Applied Accelerated Artificial Intelligence Dr. Tosin Adesuyi Department of Computer Science and Engineering Indian Institute of Technology, Palakkad

End to End Accelerated Date Learning Lecture - 31 Optimizing Deep Learning Training: Automatic Mixed Precision Part - 2

(Refer Slide Time: 00:18)



So, ok let us look at the DEMO here. So, I will go to demo here alright.

(Refer Slide Time: 00:25)



So, the first one I will be dealing with is the Mixed Precision. So, for mixed precision you have to use TensorFlow version, because I will be dealing with tensorflow first tensorflow version 2. So, I have pulled my so this is the tensorflow this is 21.12-tf2 this tf2 shows that this version 2. So, I will be running that because I am running my workstation.

(Refer Slide Time: 00:58)



So, I am just using ok.

(Refer Slide Time: 01:09)



So, I am gonna connect here, you may not have to go through all this process.

(Refer Slide Time: 01:18)



If you have your tensorflow on your workstation or on your laptop maybe via conda or virtual environment it is because I am using a container that is why I am going through this process ok.

(Refer Slide Time: 01:35)

NVIDIA. NGC CATALOG		Welcome
CATALOG A Citalog - Containes - TensorFlow Explore Catalog Cotectores	r	Pull Ta
Containes Helm Druns Models Rebounds	(19.07-py3 (1131/2019 7.45 AM 2.82 GB 1 Architecture	
	80 19.07-py2 01031021973554M 2.66 GB 1 Architecture	 بل Pull Tag
	80 19.06-py2 80282019580 AM 2.88 GB 1 Architecture	
	8 19.06-py3 ISC2020191557 AM 2.28 GB 1 Architecture	
	80 19.05-py2 ISS242019-581 AM 2.83 GB 1 Architecture	2
Collapse	10.05 2	

(Refer Slide Time: 01:41)

G gpu compute	se x W CUDA-Wilepedi x W Single-precision x W Half-precision fin x 🗞 FP64.FP32.FP16 x W Double-precision x 🚳 TensorRow N/T x 🔞 New Tab	× +	v	×
$\ \ \leftarrow \ \ \rightarrow \ \ G$	C loalhost8080			* 🗆 🌒 🛉
II App	C Home - localhost 8080		×	NPTEI
	R, Io - Google Search			as III 👔
	③ logistics box			
1	Login Charles Schwab - schwab.com/p.			
	Immigration Connect by Fragomen - immigration.fragomen.net/login			
	R mixed_precision_thr1.pymb	3 hours ago	2.62 kB	
	R mixed grecision_th/2.ipynb	3 hours ago	8.04 kB	
	🗌 🥔 Untitled ipymb	8 hours ago	13.3 kB	
localhost/8080/noteb	loks/mand_precision_t%2.bymb			
🧃 kit-1.pdf	A			Show all X
ا ۹	i 🤻 🗷 🥼 🗮 🛃 🔚 🙋 🧿 🎼 🦺 📓 🏧 🏶 🕒 6°C Cloudy 🗛 🖡 🖉	• • 6 •	11) d ⁶ EN	902 PM

So, here I have local host ok. So, I will be dealing with version 2 first of all.

(Refer Slide Time: 01:48)

JUDYTET mixed_precision_tfv2 Last Checkpoint: 3 hours ago (autosaved)	ð
File Edit View Insert Cell Kernel Help	Not Trusted 🖋 Python 3 (ipykernel) 0
B + 3< 62 K3 + ↓ ▶ Run ■ C >> Code ∨ E8	
<pre>In []: M import tensorflow as tf</pre>	
<pre>from tessorflow.import kerss from tessorflow.kerss import layers from tessorflow.kerss import mixed_precision import os print(tfversion_)</pre>	
<pre>bs.environ['IF_ENWELE_AUTO_MIXED_PRECISION'] = '1' #policy = wixed_precision.Policy('Float32')</pre>	
<pre>policy = mixed_precision.Policy('mixed_float16') mixed_precision.set_global_policy(policy)</pre>	
<pre>print('compute dtype: %s'% policy.compute_dtype) print('variable dtype: %s'% policy.variable_dtype)</pre>	
<pre>In []: W inputs = keras.Input(shape=(784,), name='digits') if tf.config.list.physical_devices('GPU'): print('The model kill nu mitth 4956 units on a GPU')</pre>	
24. 4994	. Alma

So, this is just I just put these together adapted from tensor flow website, I hope you can see my screen.

Student: Yes Tosin we can see the Jupyter Notebook.

Thank you very much. So, I just speak a simple example that so that we can actually understand very well, then on our own we can later on do complex example because of our time. So, the first thing here is that we import our tensorflow and then we from tensorflow we get our keras and then we import the layers we wanted to use here, then from tensorflow keras we import the mixed precision.

Now, please note what we are trying to do here is actually we are using mixed precision, it is not the automatic mixed precision. Mixed precision and automatic mixed precision are the same thing. So, the difference is that automatic mixed precision you do not need to do some manual job you just with 1 line or 2 everything is sets its done for you. But with mixed precision you have to do some level of manual stuff yourself. So, we are dealing with mixed precision here because, mixed precision only is possible with tensorflow version 2. So, let me comment this one out first.

(Refer Slide Time: 03:16)



So, the first thing you do is that because we are dealing with mixed precision we need to work with a float16, fp 16. So, you have to what use your mixed precision library I have imported the class then you call this method Policy. So, we set policy for one mixed precision that is you use what mixed of what float 16. So, when you set that into the policy then you can take your word you make your mixed precision set global policy then you put the policy in there.

So, when you put the policy in there you need to confirm if it is actually sets. So, for us to confirm we need to get the word the policy.compute_dtype what is the type is going to

use for computation and also policy.variable_dtype that is what is the variable type. Remember mixed precision is the use of fp16 and fp32.

(Refer Slide Time: 04:18)



So, I will run this notes book here. So, let us see what happened oh good. So, it is showing us that you can see this is a version that is version is what tensorflow version 2, then we say tensorflow mixed precision compatibility check that is mixed precision that is it is ok it is compatible. We can use also your GPU will likely to run quickly with this policy of mixed float 16, also if the compatibility is at least that is your computes capacity must be at least 7.0 which I have told you before.

(Refer Slide Time: 04:54)

			MAJO, MAJO, MALO, MALO, MATO, MATO, MATO	P3000(Mobile)		1
62		GP108 ^[41]				Tegr NA Jetson TX DRIVE PX
7.0		GV100	NVIDIA TITAN V	Quadro GV100	Tesla V100, Tesla V100S	
72	Vota	GV118 ¹⁴³				Tegra Xavi Jetson Xavie Jetson AGX X DRIVE AGX X DRIVE AGX Pegasus, O AGX
75	Turing	TU102, TU104, TU108, TU116, TU117	MVDIA TITAN RTX, GeFarce RTX 2080 TI, RTX 2080 Super, RTX 2080, RTX 2070 Super, RTX 2070, RTX 2080 Super, RTX 2080 T268, RTX 2080, GeFarce GTX 1580 TI, GTX 1580 Super, GTX 1580, GTX 1580 Super, GTX 1580, UKX50, MM450	Quadro RTX 8000, Quadro RTX 6000, Quadro RTX 5000, Quadro RTX 4000, T1000, T600, T400 T1200(mobile), T600(mobile), T500(mobile), Quadro T2000(mobile), Quadro T1000(mobile)	Testa T4	
8.0		GA100			A100 80GB, A100 40GB, A30	
8.6	Ampere	GA102, GA104, GA105, GA107	GeForce RTX 3090 TI, RTX 3090, RTX 3090 TJ, RTX 3090 125B, RTX 3080, RTX 3070 TJ, RTX 3070, RTX 3060 TJ, RTX 3090, RTX 3050, RTX 3050 Ti(mobie), RTX 3050(mobie), RTX 2050(mobie), IMS70	RTX A6000, RTX A5000, RTX A4500, RTX A4000, RTX A2000 RTX A5000(mobile), RTX A4000(mobile), RTX A3000(mobile), RTX A2000(mobile)	A40, A16, A10, A2	
87		GA108				Jetson Orin Jetson AGX (DRIVE AGX) Clara Holos
9.0	Hopper	GH100, GH202				
9x	Lovelace	AD102, AD104, AD106, AD107				
9 x		AD10B				
Compute capability (version)	Micro- architecture	GPUs	Gellorce	Quadro, NVS	Tesla/Dotacenter	Tegra, Jetson, DRIVE

So, you have to you know if you go to Wikipedia you can see from here that from Volta.

(Refer Slide Time: 04:56)



This is where we have 7.0 Pascal is lesser than the compute.

