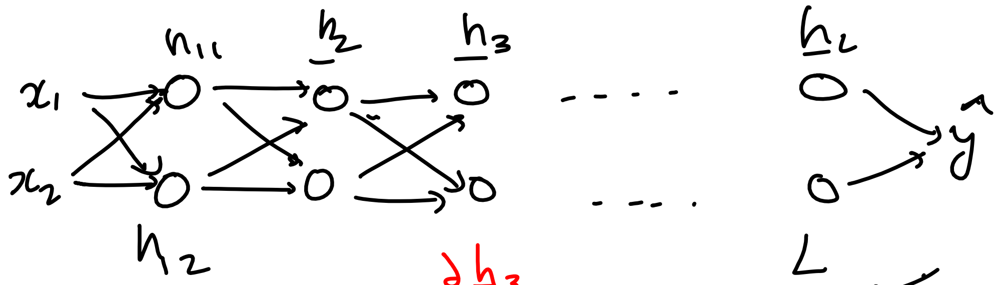
Vanishing and Explodins gradient problem



$$\frac{\partial \underline{h}_3}{\partial \underline{h}_2} \leftarrow \frac{\partial \underline{h}_L}{\partial h_{L-1}}$$

$$\underline{h}_1 = \underset{2\times2}{W} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underline{b} = W_1 \underline{x} + \underline{b}_1$$

$$\underline{h}_2 = a(W_2 \underline{h}_1 + \underline{b}_2)$$

$$\underline{h}_3 = a(W_3 \underline{h}_2 + \underline{b}_3)$$

$a(\cdot)$ = activation function

a can be ReLU
or Sigmoid
or tanh
⋮

Ignore $\boxed{a()}$ and $\boxed{\underline{b}_\ell}$

$$\underline{h}_L = W_L W_{L-1} \cdots \underbrace{W_3 \underbrace{W_2 \underbrace{W_1 \underline{x}}_{\underline{h}_1}}_{\underline{h}_2}}_{\underline{h}_3}$$

64-bit float
$10^{-300} - 10^{300}$
32-bit float
$10^{-60} - 10^{60}$

$$W_\ell = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

$$h_L = [W_\ell]^{100} \underline{x} = \begin{bmatrix} 10^{100} & 0 \\ 0 & 10^{100} \end{bmatrix} \underline{x}$$

Deep network
100-Layer

$$h_L = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}^{100} = \begin{bmatrix} 10^{-100} & 0 \\ 0 & 10^{-100} \end{bmatrix} \underline{x}$$

If you start with too small or too large
weights, the $\overset{\text{gradient}}{\text{activation}}$ $\overset{\text{values}}{\text{values}}$ in a deep network
can explode or vanish
(overflow) (underflow)

Techniques to avoid Vanish/Explode

① Normalize the input

② Normalize the weights

① Normalizing the input

Train and test

$D_{train} \sim P(D) \quad | \quad D_{test} \sim P(D)$

$$D_{train} = \{(\underline{x}_1, y_1) \dots (\underline{x}_n, y_n)\}$$

$$\boxed{\mathbb{E}[X] \simeq \frac{1}{n}\sum_{i=1}^{n} x_i}^{\text{train}} \qquad \boxed{\text{Var}[X] \simeq \frac{1}{n}\sum_{i=1}^{n}(\underline{x}_i - \mathbb{E}[X])^2}^{\text{train}}$$

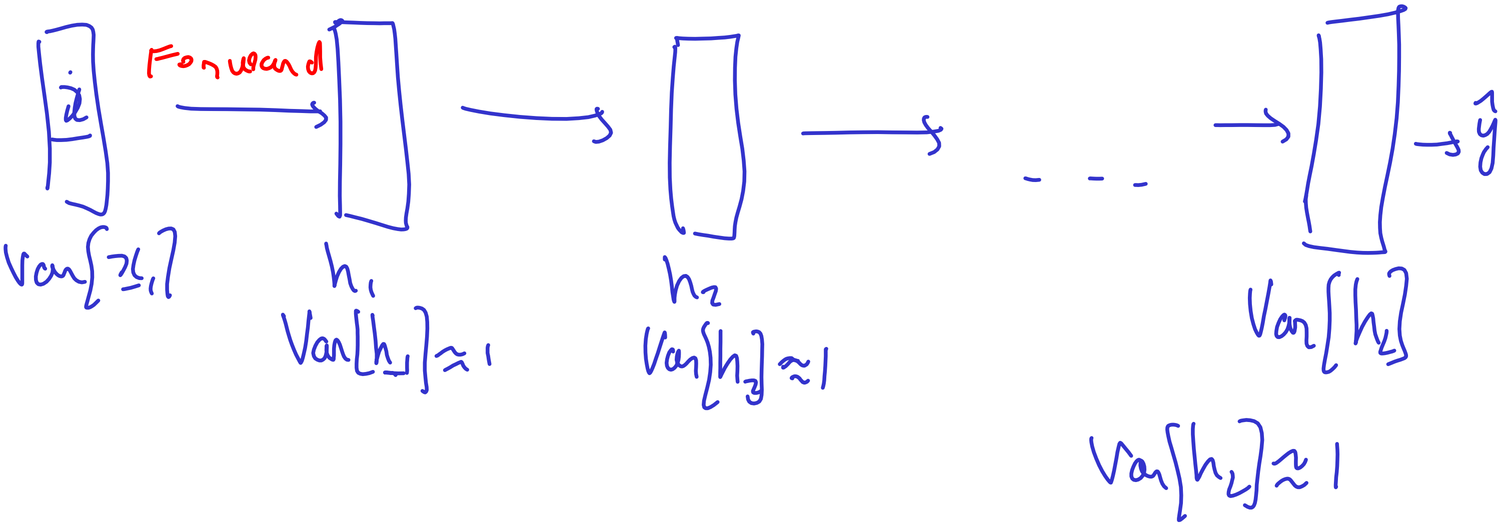
$$\hat{\underline{x}}_i = (x_i - \mathbb{E}[\underline{X}]) / \sqrt{\text{Var}[X]} \qquad \mathbb{E}[\hat{x}_i] = 0 \quad \text{Var}[\hat{X}_i] = 1$$

