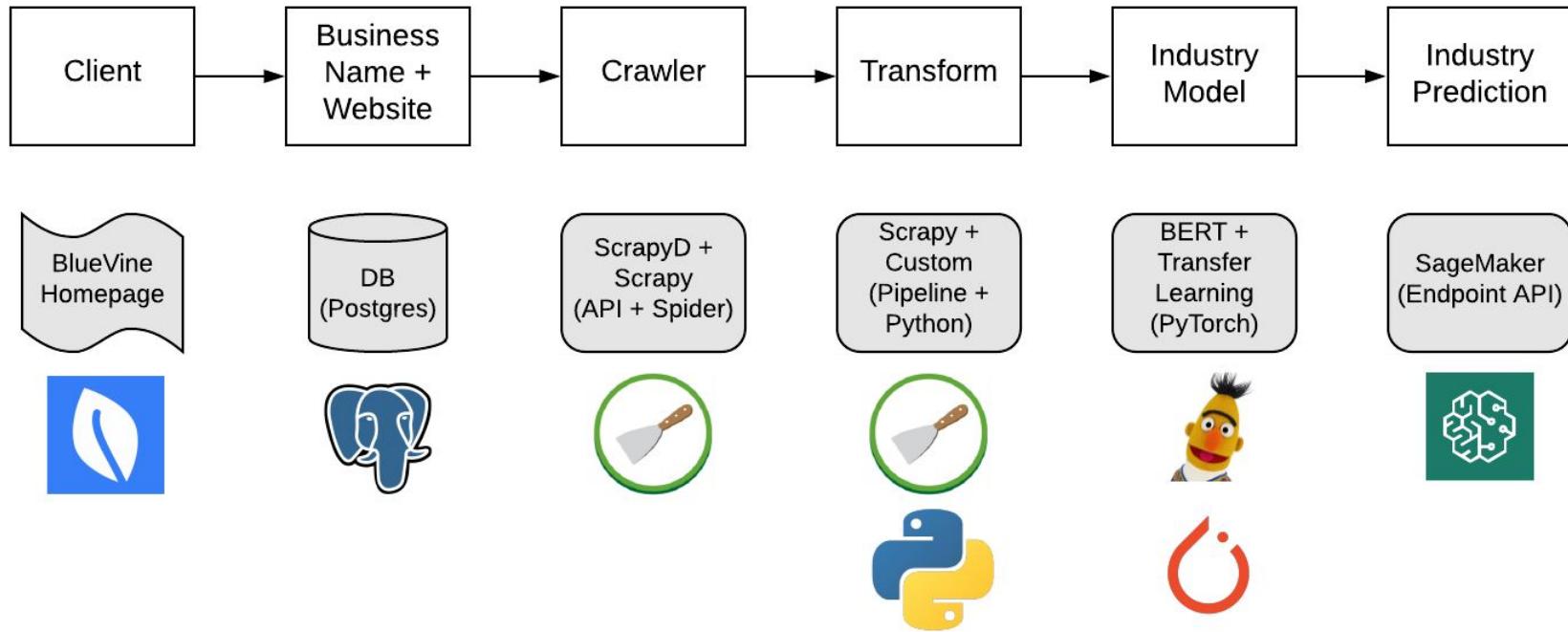


# The Point of this Presentation

- 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

## 3. Model

- Design (BERT)
- Train
- Deploy

## 2. Data

- Building the crawler
- Crawling at scale

## 4. Conclusion

- Summary
- Key takeaways

# Target - Gathering websites

Potential sources:

- Internal BV data
  - E.g: Client provided information
    - Not very big
    - Skewed scope
- Proprietary Databases
  - E.g: Various vendors
    - Big
    - Pricy
    - Outdated
- Directed web searches
  - E.g: “car dealerships in the US”
    - High effort
    - Limited scope

# 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 \$\$\$
- Train team for complex decision making:
  - Time intensive → more \$\$\$
  - Slow
- Get multiple judgements for every website:
  - If everybody are bad, result will still be bad
  - More judgements → more \$\$\$

# 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

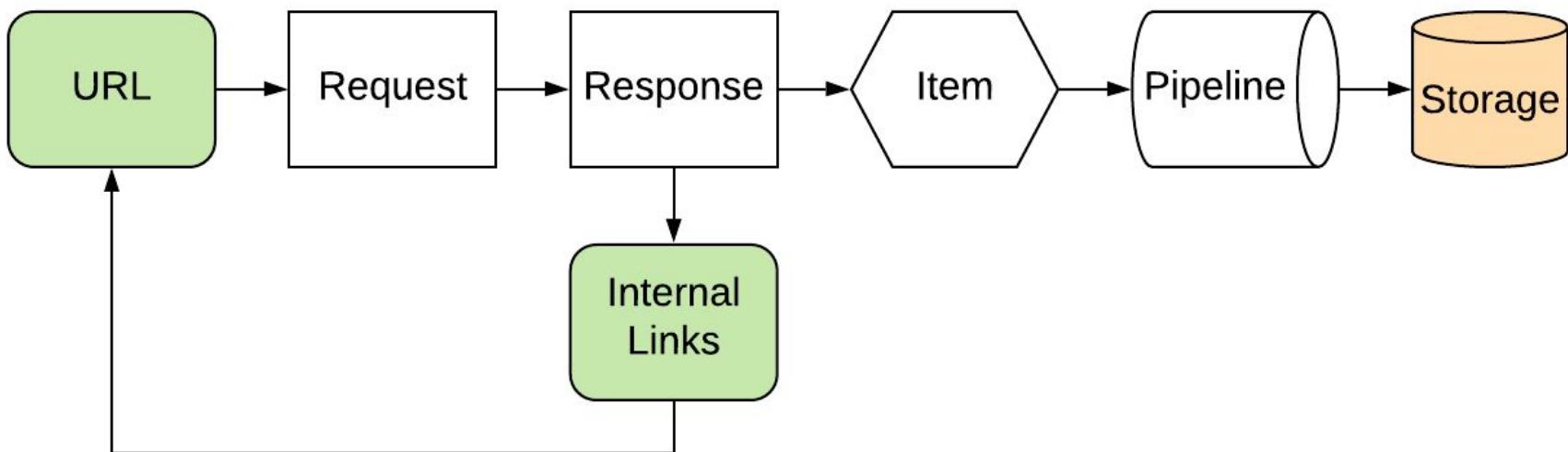
Our tool:

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

<https://github.com/scrapy/scrapy>

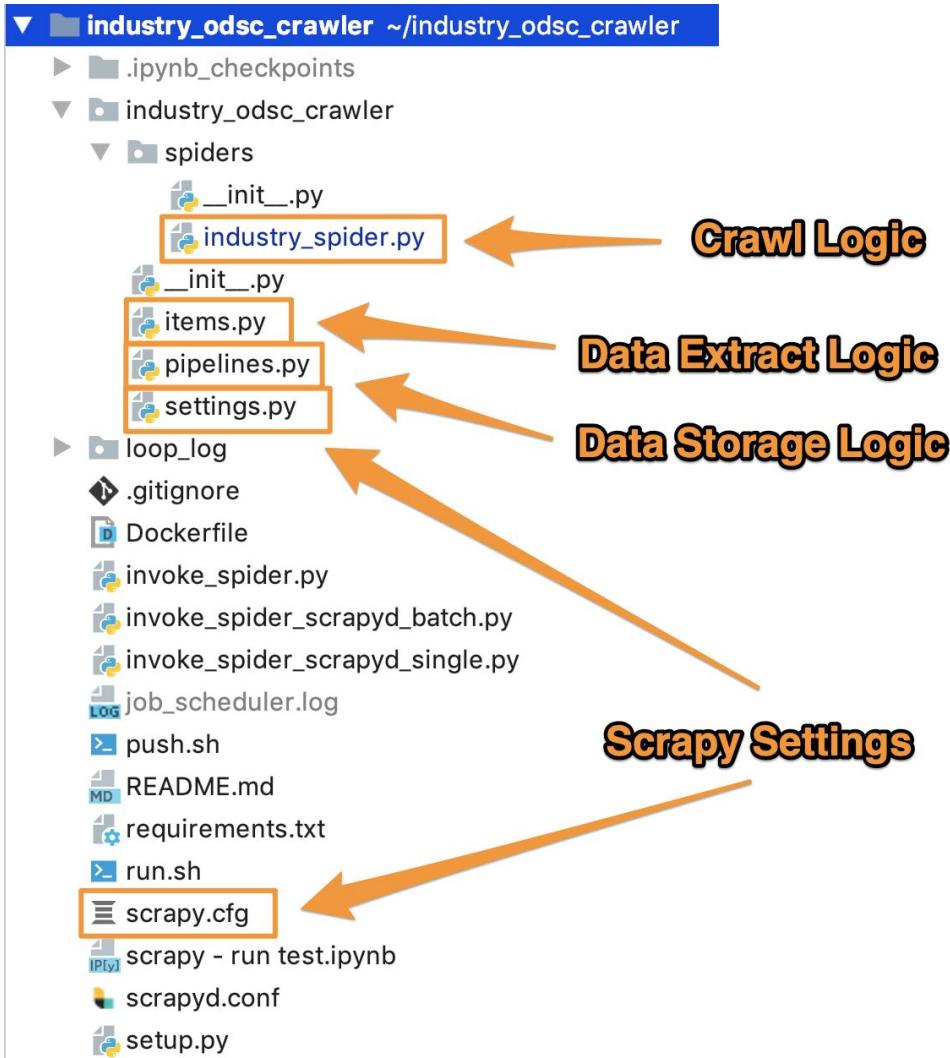
# Data - Building the Crawler

Scrapy Flow: Single URL



# Data - Building the Crawler

<scrapy - run test notebook>



```
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):...
41
42     def parse(self, response):...
```

**URLs to start with**

```
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):...
41
42     def parse(self, response):...
```

**Internal Link  
Counter**



```
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
27     def parse(self, response):...
```

For every URL,  
make 1st Request

```
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):...
41
42     def parse(self, response):...
```

**For every 1st  
Response, parse  
& follow links**

```
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
27     def parse(self, response):...
```



**For every URL,  
make 1st Request**

```
25 ⏵
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

```
def start_requests(self):
    for url_id, url in zip(self.ids, self.start_urls):
        try:
            request = Request(url=url, callback=self.parse,
                               dont_filter=True,
                               errback=self.handle_error)
        except ValueError:
            request.meta.update(url_id=url_id, url_order=1,
                               url_rank=1,
                               url_click_text=None,
                               url_domain=urlparse(url).netloc)
        yield request
```

Loop over URLs

```
25 ⏵
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

```
def start_requests(self):
    for url_id, url in zip(self.ids, self.start_urls):
        try:
            request = Request(url=url, callback=self.parse,
                               dont_filter=True,
                               errback=self.handle_error)
        except ValueError:
            request.meta.update(url_id=url_id, url_order=1,
                                url_rank=1,
                                url_click_text=None,
                                url_domain=urlparse(url).netloc)
        yield request
```

**Make 1st Request**



```
25 ⏵
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

```
def start_requests(self):
    for url_id, url in zip(self.ids, self.start_urls):
        try:
            request = Request(url=url, callback=self.parse,
                               dont_filter=True,
                               errback=self.handle_error)
        except ValueError:
            request.meta.update(url_id=url_id, url_order=1,
                                url_rank=1,
                                url_click_text=None,
                                url_domain=urlparse(url).netloc)
        yield request
```

**Make 1st  
Request**

**Point to  
parse  
method**

```
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: ...
32         request.meta.update(url_id=url_id, url_order=1,
33                             url_rank=1,
34                             url_click_text=None,
35                             url_domain=urlparse(url).netloc)
36         yield request
```

**Update Meta  
with custom  
stuff**



```
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):...
41
42     def parse(self, response):...
```

**For every 1st  
Response, parse  
& follow links**

```
42 def parse(self, response):
43     web_page = self.make_item(response)
44     yield web_page
45
46     allowed_domains = self.configure_allowed_domains(response)
47     links = self.extract_internal_links(allowed_domains, response)
48
49     for link in links:
50         is_below_limit = self.link_counter[response.meta['url_id']] < \
51                         MAX_PAGES_PER_DOMAIN
52         if is_below_limit:
53             request = self.make_internal_link_request(link, response)
54             yield request
```

**Extract things from Response and store**

**Extract all internal  
links inside the domain**

```
42 def parse(self, response):
43     web_page = self.make_item(response)
44     yield web_page
45
46     allowed_domains = self.configure_allowed_domains(response)
47     links = self.extract_internal_links(allowed_domains, response)
48
49     for link in links:
50         is_below_limit = self.link_counter[response.meta['url_id']] < \
51                         MAX_PAGES_PER_DOMAIN
52         if is_below_limit:
53             request = self.make_internal_link_request(link, response)
54             yield request
```