(Refer Slide Time: 05:04)

2.1	Ferni Kopler	GF104, GF108 GF108, GF114, GF116, GF117, GF119	Carling of DBL polymory DBL (Seriess 0) 1282, Ordenson 31 484, Ordenson 01 484, Ordenson 21 485, Ordenson 31 687, Ordenson 01 484, Ordenson 21 485, Ordenson 31 687, Ordenson 01 47684, Ordenson 21 784, Ordenson 21 4784, Ordenson 21 7834, Ordenson 21 1384, Ordenson 21 7834, Ordenson 21 7834, Ordenson 21 7844, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78344, Ordenson 21 78444, Ordenson 21 63344, Ordenson 21 78344, Ordenson 21 78444, Ordenson 21 63344, Ordenson 21 78344, Ordenson 21 78445, Ordenson 21 63344, Ordenson 21 78344, Ordenson 21 78445, Ordenson 21 63344, Ordenson 21 78344, Ordenson 21 78454, Ordenson 21 63344, Ordenson 21 78344, Ordenson 21 78344	Quadri 2000, Quadri 20000, Quadri 4000, Quadri 40000, Quadri 300000, Quadri 20000, Quadri 10000, NNS 310, NNS 315, NNS 54000, INS 52000, INS 420000		N	
3.0		GK104, GK106, GK107	Advisor CUT, 70), Orderson CUT, 700, Orderson C	Quadro K5000, Quadro K4200, Quadro K4000, Quadro (2000), Quadro X5000, Quadro K500, Quadro K500, Quadro X5000, Quadro K5100, Quadro K5000, Quadro K70000, Quadro K5000, Quadro K1000, Quadro K40004, Quadro K50000, Quadro K1000, Quadro K40004, Quadro K50000, Quadro K1000, Quadro K40004, Quadro K50000, Quadro K1000, Quadro K40000, NVS 510, Quadro K100	Tesla K10, GRD KG40, GRD K520, GRD K2		
32			GK20A				Tegra K1 Jetson TK
35			GK110, GK208	GeForce GTX Tran Z, GeForce GTX Tran Black, GeForce GTX Tran, GeForce GTX 780 TI, GeForce GTX 780, GeForce GT 640 (GEOR5), GeForce GT 302, GeForce GT 730, GeForce GT 720, GeForce GT 740M (64-bit, DDR3), GeForce GT 520M	Quadro K8000, Quadro K8200	Tesia K40, Tesia K20x, Tesia K20	
3.7		GK210			Tesla KB0		
5.0	Maxwell	GM107, GM108	GeForce GTX 750 TI, GeForce GTX 750, GeForce GTX 980M, GeForce GTX 950M, GeForce 940M, GeForce 930M, GeForce GTX 860M, GeForce GTX 850M, GeForce 845M, GeForce 840M, GeForce 830M	Quadro K1200, Quadro K2200, Quadro K820, Quadro M2000M, Quadro M1000M, Quadro M600M, Quadro K820M, NVS 810	Tesla M10		
5.2		GM200, GM204, GM206	GeForce GTX Titan X, GeForce GTX 880 Ti, GeForce GTX 980, GeForce GTX 970, GeForce GTX 980, GeForce GTX 950, GeForce GTX 750 SE, GeForce GTX 980M, GeForce GTX 970M, GeForce GTX 955M	Quadro M6000 24GB, Quadro M6000, Quadro M6000, Quadro M4000, Quadro M2000, Quadro M5600, Quadro M5000M, Quadro M4000M, Quadro M3000M	Tesla MH, Tesla M40, Tesla M6, Tesla M60		
						Tegra X	
						Selenn 7	

(Refer Slide Time: 05:06)



The computes are capacity capability this is compute capability here the first column. So, you can see for Pascal is less than that then from Volta you can access if your GPU is between these 7.5 this turing here. If you are using GeForce RTX 380 and all the lights.

(Refer Slide Time: 05:25)

40 640 64700 1648 P100 1648 P1000 1648 P100		5.3		GM208					Jetson Nago DRIVE CK DRIVE PX
1 Particle GPUR		6.0		GP100			Quadro GP100	Tesla P100	
62 grageri production grageri production grageri production model model <td></td> <td>6.1</td> <td>Pascal</td> <td>GP102, GP104, GP106, GP107, GP108</td> <td>GeForce GTX 1080 TL GTX 1050 MX350, MX330,</td> <td>Wda TTTAN Xp, Tran X, GTX 1080, GTX 1070 T, GTX 1070, GTX 1060, T, GTX 1050, GT 1030, GT 1010, MX250, MX230, MX150, MX130, MX110</td> <td>Quadro P6000, Quadro P5000, Quadro P4000, Quadro P2200, Quadro P2000, Quadro P1000, Quadro P400, Quadro P500, Quadro P520, Quadro P600, Quadro P5000(Moble), Quadro P4000(Moble), Quadro P3000(Moble)</td> <td>Tesla P40, Tesla P6, Tesla P4</td> <td></td>		6.1	Pascal	GP102, GP104, GP106, GP107, GP108	GeForce GTX 1080 TL GTX 1050 MX350, MX330,	Wda TTTAN Xp, Tran X, GTX 1080, GTX 1070 T, GTX 1070, GTX 1060, T, GTX 1050, GT 1030, GT 1010, MX250, MX230, MX150, MX130, MX110	Quadro P6000, Quadro P5000, Quadro P4000, Quadro P2200, Quadro P2000, Quadro P1000, Quadro P400, Quadro P500, Quadro P520, Quadro P600, Quadro P5000(Moble), Quadro P4000(Moble), Quadro P3000(Moble)	Tesla P40, Tesla P6, Tesla P4	
72 0100 MODATINA V Dade 01700 Heavy Y00, Heav		62		GP108 ^[41]					Tegra X2, Jetson TX2, DRIVE PX 2
Del 7 2 Value OntgRR 72 Value OntgRR 73 Wolds in the codemane for a GPU microarchitecture developed by Nidds, a randome in Nanch 2013, although the total manufold a randome in Nanch 2013, although the total manufold a randome in Nanch 2013, although the total sector the authentication developed by Nidds, a sector the sector total May 2017. The authentication sector total manufold by Sector TX The authentication sector total manufold by Sector TX Sector TX SEC TX SE		7.0	Voita Voita micros succe a road produ The a centur	GV100		NVIDIA TITAN V	Quadro GV100	Tesla V100, Tesla V100S	
1 Walk is the concentration for GPU model, successfory percent hysical search of an anomaly in Merch 2013, alfored by Model, and the for anomaly model are to anomaly in Merch 2013, alfored by Model, and the for anomaly model are to anomaly in Merch 2013, alfored by Model, TIX ADD, RTX ADD, See, RTX, ADD, Merch 2013, alfored by Model, TIX ADD, RTX ADD, See, RTX, ADD, Merch 2013, alfored by Model, TIX ADD, RTX ADD, See, RTX, ADD, Merch 2013, alfored by Model, TIX ADD, RTX A		₽ 72		GV108 ^[42]					Tegra Xavier Jetson Xavier M Jetson AGX Xa DRIVE AGX Xa
287 Angree Angree <td></td> <td></td> <td>is the codename t architecture devel</td> <td>for a GPU oped by Nvidia, as first appounded on</td> <td></td> <td></td> <td></td> <td>DRIVE AGX Pegasus, Clar AGX</td>				is the codename t architecture devel	for a GPU oped by Nvidia, as first appounded on				DRIVE AGX Pegasus, Clar AGX
B2 Control Control And Add Add and Add Add and Add Add Add Add Add Add Add Add Add A		7.5		imap in March 20 ct was not annour rchitecture is nam ry Italian chemist i	13, although the first need until May 2017. ed after 18th–19th and physicist	WIDIA TITAN RTX, 2080 Super, RTX 2080, RTX 2070 Super, RTX Super, RTX 2080 12GB, RTX 2080, 1680 Super, GTX 1660, GTX 1650 Super, GTX 550, MX450	Quadro RTX 8000, Quadro RTX 6000, Quadro RTX 5000, Quadro RTX 4000, 11000, 1600, 1400 T1200(mobile), T800(mobile), T500(mobile), Quadro T2000(mobile), Quadro T1000(mobile)	Tesla T4	
55 OAXE_CAND Operating and proceeding and proceding and proceeding and proceding and		8.0			¢			A100 80GB, A100 40GB, A30	
0 notest. 57 Oktober 2000 2000 2000 2000 2000 2000 2000 20		8.6	Ampere	GA102, GA104, GA106, GA107	GeForce RTX 3090 Ti, 3080; RTX 3070 Ti, RTX 3050 Ti(mobile); RT	RTX 3090, RTX 3080 Ti, RTX 3080 12GB, RTX 3070, RTX 3060 Ti, RTX 3060, RTX 3050, RTX X 3050(mobile), RTX 2050(mobile), MX570	RTX A6000, RTX A5000, RTX A4500, RTX A4000, RTX A2000 RTX A5000(mobile), RTX A4000(mobile), RTX A3000(mobile), RTX A2000(mobile)	A40, A16, A10, A2	
Its incoarchiteture) Davie M	sita (mic	8.7 roarchiter	tunt)	GA108					Jetson Orin N Jetson AGX O DRIVE AGX O Clara Holmer

So, you can check the architecture of your GPU here. So, we can proceed from there. So, we know that so the next thing we do is that we want to work with mnist.

(Refer Slide Time: 05:35)

JUpyter mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved chang	ges) 🥐
File Edit View Insert Cell Kernet Help	Not Trusted Python 3 (pykernel)
B + 3 0 B + ↓ Pan B C → Code ✓ B	
2.6.2 IMFO:tensorfloweRixed precision compatibility check (mix Your GPUs will likely run quickly with dtype policy mixe compute dtype: float5 variable dtype: float52	xed float16): OK ed_Tloat16 as they all have compute capability of at least 7.0 ed_Tloat16 as they all have the transmission of transmission of the transmission of transmiss
<pre>In []: M inputs = keras.Input(shape=(764,), name='digits') if if.config.list.physical queries('OU'); print('Dim paule will run with 40% units on a GRU') man_units = 40% else: # Use four units on CPUs so the model finishes in a r print('Dim model will run with 64 units on a CPU') ma_units = 64</pre>	resionable amount of the
<pre>densel = layers.Dense(nam_units, activations'relu', name x = densel(inputs) densel = layers.Dense(num_units, activations'relu', name x = densel(x)</pre>	e='dense_1') e='dense_2')
<pre>In []: W print(densel.dtype_policy) print("x.dtype: 30 ' % x.dtype.name) # 'kernel' 'is densel's worlable print("densel.kernel.dtype.% % densel.kernel.dtype.ni</pre>	ane)

Which is like the hello world for training, which will make things simpler for us. So we create our inputs here, so the shape of the input is just 784 and then we check here if we have a GPU on this machine. So, if this GPU on this machine we want to sets our layers number of units to 4096 and if it is not we reduce it to the size this size of 64. So, these are the reflecting through dense layer here.

(Refer Slide Time: 06:07)



So, I can run this through dense layer and see that ok it has to detect that it has a GPU which is running and it is we are going to use this units here, this is the same as putting 4096 is the same as writing it to row directly here. And then we want to be show that ok each of the dense layer what policy are they using is the dense layer is it using the fp16 policy and is the kernel on fp32.

(Refer Slide Time: 06:37)



So, we need to check that because mixed precision is the use of fp16 and fp32. So, we will check that again. So, we can see saying yes the policies is mixed float 16 which was the type is float 16 and a dense kernel float 32. So, we are on the right track here.



