## Parameter initialization for Vanishing and Exploding gradient problem

```
# 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
```

MLP ~ 2 layers
CNN ~ 4 layer
100 - layers
1000 - layers

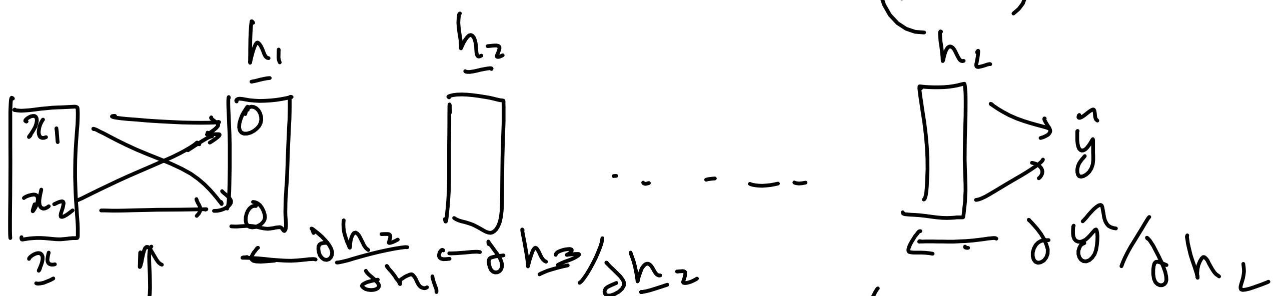
Argument 1: If you initialize the weights too small or too large, the activations in a deep network can explode or vanish.

Overflow error

$-10^{+300} \longleftrightarrow 10^{-300} \leftarrow$ float32 $\rightarrow 10^{+300}$

Underflow error (NaN)

(NaN)




Fully connected
$\equiv$ Dense Layer
$\equiv$ Linear layer

$$\underline{h}_1 = a(W_1 \underline{x} + \underline{b}_1)$$

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

$$\vdots$$

$$a(z) = \mathrm{ReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \le 0 \end{cases} \qquad h_L = a(W_L \underline{h}_{L-1} + \underline{b}_L)$$

forward pass

$$\underline{h}_L = \underbrace{W_L W_{L-1} \cdots W_2 W_1 \underline{x}} + \cdots \cdots \underline{b}_1$$

Assume

$$W_\ell \approx \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \qquad L = 100 \qquad \underline{h}_L = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}^{100} \underline{x}$$

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

can cause overflow

$$W_\ell \approx \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \longrightarrow h_L = \begin{bmatrix} 1^{-100} & 0 \\ 0 & 1^{-100} \end{bmatrix} \underline{x}$$

can cause underflow

Backward pass

$$\frac{\partial l}{\partial \underline{x}} = \frac{\partial l}{\partial \underline{\hat{y}}} \frac{\partial \hat{y}}{\partial \underline{h}_L} \cdots \underbrace{\frac{\partial \underline{h}_3}{\partial \underline{h}_2} \frac{\partial \underline{h}_2}{\partial \underline{h}_1} \frac{\partial h_1}{\partial \underline{x}}}_{\text{very big or very small}}$$

$$\begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

## Solutions

① Preprocess input to normalize it

$$\underline{\mu} = \frac{1}{N} \sum \underline{x} \qquad \text{var}(\underline{x}) = \frac{1}{(N-1)} \sum (\underline{x} - \underline{\mu})^2$$

$$\tilde{x} = \frac{\underline{x} - \underline{\mu}}{\sqrt{\text{var}(\underline{x})}}$$

zero mean
unit variance

② We want all layers activation/ hidden units to be zero mean unit variance at least initially

Initialization

③ Batch normalization (during training)

④ Gradient clipping (eventual hack) (should not be happening too much)

if $\left|\frac{\partial l}{\partial w_{\ell ij}}\right| > \max$ then $\frac{\partial l}{\partial w_{\ell ij}} = \pm \max$

(2)

## Parameter initialization

$$\underline{h}_\ell = a(W_\ell \underline{h}_{\ell-1} + \underline{b}_\ell)$$

$\underline{h}_0 = \underline{x}$ is the input

$\underbrace{\mathbb{E}[\underline{h}_0] = 0}_{\text{zero mean}}$ $\quad\quad$ $\underbrace{\text{var}[\underline{h}_{0,i}] = 1}_{\text{unit variance}}$

we want $\mathbb{E}[\underline{h}_\ell] = 0$, $\text{var}[\underline{h}_{\ell,i}] = 1$

how should we initialize $W_\ell$ and $b_\ell$?

### Linear layer

$$\text{Var}(h_\ell) \stackrel{?}{=} \text{Var}(W_\ell \underline{h}_{\ell-1} + \underline{b}_\ell) \quad ?$$

### ReLU layer

$$\text{Var}(\text{ReLU}(z)) \stackrel{?}{=} \quad \text{Var}(z)$$

$$\underbrace{\begin{bmatrix} h_{l1} \\ \vdots \\ \boxed{h_{li}} \\ \vdots \\ h_{ln} \end{bmatrix}}_{\underline{h}_l} = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ \boxed{\quad\quad\quad\quad} & & \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{bmatrix} \underbrace{\begin{bmatrix} h_{l-1\,1} \\ \vdots \\ h_{l-1\,m} \end{bmatrix}}_{\underline{h}_{l-1}} + \underline{b}_l$$

$$h_{li} = \underbrace{\sum_{j=1}^{m} w_{ij}\, h_{l-1,j} + b_{li}}_{\text{var}}$$

$$\text{var}(h_{li}) =$$

$$\mu = \mathbb{E}[z]$$

$$\begin{aligned}
\text{var}(z) &= \mathbb{E}[(z-\mu)^2] \\
&= \mathbb{E}[z^2 - 2\mu z + \mu^2] \\
&= \mathbb{E}[z^2] - 2\mu \underbrace{\mathbb{E}[z]}_{\mu} + \mathbb{E}[\mu^2] \\
&= \mathbb{E}[z^2] - \mu^2
\end{aligned}$$

$$\mathrm{var}(h_{il}) = \mathbb{E}\left[\left(\sum_{j=1}^{m} w_{ij} h_{l-1,j} + \underline{b_{li}}\right)^2\right]$$

$$- \mathbb{E}\left[\left(\sum_{j=1}^{m} w_{ij} h_{l-1,j} + b_{li}\right)\right]$$