```
42 def parse(self, response):
43     web_page = self.make_item(response)
44     yield web_page
45
46     allowed_domains = self.configure_allowed_domains(response)
47     links = self.extract_internal_links(allowed_domains, response)
48
49     for link in links:
50         is_below_limit = self.link_counter[response.meta['url_id']] < \
51                         MAX_PAGES_PER_DOMAIN
52         if is_below_limit:
53             request = self.make_internal_link_request(link, response)
54             yield request
```

**Check domain link counter,  
break if limit passed**

```
42 ⚡ 42 def parse(self, response):
43     web_page = self.make_item(response)
44     yield web_page
45
46     allowed_domains = self.configure_allowed_domains(response)
47     links = self.extract_internal_links(allowed_domains, response)
48
49     for link in links:
50         is_below_limit = self.link_counter[response.meta['url_id']] < \
51                         MAX_PAGES_PER_DOMAIN
52         if is_below_limit:
53             request = self.make_internal_link_request(link, response)
54             yield request
```



**Make Request for  
internal link**

```
42 def parse(self, response):
43     web_page = self.make_item(response)
44     yield web_page
45
46     allowed_domains = self.configure_allowed_domains(response)
47     links = self.extract_internal_links(allowed_domains, response)
48
49     for link in links:
50         is_below_limit = self.link_counter[response.meta['url_id']] < \
51                         MAX_PAGES_PER_DOMAIN
52         if is_below_limit:
53             request = self.make_internal_link_request(link, response)
54             yield request
```

**Extract things from Response and store**

```
88 def make_item(self, response):
89     web_page = WebPage()
90     web_page['url_id'] = response.meta['url_id']
91     web_page['url_order'] = response.meta['url_order']
92     web_page['input_url'] = response.meta['download_slot']
93     web_page['depth'] = response.meta['depth'] if \
94         'depth' in response.meta.keys() else None
95     web_page['url'] = response.url
96     web_page['path'] = urlparse(response.url).path
97     web_page['url_click_text'] = response.meta['url_click_text']
98     web_page['response_raw'] = response.text
99     web_page['url_rank'] = response.meta['url_rank']
100    web_page['response_text'], web_page['url_title'] = \
101        text_from_html(response.text)
102    web_page['status_code'] = response.status
103    web_page['is_error'] = False
104    self.link_counter[response.meta['url_id']] += 1
105    return web_page
```

Extract  
clean text

```
88 def make_item(self, response):
89     web_page = WebPage()
90     web_page['url_id'] = response.meta['url_id']
91     web_page['url_order'] = response.meta['url_order']
92     web_page['input_url'] = response.meta['download_slot']
93     web_page['depth'] = response.meta['depth'] if \
|       'depth' in response.meta.keys() else None
94     web_page['url'] = response.url
95     web_page['path'] = urlparse(response.url).path
96     web_page['url_click_text'] = response.meta['url_click_text']
97     web_page['response_raw'] = response.text
98     web_page['url_rank'] = response.meta['url_rank']
99     web_page['response_text'], web_page['url_title'] = \
|       text_from_html(response.text)
100    web_page['status_code'] = response.status
101    web_page['is_error'] = False
102    self.link_counter[response.meta['url_id']] += 1
103    return web_page
104
105
```

Update  
domain  
counter

**Extract all internal  
links inside the domain**

```
42 def parse(self, response):
43     web_page = self.make_item(response)
44     yield web_page
45
46     allowed_domains = self.configure_allowed_domains(response)
47     links = self.extract_internal_links(allowed_domains, response)
48
49     for link in links:
50         is_below_limit = self.link_counter[response.meta['url_id']] < \
51                         MAX_PAGES_PER_DOMAIN
52         if is_below_limit:
53             request = self.make_internal_link_request(link, response)
54             yield request
```

```
~~  
67  
68  
69  
70  
71  
72  
73
```

```
def extract_internal_links(self, allowed_domains, response):  
    links = LinkExtractor(canonicalize=True, unique=True). \  
        extract_links(response)  
    links = [link for link in links if self.is_link_allowed(  
        allowed_domains, link.url)]  
    links = sorted(links, key=self.get_link_rank)  
    return links
```

Sort links by  
custom ranking

```
107 def get_link_rank(self, link):
108     priority_keywords = [
109         ('about', 2),
110         ('story', 2),
111         ('what we do', 3),
112         ('capabilities', 3),
113         ('services', 4),
114         ('products', 4),
115         ('amenities', 5),
116         ('overview', 6)
117     ]
118     default_page_url_rank = 100
119     default_page_text_rank = 100
120     link_rank = self.set_link_rank(
121         link, priority_keywords,
122         default_page_text_rank, default_page_url_rank)
123     return link_rank
```

Prioritize  
more  
informative  
links

```
42 def parse(self, response):
43     web_page = self.make_item(response)
44     yield web_page
45
46     allowed_domains = self.configure_allowed_domains(response)
47     links = self.extract_internal_links(allowed_domains, response)
48
49     for link in links:
50         is_below_limit = self.link_counter[response.meta['url_id']] < \
51                         MAX_PAGES_PER_DOMAIN
52         if is_below_limit:
53             request = self.make_internal_link_request(link, response)
54             yield request
```

**Extract things from Response and store**

```
42 ⚡↑ ⏹  
43  
44 def parse(self, response):  
45     web_page = self.make_item(response)  
46     yield web_page ← Send item to pipeline  
47  
48     allowed_domains = self.configure_allowed_domains(response)  
49     links = self.extract_internal_links(allowed_domains, response)  
50  
51     for link in links:  
52         is_below_limit = self.link_counter[response.meta['url_id']] < \  
53             MAX_PAGES_PER_DOMAIN  
54         if is_below_limit:  
55             request = self.make_internal_link_request(link, response)  
56             yield request
```

```
9  class ConvertToDataFramePipeline:  
10     def __init__(self):  
11         self.item_list = []  
12  
13     def process_item(self, item, spider):  
14         self.item_list += [item]  
15  
16     def close_spider(self, spider):  
17         data_clean_json, fn_clean = combine_items(  
18             self.item_list, spider, is_raw=False)  
19  
20         data_raw_json, fn_raw = combine_items(  
21             self.item_list, spider, is_raw=True)  
22  
23     if spider.save_local:  
24         self.save_data_locally(data_clean_json, data_raw_json,  
25                             fn_clean, fn_raw)  
26     else:  
27         self.save_data_in_s3(data_clean_json, data_raw_json,  
28                             fn_clean, fn_raw)
```

For every item, collect

```
9 class ConvertToDataFramePipeline:  
10     def __init__(self):  
11         self.item_list = []  
12  
13     def process_item(self, item, spider):  
14         self.item_list += [item]  
15  
16     def close_spider(self, spider):  
17         data_clean_json, fn_clean = combine_items(  
18             self.item_list, spider, is_raw=False)  
19  
20         data_raw_json, fn_raw = combine_items(  
21             self.item_list, spider, is_raw=True)  
22  
23     if spider.save_local:  
24         self.save_data_locally(data_clean_json, data_raw_json,  
25                             fn_clean, fn_raw)  
26     else:  
27         self.save_data_in_s3(data_clean_json, data_raw_json,  
28                             fn_clean, fn_raw)
```

**Organize all items, clean and raw**

```
9  class ConvertToDataFramePipeline:  
10     def __init__(self):  
11         self.item_list = []  
12  
13     def process_item(self, item, spider):  
14         self.item_list += [item]  
15  
16     def close_spider(self, spider):  
17         data_clean_json, fn_clean = combine_items(  
18             self.item_list, spider, is_raw=False)  
19  
20         data_raw_json, fn_raw = combine_items(  
21             self.item_list, spider, is_raw=True)  
22  
23     if spider.save_local:  
24         self.save_data_locally(data_clean_json, data_raw_json,  
25                                 fn_clean, fn_raw)  
26     else:  
27         self.save_data_in_s3(data_clean_json, data_raw_json,  
28                                 fn_clean, fn_raw)
```

**Send to 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

```
1 [scrapyd]
2 eggs_dir      = eggs
3 logs_dir      = logs
4 items_dir     =
5 jobs_to_keep  = 500
6 dbs_dir       = dbs
7 max_proc      = 0
8 max_proc_per_cpu = 4
9 finished_to_keep = 100
10 poll_interval = 5.0
11 bind_address  = 0.0.0.0
12 http_port    = 6800
13 debug        = off
14 runner       = scrapyd.runner
15 application  = scrapyd.app.application
16 launcher     = scrapyd.launcher.Launcher
17 webroot      = scrapyd.website.Root
18
19 [services]
20 schedule.json = scrapyd.webservice.Schedule
21 cancel.json   = scrapyd.webservice.Cancel
22 addversion.json = scrapyd.webservice.AddVersion
23 listprojects.json = scrapyd.webservice.ListProjects
24 listversions.json = scrapyd.webservice.ListVersions
25 listspiders.json = scrapyd.webservice.ListSpiders
26 delproject.json = scrapyd.webservice.DeleteProject
27 delversion.json = scrapyd.webservice.DeleteVersion
28 listjobs.json = scrapyd.webservice.ListJobs
29 daemonstatus.json = scrapyd.webservice.DaemonStatus
```

**Parallelism settings**

**Webserver port**

```
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)
40
```

## Import ScrapyD API constructor

```
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)
40
```

**Specify ScrapyD host and Scrapy 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)
```

**Load big file with URLs, split into chunks**



```
55 max_running = 10
56
57 max_pending = 5
58
59 sleep_interval = 15
60
61 while len(url_id_chunks) > 0:
62     server_status = scrapyd.list_jobs(project_name)
63
64     num_pending = len(server_status['pending'])
65     num_running = len(server_status['running'])
66
67     if (num_pending > max_pending) or \
68         (num_running > max_running): ...
69
70     i, url_id_chunk = url_id_chunks.pop(0)
71     urls = get_urls_from_ids(data_df, url_id_chunk)
72
73     scrapyd_schedule_job(urls)
```

**Define params  
for parallelism**

**Check what's  
running /  
pending**



```
55 max_running = 10
56
57 max_pending = 5
58
59 sleep_interval = 15
60
61 while len(url_id_chunks) > 0:
62     server_status = scrapyd.list_jobs(project_name)
63
64     num_pending = len(server_status['pending'])
65     num_running = len(server_status['running'])
66
67     if (num_pending > max_pending) or \
68         (num_running > max_running):...
69
70     i, url_id_chunk = url_id_chunks.pop(0)
71     urls = get_urls_from_ids(data_df, url_id_chunk)
72
73     scrapyd_schedule_job(urls)
74
75
76
77
78
79
80
81
82
```

```
55 max_running = 10
56
57 max_pending = 5
58
59 sleep_interval = 15
60
61 while len(url_id_chunks) > 0:
62     server_status = scrapyd.list_jobs(project_name)
63
64     num_pending = len(server_status['pending'])
65     num_running = len(server_status['running'])
66
67     if (num_pending > max_pending) or \
68         (num_running > max_running): ...
69
70     i, url_id_chunk = url_id_chunks.pop(0)
71     urls = get_urls_from_ids(data_df, url_id_chunk)
72
73     scrapyd_schedule_job(urls)
```

**If limits exceeded, break**

**Otherwise, schedule job**

# 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)

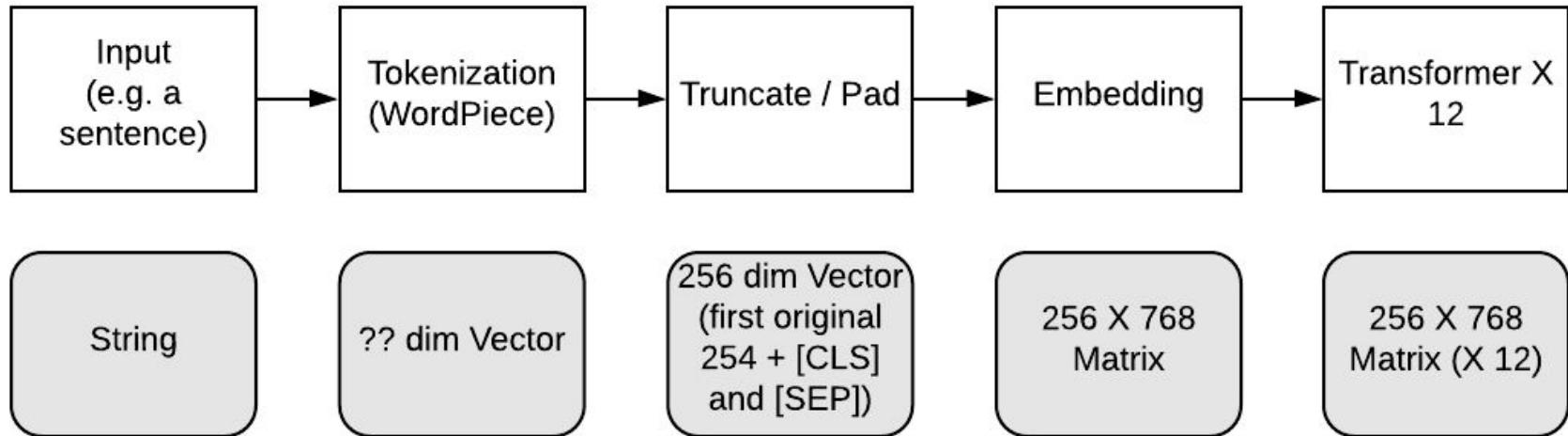
BERT (grossly simplified):