(Refer Slide Time: 06:59)

So, the next one here is what to do after the 2 layer then we get our last layer dense layer which is where we supply this softmax here. So, and we check that this is float 16. So, the rule here is that when you are dealing with what the softmax it has to be because you are about to move into doing your calculating your loss and gradient descent.

So, it has to be float 32 there in order to escape the dynamic rate limitation that fp 16 has. So, in order to do that is that we will separate what we have here we separate this layer in toward just the dense layer and then separate the activation layer from it. So, that is what we have done here, if you look at this line here we just have the dense layer here and the second one the output will be just the layer of activation function.

And how do we do that because we are using fp 16 a policy. So, we need to write this out to cast it back to fp 32 here. So, we are only casting back the softmax layer only. So now we are good to go. So, this is just the exception that we have there.

(Refer Slide Time: 08:17)



And then so you can also do that using this line of course, you have a linear here it can also cast it back with that. So, what we need to do next is what to instantiate our model we have the input into the model, we have the output into the model and then we can compile our model by setting the loss and also the optimizer set here and our metrics is what for calculate the accuracy.

Then we get here what we do here is to get our data you know the mnist data, you get your mnist data the training will be 60000 and the test will be 10000 as well. So, it is downloading we download that, so after downloading that so the next thing to do is to what to initialize our weights.

(Refer Slide Time: 09:10)

C jupyter	mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		e
File Edit	View Insert Cell Kernel Help	Not Trusted	Python 3 (ipykernel) O
B + % 6	2 15 ↑ ↓ ▶ Run II C >> Mariadown ✓ 🖾		
In [7]:	M initial_weights = model.get_weights()		
In [8]:	<pre>M history = model.fit(r_rais, r_train,</pre>	185] None of the MLI	IR Optimization Pa
	sees are enabled (registered 2) Epoch J/5 6/6 [Epoch J/5 6/6	val_loss: 0.9415 - val_loss: 0.4111 -	val_accuracy: 0.8 val_accuracy: 0.8
	Epoch 3/5 6/6 [val_loss: 0.4232 - val_loss: 0.2493 -	val_accuracy: 0.8 val_accuracy: 0.9
	Epoch 5/5 6/6 [=======] - 0s 43ms/step - loss: 0.2979 - accuracy: 0.9051 - 513	val_loss: 0.1733 -	val_accuracy: 0.9

So, we initialize the weights here that we will initialize the weights, then we have to work do what and train the model here. So, we can see the model here. So, what I want you to notice here is that this is just put it on the 5 epoch that now we times.

(Refer Slide Time: 09:27)

💭 jupyter 🛛	nixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		e 🕹
File Edit V	ew Insert Cell Kernel Help	Not Trusted	Python 3 (ipykernel) C
8 + % 0	K ↑ ↓ ▶ Run ■ C ≫ Markdown ∨ 🖾		
	<pre>print('Test accuracy:', test_scores[1])</pre>		
	2022-03-14 12:10:22.949574: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.c sses are enabled (registered 2)	cc:185] None of the ML1	IR Optimization Pa
	Epoch 1/5 6/6 [] - 1s 66ms/step - loss: 5.4120 - accuracy: 0.3583 104	3 - val_loss: 0.9415 -	val_accuracy: 0.8
	ppcn 2/5 6/6 [===================================	2 - val_loss: 0.4111 -	val_accuracy: 0.8
	6/6 [=======] - 05 41m5/step - loss: 0.3153 - accuracy: 0.9070 604 Epoch 4/5 6/6 [===============] - 0: 44mc/step - loss: 0.3450 = sccuracy: 0.8091	0 - val_loss: 0.4232 -	val_accuracy: 0.8
	0/6 [1 - val loss: 0.1733 -	val_accuracy: 0.9
	513 313/313 - 1s - loss: 0.1759 - accuracy: 0.9478 Test loss: 0.17591698467331476 Test accuracy: 0.9477999806404114		
I	.oss Scaling		
	<pre>loss_scale = 1024 loss = model(inputs)</pre>		
	iss_scale		

If you can see the number the time it takes for the for each steps, you can see the first one was the one the first step is where it takes longer which is 1 seconds 66 the others are just 0 seconds 0 seconds 0 second 1 second is that it is doing the casting here, this is the first epoch. So, at the subsequent epoch it becomes more faster there.

(Refer Slide Time: 09:53).



(Refer Slide Time: 09:54)



So, let us see if this were to be trained without using mixed precision. So, if we are not using mixed precision. So, what do we do? If we change from mixed precision to maybe we want to use fp 32.

(Refer Slide Time: 10:09)



So, I can just come here and say like let me the connect first clear the output just to show us the difference because this is very important we have limited time ok.

(Refer Slide Time: 10:13)



I have cleared that, so what I would do is what this mixed precision policy? I erase it then I just turn it back to floating point 32. So, this is without using mixed precision here without using mixed precision. So, if you compute here, so what we will see is that you see the compute type is no longer fp 16 it is fp 32 the variable type is fp 32. So, let me quickly run all of these.

(Refer Slide Time: 10:49)



(Refer Slide Time: 10:51)

🔵 jupyter	mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		2
File Edit	llew Insert Cell Kernel Help	Not Trusted	Python 3 (ipykernel) O
5 + 2 (2	K + ↓ ► Run ■ C ≫ Code v E		
	0/105/10/2022/2022/2022 MICH 200/00 No MEMNY: VESICE, 23 NOME, QUANIO 102200, PL2 005 20, 00 7.0		where repertury.
In [3]:	<pre>M print(dense1.dtype_policy) print('x.dtype.is% % x.dtype.name) # "hernel' is dense1's vuriable print('dense1.dtype.is% & dense1.kerne1.dtype.name)</pre>		
	<pre>@olicy *float32"> x.dtype: float32 dense1.kernel.dtype: float32</pre>		
In [4]:	# # JUCORECT: softmax and model output will be float16, when it should be float32 outputs = layers.Dense(10, activation*softmax', name*predictions')(x) print('Outputs dtype: %s' % outputs.dtype.name)		
	Outputs dtype: float32		
In []:	# # CORRECT: softmax and model output are float32 x = layers.Dense(10, name 'dense_log(1x')(x) outputs = layers.Activation('orthorn', dyppe'float32', name='predictions')(x) print('Outputs dype: Ks' % outputs.dtype.name)		
	# The Linear activation is an identity function. So this simply costs 'outputs' # to float32. In this particular cose, 'outputs' is already float32 so this is a # no-op.		
	<pre>outputs = layers.Activation('linear', dtype='float32')(outputs)</pre>		

All of this which I have run before I am only interested in the results, so everything is fp 32.

(Refer Slide Time: 11:00)

@ localhost8080/hotebooks/mixed_precision_thr2.pynb	u & r
JUpyter mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)	e -
File Edit View Insert Cell Kernel Help	Not Trusted 🥒 Python 3 ((pykernel) O
B + 3× 62 K + ↓ Pan ■ C → Code. ✓ □	
₽.	
In [5]: W model = kerss Model(inputs_inputs_ outputs_outputs)	
in [5]), in most - incommendation related and not and not	
<pre>model.compile(loss='sparse_categorical_crossentropy',optimize</pre>	<pre>r=keras.optimizers.RMSprop(),metrics=['accuracy'])</pre>
<pre>(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.l</pre>	oad_data()
<pre>x_train = x_train.reshape(60000, 784).astype('float32') / 255 x_test = x_test_reshape(10000, 784).astype('float32') / 255</pre>	
A contra A contra competitioned in an analysed in contra f f into	
<pre>In [6]: M initial_weights = model.get_weights()</pre>	
<pre>In []: M history = model.fit(x_train, y_train, batch size=8192.</pre>	
epochs=5,	
validation_split=0.2) test scores = model.evaluate(x test, v test, verbose=2)	
<pre>print('Test loss:', test_scores[0])</pre>	
<pre>print('Test accuracy:', test_scores[1])</pre>	
Loss Scaling	
loss_scale = 1024	
<pre>loss = model(inputs)</pre>	
loss *= loss_scale	
# Assume "grads" are float32. You do not want to divide float	16 gradients,

Since its fp 32 I do not need to run that one again to convert.

(Refer Slide Time: 11:14)

				200
💭 jupyter m	xed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		4	NPT
File Edit View	Insert Cell Kernel Help	Not Trusted	Python 3 (ipykernel) O	
B + % @ I	h ✦ ✦ ▶ Run ■ C 胂 Markdown ✓ 📾			
	<pre>(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data() x_train = x_train.reshape(60000, 784).astype('float32') / 255 x_test = x_test.reshape(10000, 786).astype('float32') / 255</pre>			
In [6]: 🕅	<pre>initial_weights = model.get_weights()</pre>			
In[7]: M	<pre>history = model.fit(t_trim, y_trim,</pre>			
	2022-03-14 12:12:19.892533: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc sses are enabled (registered 2)	185] None of the MLI	R Optimization Pa	
	Epsch 1/5 6/6 [- val_loss: 0.8017 - • val_loss: 0.3334 -	val_accuracy: 0. val_accuracy: 0.9	
	Epoch 3/5 6/5 [===========] - 1s 94ms/step - loss: 0.4370 - accuracy: 0.8558 276 Epoch 4/5 6/5 [==============] - 1s 94ms/step - loss: 0.2948 - accuracy: 0.9890	val_loss: 0.2468 - ·	val_accuracy: 0.9 val_accuracy: 0.8	
	Epoch 5/5			

So, let us run this. So now, what I want us to see here is that.