Normalizing the input to zero mean and unit variance

Forward pos

$\mathbb{E}[\underline{x}_i] = 0$

$\mathrm{Var}[\underline{x}_i] = 1$


$\underline{x}$
Forward
$\hat{y}$
$\mathrm{Var}[\underline{x}_i]$
h_1
$\mathrm{Var}[\underline{h}_1] \approx 1$
h_2
$\mathrm{Var}[h_2] \approx 1$
- - -
$\mathrm{Var}[h_L]$
$\mathrm{Var}[h_L] \approx 1$

$$\underline{z}_l = W_l \underline{x}_l + \underline{b}_l^{\nearrow 0}$$

$$\underline{h}_{l+1} = a(\underline{z}_l)$$

$$\underline{x}_l = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathrm{Var}(x_{li}) = 1$$

$$Z_l = \begin{bmatrix} z_{l1} \\ z_{l2} \\ \vdots \\ z_{lm} \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} z_{l1} \\ z_{l2} \\ \vdots \\ z_{lm} \end{bmatrix}}_{} = \begin{bmatrix} \\ w_{i1} \; w_{i2} \; \dots \; w_{in} \\ \\ \end{bmatrix} \begin{bmatrix} x_{l1} \\ x_{l2} \\ \vdots \\ x_{ln} \end{bmatrix}$$

$$\begin{array}{ccc} z_l & W & x \end{array}$$

$$z_{li} = \sum_{j=1}^{n} w_{ij} x_{lj}$$

$$\mathrm{Var}(x_{lj}) = \mathrm{Var}(x_l)$$

$$\mathrm{Var}(W_{ij}) = \mathrm{Var}(w_l)$$

$$\mathrm{Var}(Z_{li}) = \mathrm{Var}(Z_l)$$

$$\mathrm{Var}(z_{l}) = n \; \mathrm{Var}(w_l x_l)$$

$$w_l \perp x_l$$

$$\mathrm{Var}(w_l x_l) = \mathrm{Var}(w_l) \, E(x_l^2)$$ | why ?

Forward pass

$$\mathrm{Var}(w_l x_l) = E[(w_l x_l)^2] - (E[w_l x_l])^2$$

$$E[w_l x_l] = \iint_{w_l \, x_l} w_l x_l \, f(w_l x_l) \, dw \, dx$$

when $w_l \perp x_l$ $\qquad f(w_l x_l) = f_w(w_l) \, f_x(x_l)$

$$E[w_l x_l] = \int w_l f_w(w_l) \, dw \int x_l f_x(x_l) \, dx$$

$$= E[w_l] \, E[x]$$

$$\mathrm{Var}(w_l x_l) = E[w_l^2] \, E[x^2] - \underbrace{(E[w_l])^2}_{=0} \underbrace{E[x_l]^2}_{\substack{=0 \text{ for 1st layer} \\ \neq 0 \text{ for others}}}$$

$$= E[w_l^2] \, E[x^2]$$

$$= \mathrm{Var}(w_l) \, E[x_l^2]$$

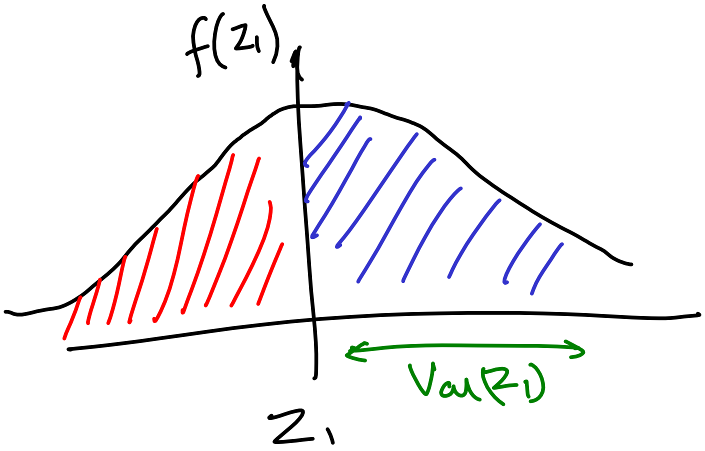
$$E[w_l^2] = \mathrm{Var}(w_l) - E[w_l]^2$$

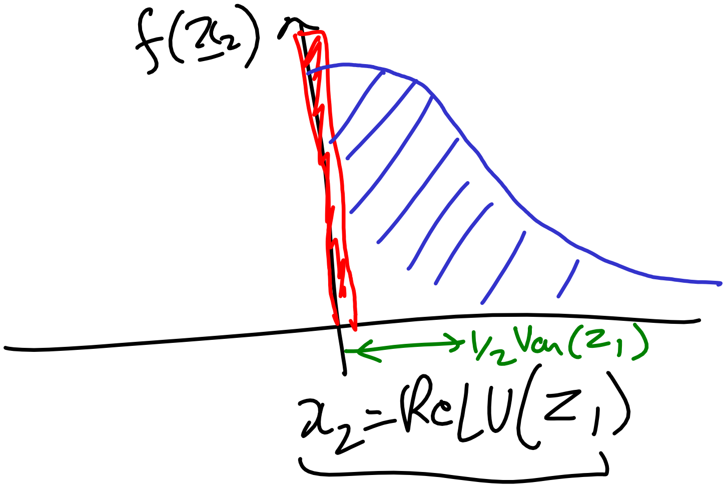
$$E[x_l^2] = \mathrm{Var}(x_l) - E[x_l]^2$$