- 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>

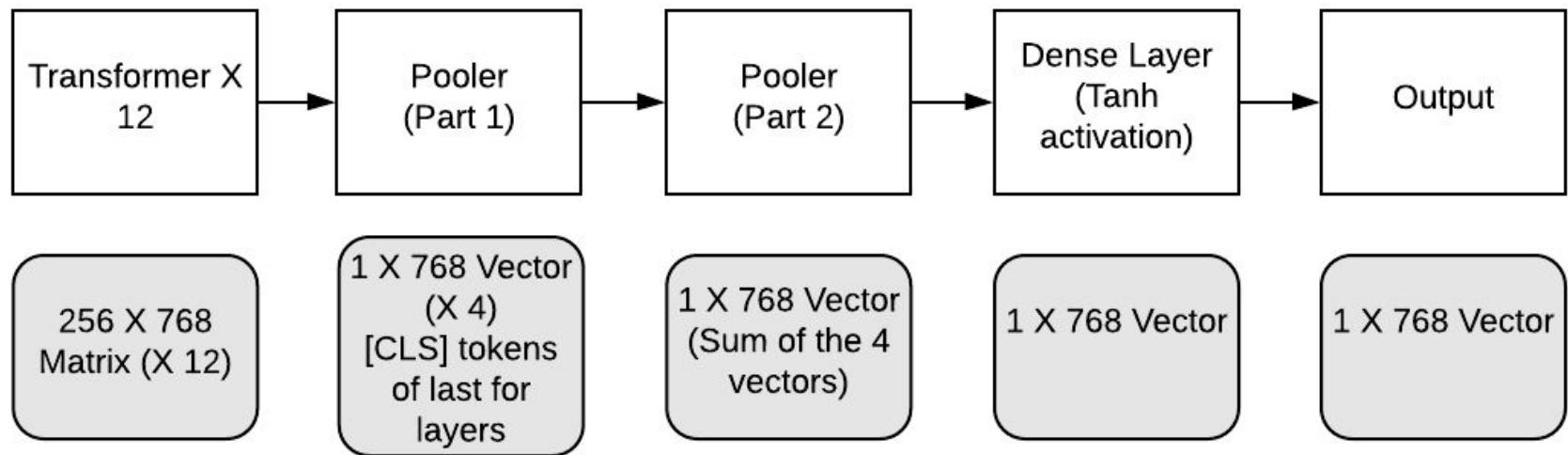
A bit more context:

- 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

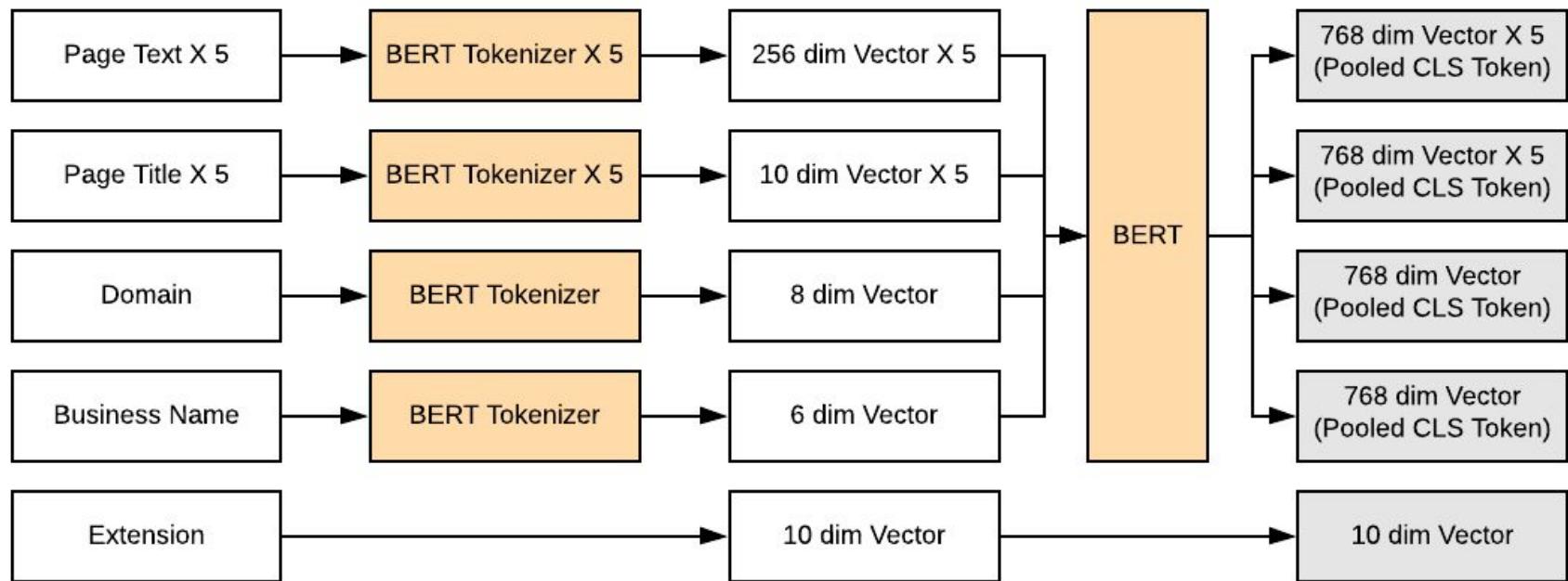
The basic idea:

- 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

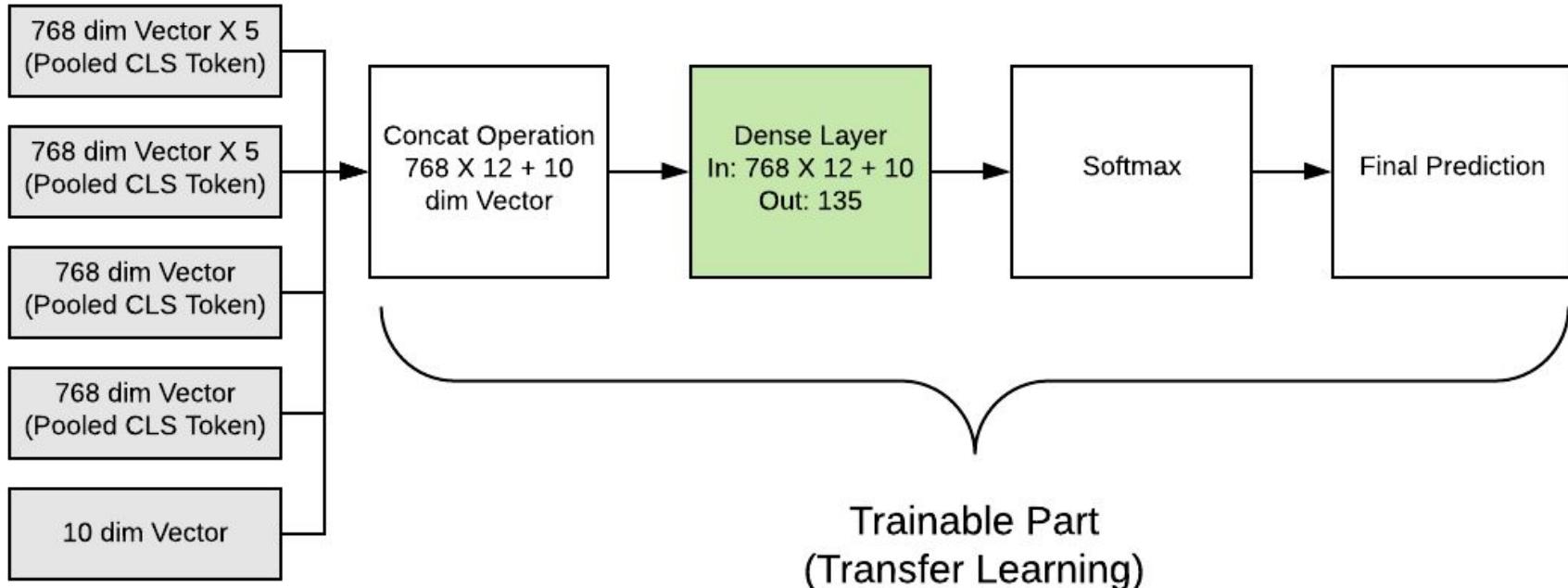
Implementation (moderate complexity)

- 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

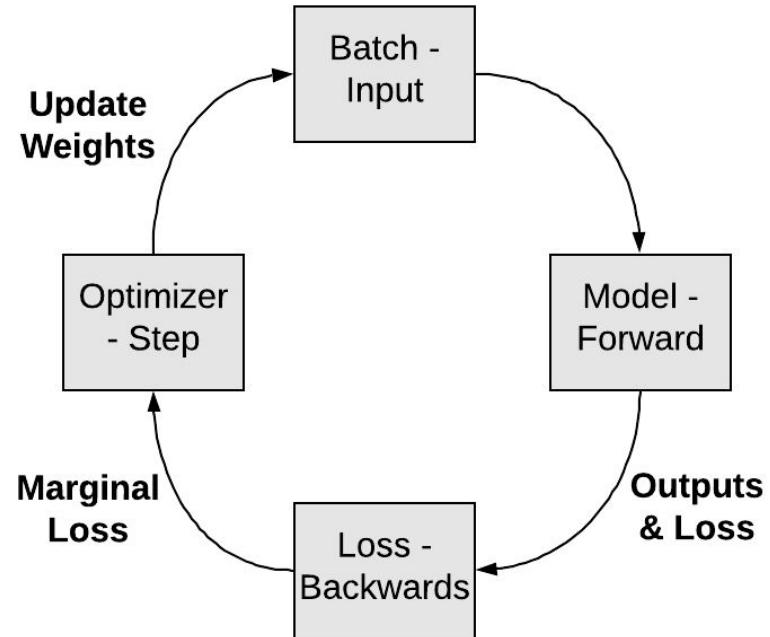
Tool:

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

# Model - Train

NN / PyTorch Train Cycle:

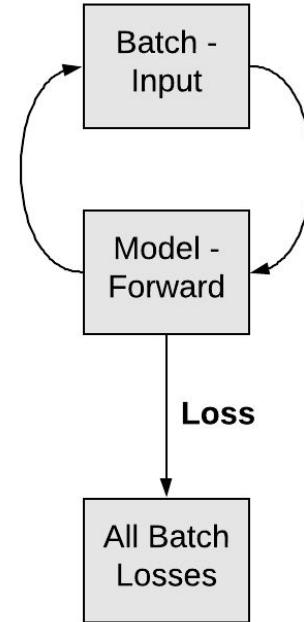
1. Select input batch (e.g. size 16)
2. Calc current outputs (forward)
3. Calc loss: Outputs VS labels
4. Calc marginal loss (backwards)
5. Update weights (step)
6. Again! Until batches / epoch done.



