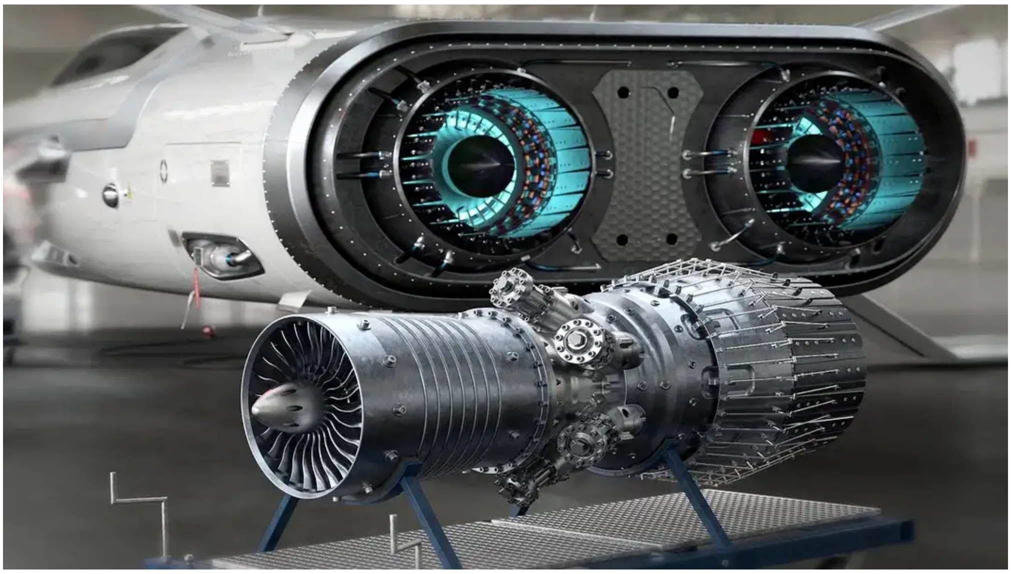
lished in PyTorch

inghui Ouyang

pr 8, 2021 · 7 min read · Listen

We Used AWS Inferentia to Boost PyTorch Model Performance by 4.9x for the desk Ava Chatbot



k is a multinational software company with world-renowned s in areas such as Architecture, Engineering, & Construction, cturing, and Media & Entertainment. Amongst Autodesk's best- products are AutoCAD, Revit, Maya, and Fusion 360. The company ions of customers around the world, and many of them have need ort to make best use of their products.

of the process of improving the customer support experience, the y developed AVA, the Autodesk Virtual Agent. Ava is Autodesk's r support chatbot. The front end consists of a dynamic web ent, which can be embedded in different sites and applications.

e six NLP models that comprise part of the backend of AVA that the best response or next action presented to the customer, based on ut. For example, one of the NLP models is the Intent Model, which s a customer's natural language input into tasks such as 'introducing information', 'initiating product downloads', and 'helping manage tions'. AVA answers over 100,000 customer questions per month by natural language understanding (NLU). Therefore, both the speed of model inference is important to ensure good customer ice with AVA.

erentia is the first Machine Learning chip by AWS, which promises ve the highest throughput at almost half the cost per inference when ed with GPUs. Given the need for a high-quality, efficient service, we to test and benchmark the performance of Inferentia on the Intent

ing the Benchmark

Intent Model is a BERT Sequence Classification model using 1 and the Huggingface library (version 3.4.0) 2. AWS Neuron is a development kit (SDK) for running machine learning inference VS Inferentia chips. It is integrated into PyTorch to run inference. uron, ML developers could compile a pretrained BERT model, and un-time, and profiling tools to benchmark the performance of the e.

odels represent a popular example of a Transformer model. These are large, with hundreds of millions or more parameters, and y are built in two stages. The first is the training of the base language nd the second is the creation of a task-specific fine-tuned model.

e in transformer models is computationally expensive, and y inefficient and memory-intensive on CPU-based architectures. PU-based systems deliver performance, they can be costly. It is to use quantization, distillation, or other approaches to create a model that is less costly, but in the end the use of another ture is appealing in the AVA Case.

ne key factors that led to interest in Inferentia is that chatbots have a predictable, lower-latency responses. Since chatbot requests often dependently instead of as a group, the inference needs to perform low batch size setting. As models are chained or called in parallel, ty to have stable, scalable throughput is essential. It is more difficult batches or queues of inputs for inference. This need for stability, put, cost-efficiency, and small batch sizes made the choice of ial potentially attractive.