$$E[\underline{x}_1] = 0$$

$$\underline{z}_1 = W_1 \underline{x}_1 + \underline{b}_1$$

$$\underline{x}_2 = \max(0, z_1) = \mathrm{ReLU}(0, \underline{z}_1)$$





$$Var(\underline{x}_2) \approx \frac{1}{2} Var(z_1)$$

For ReLU activation

input vector n

$$Var(x_2) = \boxed{\frac{1}{2}} n \, Var(w_1) \, Var(x_1)$$

$$W_{m \times n}$$

$$Var(h_{\ell+1}) = \frac{1}{2} \underbrace{n_\ell}_{\text{fan in}} Var(W_\ell) \, Var(h_\ell)$$

n_ℓ = size of $\underline{h}_\ell$

we want

$$Var(h_{\ell+1}) = Var(h_\ell)$$

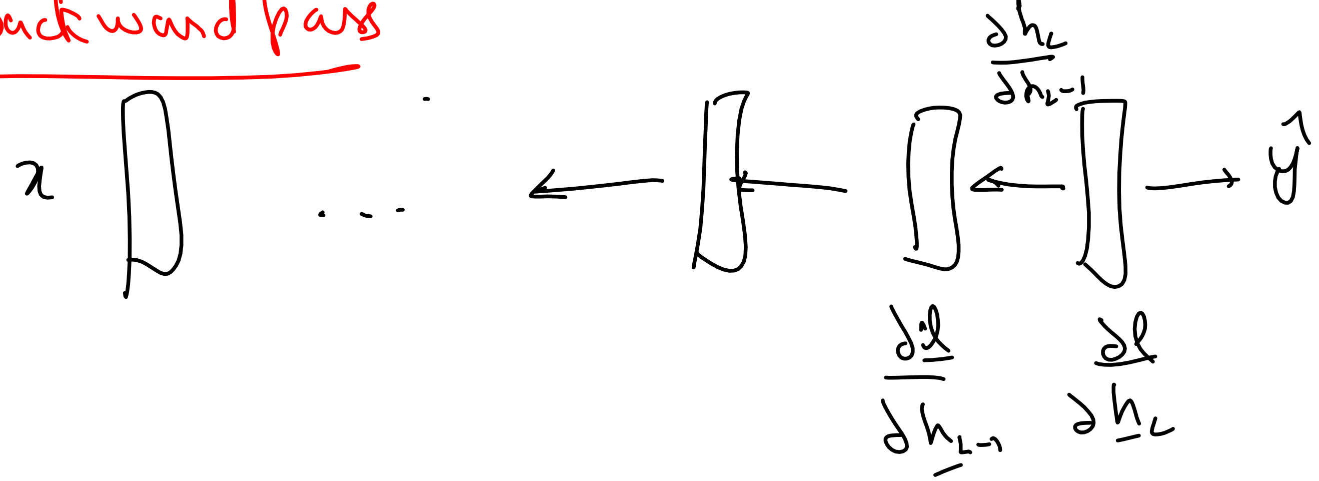
$$\Rightarrow 1 = \frac{1}{2} n_\ell \, Var(W_\ell)$$

$$\Rightarrow Var(w_\ell) = \frac{2}{n_\ell}$$

for ReLU activat

The factor $\sqrt{2}$ for ReLU is known as $\boxed{\text{gain factor}}$ for the activation

Backward pass


$\frac{\partial h_L}{\partial h_{L-1}}$
x
$\hat{y}$
$\frac{\partial l}{\partial h_{L-1}}$
$\frac{\partial l}{\partial h_L}$

$$\mathrm{Var}\Big(\underbrace{\frac{\partial l}{\partial h_l}}_{\substack{n\times 1\\ \text{vecto}}}\Big) = \frac{1}{2}\,\underbrace{n_{l+1}}_{n}\,\mathrm{Var}(\underbrace{W_l^T}_{n\times m})\,\mathrm{Var}\Big(\underbrace{\frac{\partial l}{\partial h_{l+1}}}_{\substack{m\times 1\\ \text{vecto}}}\Big)$$