(Refer Slide Time: 11:20)

💭 jupyte	r mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)	e
File Edit	View Insert Cell Kernel Help	Not Trusted Python 3 (ipykernel) O
8 + %	② L ↑ ↓ ► Run ■ C ≫ Markdown ∨ □	
	<pre>print('Test accuracy:', test_scores[1])</pre>	
	2022-03-14 12:12:19.892533: I tensorflow/compiler/mlir/mlir_graph_optimiza sses are enabled (registered 2)	ation_pass.cc:185] None of the MLIR Optimization Pa
	↓ Epoch 1/5 6/6 [************************] - 1s 128ms/step - loss: 3.4745 - accu 7428 Epoch 2/5	uracy: 0.4305 - val_loss: 0.8017 - val_accuracy: 0.
	6/6 [***********************************	acy: 0.7826 - val_loss: 0.3334 - val_accuracy: 0.9 acy: 0.8658 - val_loss: 0.2468 - val_accuracy: 0.9
	Epoch 4/5 6/6 [======] - 1s 94ms/step - loss: 0.2948 - accur 688 Epoch 5/5	racy: 0.9090 - val_loss: 0.3883 - val_accuracy: 0.8
	6/6 [***********************************	acy: 0.9126 - val_loss: 0.1579 - val_accuracy: 0.9
	Loss Scaling	
	loss_scale = 1024 loss = model(inputs)	
	loss *= loss scale	

If you look at this we have at the initial is 1 seconds subsequent epoch is also 1 seconds 1 seconds 94 the 1 seconds 94 you can see that the time it takes when it is not mixed precision is higher than when it is mixed precision. It might looks small is because we are dealing with smaller data and smaller model size here that is why. So, it is just to show you the differences that are there, when it was mixed precision fp 16 you can see only have 1 here and all other subsequent epoch I just took only took 0 seconds.

(Refer Slide Time: 11:57)

	ard united Total and Total Constrained	
Öjupyter i	mixed_precision_tfv2 Last Checkpoint: 3 hours ago (autosaved)	4
File Edit V	iew Insert Cell Kernel Help	Trusted Python 3 (ipykemel) C
ම 🕈 🛪 එ	🗓 🛧 ¥ 🕨 Run 📕 C 🍽 Markdown 🔹 🖾	
In [1];	<pre>M import tensorflaw as tf from tensorflaw tenss from tensorfl</pre>	
10 [2]:	<pre>if tf.config.list_physical_devices('GPU'): print('The model will run with 4096 units on a GPU')</pre>	

So, I will change it back because I need to show us in case you are now you are the way you write your code you are using custom loop within your code.

G gpu compute: x | W CUDA-Wile; x | W Single-predict x | W Half-predictor: x | 🕲 FF64.FF32,Fi x | W Double-predict x | 🖷 TessorFlow | t x | 🔿 Home-Page- x 😵 mixed_predict x + 🗸 Q 🖻 🖈 🔺 🚺 → X ② localhost:8080/notebo NPTEL Jupyter mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes) File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel) O v 🖬 E + 3< 2 E + ↓ ► Run Interrupt 1 1 - □ = Restart 0 0 Restart & Clear Output In [1]: M import tensorflow Rebart & Run Al from tensorflow; from tensorflow. from tensorflow. oort os int(tf._versic_ Change kernel viron['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1 #policy = mixed_precision.Policy('float32') blicy = mixed_precision.Policy('mixed_float16')
ixed_precision.set_global_policy(policy) print('compute dtype: %s'% policy.compute_dtype)
print('variable_dtype: %s'% policy.variable_dtype 2.6.2 compute dtype: float32 variable dtype: float32 In [2]: M inputs = keras.Input(shape=(784,), name='digits') if tf.config.list_physical_devices('GPU'): print('The model will run with 4096 units on a GPU') Cloudy ^ 🖡 🛆 🖼 🦟 (4) 🖧 ENG 🔐 313 PM ang 10%

(Refer Slide Time: 12:04)

So, if you use custom loop within your code, then how do you use mixed precision when you use custom loop? So, to use when you use custom loop, then I will show you that 1 in a GV sorry I do not want to mess things up that is why I keep now. So, we have set back we set back to our mixed precision here.

(Refer Slide Time: 12:34)



So, I can run this now back run this.

(Refer Slide Time: 12:41)



(Refer Slide Time: 12:49)

0	jupyter	mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		2	
File	Edit	View Insert Cell Kernel Help	Trusted	Python 3 (ipykernel) O	
8	+ * 2	K + + ► Run ■ C → Markdown			
5		<pre></pre>			
	In [4]:	# INCORECT: softmax and model output will be float15, when it should be float32 outputs = layers.Bense(10, activations'softmax', name='predictions')(x) print('Outputs dtype: %s' % outputs.dtype.name)			
		Outputs dtype: float16			
	In [5]:	N # CONECT: softmax and model output are float32 x = layers.Dense(10, name='dense_logis')(x) outputs = layers.Activation('softmax', dype='float32', name='predictions')(x) print('Outputs dype: &S' & Soutput.Schype.name)			
		Outputs dtype: float32			
		# The Linear activation is an identity function. So this simply casts 'outputs' # to float22. In this particular case, 'outputs' is already float32 so this is a # no-op. outputs = layers.Activation('linear', dtypes'float32')(outputs)			
	In []:	<pre>M model = keras.Model(inputs=inputs, outputs=outputs) model.compile(loss='spurse_categorical_rrossentropy',optimizer=keras.optimizers.MMSprop(),</pre>	metrics=['accura	cy'])	
alhost		(x train, y train), (x test, y test) = keras.datasets.mnist.load data()			

(Refer Slide Time: 12:52)



(Refer Slide Time: 13:03)

~ Jubler	mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)	4
File Edit	View Insett Cell Kernel Help Trusted Python 3 (ip	iykemel) 🌒
8 + % 0	b 1b + ↓ ▶ Run ■ C ≫ Maridown ✓ 🖾	
	<pre>(x_train, y_train), (x_test, y_test) = kerss.ditasets.mist.load_data() x_train = x_train.reslape(2000, 78).astype(*[loat22] / 255 x_test = x_test.reslape(2000, 78).astype(*[loat22] / 255</pre>	
In [7]:	<pre>M initial_weights = model.get_weights()</pre>	
In [*]:	<pre>M history = model.fit(t_trin, y_trin, bath_sizesUD, epoths, villation.split=0.) test_scores = model_evaluat(x_test, y_test, verbose=2) primt("Test loss", test_scores[1]) primt("Test accuracy", test_scores[1]) 2022-03-14 12:16:09.1369%1: tessorflow/compiler/milr/milr_graph_optimization_pass.ccc185] None of the MLR Optimizati sets are emable (registered 2)</pre>	on Pa
	Epoch 1/5 6/6 [] - 1x 60ms/step - loss: 3.9182 - accuracy: 0.4074 - val_loss: 0.6007 - val_accuracy 282 Epoch 2/5 6/5 [: 0.8 : 0.8

Run that is then?

(Refer Slide Time: 13:10)

~ upyter	mixed precision ffv2 Last Checkpoint 3 hours and (unsaved channes)		2
Jupyter	Hinkoa "Productori" nya rasi onekonin sinoina alo (ananoo anangos)		¥
File Edit	View Insert Cell Kernel Help	Trusted	Python 3 (ipykernel) C
8 + 2: 2	S K + ↓ ▶ Run ■ C ≫ Maridown v 100		
	2022-03-14 12:14:09.136991: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] N sses are enabled (registered 2)	one of the Mi	IR Optimization Pa
	Epoch 1/5 6/6 [=============] - 1s 68ms/step - loss: 3.9182 - accuracy: 0.4074 - val_1 282	oss: 0.6807 ·	val_accuracy: 0.8
	Epoch 2/5 6/6 [oss: 0.3928 ·	val_accuracy: 0.8
	cpcin 3/2 6/5 [- @\$ 42ms/step - loss: 0.3479 - accuracy: 0.8904 - val_1 266 Epoch 4/5	oss: 0.2470 ·	val_accuracy: 0.9
	6/f [oss: 0.2463 ·	val_accuracy: 0.9
	6/6 [] - 0s 43ms/step - loss: 0.2281 - accuracy: 0.9329 - val_3 927 313/313 - 1s - loss: 0.3536 - accuracy: 0.8893	oss: 0.3491 -	val_accuracy: 0.8
	105T 105S: 0.35363/45889392009 Test accuracy: 0.8892999887466431		
	Loss Scaling		
	loss_scale = 1024		
	<pre>loss = model(inputs)</pre>		
	loss "= loss_scale		
1	# Assume "arads" are float32. You do not want to divide float16 aradients.		

So, look at it when we are back it is only the first step that we have 1 seconds and the others just $0\ 0\ 0$ there.

(Refer Slide Time: 13:17)



So, when now we want to do with what loss scaling here. So, loss scaling you need to compute the loss scaling when you are using custom training loops, here we do not need to what compute that. But when we are using custom loop it will be done automatically by itself.

(Refer Slide Time: 13:34)



But in this case when you use custom loop here you need to specify a mixed precision loss scaling optimize this is the scaling we are talking about. So, when you try to use custom loop you must specify the scaling optimizer. So, you get your optimizer here this is our optimizer here is the optimizer root mean square props. So, we use the mixed precision loss scale optimizer to wrap it, after we wrap it through that we get our loss objects here using the cross entropy then we here we get our data again here.

So, let me run this get our data then. So, we want to compute the training step. So, in this training step here we get our gradient tape, this is the gradient tape under the gradient tape. So, what happened get your prediction this is the model supply the input x into it, this is a function we are just writing this function graph from then we compute our loss using this loss object you know we have calculated the loss object here.

So, we call it takes this value, then we scale our loss with what we have been talking about before. So, this optimizer from the mixed precision loss scale optimizer we use it to scale. So, we get the scale loss here, after we get the scale loss, then we also get the what the scale gradient as well the scale gradients using tape loss scale gradient add your scale loss and the model variable add trainable into it.

Then after we have gone that is our scale gradient then we will get our re gradients. So, what happened after your scale your gradient then you what on scale back? So, our optimizer dot on scale the gradient back. So, we will scale the gradient back then we can

now use our optimizer to apply to our normal trainable gradient. So, we return the loss here.

(Refer Slide Time: 15:31)



So, as we have done that we need to do also for testing, but our testing we do not need to go through this. So, we just write the model testing this way that you see sorry about this for now I do not include this here and here also I do not want to include this here.

(Refer Slide Time: 15:50)



So, this @tf function is showing this is a graph formats of writing your model. So, then ok let me run this as well if I run the first one here.