ng a Model using Neuron

ally, there was a large separation between training and inference cture in many deep learning models. For example, before the merged, many models trained in PyTorch were ported to Caffe2 for e. One lesson from this era is that any new approach to inference ep the process of cross-compilation as short and simple as possible longer feasible to devote significant time purely to re-engineering or production.

entia, the process is described below, and is largely automatic. The re is to take the conventionally-trained model, and perform a 'trace', mpiling it for the new hardware. There are a few lines of code to at traced model, and then the remaining inference code is as The same code can be used for inference on custom hardware, with model files. This is a great advantage for testing and engineering dels quickly.

lel compilation was done within SageMaker. We first loaded the fine- tent Model in a SageMaker Notebook using delForSequenceClassification from Hugging Face transformers

```
zer = AutoTokenizer.from_pretrained(model_path)
=
```

```
delForSequenceClassification.from_pretrained(model_path)
```

use torch.neuron.trace from AWS Neuron to generate a Torchscript ptimized by AWS Neuron, and save the script for later use.

```
neuron = torch.neuron.trace(model, example_inputs,
er_args=['-O2'], verbose=1, compiler_workdir='./compile')

 the TorchScript for later use
neuron.save('intent_neuron.pt')
```

d now test the model inference within SageMaker notebook. The input we used for the test was “change company address”, which is n intent of changing customer information.

```
p some example inputs
 "change company address"

s = tokenizer.encode_plus(docs, max_length=512,
_max_length=True, return_tensors="pt")

e_inputs = tensors['input_ids'], tensors['attention_mask'],
s['token_type_ids']

ing the prediction result for input example
fication_logits_neuron = model_neuron(*example_inputs)
 =
nn.functional.softmax(classification_logits_neuron[0]).detach()
.numpy()
```

ting, we find that the model optimized by AWS Neuron returns the predicted intents. Therefore, we now move onto deploy the model. to deploy the compiled model, we need to upload the compiled neuron.pt’ Torchscript file onto an S3 bucket. From there, we can n Amazon EC2 Inf1 instance, and copy paste and deploy the ript file into the Inf1 instance.

arking

ve test model inference on an Inferentia chip, we need to create an Learning Inf1 instance. This webpage shows the process of how to e instance.

use the following command to SSH into the Inf1 instance in our ıd line. Note that you need to save your AWS pem key file in your directory when doing the SSH. You also need to make sure that the rule of your Inf1 instance allows your IP address to SSH into it. You t this up in the security settings of your Inf1 instance.

```
 [pem key file] ec2-user@[IP address of your Inf1 instance]
```

owing code runs the benchmark process for the model inference, could save the benchmark result into a CSV file.

```
 []
t_num_infers = []
hputs = []
 []
 []

um_infer = num_infer
in range(args.throughput_time // args.throughput_interval):
rrent_num_infer = num_infer
roughput = (current_num_infer - last_num_infer) /
hroughput_interval
0 = 0.0
0 = 0.0
 latency_list:
  p50 = np.percentile(latency_list[-args.latency_window_size:],

  p90 = np.percentile(latency_list[-args.latency_window_size:],

ds.append(os.getpid())
rrent_num_infers.append(current_num_infer)
roughputs.append(throughput)
0s.append(p50)
0s.append(p90)

'pid {}: current infers {} throughput {:.3f}, latency p50=
 p90={:.3f}'.format(os.getpid(), current_num_infer,throughput,
90))
s.stdout.flush()
st_num_infer = current_num_infer
me.sleep(args.throughput_interval)
 live
 False
p = pd.DataFrame({'pid':pids,
nt_num_infer':current_num_infers,
hroughput':throughputs,'p50':p50s,'p90':p90s})
p.to_csv('benchmark_dump_neuron_v3.csv', index=False)
```

ilar Benchmark Steps for the Same Model deployed in a G4

e

the model in a SageMaker notebook, upload it onto S3. After we
an EC2 G4 instance, we copy the model files from S3 to the instance.
OC, we used a g4dn.xlarge instance. SSH into the G4 instance, and
similar inference and benchmark scripts as we did for the Inf1
.

n is a comparable instance chosen as one of the most popular for
erence.

nark Result

ferentia, we were able to obtain a 4.9x higher throughput over g4dn
ntent Model for AVA.