# 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)

```
202  
203 def make_dataset(input_df,  
204     bert_tokenizer, num_pages=5, max_len_page_text=256,  
205     max_len_page_title=16, max_len_domain=8, max_len_biz_name=10,  
206     chunk_size=300, n_jobs=None):  
207     """transform crawl data input into model readable tensor dataset..."""\n208  
209 # split input data into separate pages  
210 data_pages = split_data_into_pages(input_df.copy(), num_pages)  
211  
212 # generate dummies for top 10 url extensions  
213 top_extensions = [...]  
214 url_ext_mat = gen_url_extension_dummies(data_pages[0], top_extensions)  
215  
216 # extract domain from url  
217 data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(  
218     extract_domain_from_url)  
219  
220 # tokenize & truncate all input fields  
221 biz_name_masks, biz_name_tokenized, domain_masks, domain_tokenized, \  
222     page_text_masks, page_text_tokenized, page_title_masks, \  
223     page_titles_tokenized = tokenize_and_truncate_fields(...)  
224  
225 # combine into tensor dataset  
226 data_set = combine_inputs_into_tensor_dataset(...)  
227  
228 return data_set
```

BERT  
settings

Parallelism  
settings

```
202  
203     def make_dataset(input_df,  
204  
205         bert_tokenizer, num_pages=5, max_len_page_text=256,  
206         max_len_page_title=16, max_len_domain=8, max_len_biz_name=10,  
207  
208             chunk_size=300, n_jobs=None):  
209     """transform crawl data input into model readable tensor dataset..."""\n210  
211     # split input data into separate pages  
212     data_pages = split_data_into_pages(input_df.copy(), num_pages)  
213  
214     # generate dummies for top 10 url extensions  
215     top_extensions = [...]  
216     url_ext_mat = gen_url_extension_dummies(data_pages[0], top_extensions)  
217  
218     # extract domain from url  
219     data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(  
220         extract_domain_from_url)  
221  
222     # tokenize & truncate all input fields  
223     biz_name_masks, biz_name_tokenized, domain_masks, domain_tokenized, \  
224         page_text_masks, page_text_tokenized, page_title_masks, \  
225         page_titles_tokenized = tokenize_and_truncate_fields(...)  
226  
227     # combine into tensor dataset  
228     data_set = combine_inputs_into_tensor_dataset(...)  
229  
230     return data_set
```

**Split and limit data to X pages per domain**

```
202  
203     def make_dataset(input_df,  
204  
205             bert_tokenizer, num_pages=5, max_len_page_text=256,  
206             max_len_page_title=16, max_len_domain=8, max_len_biz_name=10,  
207             chunk_size=300, n_jobs=None):  
208         """transform crawl data input into model readable tensor dataset..."""  
209  
210         # split input data into separate pages  
211         data_pages = split_data_into_pages(input_df.copy(), num_pages)  
212  
213         # generate dummies for top 10 url extensions  
214         top_extensions = [...]  
215         url_ext_mat = gen_url_extension_dummies(data_pages[0], top_extensions)  
216  
217         # extract domain from url  
218         data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(  
219             extract_domain_from_url)  
220  
221         # tokenize & truncate all input fields  
222         biz_name_masks, biz_name_tokenized, domain_masks, domain_tokenized, \  
223             page_text_masks, page_text_tokenized, page_title_masks, \  
224             page_titles_tokenized = tokenize_and_truncate_fields(...)  
225  
226         # combine into tensor dataset  
227         data_set = combine_inputs_into_tensor_dataset(...)  
228  
229     return data_set
```

← **Tokenize  
and  
Truncate all  
fields**

```
202  
203     def make_dataset(input_df,  
204  
205         bert_tokenizer, num_pages=5, max_len_page_text=256,  
206         max_len_page_title=16, max_len_domain=8, max_len_biz_name=10,  
207         chunk_size=300, n_jobs=None):  
208         """transform crawl data input into model readable tensor dataset..."""\n209  
210         # split input data into separate pages  
211         data_pages = split_data_into_pages(input_df.copy(), num_pages)  
212  
213         # generate dummies for top 10 url extensions  
214         top_extensions = [...]  
215         url_ext_mat = gen_url_extension_dummies(data_pages[0], top_extensions)  
216  
217         # extract domain from url  
218         data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(  
219             | extract_domain_from_url)  
220  
221         # tokenize & truncate all input fields  
222         biz_name_masks, biz_name_tokenized, domain_masks, domain_tokenized, \  
223             page_text_masks, page_text_tokenized, page_title_masks, \  
224             page_titles_tokenized = tokenize_and_truncate_fields(...)  
225  
226         # combine into tensor dataset  
227         data_set = combine_inputs_into_tensor_dataset(...)  
228  
229         return data_set
```

Combine  
into  
tensor  
data set

```
202  
203     def make_dataset(input_df,  
204  
205         bert_tokenizer, num_pages=5, max_len_page_text=256,  
206         max_len_page_title=16, max_len_domain=8, max_len_biz_name=10,  
207  
208             chunk_size=300, n_jobs=None):  
209     """transform crawl data input into model readable tensor dataset..."""  
210  
211  
212     # split input data into separate pages  
213     data_pages = split_data_into_pages(input_df.copy(), num_pages)  
214  
215     # generate dummies for top 10 url extensions  
216     top_extensions = [...]  
217     url_ext_mat = gen_url_extension_dummies(data_pages[0], top_extensions)  
218  
219     # extract domain from url  
220     data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(  
221         extract_domain_from_url)  
222  
223  
224     # tokenize & truncate all input fields  
225     biz_name_masks, biz_name_tokenized, domain_masks, domain_tokenized, \  
226         page_text_masks, page_text_tokenized, page_title_masks, \  
227         page_titles_tokenized = tokenize_and_truncate_fields(...)  
228  
229  
230     # combine into tensor dataset  
231     data_set = combine_inputs_into_tensor_dataset(...)  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249     return data_set
```

← **Tokenize  
and  
Truncate all  
fields**

```
79 def _tokenize_and_truncate_chunk(page_sequences_text, bert_tokenizer, max_len):
80     page_sequences_tokenized = []
81     for seq_text in page_sequences_text:
82         seq_tokenized = bert_tokenizer.tokenize(seq_text)
83
84         seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
85         if hasattr(seq_ids, '__iter__'):
86             seq_ids = pad_sequences([seq_ids], maxlen=max_len, dtype="long",
87                                    truncating="post", padding="post")[0]
88         else:
89             seq_tokenized = bert_tokenizer.tokenize("[CLS] [SEP]")
90             seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
91
92         page_sequences_tokenized.append(seq_ids)
93
94     return page_sequences_tokenized
```

**For every text  
(i.e sentence) ...**

```
79 def _tokenize_and_truncate_chunk(page_sequences_text, bert_tokenizer, max_len):
80     page_sequences_tokenized = []
81     for seq_text in page_sequences_text:
82
83         seq_tokenized = bert_tokenizer.tokenize(seq_text) Tokenize
84
85         seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
86         if hasattr(seq_ids, '__iter__'):
87             seq_ids = pad_sequences([seq_ids], maxlen=max_len, dtype="long",
88                                    truncating="post", padding="post")[0]
89         else:
90             seq_tokenized = bert_tokenizer.tokenize("[CLS] [SEP]")
91             seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
92
93         page_sequences_tokenized.append(seq_ids)
94
95     return page_sequences_tokenized
```

```
79 def _tokenize_and_truncate_chunk(page_sequences_text, bert_tokenizer, max_len):
80     page_sequences_tokenized = []
81     for seq_text in page_sequences_text:
82
83         seq_tokenized = bert_tokenizer.tokenize(seq_text)
84
85         seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized) # Boxed
86         if hasattr(seq_ids, '__iter__'):
87             seq_ids = pad_sequences([seq_ids], maxlen=max_len, dtype="long",
88                                     truncating="post", padding="post")[0]
89         else:
90             seq_tokenized = bert_tokenizer.tokenize("[CLS] [SEP]")
91             seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
92
93         page_sequences_tokenized.append(seq_ids)
94
95     return page_sequences_tokenized
```

Convert to ids  
(i.e. integers)

```
79 def _tokenize_and_truncate_chunk(page_sequences_text, bert_tokenizer, max_len):
80     page_sequences_tokenized = []
81     for seq_text in page_sequences_text:
82
83         seq_tokenized = bert_tokenizer.tokenize(seq_text)
84
85         seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
86         if hasattr(seq_ids, 'iter'):
87             seq_ids = pad_sequences([seq_ids], maxlen=max_len, dtype="long",
88                                    truncating="post", padding="post")[0]
89         else:
90             seq_tokenized = bert_tokenizer.tokenize("[CLS] [SEP]")
91             seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
92
93         page_sequences_tokenized.append(seq_ids)
94
95     return page_sequences_tokenized
```

Truncate



```
79     def _ tokenize_and_truncate_chunk(page_sequences_text, bert_tokenizer, max_len):
80         page_sequences_tokenized = []
81         for seq_text in page_sequences_text:
82
83             seq_tokenized = bert_tokenizer.tokenize(seq_text)
84
85             seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
86             if hasattr(seq_ids, '__iter__'):
87                 seq_ids = pad_sequences([seq_ids], maxlen=max_len, dtype="long",
88                                         truncating="post", padding="post")[0]
89             else:
90                 seq_tokenized = bert_tokenizer.tokenize("[CLS] [SEP]")
91                 seq_ids = bert_tokenizer.convert_tokens_to_ids(seq_tokenized)
92
93             page_sequences_tokenized.append(seq_ids)
94
95     return page_sequences_tokenized
```

Handle empty string

```
202  
203     def make_dataset(input_df,  
204  
205         bert_tokenizer, num_pages=5, max_len_page_text=256,  
206         max_len_page_title=16, max_len_domain=8, max_len_biz_name=10,  
207  
208             chunk_size=300, n_jobs=None):  
209     """transform crawl data input into model readable tensor dataset..."""  
210  
211  
212     # split input data into separate pages  
213     data_pages = split_data_into_pages(input_df.copy(), num_pages)  
214  
215     # generate dummies for top 10 url extensions  
216     top_extensions = [...]  
217     url_ext_mat = gen_url_extension_dummies(data_pages[0], top_extensions)  
218  
219     # extract domain from url  
220     data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(  
221         extract_domain_from_url)  
222  
223  
224     # tokenize & truncate all input fields  
225     biz_name_masks, biz_name_tokenized, domain_masks, domain_tokenized, \  
226         page_text_masks, page_text_tokenized, page_title_masks, \  
227         page_titles_tokenized = tokenize_and_truncate_fields(...)  
228  
229  
230     # combine into tensor dataset  
231     data_set = combine_inputs_into_tensor_dataset(...)  
232  
233  
234  
235     return data_set
```

← **Tokenize  
and  
Truncate all  
fields**

```
202  
203     def make_dataset(input_df,  
204  
205         bert_tokenizer, num_pages=5, max_len_page_text=256,  
206         max_len_page_title=16, max_len_domain=8, max_len_biz_name=10,  
207         chunk_size=300, n_jobs=None):  
208         """transform crawl data input into model readable tensor dataset..."""\n209  
210         # split input data into separate pages  
211         data_pages = split_data_into_pages(input_df.copy(), num_pages)  
212  
213         # generate dummies for top 10 url extensions  
214         top_extensions = [...]  
215         url_ext_mat = gen_url_extension_dummies(data_pages[0], top_extensions)  
216  
217         # extract domain from url  
218         data_pages[0]['url_domain_text'] = data_pages[0]['url'].apply(  
219             | extract_domain_from_url)  
220  
221         # tokenize & truncate all input fields  
222         biz_name_masks, biz_name_tokenized, domain_masks, domain_tokenized, \  
223             page_text_masks, page_text_tokenized, page_title_masks, \  
224             page_titles_tokenized = tokenize_and_truncate_fields(...)  
225  
226         # combine into tensor dataset  
227         data_set = combine_inputs_into_tensor_dataset(...)  
228  
229         return data_set
```

Combine  
into  
tensor  
data set

```
71 bert_model_name = 'bert-base-cased'  
72 bert_tokenizer = BertTokenizer.from_pretrained(bert_model_name,  
73                                     do_lower_case=False)  
74  
75 train_data = make_dataset(...)  
76 test_data = make_dataset(...)  
77  
78 train_sampler = RandomSampler(train_data)  
79 train_dataloader = DataLoader(train_data, sampler=train_sampler,  
80                             batch_size=16)  
81  
82 test_sampler = SequentialSampler(test_data)  
83 test_dataloader = DataLoader(test_data, sampler=test_sampler,  
84                             batch_size=16)
```

Load pertained BERT Tokenizer



```
71 bert_model_name = 'bert-base-cased'  
72 bert_tokenizer = BertTokenizer.from_pretrained(bert_model_name,  
73                                     do_lower_case=False)  
74  
75 train_data = make_dataset(...)  
76 test_data = make_dataset(...)   
77  
78 train_sampler = RandomSampler(train_data)  
79 train_dataloader = DataLoader(train_data, sampler=train_sampler,  
80                             batch_size=16)  
81  
82 test_sampler = SequentialSampler(test_data)  
83 test_dataloader = DataLoader(test_data, sampler=test_sampler,  
84                             batch_size=16)
```

**Convert  
raw train /  
test into  
tensor  
datasets**

```
71 bert_model_name = 'bert-base-cased'  
72 bert_tokenizer = BertTokenizer.from_pretrained(bert_model_name,  
73                                do_lower_case=False)  
74  
75 train_data = make_dataset(...)  
76 test_data = make_dataset(...)  
77  
78 train_sampler = RandomSampler(train_data)  
79 train_dataloader = DataLoader(train_data, sampler=train_sampler,  
80                               batch_size=16)  
81  
82 test_sampler = SequentialSampler(test_data)  
83 test_dataloader = DataLoader(test_data, sampler=test_sampler,  
84                               batch_size=16)
```

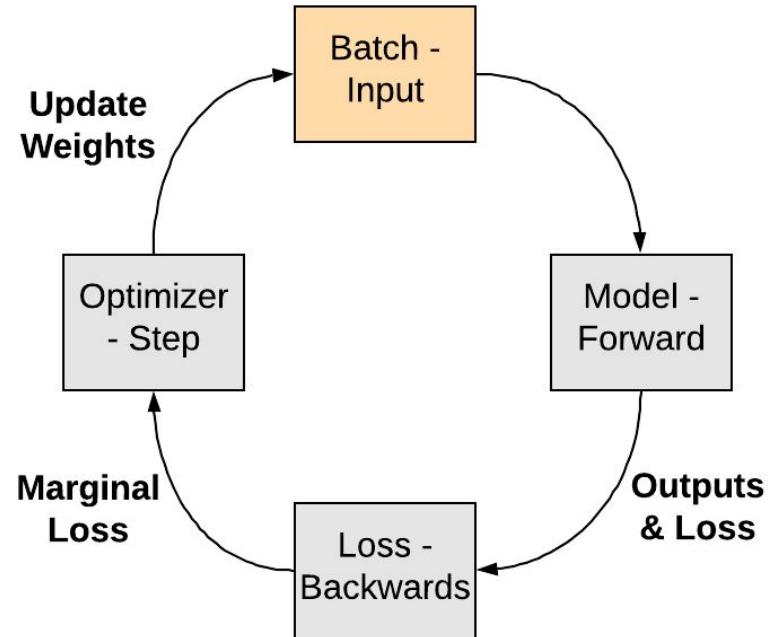
Create  
test/train  
data  
loaders  
for GPU