$$\mathrm{Var}(W_l) = \frac{2}{n_{l+1}}$$

① He initialization or Kaiming initialization

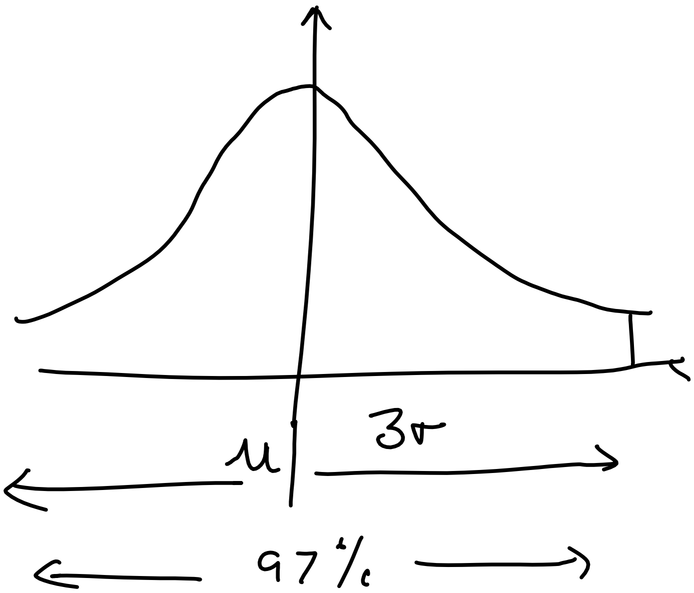
either use fan in n_l

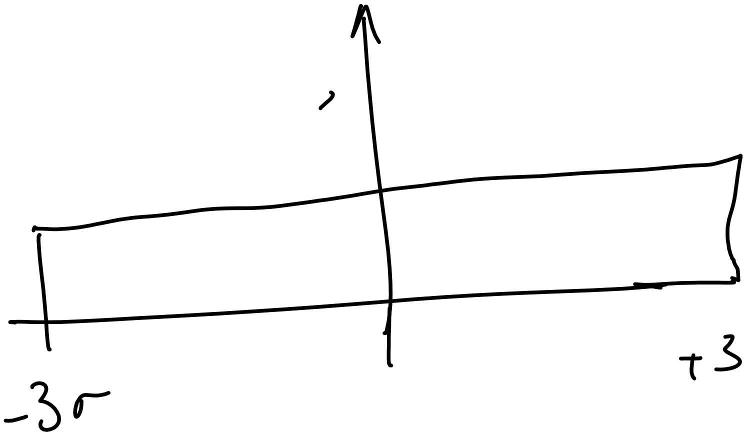
or use fan out n_{l+1}

$$w_l \sim N\left(0, \frac{gain}{\sqrt{n_l}}\right) \leftarrow \text{fan in}$$

or

$$w_l \sim N\left(0, \frac{gain}{\sqrt{n_{l+1}}}\right) \leftarrow \text{fan out}$$





$$w_l \sim U\left[\frac{-3\,gain}{\sqrt{n_l}}, \frac{+3\,gain}{\sqrt{n_l}}\right]$$

or $$w_l \sim U\left[\frac{-3\,gain}{\sqrt{n_{l+1}}}, \frac{+3\,gain}{\sqrt{n_{l+1}}}\right]$$

② Glorot or Xavier Initialization

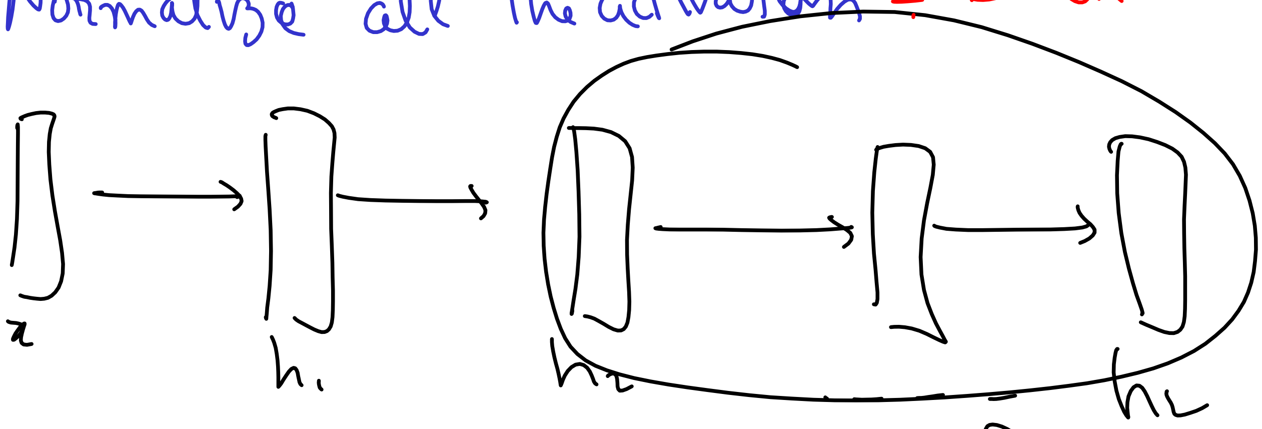
$$Var(w_l) = \frac{4}{\underbrace{n_l}_{\text{fan in}} + \underbrace{n_{l+1}}_{}} \leftarrow \text{fan out} \quad \text{for ReLU}$$

$$w_\ell \sim \mathcal{N}\left(0, \frac{gain \times 2}{\sqrt{n_\ell + n_{\ell+1}}}\right)$$

① Normalizing The input

② Initializing the weights

③ → Normalize all the activations = Batch Normalization



$$\left.\begin{aligned}
\hat{h}_\ell &= \frac{h_\ell - \frac{1}{B}\sum_{i=1}^{B} h_{\ell i}}{\sqrt{\frac{1}{B}\sum_{i=1}^{B}(h_\ell - \mu_\ell)^2}} \\
\tilde{h}_\ell &= \underbrace{\begin{bmatrix}\gamma_1 & & 0\\ & \gamma_2 & \\ 0 & & \ddots\ \gamma_n\end{bmatrix}}_{\text{Learned by SGD}} \hat{h}_\ell + \underbrace{\begin{bmatrix}\beta_1\\ \beta_2\\ \vdots\\ \beta_n\end{bmatrix}}_{\text{Bias}}
\end{aligned}\right\}\text{BatchNorm}$$