(Refer Slide Time: 16:19)

localhost:8080/notebooks/mixed_precision_th/2.ipynb	Q & ±
<pre>JUpyter mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)</pre>	e 👌
File Edit View Insert Cell Kernel Help	Trusted Python 3 (pykernel) C
E + 3× 2 K + ↓ ► Run ■ C > Code	
<pre>In [10]: W #(jit_complication)</pre>	
<pre>In []: M model.set_weights(initial_weights)</pre>	
<pre>In []: W for epoch in range(3): epoch loss_mp = tf.kers.metrics.Hean() test_accuracy = tf.kerss.metrics.Spars&CategoricalAccuracy(name='test_ac for x, y in train_dataset: loss = train_step(x, y) epoch_loss_yel(loss)</pre>	conacy.)
for x. v in test dataset:	

So, we run that. So, run this cell which I have explained then the one also that compute for our test step. So, remember this is for training step and this is for test step.

(Refer Slide Time: 16:26)

	MIXEd_pfeciSion_IIV2 Last Checkpoint: 3 hours ago (unsaved changes)	4
File Edit	View Insert Cell Kernel Help	Trusted Python 3 ((pykernel)
8 + % 0	K + + ► Run ■ C >> Markdown - □	
	<pre>def test_step(x): return model(x, training=Fmlse)</pre>	
In [12]:	₩ model.set_weights(initial_weights)	
	<pre>epoch_loss_pg = tf.keras.metrics.Heen() test_accuracy = tf.keras.metrics.Heen() test_accuracy = tf.keras.metrics.jence loss = train_step(x, y) epoch_loss_pg(loss) for x, y in test_dataset: predictions = test_ttep(x) test_accuracy.update_statk(y, predictions) print("Epoch (): loss(), test_accuracy-()'.format(epoch, epoch_loss)</pre>	est_accuracy") ss_avg.result(), test_accuracy.result()))
	Epoch 1: 1055-3.1414408/09530806, test accuracy-0.7559775385937 Epoch 1: 1055-0.48655716057525, test accuracy-0.87269975367072 Epoch 2: 1055-0.48655716057552, test accuracy-0.872699950487265 Epoch 3: 1055-0.4159829315567, test accuracy-0.4973999783893237 Epoch 4: 1055-0.14815224707126617, test accuracy-0.5739999961853027	
8	Open 1: 1035-3: 14144800007500506; test & accuracyole. 705097755303713 Open 1: 1035-3: 1046807160975526; test & accuracyole. 705097755078127 Open 1: 1035-4: 10459716075576712 Open 1: 1035-4: 10459716075576712 Open 1: 1035-4: 104597160756712 Open 1: 1035-4: 10459717166712 Open 1: 1035-4: 14051224707126617 Fopen 4: 1035-4: 14051224707126617 You can optimize further using the Tensorflow XLA compiler by adding: You	

Then the weight we set our weight to initialize our weight again, then this is the customized loop now. So, we are inside the loop. So, inside the loop we calculate our epoch loss average here using the metric.Mean() then our test accuracy is set here. So, in each value inside our training data set we train with these steps get the loss and the loss average loss is calculated here.

Also inside our test set so we get the training test step here training step and then we our test accuracy will update the states and then we can compute here we can print all the results here. So, when you do that so you can see also it is a very fast order we did not calculate time here, but if you test this with bigger model you will be able to see and that it run faster than when you are not using mixed precision.

So, I have just showed you 2 for when you are using custom loop training loop and when you do not use custom training loop. So, with custom training loop you know you have to set your mixed precision loss scaling all of that then you have to scale you have to unscale back or scale gradients and all that which all the theoretical which we have explained before.

(Refer Slide Time: 17:54)



So, here one thing you can do again is to optimize for that is what we call the XLA compiler. So, you can do use the XLA compiler by what adding this to you add this to the environment variable then you enable a jit compiler on each of the top of custom loop that you have. That is you know our custom loop exist between on the each of this function.

So, what we need to do here is our first thing is to have that is let us come here then I can enable the mixed precision here. Remember we do not get confused this the enablement we are doing here is for the XLA. So, we enable it like this then let me run this as I have enabled that. So, we are still on the float 16 then I can come here then add the jit to it as well, just to show you that you can have it in front of the function here just in time also you can add that here just in time and add that here.

(Refer Slide Time: 19:10)

×		
Jupyter n	IXEd_precision_tVZ Last Checkpoint: 3 hours ago (unsaved changes)	~
File Edit Vie	w Insert Cell Kernel Help	Trusted 🖋 Python 3 (ipykernel) C
8 + % 8	K + ↓ ▶ Run ■ C ≫ Code ✓ III	
In [10];	<pre>figit_compilering figit_compilering def train_step(x, y): def train_step(x, y): def train_step(x, y): def train_step(x, y): scalet_paintral(sel) as tape: predictions = nodel(x) scalet_paintral(sel) = tape_predictions) scalet_paintral(sel) = tape_predictions(loss) scalet_paintral(sel) = tape_predictions(loss) gradients = optimizer.get_uncided_predictis(scalet_predictis(sel) gradients = optimizer.get_uncided_predictis(scalet_predictis) return loss</pre>	
In [11]: 1	f(jt:compile/rmc) gf:fnctmcin(jt;compilertee) def test_step(2): reterm models(t, training=false) model.set weights(initial weights)	
In [13];)	for epoch in range(): epoch_loss_ang = trikerss.metrics.Mean() test_accuracy = trikerss.metrics.Spars&Categorical&ccuracy(name='test_accuracy') for x, y in train_dataset: loss = train_ste(x, y) epoch_loss_ang(loss) for x, y in test_dataset: predictions = test_tdataset: predictions = test_tdataset:	

So, then I can run this over again and run this over again.

(Refer Slide Time: 19:18)

💭 jupyter	mixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		e 🕹
File Edit 1	/iew Insert Cell Kernel Help	Trusted	Python 3 (ipykernel)
B + % Ø	B + ↓ ► Run ■ C ₩ Code v □		
	# Assume "grads" are float33. You do not wont to divide float16 gradients. grads = compute gradient(loss, model.trainable_variables) grads /= loss_scale		
In [15]:	<pre>M optimizer = keras.optimizers.MSprop() primizer = misers.optimizer = misers.optimizers.op</pre>		
In [10]:	<pre>test_dataset = tf_data.Dutaset_from_temsor_slices((x_test, y_test)).batch(000) # (j(:,comp(=n-ruw) # (j(:,comp(=n-ruw) # fright_completrue) # tright_data(tspel) as tape: predictions = mode((n) loss = loss goigt(xy, predictions) scaled_loss = optimize_get_scaled_loss(loss) scaled_loss = tpe_grade(tscaled_loss, model.trainable_variables) gradients = optimize_grade(scaled_gradients)(scaled_gradients))</pre>		

Then run this also over and run this as well.

(Refer Slide Time: 19:20)



(Refer Slide Time: 19:27)

Jupyter	ixed_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		e 🕹
File Edit V	w Insert Cell Kernel Help	Trusted	Python 3 (ipykemel) C
B + % Ø	Č ↑ ↓ PRun ■ C » Code v III		
	<pre>graulents = optimizer.get_unscaled_graulents(scaled_graulents) optimizer_apply_gradients(rip(gradients, model.trainable_variables)) return loss</pre>		
In [17]:	f(jit.compile=fram) @f.function(jit.compile=Trome) def test.step(); return model(x, training=False)		
In [18]:	model.set_weights(initial_weights)		
In [13]:	<pre>for epoch in range(3): spot [Joss_mp] = tikers.metrics.Heam() test_accuracy = tikers.metrics.SursGategoricalAccuracy(name='test_accuracy') for x, y in train_dataset: loss = train_step(x, y) epoch_loss_avg(loss) for x, y in test_dataset: productions = test_step(x) test_accuracy.publes_step(x), preductions) print('ipoch []: loss-[], test_accuracy[]'.format(epoch, epoch_loss_avg.result(), test_accuracy </pre>	acy.result()))
	Epoch 0: 10ss-3,1414480289556566, test accuracy=0.7555999755859375 Epoch 1: 10ss=0.48657166575552, test accuracy=0.87259997564872 Epoch 2: 10ss=0.4395497489557657, test accuracy=0.895999984847266 Epoch 2: 10ss=0.419548231956567, test accuracy=0.89739997818531207 Epoch 4: 10ss=0.4185214970715677, test accuracy=0.89739997818531207		

(Refer Slide Time: 19:31)

💭 jupyter mixed_precision_tfv2 u	ast Checkpoint: 3 hours ago (unsaved changes)		0
File Edit View Insert Cell Kerr	iel Help	Trusted	Python 3 (ipykernel)
8 + 3K 연 16 + 4 > Run 18	C 🗰 Markdown 🗸 🖾		
In [*]: W for each in range(5): emoth loss, one = th test_accuracy = tf. for x, y in train, do not a train step each loss = train step each loss = units dat predictions = test test_accuracy.upd test_accuracy.upd	<pre>keras.metrics.Hean() eras.metrics.Jean() eras.metrics.SparseCategoricalAccuracy(name='test_accur taset:</pre>	acy')	
2022-03-14 12:20:37.44 m CUDA (this does not 2022-03-14 12:20:37.44 putc Capability 7.0 2022-03-14 12:20:37.44 putc Capability 7.0 2022-03-14 12:20:37.44 er, set emv var "MLIE	<pre>Fill tensorflaw(complier(x)/wrvice/service.cc:17) guranter that XLA will be used). Derices: 6501: 1 tensorflaw(complier(x)/service/service.cc:17) 6507: 1 tensorflaw(complier/x)/service/service.cc:17) 6507: 1 tensorflaw(complier/x)/service/service.cc:17) 6507: 1 tensorflaw(complier/x)/service/service/service.cc:17) 6507: 1 tensorflaw(complier/x)/service/service/service.cc:17) 6507: 1 tensorflaw(complier/x)/service/service/service/service.cc:17) 6507: 1 tensorflaw(complier/x)/service/service/service.cc:17) 6507: 1 tensorflaw(complier/x)/service/se</pre>	XLA service 0x2279b3f0 initia StreamExecutor device (0): StreamExecutor device (1): mlir_util.cc:210] disabling ML	// lized for platfor Quadro GV100, Com Quadro GV100, Com IR crash reproduc
You can optimize further using t 1. os.environ['TF_ENABLE 2. (jit_compile=True) to	he Tensorflow XLA compiler by adding: _AUTO_MIXED_PRECISION'] = '1' to the top of the script the top of the custom loop		

Run this as well then here. So, we can also run this as well. So, it will make it faster you can see if you can see here.