# Model - Train

Batch - Input:

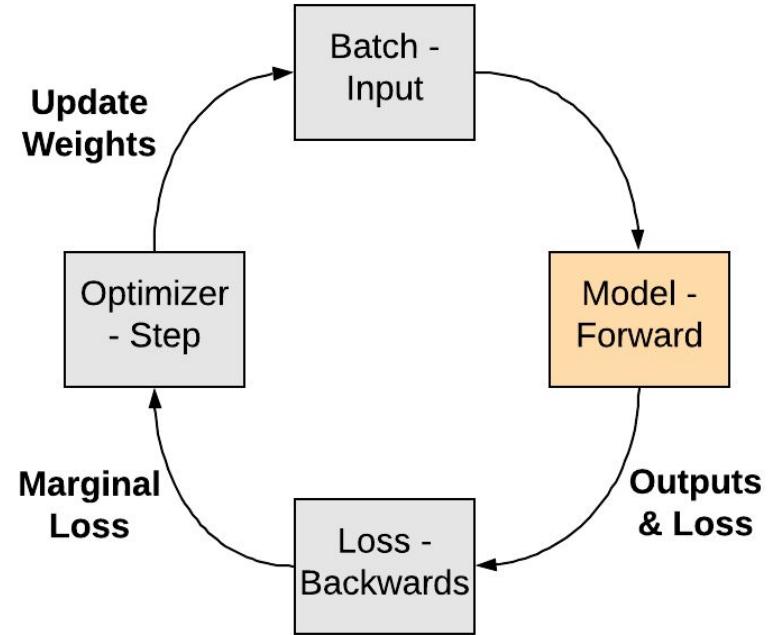
- Create Train / Test Dataloaders
- Extract Batch



# 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



## Extending a BERT pre- trained model class

```
17 class BertWebsiteClassifier(BertPreTrainedModel): ←
18     def __init__(self, config):
19         super(BertWebsiteClassifier, self).__init__(config)
20         self.bert = BertModel(config)
21         self.dropout = nn.Dropout(config.hidden_dropout_prob)
22         self.classifier = nn.Linear(config.hidden_size * 12 + 10,
23                                     self.config.num_labels)
24
25         self.num_labels = None
26         self.num_pages = None
27         self.max_len_page_text = None
28         self.max_len_page_title = None
29         self.max_len_domain = None
30         self.max_len_biz_name = None
31         self.industry_map_dict = None
32
33         self.init_weights()
```

## Loading actual BERT pre- trained model

```
17 class BertWebsiteClassifier(BertPreTrainedModel):
18     def __init__(self, config):
19         super(BertWebsiteClassifier, self).__init__(config)
20         self.bert = BertModel(config) ←
21         self.dropout = nn.Dropout(config.hidden_dropout_prob)
22         self.classifier = nn.Linear(config.hidden_size * 12 + 10,
23                                     self.config.num_labels)
24
25         self.num_labels = None
26         self.num_pages = None
27         self.max_len_page_text = None
28         self.max_len_page_title = None
29         self.max_len_domain = None
30         self.max_len_biz_name = None
31         self.industry_map_dict = None
32
33         self.init_weights()
```

```
17 class BertWebsiteClassifier(BertPreTrainedModel):
18     def __init__(self, config):
19         super(BertWebsiteClassifier, self).__init__(config)
20         self.bert = BertModel(config)
21         self.dropout = nn.Dropout(config.hidden_dropout_prob)
22         self.classifier = nn.Linear(config.hidden_size * 12 + 10,
23                                     self.config.num_labels)
24
25         self.num_labels = None
26         self.num_pages = None
27         self.max_len_page_text = None
28         self.max_len_page_title = None
29         self.max_len_domain = None
30         self.max_len_biz_name = None
31         self.industry_map_dict = None
32
33         self.init_weights()
```



**Trainable linear  
layer**

```
47  o↑ def forward(self,  
48      page_text_ids, page_text_masks,  
49      page_title_ids, page_title_masks,  
50      domain_ids, domain_masks,  
51      biz_name_ids, biz_name_masks,  
52      url_exts,  
53      token_type_ids=None,  
54      position_ids=None, head_mask=None,  
55      labels=None):  
56  
57      biz_name_pooled_output = self.bert(...)[1]  
63      domain_pooled_output = self.bert(...)[1]  
69  
70      page_text_outputs = []  
71      for page_text_id, page_text_mask in \  
72          zip(page_text_ids, page_text_masks):  
73          page_text_outputs += [self.bert(...)[1]]  
79  
80      page_title_outputs = []  
81      for page_title_ids, page_title_masks in \  
82          zip(page_title_ids, page_title_masks):  
83          page_title_outputs += [self.bert(...)[1]]  
84
```

**All inputs:  
tokens, masks  
and dummies**

```
47 ⚡
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
```

```
def forward(self,
            page_text_ids, page_text_masks,
            page_title_ids, page_title_masks,
            domain_ids, domain_masks,
            biz_name_ids, biz_name_masks,
            url_exts,
            token_type_ids=None,
            position_ids=None, head_mask=None,
            labels=None):

    biz_name_pooled_output = self.bert(...)[1]
    domain_pooled_output = self.bert(...)[1]

    page_text_outputs = []
    for page_text_id, page_text_mask in \
        zip(page_text_ids, page_text_masks):
        page_text_outputs += [self.bert(...)[1]]

    page_title_outputs = []
    for page_title_ids, page_title_masks in \
        zip(page_title_ids, page_title_masks):
        page_title_outputs += [self.bert(...)[1]]
```

**Pass  
everything  
through pre-  
trained BERT**

```
47  ⚡
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
```

```
def forward(self,
            page_text_ids, page_text_masks,
            page_title_ids, page_title_masks,
            domain_ids, domain_masks,
            biz_name_ids, biz_name_masks,
            url_exts,
            token_type_ids=None,
            position_ids=None, head_mask=None,
            labels=None):
    biz_name_pooled_output = self.bert(...)[1]
    domain_pooled_output = self.bert(...)[1]

    page_text_outputs = []
    for page_text_id, page_text_mask in \
        zip(page_text_ids, page_text_masks):
        page_text_outputs += [self.bert(...)[1]]

    page_title_outputs = []
    for page_title_ids, page_title_masks in \
        zip(page_title_ids, page_title_masks):
        page_title_outputs += [self.bert(...)[1]]
```

**Vectors of  
size 1X768**

**Pass  
everything  
through pre-  
trained BERT**

```
...
90 # combine all outputs
91 outputs_all = page_text_outputs + page_title_outputs + \
92     [domain_pooled_output, biz_name_pooled_output, url_exts]
93 pooled_output = torch.cat(outputs_all, dim=1)
94 pooled_output = self.dropout(pooled_output)
95
96 logits = self.classifier(pooled_output)
97
98 outputs = (logits,)
99
100 if labels is not None: ...
101 return outputs
```



**Concat all  
vectors into one  
big vector**

```
...
90 # combine all outputs
91 outputs_all = page_text_outputs + page_title_outputs + \
92     [domain_pooled_output, biz_name_pooled_output, url_exts]
93 pooled_output = torch.cat(outputs_all, dim=1)
94 pooled_output = self.dropout(pooled_output)
95
96 logits = self.classifier(pooled_output) Run big vector  
thru trainable  
layer
97 outputs = (logits,)
98
99 if labels is not None: ...
100 return outputs
111
```

```
...
90      # combine all outputs
91      outputs_all = page_text_outputs + page_title_outputs + \
92                  [domain_pooled_output, biz_name_pooled_output, url_exts]
93      pooled_output = torch.cat(outputs_all, dim=1)
94      pooled_output = self.dropout(pooled_output)
95
96      logits = self.classifier(pooled_output)
97
98      outputs = (logits,)
99
100     if labels is not None: ...
101     return outputs
```

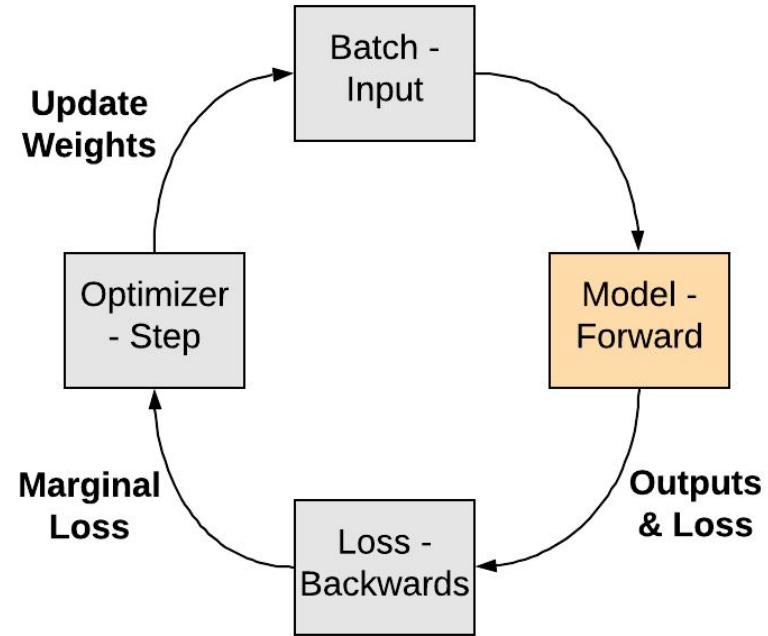
**Return outputs  
(+ Loss for  
training)**



# 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



236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246

```
bert_website_mdl = BertWebsiteClassifier.from_pretrained(  
    sample_meta['bert_model_name'], sample_meta['num_labels'])  
  
if torch.cuda.device_count() > 1:  
    print("Let's use", torch.cuda.device_count(), "GPUs!")  
    bert_website_mdl = nn.DataParallel(bert_website_mdl)  
  
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  
bert_website_mdl.to(device)
```

**Init  
classifier**

```
236  
237 bert_website_mdl = BertWebsiteClassifier.from_pretrained(  
238     sample_meta['bert_model_name'], sample_meta['num_labels'])  
239  
240 if torch.cuda.device_count() > 1:  
241     print("Let's use", torch.cuda.device_count(), "GPUs!")  
242     bert_website_mdl = nn.DataParallel(bert_website_mdl)  
243  
244 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  
245 bert_website_mdl.to(device)
```

Set up for  
multiple  
GPUs

```
265 for _ in trange(epochs, desc="Epoch"):
266
267     bert_website_mdl.train()           ← Allow model to
268                                         update weights
269     tr_loss, tr_accuracy = 0, 0
270     nb_tr_examples, nb_tr_steps = 0, 0
271
272     for step, batch in enumerate(train_dataloader):
273
274         # Add batch to GPU
275         batch = tuple(t.to(device) for t in batch)
276
277         # Unpack the inputs from our dataloader
278         page_text_ids1, page_text_ids2, page_text_ids3, \
279             page_text_ids4, page_text_ids5, \
280             page_text_masks1, page_text_masks2, page_text_masks3, \
281             page_text_masks4, page_text_masks5, \
282             page_title_ids1, page_title_ids2, page_title_ids3, \
283             page_title_ids4, page_title_ids5, \
284             page_title_masks1, page_title_masks2, page_title_masks3, \
285             page_title_masks4, page_title_masks5, \
286             domain_ids, domain_masks, biz_name_ids, biz_name_masks, \
287             url_exts, batch_labels = batch
288
```

```
265     for _ in trange(epochs, desc="Epoch"):
266         bert_website_mdl.train()
267
268         tr_loss, tr_accuracy = 0, 0
269         nb_tr_examples, nb_tr_steps = 0, 0
270
271
272         for step, batch in enumerate(train_dataloader):
273
274             # Add batch to GPU
275             batch = tuple(t.to(device) for t in batch)
276
277             # Unpack the inputs from our dataloader
278             page_text_ids1, page_text_ids2, page_text_ids3, \
279                 page_text_ids4, page_text_ids5, \
280                 page_text_masks1, page_text_masks2, page_text_masks3, \
281                 page_text_masks4, page_text_masks5, \
282                 page_title_ids1, page_title_ids2, page_title_ids3, \
283                 page_title_ids4, page_title_ids5, \
284                 page_title_masks1, page_title_masks2, page_title_masks3, \
285                 page_title_masks4, page_title_masks5, \
286                 domain_ids, domain_masks, biz_name_ids, biz_name_masks, \
287                 url_exts, batch_labels = batch
288
```