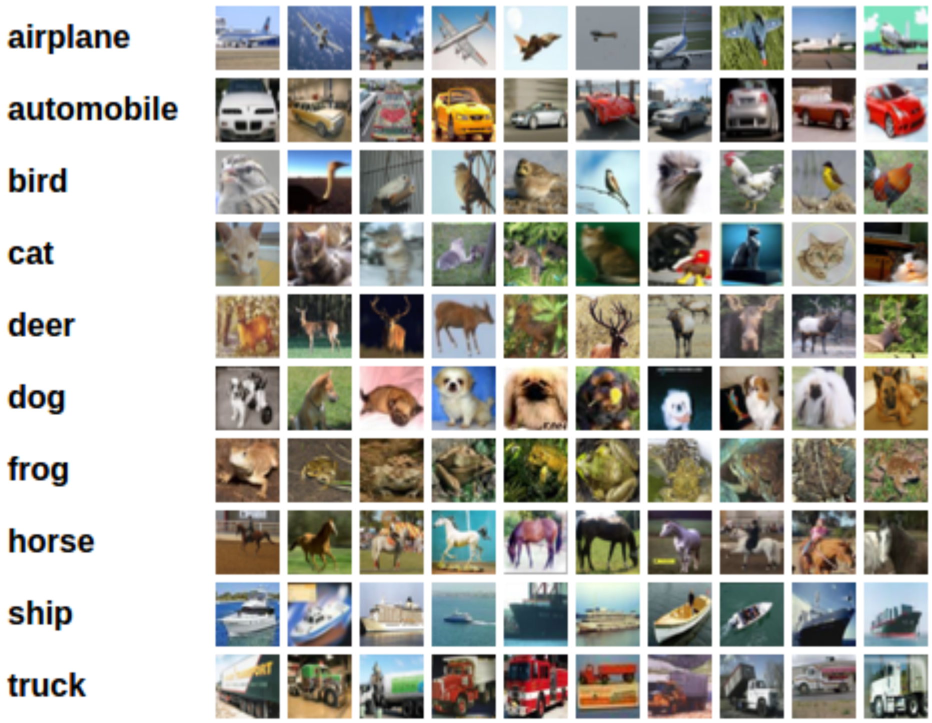
```
# Adapted from: Chapter 7 and 8 of Deep Learning with Pytorch by Eli Stevens (2020)
# References
# 1. 2010-glorot.pdf from milestone papers
# 2. 2015-HeInitialization.pdf from milestone papers
# 3. 2015-BatchNorm.pdf from milestone papers
# 4. Section 11.4 of UDLBook
# 5. Chapter 7 of UDLBook
try:
    import torch as t
    import torch.nn as tnn
except ImportError:
    print("Colab users: pytorch comes preinstalled. Select Change Ru")
    print("Local users: Please install pytorch for your hardware using instructions
    print("ACG users: Please follow instructions here: https://vikasdhiman.info/ECE

    raise

if t.cuda.is_available():
    DEVICE="cuda"
elif t.mps.is_available():
    DEVICE="mps"
else:
    DEVICE="cpu"

DTYPE = t.get_default_dtype()


## Doing it the Pytorch way without using our custom feature extraction

import torch
import torch.nn
import torch.optim
import torchvision
from torchvision.transforms import ToTensor, Compose, Normalize
from torch.utils.data import DataLoader

torch.manual_seed(17)
DATASET_MEAN = [0.4914, 0.4822, 0.4465]
DATASET_STD = [0.2470, 0.2435, 0.2616]
# Getting the dataset, the Pytorch way
all_training_data = torchvision.datasets.CIFAR10(
    root="data",
    train=True,
    download=True,
    transform=Compose([ToTensor(),
                       Normalize(DATASET_MEAN, # dataset mean
                                 DATASET_STD)]) # dataset std
)

test_data = torchvision.datasets.CIFAR10(
```

▸ Executing (54s) <cell li... › tr... › __n... › _next... › f... › <list... › __geti... › __geti... › __c... › __c... › to_te... ••• ✕

```
    train=False,
    download=True,
    transform=Compose([ToTensor(),
                       Normalize(DATASET_MEAN, # dataset mean
                                 DATASET_STD)]) # dataset std
)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to data/c
100%|██████████| 170498071/170498071 [00:02<00:00, 73180943.58it/s]
Extracting data/cifar-10-python.tar.gz to data
Files already downloaded and verified
```

```
training_data, validation_data = torch.utils.data.random_split(all_training_data, |
```

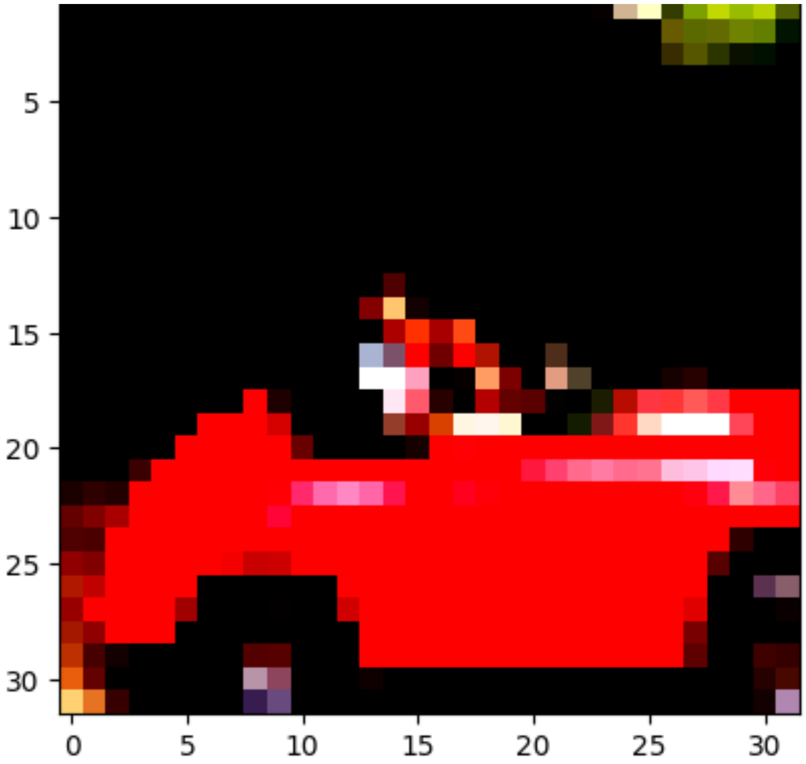


```
img, label = all_training_data[99]
img.shape, label
```

```
(torch.Size([3, 32, 32]), 1)
```