(Refer Slide Time: 19:35)

💭 jupyter	mixed_precision_tfv2 Last Checkpoint: 3 hours ago (autosaved)		e
File Edit	View Insert Cell Kernel Help	Trusted	Python 3 (ipykemel) O
8 + % 6] 15 + ↓ ▶ Run ■ C >> Code - ✓ 55		
	<pre>for x, y is train_distant: loss = train_step(x, y) epoch_loss_ang(loss) for x, y is test_distant: predictions = test_step(x), predictions) test_accurey.update state(y, predictions) print("fook" (i losse(), test_accuracy['.format(epoch, epoch)</pre>	loss avg.result(), test accuracy.result	()))
	2822-34.44 12:38:73.46842:1 1tesorflaw1compiler/hild/service/serv n OMA (this des not guarante tett XA will be used). Bedies: 2822-34.54 12:28:37.46827:1 1tesorflaw1compiler/hild/service/serv pite Capability 7.0 2822-39.14 12:28:37.46827:1 1tesorflaw1compiler/hild/service/serv pite (spability 7.0 2822-39.14 12:28:37.46828:1 1tesorflaw1compiler/hild/service/serv er, set ew ver mill.c050.8 #RMMCL_BIECTOW reads. 2022-33.41 12:28:64.8558:1 1tesorflaw1compiler/hild/2016.000147 5 logged at not cone for the life.ife of the process.	<pre>ice.cc:171] XLA service 0x22706376 init ice.cc:179] StreamExecutor device (0) ice.cc:179] StreamExecutor device (1) utils/dump_mlir_util.cc:210] disabling ice_cache.cc:368] Compiled cluster usin</pre>	ialized for platfor : Quadro GV100, Com : Quadro GV100, Com MLIR crash reproduc g XLA! This line i
	Epoch 4: loss=J.5753080866980037, test accuracy=0.759099779701233 Epoch 1: loss=0.476659360106293, test accuracy=0.815199971199015 Epoch 2: loss=0.353744057228, test accuracy=0.4813999711512286 Epoch 3: loss=0.35224002456655, test accuracy=0.492090010013080 Epoch 4: loss=0.17452841241943, test accuracy=0.49400010013087		
	You can optimize further using the Tensorflow XLA compiler by adding:		
	 os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1' to the top of the sc (iit compile=True) to the top of the custom loco 	cript	

These are enabling the what the jit XLA compilation this is tensorflow special tensorflow compiler which will run with what using the jit there. So, you can also use this and if you use it that is this is combination of mixed precision and the XLA. So, which give more optimization.

(Refer Slide Time: 20:01)

> Jupyter mixed_precision_ti	fv2 Last Checkpoint: 3 hours ago (autosaved)	ê
File Edit View Insert Cell	Kernel Help	Trusted Python 3 (pykemel) O
B + 3× 20 K + ↓ ► Run	Interrupt II v CO Restart 0.0	
In [14]: W import tensorflow from tensorflow.from tensorflow.f from tensorflow.f import os	Restart & Run All Restart & Run All Reconnect Shudown ision	
os.environ['IF_EN #policy = mixed_p policy = mixed_precisions mixed_precisions print('compute dt print('variable d	MBLE_ANTO_MIXED_PRECISION'] = '1' recision.Policy('floot12') ecision.Policy('mixed_floot15') et_glabel_policy(onito) ype: B.'% policy.compute_dtype) type: B.'% policy.compute_dtype)	
2.6.2 compute dtype: fl variable dtype: f	loat16 loat32	
In [2]: M inputs = keras.In	<pre>put(shape=(784,), name='digits')</pre>	

(Refer Slide Time: 20:07)

_	g local as cool in ote cool	edunizar Suzeranu Lutrishun	4 E A	
	Ç jupyter n	nixed_precision_tfv2 Last Checkpoint: 3 hours ago (autosaved)	-	INC.
	File Edit. Vie	w Insen Restart kernel and clear all output? * usind	Python 3 (ipykemel) O	
	8 * * @	O you want to restart the current kernel and clear all output? All variables and outputs will be lost.		
	In [14]:)	I import to Continue Running Restant and Clear Al Oxfords		
		from tensorflow laport keras		
		from tensorflow.keras import mixed_precision		
		print(tfversion_)		
		os.environ['TF_EHABLE_AUTO_MIXED_PRECISION'] = '1'		
		<pre>Ppolicy = mixed_precision.Policy('float32')</pre>		
		<pre>policy = mixed_precision.Policy('mixed_float16') mixed_precision.set_global_policy(policy)</pre>		
		mint('compute diver: %s'% policy.compute diver)		
		print('variable dtype: %s'% policy.variable_dtype)		
		2.6.2 commite divine: Float15		
		variable dtype: float32		
	In [2];)	<pre>inputs = keras.Input(shape=(784,), name='digits')</pre>		
		<pre>if tf.config.list_physical_devices('GPU'): print('The model will run with 40% units on a GPU')</pre>		
n for jocalhout		And the second se		

So, this is it on mixed precision then.

(Refer Slide Time: 20:12)



So, I am about to round up just 2 more slides.

(Refer Slide Time: 20:19)



Let me shut down this here.

(Refer Slide Time: 20:20)

11110110100				60
Jupyter m	XEd_precision_tfv2 Last Checkpoint: 3 hours ago (unsaved changes)		e	NPT
File Edit View	/ Insert Cell Kernel Help	No kernel Trusted	Python 3 (ipykernei)	
B + % @ I	5 ♠ ♥ ▶ Run ■ C ₩ Code v □			
in[]: W	<pre>import tensorflaw as tf fram tensorflaw import kerns fram tensorflaw.kerns import layers fram tensorflaw.kerns import mixed_precision import os print(tfversion_) os.environ('TF_EMANLE_ANTO_MIXED_PRECISION') = '1' Apulity = mixed_precision.Policy('float22') policy = mixed_precision.Policy('float22') policy = mixed_precision.Policy('float22') </pre>			
In[]: M	print('umpute ditype: No'T policy.compute ditype) print('umpute No'T policy.variable_dype) imputs = keras.input(shape('dys), names'i(gits') if tf.comfig.list_physical_devices('dW'): print('The wood ull run with Web utts on a CBU')			

Then you can leave also I leave here.

(Refer Slide Time: 20:24)

3 gou compute ce: x W CUDA - Wikipeli x W Single-precision : x W Half-precision fin x 🕲 FF64, FF32, FF16 x W Double-precision x 🚳 Tensorflow (N	W 🗴 🙄 Home Page	-Sele x +	v
→ C O localhost:000/tree?		(0.04
📁 jupyter			Quit
Files Running Clusters			
Select items to perform actions on them.		Upload	New- 2
	Name 🗣	Last Modified	File size
Brited_precision_thr1.pynb	Ð	3 hours ago	2.82 kB
& mixed_precision_th/2.ipynb		seconds ago	12.1 kB
		8 hours and	13.3 kB

1	kit-1.pd		^					_	_												Show	all	×
		4	$\mathbf{\Sigma}$	4	=		0			- 🚯	e)	2	- 63	4	6°	C Cloud		۵	۰.		3/14/20	2	6

(Refer Slide Time: 20:32)



So.

(Refer Slide Time: 20:43)

	AMP: lensorflow	
set the environme	ent variable inside a TensorFlow Python script	
os.environ['TF_ENAB	LE_AUTO_MIXED_PRECISION'] = '1'	
Graph-based:		
opt = tf.train.AdamO	ptimizer()	
opt = tf.train.experin	nental.enable_mixed_precision_graph_rewrite(opt)	
train_op = opt .mimir	nize(loss)	
Keras-based:		
opt = tf.keras.optimi	zers.Adam() b	
opt = tf.train.experin	nental.enable_mixed_precision_graph_rewrite(opt)	
model.compile(loss=	loss, optimizer=opt)	
model.fit()		

So, back from the demo, so there is another better way to do what we have done or so in that other way we which to use now automatic mixed precision with our tensorflow is that you set this environment although. We have seen how we can set this environment. So, if you are working with graph base what you need to include is just this enable_ mixed_ precision_ graph_ rewrite you use that to wrap your optimizer and it is done.

So, you do not need to on scale on the scale you do not need to do that, that is how automatic mixed precision is very easy to use. And if you are using keras based as well you can just do that as well enable_ mixed_ precision_ graph_ rewrite to wrap your optimizer and everything is done. It does the scaling and on scaling itself within it does that within.