Iterate over all batches in train



```
265     for _ in trange(epochs, desc="Epoch"):
266         bert_website_mdl.train()
267
268         tr_loss, tr_accuracy = 0, 0
269         nb_tr_examples, nb_tr_steps = 0, 0
270
271         for step, batch in enumerate(train_dataloader):
272
273             # Add batch to GPU
274             batch = tuple(t.to(device) for t in batch)
275
276
277             # Unpack the inputs from our dataloader
278             page_text_ids1, page_text_ids2, page_text_ids3, \
279                 page_text_ids4, page_text_ids5, \
280                 page_text_masks1, page_text_masks2, page_text_masks3, \
281                 page_text_masks4, page_text_masks5, \
282                 page_title_ids1, page_title_ids2, page_title_ids3, \
283                 page_title_ids4, page_title_ids5, \
284                 page_title_masks1, page_title_masks2, page_title_masks3, \
285                 page_title_masks4, page_title_masks5, \
286                 domain_ids, domain_masks, biz_name_ids, biz_name_masks, \
287                 url_exts, batch_labels = batch
```

**Unpack all  
inputs in batch**



```
289 # Clear out the gradients (by default they accumulate)
290 optimizer.zero_grad()
291
292 # Forward pass
293 loss, logits = bert_website_mdl(
294     page_text_ids=(
295         page_text_ids1, page_text_ids2, page_text_ids3,
296         page_text_ids4, page_text_ids5),
297     page_text_masks=(page_text_masks1, page_text_masks2,
298                      page_text_masks3, page_text_masks4,
299                      page_text_masks5),
300     page_title_ids=(...),
301     page_title_masks=(...),
302     domain_ids=domain_ids,
303     domain_masks=domain_masks,
304     biz_name_ids=biz_name_ids,
305     biz_name_masks=biz_name_masks,
306     url_exts=url_exts,
307     token_type_ids=None,
308     labels=batch_labels)
```

**Pass all  
inputs to  
forward  
method**

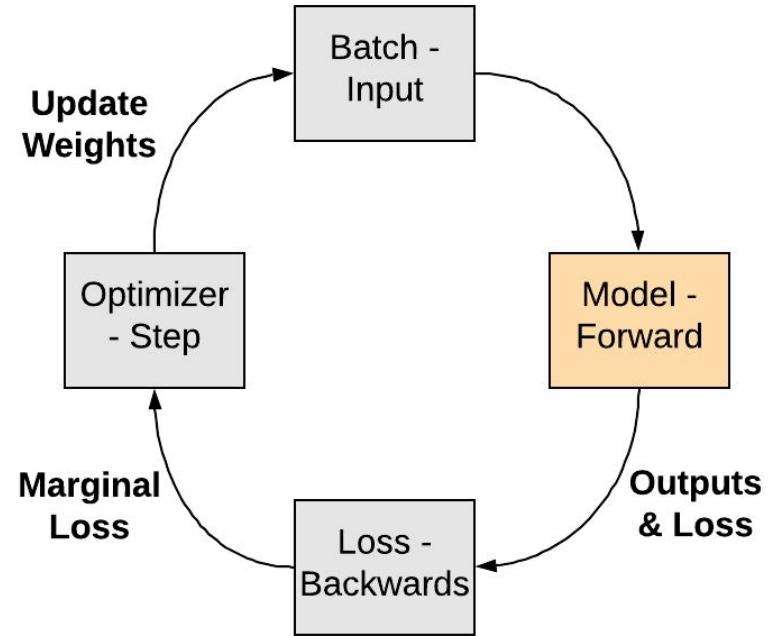
```
289     # Clear out the gradients (by default they accumulate)
290     optimizer.zero_grad()
291
292     # Forward pass
293     loss, logits = bert_website_mdl(
294         page_text_ids=(
295             page_text_ids1, page_text_ids2, page_text_ids3,
296             page_text_ids4, page_text_ids5),
297         page_text_masks=(page_text_masks1, page_text_masks2,
298                         page_text_masks3, page_text_masks4,
299                         page_text_masks5),
300         page_title_ids=(...),
301         page_title_masks=(...),
302         domain_ids=domain_ids,
303         domain_masks=domain_masks,
304         biz_name_ids=biz_name_ids,
305         biz_name_masks=biz_name_masks,
306         url_exts=url_exts,
307         token_type_ids=None,
308         labels=batch_labels)
```

Obtain batch  
output and  
loss

# Model - Train

Forward method:

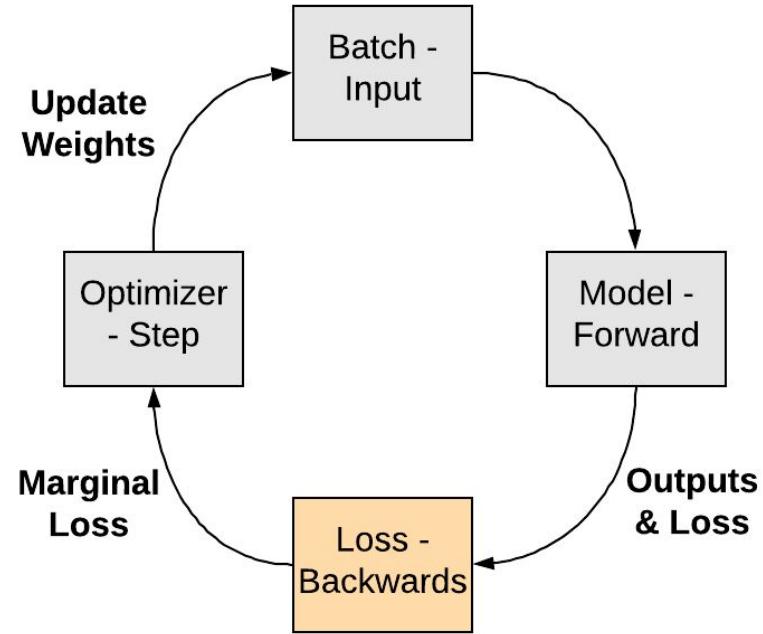
- Define the graph of the model:
  - Layers
  - Transformations
  - Combinations
- Is invoked with inputs, calculates outputs
- Using labels and a loss metric, calculates loss



# 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



```
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316
```

# Clear out the gradients (by default they accumulate)  
optimizer.zero\_grad()  
  
# Forward pass  
loss, logits = bert\_website\_mdl(...)  
  
train\_loss\_set.append(loss.mean().item())  
loss.mean().backward()  
optimizer.step()  
  
# Move logits and labels to CPU  
logits = logits.detach().cpu().numpy()  
label\_ids = batch\_labels.to('cpu').numpy()

Aggregate loss from multiple GPUs

```
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316
```

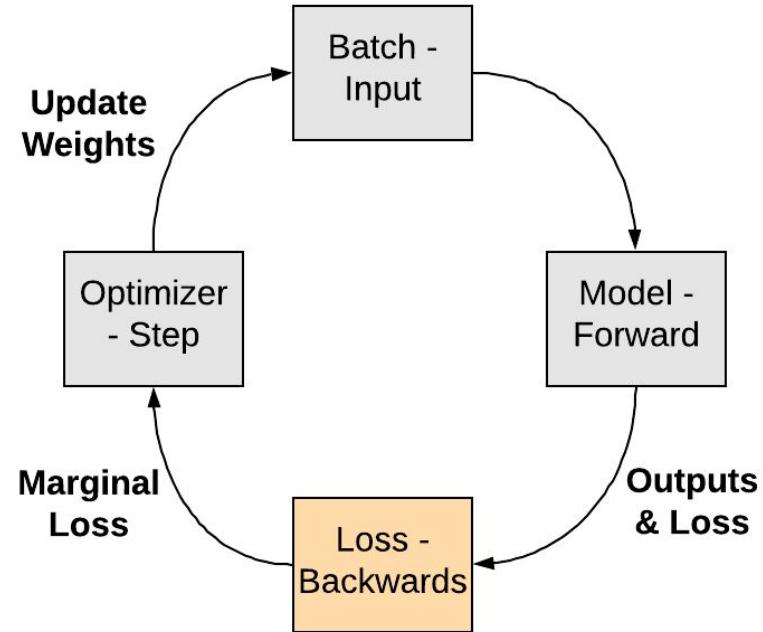
# Clear out the gradients (by default they accumulate)  
optimizer.zero\_grad()  
  
# Forward pass  
loss, logits = bert\_website\_mdl(...)  
  
train\_loss\_set.append(loss.mean().item())  
loss.mean().backward()  
optimizer.step()  
  
# Move logits and labels to CPU  
logits = logits.detach().cpu().numpy()  
label\_ids = batch\_labels.to('cpu').numpy()

**Calculate  
marginal  
loss**

# Model - Train

Loss metric:

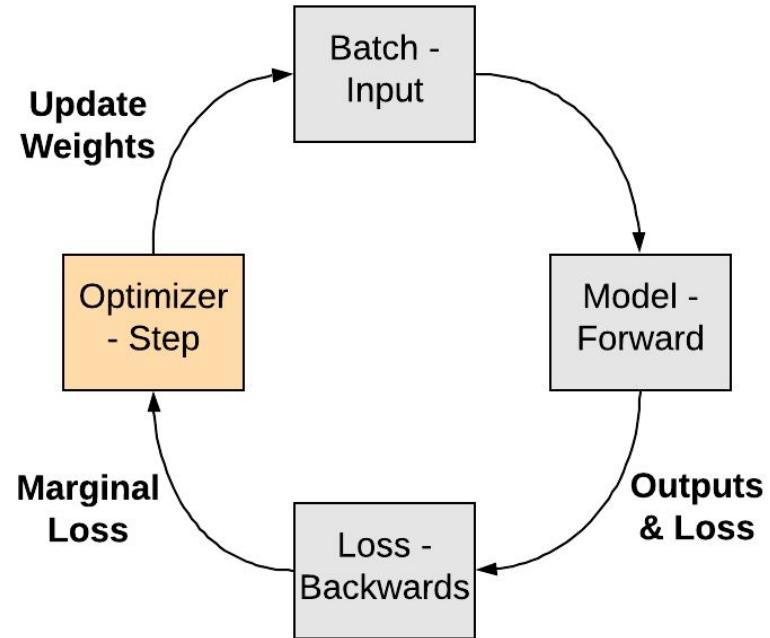
- Compares outputs with labels, calculates actual loss
- “backward” method computes the marginal loss w.r.t all the trainable parameters (i.e the loss gradient). Updates “grad” attribute of tensors.
- E.g: Cross Entropy Loss for multi-class problems



# 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



```
47 param_optimizer = list(custom_bert_mdl.named_parameters())
48 no_decay = ['bias', 'gamma', 'beta']
49 optimizer_grouped_parameters = [
50     {'params': [p for n, p in param_optimizer if
51                 not any(nd in n for nd in no_decay)],
52      'weight_decay_rate': 0.01},
53     {'params': [p for n, p in param_optimizer if
54                 any(nd in n for nd in no_decay)],
55      'weight_decay_rate': 0.0}
56 ]
57
58 optimizer = AdamW(optimizer_grouped_parameters, lr=2e-5)
```

```
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316
```

```
# Clear out the gradients (by default they accumulate)  
optimizer.zero_grad()  
  
# Forward pass  
loss, logits = bert_website_mdl(...)  
  
train_loss_set.append(loss.mean().item())  
loss.mean().backward()  
optimizer.step()  
  
# Move logits and labels to CPU  
logits = logits.detach().cpu().numpy()  
label_ids = batch_labels.to('cpu').numpy()
```

**Clear gradients  
from previous  
batches**

```
289  
290  
291  
292  
293  
309  
310  
311  
312  
313  
314  
315  
316
```

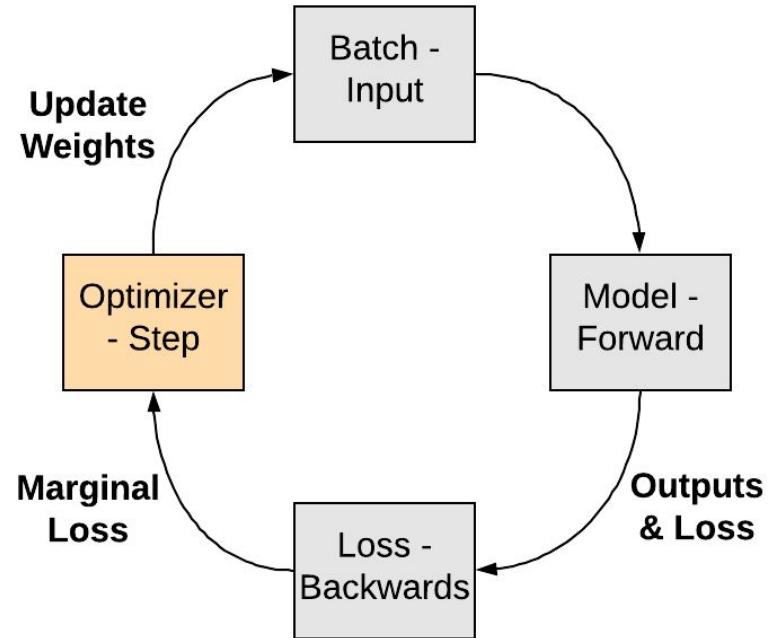
```
# Clear out the gradients (by default they accumulate)  
optimizer.zero_grad()  
  
# Forward pass  
loss, logits = bert_website_mdl(...)  
  
train_loss_set.append(loss.mean().item())  
loss.mean().backward()  
optimizer.step()  
  
# Move logits and labels to CPU  
logits = logits.detach().cpu().numpy()  
label_ids = batch_labels.to('cpu').numpy()
```