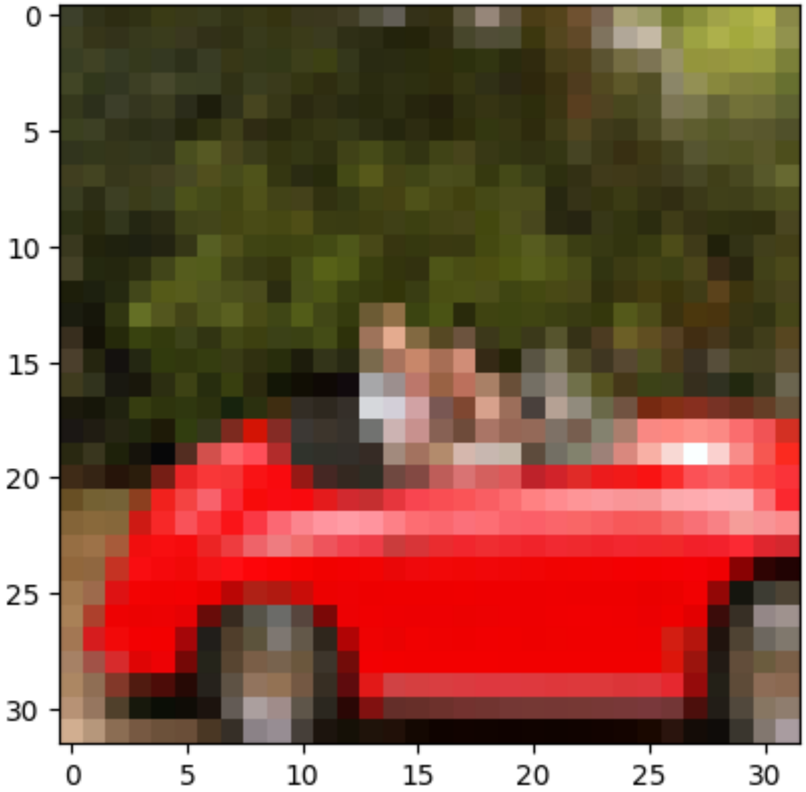
```
import matplotlib.pyplot as plt
plt.imshow(img.permute(1, 2, 0))
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wi
<matplotlib.image.AxesImage at 0x7f3fd043b430>
```



```
plt.imshow((img.permute(1, 2, 0) *  torch.Tensor(DATASET_STD)
           +  torch.Tensor(DATASET_MEAN)))
```

```
<matplotlib.image.AxesImage at 0x7f3fd02df4c0>
```



```
imgs = torch.stack([img_t for img_t, _ in all_training_data], dim=3)
imgs.reshape(3, -1).mean(dim=-1), imgs.reshape(3, -1).std(dim=-1)
```

```
    (tensor([-1.2762e-06, -1.7074e-04,  1.1819e-04]),
     tensor([1.0001, 0.9999, 1.0000]))
```

```
import pickle
cifar_meta = pickle.load(open("data/cifar-10-batches-py/batches.meta", "rb"), encod
class_names = [c.decode('utf-8') for c in cifar_meta[b'label_names']]
class_names
```

```
    ['airplane',
     'automobile',
     'bird',
     'cat',
     'deer',
     'dog',
     'frog',
     'horse',
     'ship',
     'truck']
```

```
# Hyper parameters
learning_rate = 1e-3 # controls how fast the gradient descent goes
batch_size = 64
epochs = 5
momentum = 0.9

training_dataloader = DataLoader(training_data, shuffle=True, batch_size=batch_size
validation_dataloader = DataLoader(validation_data,  batch_size=batch_size)
test_dataloader = DataLoader(test_data,  batch_size=batch_size)
X, y = next(iter(training_dataloader))
X.shape
```

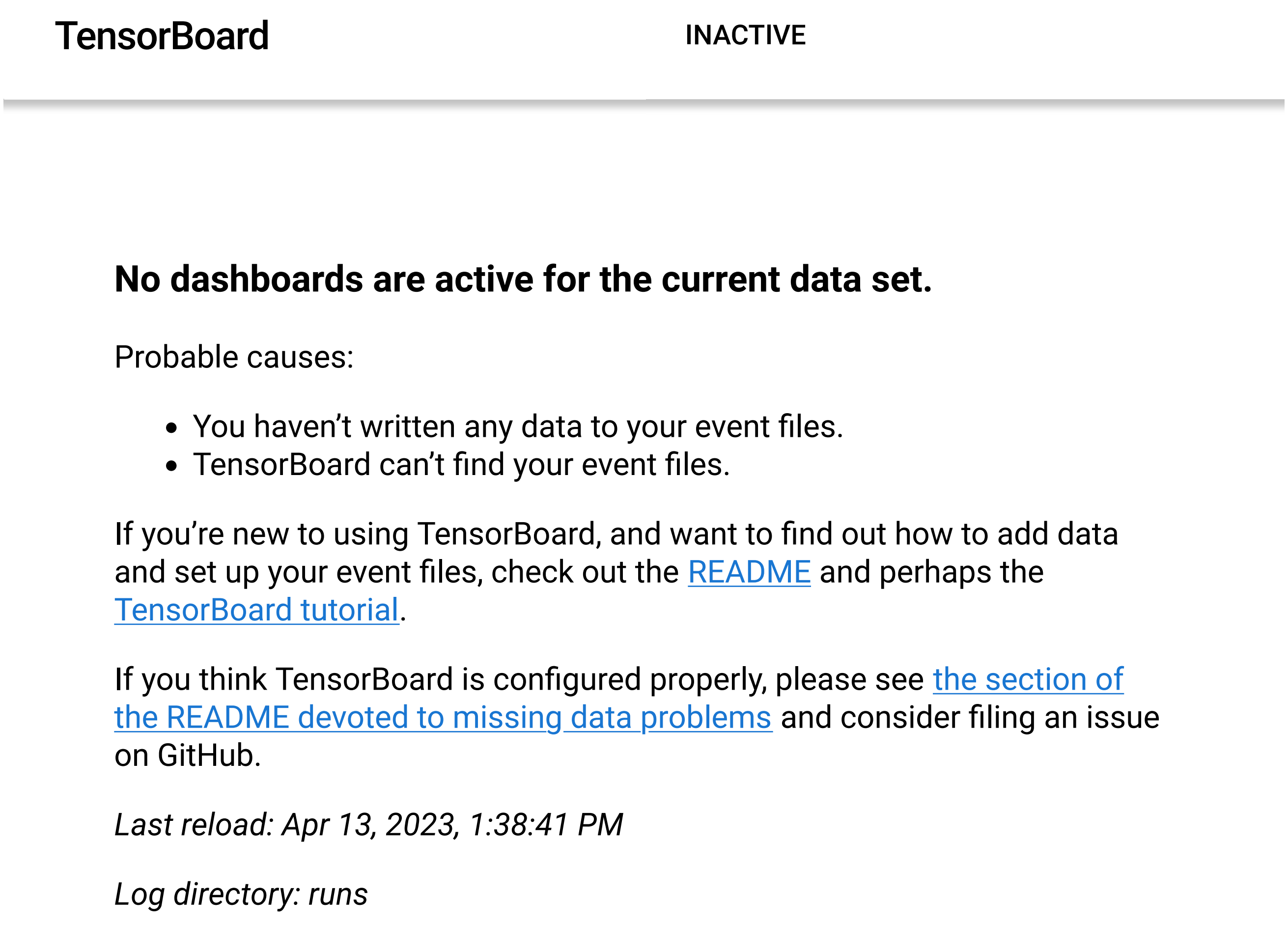
```
    torch.Size([64, 3, 32, 32])
```

```
!pip install tensorboard
```

```
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
    Requirement already satisfied: tensorboard in /usr/local/lib/python3.9/dist-pa
    Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.9/dis
    Requirement already satisfied: protobuf>=3.19.6 in /usr/local/lib/python3.9/di
    Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.9/
    Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.9/dist-
    Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib
    Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.9/dist-p
    Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.9/dis
    Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/l
    Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/li
    Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3
    Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.9/dist
```

```
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyth
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/pytho
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/py
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.9/
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.9/di
```

```
%load_ext tensorboard
%tensorboard --logdir=runs
```