owing table shows the throughput and latency of model inferences
ch size equal to one in Inf1 instance. Here throughput is defined as
of inferences per second. Latency is defined as the number of
it takes for the model inference. Latency_p50 is the 50 percentile of
tency, while latency_p90 is the 90 percentile of model latency.

pid	current_num_infer	throughput	latency_p50	latency_p90
20848	0	0.000	0.000	0.000
20848	3	0.300	5.797	8.031
20848	1176	117.300	0.234	0.239
20848	2543	136.700	0.234	0.236
20848	3914	137.100	0.233	0.236
20848	5282	136.800	0.233	0.238
20848	6651	136.900	0.233	0.235
20848	8021	137.000	0.233	0.235
20848	9391	137.000	0.233	0.235
20848	10762	137.100	0.233	0.235
20848	12134	137.200	0.233	0.235
20848	13504	137.000	0.233	0.236
20848	14872	136.800	0.233	0.236
20848	16235	136.300	0.234	0.238
20848	17606	137.100	0.234	0.235
20848	18974	136.800	0.234	0.237
20848	20347	137.300	0.233	0.235
20848	21719	137.200	0.233	0.234
20848	23085	136.600	0.233	0.238
20848	24453	136.800	0.233	0.237
20848	25822	136.900	0.233	0.235
20848	27190	136.800	0.233	0.235
20848	28559	136.900	0.233	0.235
20848	29927	136.800	0.233	0.236
20848	31293	136.600	0.234	0.238
20848	32662	136.900	0.233	0.235
20848	34032	137.000	0.233	0.236
20848	35402	137.000	0.233	0.235
20848	36776	137.400	0.233	0.235
20848	38143	136.700	0.233	0.240

›wing table shows the throughputs and latencies of model inference
:tance.

21238	0	0	0	0
21238	257	25.7	0.388	0.451
21238	538	28.1	0.388	0.412
21238	821	28.3	0.389	0.411
21238	1101	28	0.391	0.41
21238	1377	27.6	0.393	0.412
21238	1652	27.5	0.396	0.415
21238	1928	27.6	0.398	0.418
21238	2201	27.3	0.401	0.42
21238	2470	26.9	0.404	0.423
21238	2738	26.8	0.407	0.426
21238	3004	26.6	0.41	0.429
21238	3267	26.3	0.412	0.431
21238	3531	26.4	0.416	0.434
21238	3790	25.9	0.419	0.438
21238	4048	25.8	0.423	0.441
21238	4301	25.3	0.426	0.446
21238	4554	25.3	0.43	0.45
21238	4806	25.2	0.435	0.456
21238	5050	24.4	0.438	0.461
21238	5303	25.3	0.441	0.462
21238	5545	24.2	0.443	0.464
21238	5797	25.2	0.444	0.464
21238	6041	24.4	0.445	0.464
21238	6293	25.2	0.444	0.464
21238	6542	24.9	0.443	0.463
21238	6790	24.8	0.443	0.462
21238	7039	24.9	0.442	0.462
21238	7288	24.9	0.442	0.461
21238	7540	25.2	0.442	0.461

Frank Andrade in Geek Culture

Hey ChatGPT, Automate These Tasks Using Python

Anmol Tomar in CodeX

Say Goodbye to Loops in Python, and Welcome Vectorization!

Ugur Yagmur in CodeX

5 Harsh Truths from Linus Torvalds

Jacob Ferus in Level Up Coding

ChatGPT Is Mind-Blowing—Everything You Need To Know

sion

chmark results show an almost five-fold increase in the throughput tent model inference in an Inf1 instance compared to model e in a G4 instance, while having approximately half of the latency. cessful proof of concept encourages us to deploy more models in on on Inferentia in the future.

ımples of benchmark experiences on various NLP applications with 5% reduce of cost, we are looking forward to testing Inferentia on

Help Status Writers Blog Careers Privacy T
Text to speech

' models in production. When we get the benchmark results there, ıave more information regarding the cost savings.

nore general perspective, the simplicity of the process makes this an e option for models that have predicted traffic suitable for inf1 use. ı, stable throughput and lower cost make it especially helpful in s where small or fixed batches are required, as well as always-on ity. In addition, the nature of the neuron sdk cross-compilation ıat the deployment can be easily automated — adding custom e to the model can be done as part of a standard approach to ıent with only a few extra steps.

Inferentia Machine Learning Data Science Pytorch