**Update weights  
based on  
marginal loss**

# Model - Train

Optimizer:

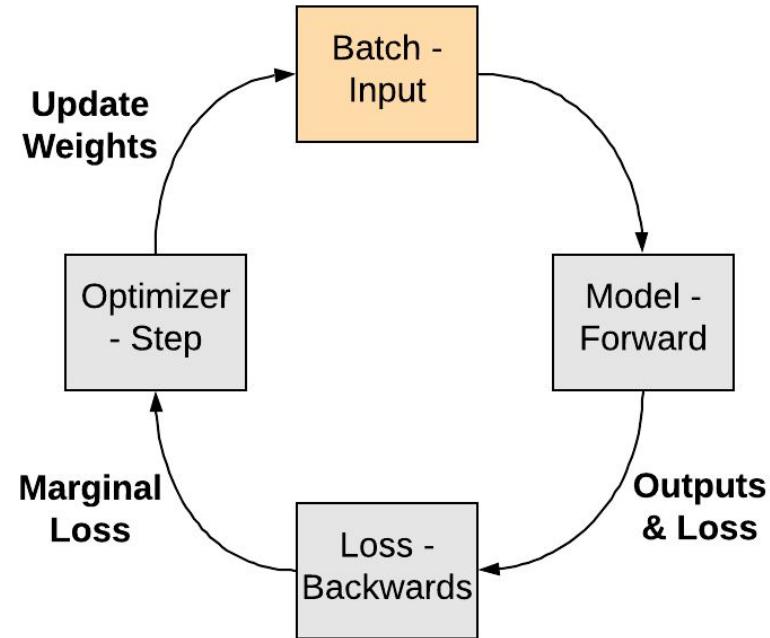
- Specifies which gradient-based optimization method to use for gradient descent
- “step” method updates all relevant weights (i.e. tensor values) according to the marginal loss (stored in the grad attribute of the tensors)
- E.g: AdamW Optimizer



# Model - Train

Batch - Input:

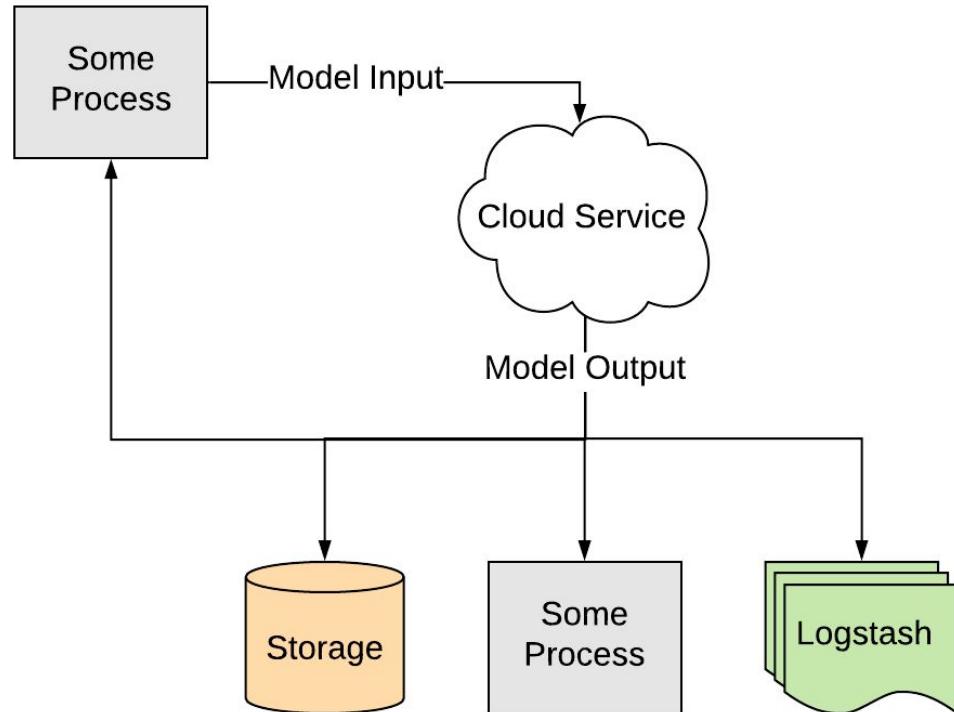
- Next Batch



# Model - Deploy

Goal:

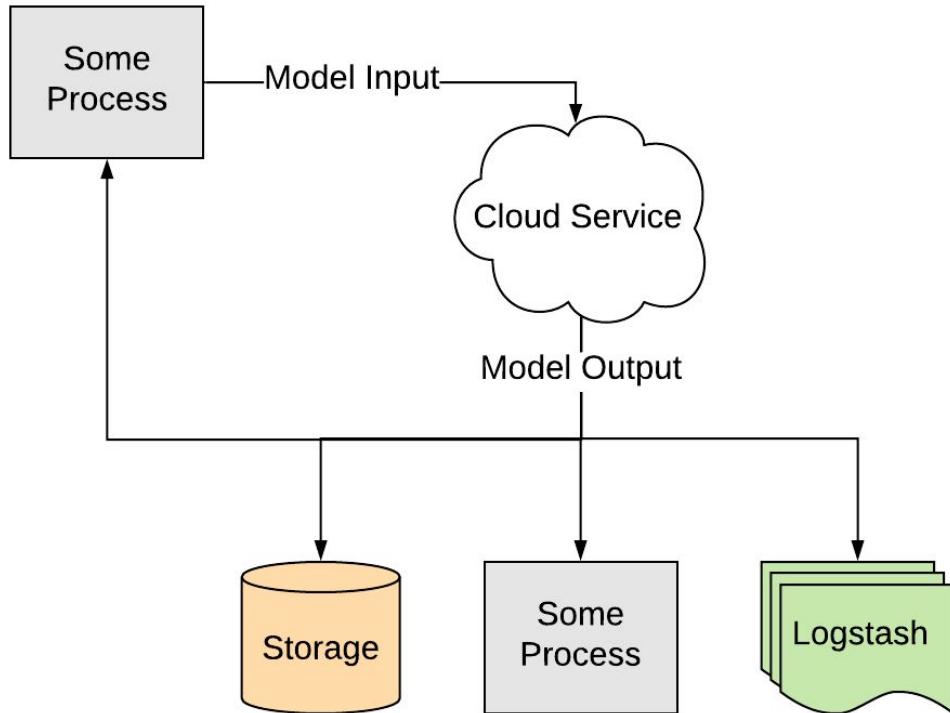
- Model hosted in the cloud
- Always up (persistence)
- Communicates via API
- Can scale



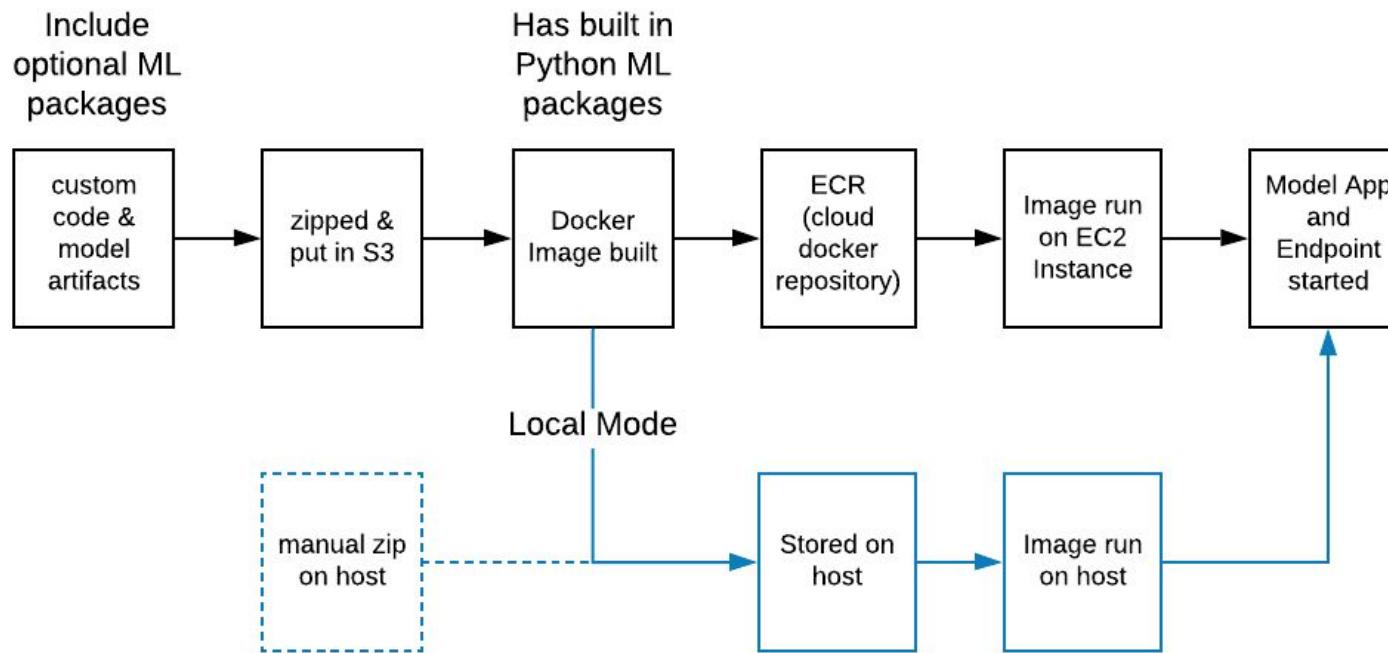
# Model - Deploy

Tool:

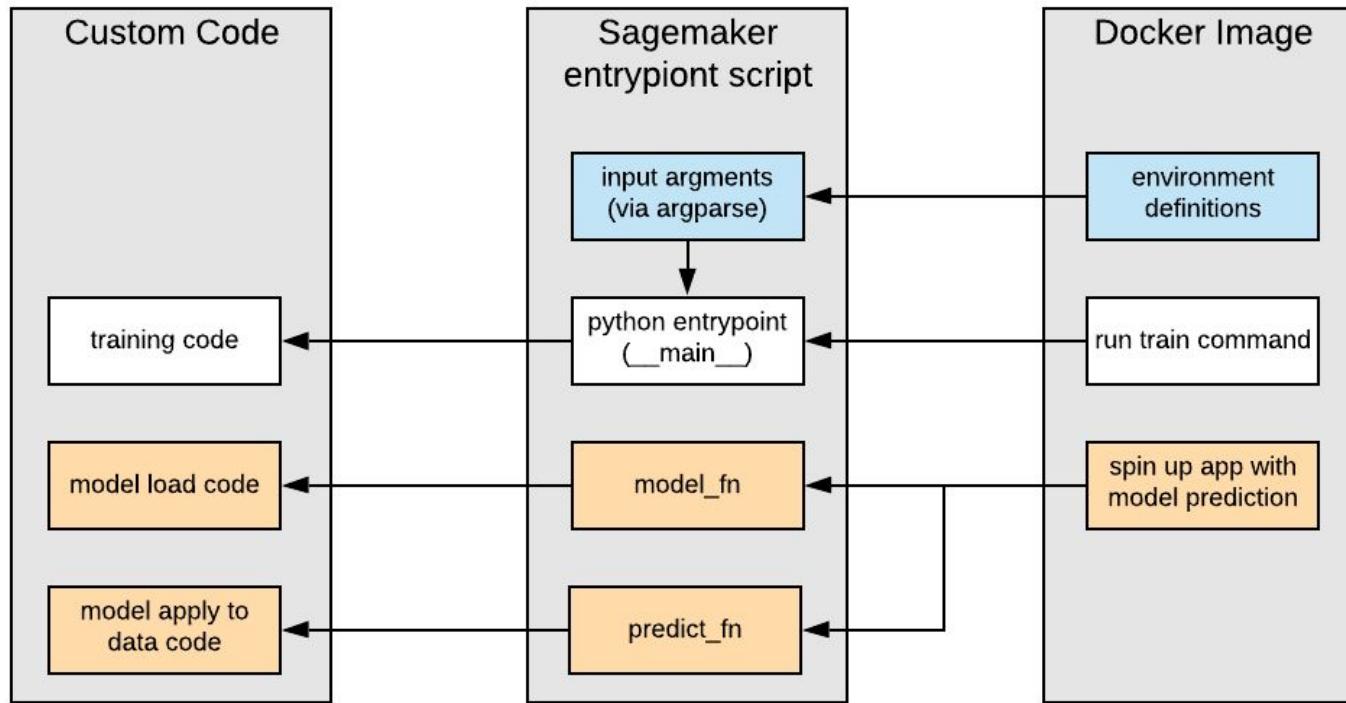
- Amazon SageMaker  
<https://aws.amazon.com/sagemaker/>
- All the pros of docker deployment without writing any docker code
- Supports four modes: Tag, Explore, Train and Deploy
- Auto Scaling