No dashboards are active for the current data set.

Probable causes:

- You haven’t written any data to your event files.
- TensorBoard can’t find your event files.

If you’re new to using TensorBoard, and want to find out how to add data and set up your event files, check out the README and perhaps the TensorBoard tutorial.

If you think TensorBoard is configured properly, please see the section of the README devoted to missing data problems and consider filing an issue on GitHub.

Last reload: Apr 13, 2023, 1:38:41 PM

Log directory: runs

```
from torch.utils.tensorboard import SummaryWriter
from torch.optim.lr_scheduler import  ReduceLROnPlateau
import os
writer = SummaryWriter()

loss = torch.nn.CrossEntropyLoss()

# class Net(tnn.Module):
#   def __init__(self):
#     super().__init__()
#     # define input size, hidden layer size, output size
#     D_i, D_k, D_o = 3*32*32, 100, 10
#     self.f = tnn.Flatten()
#     self.l1 = tnn.Linear(D_i, D_k, bias=False)
#     self.b1 = tnn.BatchNorm1d(D_k)
#     self.a1 = tnn.ReLU()
#     self.l2 = tnn.Linear(D_k, D_o)


#   def forward(self, x):
#     self.f_out = self.f(x)
#     self.l1_out = self.l1(self.f_out)
#     self.b1_out = self.b1(self.l1_out)
#     self.a1_out = self.a1(self.b1_out)
#     self.l2_out = self.l2(self.a1_out)
#     return self.l2_out

# model = Net()

# define input size, hidden layer size, output size
D_i, D_k, D_o = 3*32*32, 100, 10
model = tnn.Sequential(
    tnn.Flatten(),
    tnn.Linear(D_i, D_k, bias=False),
    tnn.BatchNorm1d(D_k),
    tnn.ReLU(),
    tnn.Linear(D_k, D_o)
)
```

```
# print(list(model.named_parameters()))

# Glorot or Xavier initialization of weights
def init_weights(m):
    if isinstance(m, (tnn.Linear, tnn.Conv2d)):
        torch.nn.init.kaiming_uniform_(m.weight, nonlinearity='relu')
        # m.bias.data.fill_(0)

model.apply(init_weights)

def loss_and_accuracy(model, loss, validation_dataloader, device=DEVICE):
        # Validation loop
        validation_size = len(validation_dataloader.dataset)
        num_batches = len(validation_dataloader)
        test_loss, correct = 0, 0

        with torch.no_grad():
            model.eval() # Put model in eval mode, affects layers like dropout and
            for X, y in validation_dataloader:
                X = X.to(device)
                y = y.to(device)
                pred = model(X)
                test_loss += loss(pred, y)
                correct += (pred.argmax(dim=-1) == y).type(DTYPE).sum()

        test_loss /= num_batches
        correct /= validation_size
        return test_loss, correct

def train(model, loss, training_dataloader, validation_dataloader, device=DEVICE, c
    # Define optimizer
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=mome
    scheduler = ReduceLROnPlateau(optimizer, 'min')
    model.to(device)
    t0 = 0
    if not ignore_chkpt and os.path.exists(f"runs/{chkpt_name}"):
        checkpoint = torch.load(f"runs/{chkpt_name}")
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        t0 = checkpoint['epoch']

    for t in range(t0, epochs):
        # Train loop
        training_size = len(training_dataloader.dataset)
        nbatches = len(training_dataloader)
        model.train() # Put model in train mode, affects layers like dropout and ba
        for batch, (X, y) in enumerate(training_dataloader):
            X = X.to(device)
            y = y.to(device)
            # Compute prediction and loss
                pred = model(X)
            pred = model(X)
            loss_t = loss(pred, y)

            # Backpropagation
            optimizer.zero_grad()
            loss_t.backward()
            optimizer.step()

            if batch % 100 == 0:
                writer.add_scalar("Train/loss_batch", loss_t,  t*nbatches + batch)
                loss_t, current = loss_t.item(), (batch + 1) * len(X)
                print(f"loss: {loss_t:>7f}  [{current:>5d}/{training_size:>5d}]", e
        valid_loss, correct = loss_and_accuracy(model, loss, validation_dataloader,
        scheduler.step(valid_loss)

        # writer.add_scalar("Layers/l1_var", model.a1_out.var(), t)
        writer.add_scalar("Train/loss", loss_t, t)
        writer.add_scalar("Valid/loss", valid_loss, t)
        writer.add_scalar("Valid/accuracy", correct, t)
        print(f"Validation Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {v
        if t % 3 == 0:
            torch.save({
                'epoch': t,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict()
                }, f"runs/{chkpt_name}")
    return model

trained_model = train(model, loss, training_dataloader, validation_dataloader, chkp

test_loss, correct = loss_and_accuracy(model, loss, test_dataloader)
print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f}
```

```
[('1.weight', Parameter containing:
tensor([[-0.0059,  0.0155, -0.0170,  ..., -0.0007,  0.0142, -0.0011],
        [ 0.0020,  0.0060, -0.0100,  ..., -0.0012, -0.0049, -0.0180],
        [ 0.0047, -0.0180, -0.0040,  ...,  0.0117, -0.0110,  0.0072],
        ...,
        [-0.0168,  0.0120, -0.0039,  ..., -0.0075, -0.0160, -0.0073],
        [-0.0117, -0.0150, -0.0128,  ..., -0.0061,  0.0112, -0.0127],
        [ 0.0035,  0.0044,  0.0070,  ...,  0.0105,  0.0122,  0.0044]],
       requires_grad=True)), ('2.weight', Parameter containing:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], requires_grad=True)), ('2.bia
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0
        0., 0., 0., 0.], requires_grad=True)), ('4.weight', Parameter contain
```

```
             0., 0., 0., 0.], requires_grad=True)), ('4.weight', Parameter contain
    tensor([[ 0.0709,  0.0782,  0.0848, -0.0909, -0.0726,  0.0927,  0.0114, -0.010
              -0.0608, -0.0433,  0.0770, -0.0703, -0.0210, -0.0316, -0.0518,  0.045
               0.0136, -0.0489, -0.0238, -0.0347,  0.0809,  0.0455,  0.0984, -0.041
              -0.0562, -0.0729,  0.0985,  0.0218, -0.0347, -0.0804,  0.0060,  0.019
               0.0298, -0.0306,  0.0793,  0.0897,  0.0392, -0.0096,  0.0931,  0.017
              -0.0718, -0.0351, -0.0133,  0.0873, -0.0747, -0.0172, -0.0958,  0.008
              -0.0508, -0.0934,  0.0348, -0.0389,  0.0372, -0.0371,  0.0141, -0.070
              -0.0675,  0.0806, -0.0965, -0.0980,  0.0127,  0.0440, -0.0584,  0.092
               0.0964, -0.0403,  0.0963,  0.0796, -0.0636, -0.0133,  0.0358, -0.010
               0.0373, -0.0487,  0.0901,  0.0995,  0.0008,  0.0702,  0.0146,  0.080
               0.0094,  0.0963,  0.0146,  0.0245,  0.0065, -0.0438, -0.0614,  0.074
               0.0128, -0.0173, -0.0965, -0.0417, -0.0960, -0.0260,  0.0025, -0.089
               0.0284,  0.0480, -0.0144, -0.0521],
             [-0.0851,  0.0805,  0.0900, -0.0173, -0.0005, -0.0925,  0.0612, -0.041
              -0.0946,  0.0524,  0.0226, -0.0501,  0.0109,  0.0450,  0.0653,  0.092
               0.0857, -0.0151, -0.0560,  0.0294, -0.0166,  0.0335,  0.0782,  0.001
               0.0454, -0.0105, -0.0878,  0.0290, -0.0168, -0.0111,  0.0344, -0.025
               0.0367,  0.0931,  0.0323,  0.0160,  0.0651, -0.0514,  0.0038, -0.019
               0.0393,  0.0193,  0.0465, -0.0680, -0.0848,  0.0457, -0.0351,  0.060
               0.0859, -0.0309, -0.0798,  0.0473,  0.0099, -0.0528,  0.0280,  0.060
              -0.0579,  0.0152, -0.0115, -0.0596, -0.0682, -0.0161, -0.0451,  0.091
               0.0425,  0.0418,  0.0512, -0.0760, -0.0100, -0.0437,  0.0616, -0.085
              -0.0113, -0.0864,  0.0253, -0.0600, -0.0555, -0.0045, -0.0403, -0.098
              -0.0446, -0.0178, -0.0258, -0.0181,  0.0706, -0.0967, -0.0053, -0.025
               0.0818,  0.0196,  0.0578, -0.0638, -0.0309, -0.0526, -0.0533,  0.005
               0.0894, -0.0606,  0.0309,  0.0490],
             [ 0.0030,  0.0755,  0.0253,  0.0273, -0.0347, -0.0118,  0.0463, -0.002
               0.0704, -0.0103, -0.0588, -0.0779,  0.0074, -0.0607,  0.0695,  0.071
               0.0119, -0.0048, -0.0443, -0.0292, -0.0643, -0.0809,  0.0356,  0.047
               0.0235,  0.0922,  0.0020, -0.0316, -0.0575, -0.0695,  0.0368,  0.010
              -0.0237,  0.0079, -0.0558, -0.0597,  0.0246, -0.0686,  0.0042, -0.094
               0.0218,  0.0899,  0.0337, -0.0536,  0.0333,  0.0166,  0.0354, -0.000
               0.0558, -0.0413,  0.0592, -0.0987,  0.0463, -0.0291, -0.0359, -0.074
              -0.0924,  0.0776,  0.0422, -0.0312, -0.0908,  0.0573,  0.0588,  0.090
              -0.0928, -0.0131, -0.0091,  0.0777, -0.0899,  0.0634,  0.0807,  0.018
              -0.0523,  0.0134,  0.0306,  0.0451,  0.0528,  0.0679, -0.0788, -0.005
              -0.0083,  0.0255, -0.0343, -0.0378,  0.0787, -0.0797,  0.0828, -0.025
              -0.0939, -0.0582, -0.0016,  0.0378,  0.0073, -0.0678, -0.0449,  0.030
```

Colab paid products - Cancel contracts here