(Refer Slide Time: 21:43)

<pre>tensorTiov:BUT-by/B Jusyler notebookport=8888ip=0.0.0.0ino-browserMolebook.aco.toker="" isual password for ten: "" TensorFlow resion :: "" TensorFlow Version 1.14.0 Container image Copyright (c) 2019. NVIDIA COMPORTION. All rights reserved. Container image Copyright (c) 2019. NVIDIA COMPORTION. All rights reserved. Copyright 2017-2019 The Metaorche Common tensor tensor tensored. W/IDIA model.comple(loss=loss, optimizer=opt) model.fit()</pre>	Shutdown 10 1212 11 1212 11 1212 tosin@to	rthis notebook server (y/[n])?y mBlob2 workshootkool Shutting com Chirned 1.40.83 totebookkool Shutting com Oternels 1.40.83 totebookkool Shutting com Oterninals sinif-sludd obcer um -gous all -11 + t -m av /home/tosin/Documents/mixed_precision:/workspacenetwork-host nu	cr.io/nv
<pre>= TensorFlow == VIDIA Release 19.07 (build 7382442) TensorFlow Version 1.14.0 Container mage Court of (c) 2019, WIDIA CORPORATION. All rights reserved. Court of the include modifications (c) WIDIA CORPORATION. All rights reserved. WIDIA modifications are covered by the license tense that apoly to the underlying project or file. WIDIE WIFED driver for mult-mode communication was not detected. Nulti-mode communication may be reduced. WIDIE WIFED driver for mult-mode commination was not detected. Nulti-mode commination may be reduced. WIDIE WIFED Griver for mult-mode commination was not detected. Nulti-mode commination may be reduced. WIDIE WIFED for TensorFlow, WIDIA recommends the was of the too lowing frags: nridiarebocker run matersize=1gulinait sealcok=1 +-ulinait stack=0100004 model.compile(loss=loss, optimizer=opt) model.fit()</pre>	tensorf [sudo] (gw:19.07-py3 jugyter notebookport=8888ip=0.0.0.0no-browserNotebookApp.token=** assword for tosin:	
WIDIA Release 19.07 (build 7332442) Tengorfiko Version 1.14.0 Container image Copyright (c) 2019, NVIDIA COMPORTION, All rights reserved. Sovyright 2017-4019 The heardon without a served. Version files include modifications (c) NVIDIA COMPORTION, All rights reserved. NVIDIA modifications are covered by the license tenses that apply to the underlying project or file. NVIDIE: NVFCD error multi-mode commission was not detected. NUTE: The SMEM allocation performance may be reduced. NUTE: The SMEM allocation inimit is set to the default of 6406. This may be insufficient for TenserFile. NVIDIA recommends the use of the following flags: notification for TenserFile. NVIDIA recommends notification for TenserFile. NVIDIA recommends model.compile(loss=loss, optimizer=opt) model.fit()	== Tensi		
Container inage Copyright (c) 2019. NVIDIA CORPORATION. All rights reserved. Copyright 2017-2019 The Hencorf To Autors, All rights reserved. Various files include modifications (c) NVIDIA CORPORATION, All rights reserved. NVIE: NEED eriver for multi-mode communication was not detected. NVIE: The SMEW allocation inter is set to the default of BAME. This may be insufficient for TensorFlow, NVIDIA recommends the use of the following flags: nvidia-docker run -=shm=ize=Tgulimit semicol=1ulimit stack=E7100004 modeL.compile(loss=loss, optimizer=opt) modeL.fit()	NVIDIA F TensorF	kelesse 19.07 (build 7332442) dw Version 1.14.0	
<pre>various files include modifications (c) NVIDIA COSPORTION, All rights reserved. NVIDIA modifications are covered by the license tens that acoly to the underlying project or file. NVIE: NVEED incert form all transfer commission was not detected. NVIE: NVEE the SHEEM solution isn't is set to the default of 64MB. This may be insufficient for femotificw, NVIDIA recommends the use of the tollowing flags: nvidia-docker run ==hm=size=10 ==-ulinit stack=ST108884 model.compile(loss=loss, optimizer=opt) model.fit()</pre>	Containe Copyrigh	er image Copyright (c) 2019, NVIDIA CORPORATION. All rights reserved. nt 2017-2019 The TensorFlow Authors. All rights reserved.	
NOTE: WEED driver for multi-inode communication was not detected. Nulti-inode communication performance may be reduced. NUTE: The SRME all location inter is next to the default of EARE. This may be multificient for Tomorflow. WIDLA recommends the use of the following flags: mvidia-docker run ==damsize=tg ==ullmit mealock=:1 ==ullmit stack=E100884 model.comple(loss=loss, optimizer=opt) model.fit()	Various NVIDIA m	files include modifications (c) NVIDIA CORPORATION. All rights reserved. modifications are covered by the license terms that apply to the underlying project or file.	
NUTE: The SMEW allocation limit is set to the default of 6406. This may be insufficient for TendorFlow, NVIDLA recommends the use of the following flags: nvidia-docker runshmesize=10uinit meniodx=1uinit stack=0108884 model.compile(loss=loss, optimizer=opt) model.fit()	NOTE: MI Mi	FED driver for multi-node communication was not detected. Jiti-node communication performance may be reduced.	
model.compile(loss=loss, optimizer=opt) model.fit()	NOTE: TH insut nvid	e GMEM allocation limit is set to the default of BAMG. This may be ficient for TendorFlow. WUDLA recommends the use of the following flags: a-docker runsharsize=fgulimit memicok≈1ulimit stack=87108884	
modeLfit()	- m	odel.compile(loss=loss, optimizer=opt)	
	m	odel.fit()	

So, to show you that I can just because we can only do that with tensorflow version 1, if you can see here this is version 1 here container. So, I will run this again. So, we would not waste time on this ok.

(Refer Slide Time: 22:00)

20 Op	er65H SSH dient - [] 18. Parlament 40: 107. (hou 14.7929.019)
Tensi	a herease is.ur (build (552442) pFlow Version 1.14.0
Cont: Copy	ainer image Copyright (c) 2019. NYIDIA CORPORATION. All rights reserved. right 2017-2019 The TensorFlow Authors. All rights reserved.
Varii NVID	us files include modifications (c) NVIDIA CORPORATION. All rights reserved. IA modifications are covered by the license terms that apoly to the underlying project or file.
NOTE	NOFED driver for multi-node communication was not detected. Nulti-node communication performance may be reduced.
NOTE ii	The SHNEM allocation limit is set to the default of 64MB. This may be sufficient for TensorFlav. MVIDLA recommends the use of the following flags: vida-obder row - matherizatell are unlimit stacket-for Unlimit stacket-for0684
2022 art tsMa	(2017) 152 Notebook op Writing notebook server cook is secret to /root/.local/share/jupyter/runtime/hotebook_cook/is_secret (2017) 74 Notebook op 141 suthentication is disabled. Anyone who can connect to this server will be able to moode. (94) 14 (22:010:401947 + iteraser/obs/isatoma/deful/
	122-08.274 korebookap; Berving notebooks from local directory:/workspace 1220-08.274 korebookap; The Jugyter Kokebok; sz. onning at 1 1220-08.274 korebookap; The Jugyter Kokebok; sz. onning at 1 1220-08.274 korebookap; Ube Control-1: to stoo this server and shut down all kernels (twice to skip confirmation).
-	model.compile(loss=loss, optimizer=opt)
	model fit()

I think I can come here then let me have that is ok.

(Refer Slide Time: 22:08)

Jupyter		Quit
Files Running Clusters		
Select items to perform actions on them.	Upload	New-
0 • 1	Name 🕹 Last Modified	File size
# mixed_precision_thr1.ipynb	3 hours ago	2.82 kB
Brixed_precision_th/2.ipynb	2 minutes ago	12.1 kB
🗋 🖉 Untitled.ipynb	8 hours ago	13.3 kB

Now, I will run this is our version 1 here.

(Refer Slide Time: 22:18)

JUpyter mixed_precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)	6
File Edit View Insert Cell Kernel Help	Trusted Python 3
B + 3× ② 15 + ↓ HRun ■ C ≫ Code ∨ □	
<pre>in [1]: W import tensorflaw as tf from tensorflaw import kerss from tensorflaw import insorflaw import in provide the tensorflaw import in provide the tensorflaw import in provide the tensorflaw import in os.emvire("TF_ENRLE_AUTO_NIXE_PRECISION") = '1' 1.14.0</pre>	
<pre>In []: W [x_train, y_train], (x_test, y_test) = keras.datasets.mist.load_data() x_train = x_train.reshape(10000, 784).astype('float32') / 255 x_test = x_test.reshape(10000, 784).astype('float32') / 255 inputs = kerss.input(happe('784.), name'digits') dessel = layers.Desse(4095, activation'relw', name'dense_1') x = densel(inputs) dessel = layers.Desse(4095, activation'relw', name'dense_2') x = densel(inputs) outputs = layers.Desse(4095, activation'relw', name'dense_2') x = densel(inputs) </pre>	Ι

So, in the version 1 this is what with import tensorflow we have import our keras, our layer, OS do you see we do not import mixed precision here. So, what you can do here is that here I have set my environment for cuda device because I want to use GPU, then I set the environment here. So, after setting the environment here let me just run this that is done. So, you can see the version of the tensorflow is 1.14 yeah for automatic mixed precision.

(Refer Slide Time: 22:49)

,	nixed_precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)		
File Edit Vie	aw Insert Cell Kernel Help	Trusted	Python
B + % Ø	Ko + ↓ NRun ■ C >> Code → ⊠		
	os.environ["CUDA_VISIBLE_DEVICES"]="0,1" os.environ["TF_EMABLE_AUTO_MIXED_PRECISION"] = '1'		
	1.14.0		
In [2]: 1	<pre>(v Taih v Traih) (v Terk v Terks v Terks addaseds.amid.Load.dta() v Traih v Terks nethod(0000, 700).https://load.ttaih// 285 v.test = x_test.reshape(0000, 700).https://load.ttaih// 285 inputs = karas.Taput(shape(7000, 700).https://load.ttaih// 285 inputs = karas.Taput(shape(700, attivation='rels', name='dense_1') dense2 = layers.Dense(4006, attivation='softam', name='dense_2') x = dense[(not set inputs/softam', name='softam'/(kense_inputs/softam'/(kense_inputs/softam'/(kense_inputs/softam'/(kense_inputs/softam'/(kense_inputs/softam'/(kense_inputs/softam'/)(x) Denloading data from https://torage.populesion.com/comserf/au/t/kense_idatsets/mist.pp </pre>		
	1193376/1198814 [sorflow/p precated	ython/ops/ and will b

So, here we are getting our value here loading the data from mnist, then the training the train and the test data then the input size which is the shape of the input is 784. Then we have our 2 dense layer and the output layer. So, here we care less whether our activation for sure it is on float 32 or is on float 16 we do not need to bother about that. So, I run that so downloading putting our download the data that we need.