# Model - Deploy (SageMaker)



# Model - Deploy (SageMaker)



```
-- 11  def model_fn(model_dir):  
12      """  
13          initialize resources during service startup  
14      """  
15      # load model  
16      from industry_sagemaker_service.bert_website_classifier \  
17          import BertWebsiteClassifier  
18      bert_mdl = BertWebsiteClassifier.from_pretrained(  
19          os.path.join(model_dir, 'data_files', 'model_e'))  
20  
21      # set up for GPU if exists  
22      device = torch.device("cuda" if torch.cuda.is_available()  
23                      else "cpu")  
24  
25      if device.type == 'cuda':  
26          bert_mdl.cuda()  
27  
28      # load tokenizer  
29      bert_tokenizer = BertTokenizer.from_pretrained('bert-base-cased',  
30                                         do_lower_case=False)  
31  
32      return bert_mdl, bert_tokenizer  
33
```

Load  
trained  
model

```
11 def model_fn(model_dir):  
12     """  
13         initialize resources during service startup  
14     """  
15     # load model  
16     from industry_sagemaker_service.bert_website_classifier \  
17         import BertWebsiteClassifier  
18     bert_mdl = BertWebsiteClassifier.from_pretrained(  
19             os.path.join(model_dir, 'data_files', 'model_e'))  
20  
21     # set up for GPU if exists  
22     device = torch.device("cuda" if torch.cuda.is_available()  
23                         else "cpu")  
24  
25     if device.type == 'cuda':  
26         bert_mdl.cuda()  
27  
28     # load tokenizer  
29     bert_tokenizer = BertTokenizer.from_pretrained('bert-base-cased',  
30                                                 do_lower_case=False)  
31  
32     return bert_mdl, bert_tokenizer
```

Load  
pre-  
trained  
tokenizer



```
11 def model_fn(model_dir):  
12     """  
13         initialize resources during service startup  
14     """  
15     # load model  
16     from industry_sagemaker_service.bert_website_classifier \  
17         import BertWebsiteClassifier  
18     bert_mdl = BertWebsiteClassifier.from_pretrained(  
19             os.path.join(model_dir, 'data_files', 'model_e'))  
20  
21     # set up for GPU if exists  
22     device = torch.device("cuda" if torch.cuda.is_available()  
23                             else "cpu")  
24  
25     if device.type == 'cuda':  
26         bert_mdl.cuda()  
27  
28     # load tokenizer  
29     bert_tokenizer = BertTokenizer.from_pretrained('bert-base-cased',  
30                                                 do_lower_case=False)  
31  
32     return bert_mdl, bert_tokenizer
```

**Return  
all model  
artifacts**

```
66     def predict_fn(urls, crawl_df, model_fn_input):
67         """
68             apply model to input, return inference
69         """
70         bert_mdl, bert_tokenizer = model_fn_input
71
72         from industry_sagemaker_service.bert_website_classifier \
73             import predict_industry
74         output_with_urls = predict_industry(urls, crawl_df, bert_mdl,
75                                             bert_tokenizer, batch_size=16,
76                                             top_n=3)
77
78     return json.dumps(output_with_urls)
79
```



**Get input  
to predict  
on**

**Get model artifacts**

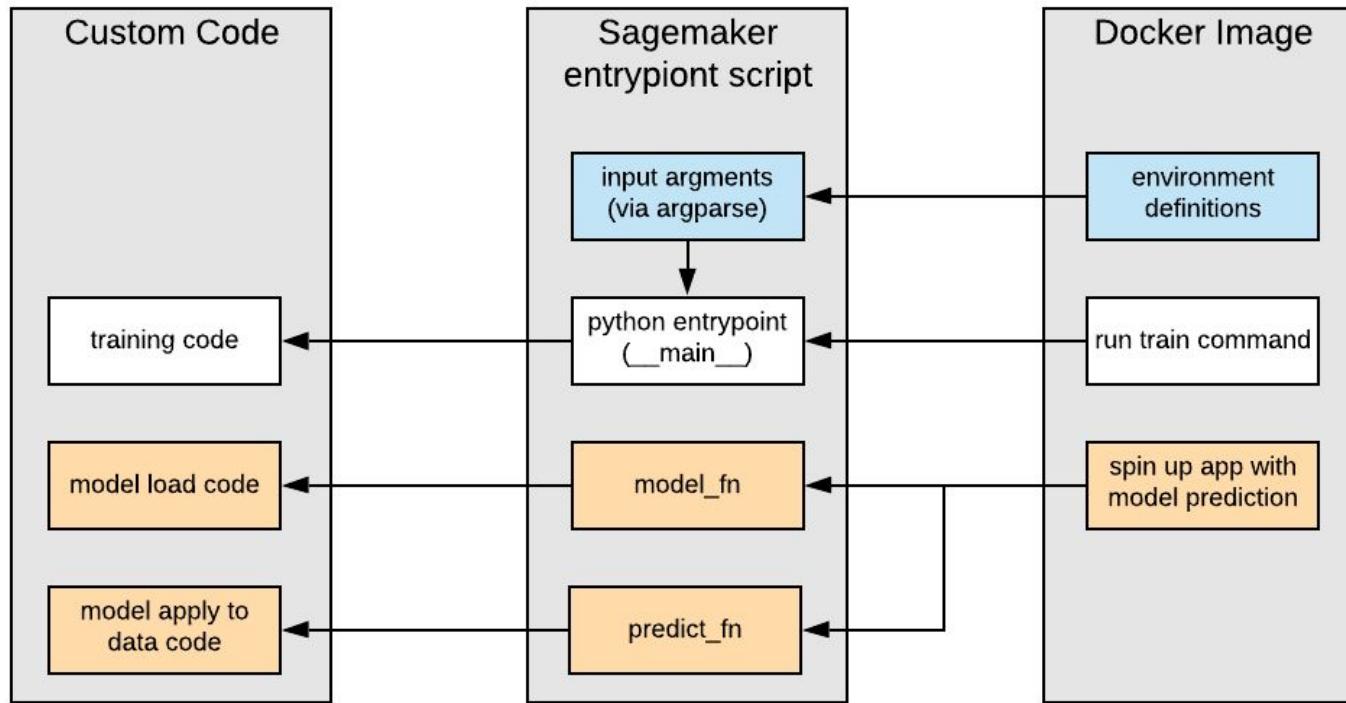
```
65
66  def predict_fn(urls, crawl_df, model_fn_input):
67      """
68          apply model to input, return inference
69      """
70      bert_mdl, bert_tokenizer = model_fn_input
71
72      from industry_sagemaker_service.bert_website_classifier \
73          import predict_industry
74      output_with_urls = predict_industry(urls, crawl_df, bert_mdl,
75                                          bert_tokenizer, batch_size=16,
76                                          top_n=3)
77
78      return json.dumps(output_with_urls)
79
```

```
66     def predict_fn(urls, crawl_df, model_fn_input):
67         """
68             apply model to input, return inference
69         """
70         bert_mdl, bert_tokenizer = model_fn_input
71
72         from industry_sagemaker_service.bert_website_classifier \
73             import predict_industry
74         output_with_urls = predict_industry(urls, crawl_df, bert_mdl,
75                                             bert_tokenizer, batch_size=16,
76                                             top_n=3)
77
78     return json.dumps(output_with_urls)
79
```

Predict



# Model - Deploy (SageMaker)



```
1 import sagemaker
2 from boto3 import Session as BotoSession
3
4 # create sagemaker session
5 boto_session = BotoSession(profile_name='default',
6                             region_name='us-east-1')
7 sagemaker_session = sagemaker.Session(boto_session=boto_session)
8 sagemaker_role = 'ido-sagemaker-test'
9
10 from sagemaker.pytorch import PyTorchModel
11
12 industry_cloud_model = PyTorchModel(
13
14     entry_point='sagemaker_entry_point.py',
15
16     model_data='s3://ido-sagemaker-test/ind_mdl_data.tar.gz',
17
18     framework_version='1.1.0',
19     py_version='py3',
20
21     role=sagemaker_role,
22     source_dir='.')
23
```

## Authenticate with SageMaker service



```
1 import sagemaker
2 from boto3 import Session as BotoSession
3
4 # create sagemaker session
5 boto_session = BotoSession(profile_name='default',
6                             region_name='us-east-1')
7 sagemaker_session = sagemaker.Session(boto_session=boto_session)
8 sagemaker_role = 'ido-sagemaker-test'
9
10 from sagemaker.pytorch import PyTorchModel
11
12 industry_cloud_model = PyTorchModel(
13
14     entry_point='sagemaker_entry_point.py',
15
16     model_data='s3://ido-sagemaker-test/ind_mdl_data.tar.gz',
17
18     framework_version='1.1.0',
19     py_version='py3',
20
21     role=sagemaker_role,
22     source_dir='.')
23
```



**Specify SageMaker  
PyTorch Docker  
container**

```
1 import sagemaker
2 from boto3 import Session as BotoSession
3
4 # create sagemaker session
5 boto_session = BotoSession(profile_name='default',
6                             region_name='us-east-1')
7 sagemaker_session = sagemaker.Session(boto_session=boto_session)
8 sagemaker_role = 'ido-sagemaker-test'
9
10 from sagemaker.pytorch import PyTorchModel
11
12 industry_cloud_model = PyTorchModel(
13
14     entry_point='sagemaker_entry_point.py', ←
15     model_data='s3://ido-sagemaker-test/ind_mdl_data.tar.gz',
16
17     framework_version='1.1.0',
18     py_version='py3',
19
20     role=sagemaker_role,
21     source_dir='.')
22
23
```

**Location of  
entrypoint script**

```
1 import sagemaker
2 from boto3 import Session as BotoSession
3
4 # create sagemaker session
5 boto_session = BotoSession(profile_name='default',
6                             region_name='us-east-1')
7 sagemaker_session = sagemaker.Session(boto_session=boto_session)
8 sagemaker_role = 'ido-sagemaker-test'
9
10 from sagemaker.pytorch import PyTorchModel
11
12 industry_cloud_model = PyTorchModel(
13
14     entry_point='sagemaker_entry_point.py',
15
16     model_data='s3://ido-sagemaker-test/ind_mdl_data.tar.gz',  
17
18     framework_version='1.1.0',
19     py_version='py3',
20
21     role=sagemaker_role,
22     source_dir='.')
23
```

**Location of  
model artifacts  
in S3**

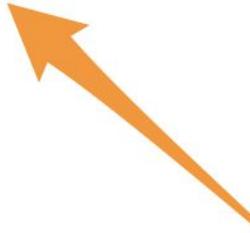


```
1 import sagemaker
2 from boto3 import Session as BotoSession
3
4 # create sagemaker session
5 boto_session = BotoSession(profile_name='default',
6                             region_name='us-east-1')
7 sagemaker_session = sagemaker.Session(boto_session=boto_session)
8 sagemaker_role = 'ido-sagemaker-test'
9
10 from sagemaker.pytorch import PyTorchModel
11
12 industry_cloud_model = PyTorchModel(
13
14     entry_point='sagemaker_entry_point.py',
15
16     model_data='s3://ido-sagemaker-test/ind_mdl_data.tar.gz',
17
18     framework_version='1.1.0',
19     py_version='py3',
20
21     role=sagemaker_role,
22     source_dir='.')
23
```

**Specify PyTorch  
and Python  
versions**

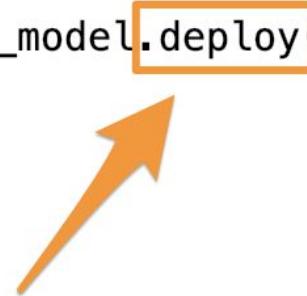


```
24 industry_cloud_predictor = industry_cloud_model.deploy(  
25     initial_instance_count=1,  
26     instance_type='ml.p2.xlarge')
```



**Specify desired  
EC2 instance**

```
24     industry_cloud_predictor = industry_cloud_model.deploy  
25         initial_instance_count=1,  
26         instance_type='ml.p2.xlarge')
```



**Call "deploy"  
method on the  
model**

EndpointsUpdate endpointActions ▾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	<span>✓ InService</span>	Oct 30, 2019 04:00 UTC
sagemaker-scikit-learn-2019-10-08-21-54-08-741	[REDACTED]	Oct 08, 2019 21:54 UTC	<span>✓ InService</span>	Oct 25, 2019 19:34 UTC

```
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)  
38
```

**Specify  
input will be  
JSON**



```
--  
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 "predict"  
method**



# 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](mailto:ido.shlomo@bluevine.com)