(Refer Slide Time: 23:24)

C jupyter	mixed_precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)
File Edit	View Insert Cell Kernel Help Trusted Python 3 (
8 + % 0	B ↑ ↓ HRan ■ C >> Matidown ~
	Call initializer instance with the dtype argument instead of passing it to the constructor
In [3]:	<pre>M model = keras.Hodel(inputs=inputs, outputs=outputs)</pre>
	<pre>opt = tf.keras.optimizers.Adam()</pre>
	<pre>opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)</pre>
	<pre>model.compile(loss='sparse_categorical_crossentropy', optimize==opt, metrics=['accuracy'])</pre>
In [4]:	<pre>M initial_weights = model.get_weights()</pre>
In [*]:	M history = model.fit(x_train, y_train, betch;jsze4193, epochas; validation.glite1.); test_scores andule.audute(zetx, y_test, verbose2) print("test_scores", scores(1)) print("test_scores"; test_scores(1))
	Trzin on 48000 samples, validate on 12000 samples [ppch 1/5 48000 (48000 [==================================

So, this is the crucial part here. So, we define our model here the input and output. So, then we load what our optimizer. So, after loading optimizer the next thing to do so to wrap the optimizer under tensorflow dot train dot experimental, then you wrap it with enable mixed precision graph rewrite. So, you wrap that against your optimizer. So, once you do that this is just a single line here.

So, which makes automatic mixed precision makes life easier. So, you wrap that here then you compile this is the normal line for to compile, you have your loss your optimizer your metrics that you want to use to compile. So, let me run that cell after running that cell then we can initialize when you initialize or initialization is done then.

(Refer Slide Time: 24:27)

💭 jupyter	mixed_precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)	2
File Edit	View Insert Cell Kernel Help Trusk	ed Python 3
B + % 6	8 K + ↓ HRun ■ C >> Markdown ✓ 55	
	<pre>print('Test loss:', test_scores[0]) print('Test accuracy:', test_scores[1])</pre>	
	Epoch J/5	<pre>val_acc: 0.8 val_acc: 0.90 val_acc: 0.94 val_acc: 0.95 val_acc: 0.96</pre>
	References This script was adapted from: https://www.tensorflow.org/suid/simised_precision	

So, we can now run our model as well. So, you see we do not need to do anything for that here as well. So, just run the model you can see how the model runs, you see it is even almost it is faster a little bit faster. But we cannot judge based on the little more that we have done is 0. This is first one took 1 seconds and all that subsequent epoch, this number of steps just to 0 0 seconds and you can see this is just micro seconds yes 6 6 whether I want more than that ok.

So, this is just how to use automatic mixed precision. So, how do we check with this tensorflow one if we are not using automatic mixed precision or let us compare with when no automatic mixed precision is used or mixed precision is used.

(Refer Slide Time: 25:09)

File Edit View Inser	ert Cell Kernel Help	Tristed Python 3 C
B + x 0 B + 4		Transa 1.3 miles a
In [1]: M import : from te from te import print(t os.envi os.envi	HRung Hemopt Restart Restart	
In [2]: W (x_train x_train x_t	<pre>in,train, (x_text, y_text) + keras.datasets.mist.load_data() n * x_train.reshape(0000, 700).astype('float22') / 255 = x_text=reshape(10000, 700).astype('float22') / 255 = texess.logut(shape(124_), asser:digits') = layers.bese(0005, activations'rels', name='dense_1') nest(ippus) = layers.bese(005, activations'rels', name='dense_2') nest(x) s = layers.bese(00, activations'roltar', name='dense_2') (x)</pre>	

(Refer Slide Time: 25:13)

jupyter mixed_	precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)	4	6
File Edit View In	Restart kernel and clear all output?	Trusted	Python 3
8 + × & 6 +	Do you want to restart the current kernel and clear all output? All variables and outputs will be lost.		
In [1]: H impor	Continue Running Restart and Clear Al Outputs		
from from impor print	tensorflow import keras tensorflow.keras import layers rt os (ff. version)		
os.en	wiron("filon_visible_devices")="0,1" wiron("TF_dnale_and_aleed_stor") = "1"		
1.14.	9		
In [2]: M (x_tra x_tra x_tes	nain, y_train), (x_test, y_test) = keras.datasets.mist.load.data() in = x_train.reshape(60000, 784).astype("float32") / 755 it = x_test.reshape(10000, 784).astype("float32") / 755		
input	ts = keras.Input(shape=(784,), name='digits')		
dense x = d	<pre>el = layers.Dense(40%, activations'relu', mameridense_b') lensel(inputs)</pre>		
dense x = d	o2 = layers.Dense(4096, activation='relu', name='dense_2') dense2(x)		
outpu	<pre>its = layers.Dense(10, activation='softmax', name='predictions')(x)</pre>		
Down1	loading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz		

(Refer Slide Time: 25:15)

C JUpyter mixed_precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)	
File Edit View Insert Cell Kernel Help Trusted Python	
B + 34 <2 K + ↓ HRun ■ C > Code ∨ □	
<pre>In [1]: # import tensorflow as tf from tensorflow layort terms from tensorflow.terms from tensorflow.terms from tensorflow.terms print(tfversion_) oem/rom("TE_EDBLE_AUTO_NIXE_PRECISION"] = "1" 1.14.0</pre>	
<pre>In []: W (x_train, y_train), (x_test, y_test) = keras.datasets.mnit.load_data() x_train = x_train.reshape(00000, 784).astype('loat22') / 255 x_test = x_test-reshape(0000, 784).satype('loat22') / 255 inputs = keras.lnput(shape=(786,), name='digits') densel = layers.Dense(00%, activation='relu', name='dense_1') x = densel(inputs) dense2 = layers.Dense(00%, activation='relu', name='dense_2') x = densel(inputs) outputs = layers.Dense(00%, activation='softmar', name='dense_2') x = densel(inputs)</pre>	

So, if I clear the output here I cleared then I would comment out this, that it is not used it is not enabled.

(Refer Slide Time: 25:22)

>Jublici III	nixed_precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View	w Insert Cell Kernel Help Trusted Python 3 C
B + % @ I	K ↑ ↓ HRun II C >> Code ✓ 🖾
	os.environ["CLDA_VISIBLE_DEVICES"]="0,1" Bos.environ['TF_EMABLE_AUTO_MIXED_PRECISION'] = '1'
	1.14.0
In [2]: M	<pre>(k_train x_train.reshape(00000, 704).astype('float2)' / 255 k_train x_train.reshape(0000, 704).astype('float2)' / 255 inputs = kerss.Input(shape(706, 1), name='dense_1') dessel = layers.Dense(4000, activation='relu', name='dense_1') x = densel(Inputs) dessel = layers.Dense(4000, activation='relu', name='dense_2') x = densel(0, activation='relu', name='dense_2') s = densel(0, activation='softmas', name='predictions')(x)</pre>
	MARINE: loggither flag participation of the presenter (p) MARINE: loggither flag participations of the present
-	<pre>model = keras.Hodel(inputs=inputs, outputs=outputs)</pre>

And I will also come here this is the line here and I comment this line out yeah. So now, we are just run in our normal tensorflow. So, we can see here I run this slide ok and also run this.

(Refer Slide Time: 25:41)

× uputor	nived exercision fful
Jupyter	litiked_precision_uvi Last checkpoint: 3 hours ago (unsaved changes)
File Edit V	ew Insert Cell Kernel Help Trusted Python 3 (
8 + % @	Kh ↑ ↓ N Run III C >> Markdown ∨ □
	Call initializer instance with the dtype argument instead of passing it to the constructor
In [3]:	<pre>model = keras.Hodel(inputs=inputs, outputs=outputs)</pre>
	<pre>opt = tf.keras.optimizers.Adam()</pre>
	#opt = tf.train.experimental.enable mixed precision graph rewrite(opt)
	model compile(losse'snames categorical crossentrony' entinizament matrices['accumacy'])
	montricederefreese she schereder reactions and A observes why accured account IV
In [4]:	<pre>M initial_weights = model.get_weights()</pre>
In [*]:	<pre>history = model.fit(x_train, y_train,</pre>
	batch_size=8192, epochs=5,
	validation_split=0.2) test crones = model evaluate(v test v test verbose=2)
	<pre>print('Test loss:', test_scores[0])</pre>
	<pre>print('Test accuracy:', test_scores[1])</pre>
	Frain on 480000 samples, validate on 12000 samples Epoch 1/5
	48000/48000 [==================================
	Epoch 2/5
	48000/48000 [==================================
	Epoch 3/5

So, we do not actually need to put initialization we might not run that let us run it. So, and then let us see how long we took that.

(Refer Slide Time: 26:05)

> Jupyte	" mixed_precision_tfv1 Last Checkpoint: 3 hours ago (unsaved changes)	e 19
File Edit	View Insert Cell Kernel Help T	iusted Python 3 O
8 + %	C ▶ ↓ HRun ■ C ▶ Maskdown ∨ □	
	<pre>print('Test accuracy:', test_scores[1])</pre>	
	1:si 1/2 0.5 Million (million) (million) (million) (million) (million) (million) 44000 /4000 [**********************************	1 - val_acc: 0.8 0 - val_acc: 0.9 9 - val_acc: 0.9 66 - val_acc: 0.9 18 - val_acc: 0.9
	References This script was adapted from: https://www.lensorflow.org/guidelmixed_precision	

So, you can see when you run it normally this is steps it takes 1 second one seconds 14 microseconds 1 seconds second. So, this shows the efficacy imagine why you are running a bigger model, the scale up will be the differences will be very glaring that you can very easily see as compared to small model this is just a small demo I must say.

So, from here so we are back here, so I have explained how to use the automatic mixed precision with tensorflow rather it also exists in pytorch. So, this is how you can do that in pytorch. So, you import from cuda your automatic mixed precision with grad scalar, then for optimization here you auto cast as well you get your loss.

(Refer Slide Time: 26:37)

<pre>import torch # Creates once at the beginning of training scaler = torch.cuda.amp.GradScaler()</pre>	
for data, label in data_iter: optimizer.zero_grad() # Casts operations to imixed precision with torch.cuda.amp.autocast(): loss = model(data)	
# Scales the loss, and calls backward() # to create scaled gradients scaler.scale(toss).backward()	
<pre># Unscales gradients and calls # or skips optimizer.step() scaler.step(optimizer)</pre>	
<pre># Updates the scale for next iteration scaler.update()</pre>	

Then also you scale the loss the backward then you also would get your scaler dot step and then you update the weight as well. So, for more details you can check more details here and how to do this well, I did not prepare for the PyTorch version. That is why but I will if you check on this link it is very explanatory you can easily get how to do it very well so.

Thank you.