---

$$\underline{\mathrm{Var}(x+y)} = \mathbb{E}\left[(x+y)^2\right] - \mathbb{E}[x+y]$$

$$= \underbrace{\mathbb{E}\left[x^2 + y^2 + 2xy\right]} - \mathbb{E}[x] - \mathbb{E}[y]$$

$$= \mathbb{E}[x^2] + \mathbb{E}[y^2] + 2\mathbb{E}[xy] - \mathbb{E}[x] - \mathbb{E}[y]$$

$$= \mathrm{Var}(x) + \mathrm{Var}(y) + 2\underbrace{\mathbb{E}[xy]}_{\sim \text{covariance}}$$

If $\underline{x}$ is independent of $\underline{y}$

$$P(x, y) = P(x)P(y)$$

$$P(x \mid y) = P(x)$$

then $\mathbb{E}[xy] = \mathbb{E}[x]\,\mathbb{E}[y]$

$$= \int_y \int_x x\,y\,\underbrace{f_{xy}(x,y)}\,dx\,dy$$

$$= \int\int x\,y\,\overbrace{f_x(x)\,f_y(x)}\,dx\,dy$$

$$= \int x f_x(x)\,dx \int y f_y(x)\,dy$$

$$= \mathbb{E}[x]\,\mathbb{E}[y]$$

If $x \perp y$ then $\mathrm{Var}(x+y) = \mathrm{Var}(x) + \mathrm{Var}(y) + 2\,\mathbb{E}[x]\,\mathbb{E}[y]$

If $x \perp y$, $\left.\begin{matrix}\mathbb{E}[x] = 0, \\ \mathbb{E}[y] = 0\end{matrix}\right\}$ $\mathrm{Var}(x+y) = \mathrm{Var}(x) + \mathrm{Var}(y)$

$$\mathrm{Var}(h_{il}) = \underbrace{\mathrm{Var}\left(\sum_{j=1}^{m} w_{ij} h_{l-1,j}\right)} + \underbrace{\mathrm{Var}(b_{ij})}$$

$$\mathbb{E}\left[\sum_{j=1}^{m} w_{ij} h_{l-1,j}\right] = 0 \qquad \underbrace{\mathbb{E}[b_{ij}] = 0}$$

$$\mathbb{E}[h_{l-1,j}] = 0$$

① Initialze

$b_{ij}$ = zero mean

$$\mathrm{Var}(w_{ij}) = \sigma^2 \text{ for all } ij$$

$$\mathbb{E}[w_{ij} \underbrace{h_{l-1,j}}] = 0$$

$$\mathrm{var}\left(\sum_{j=1}^{m} w_{ij} h_{l-1,j}\right) = m\, \underbrace{\mathrm{var}(w_{ij} h_{l-1,j})}$$

$$\mathbb{E}[xy] = \mathbb{E}[x]\,\mathbb{E}[y] \text{ when } x \perp y$$

$$\mathrm{Var}[xy] = \mathrm{Var}[x]\,\mathrm{Var}[y] \quad \text{when } x \perp y$$

and $\mathbb{E}[x] = 0$

$\mathbb{E}[y] = 0$

$$\overbrace{\mathrm{var}\left(\sum_{j=1}^{m} w_{ij} h_{\ell-1,j}\right)}^{\mathrm{var}(h_{\ell,j}) - \mathrm{var}(b_{i,j})} = m\,\mathrm{Var}(w_{ij})\,\mathrm{var}(h_{\ell-1,j})$$

$$\mathrm{var}(w_{ij}) = \frac{1}{m}\,\frac{\mathrm{var}(h_{\ell,j}) \simeq 1}{\mathrm{var}(h_{\ell-1,j}) \simeq 1}$$

$$\underbrace{\mathrm{var}(w_{ij}) = \frac{1}{m}}$$

When you are initializing weights $\underline{W}_\ell$ and biases $\underline{b}_\ell$

then Initialize $\underline{b}_\ell = 0$

STD

$\underline{W}_\ell \in \mathbb{R}^{n \times m}$, $\underline{w_{\ell ij} \sim \mathcal{N}\left(0, \sqrt{\frac{1}{m}}\right)}$ ← Forward pass

when

$W \in \mathbb{R}^{n \times m}$

$$\frac{\partial l}{\partial \underline{h}_{l-1}} = \underbrace{\frac{\partial l}{\partial \underline{h}_l}}_{1 \times n} \underbrace{\boxed{\frac{\partial \underline{h}_l}{\partial \underline{h}_{l-1}}}}_{n \times m}$$

$1 \times m$

$var(\ ) = 1$ $\quad$ $var(\cdot) = 1$ $\quad$ $var(\ )?$

$$(\underline{h}_l = W \underline{h}_{l-1} + \underline{b})$$

Backward pass

STD

$$w_{ij} \sim \mathcal{N}\left(0, \sqrt{\frac{1}{n}}\right)$$

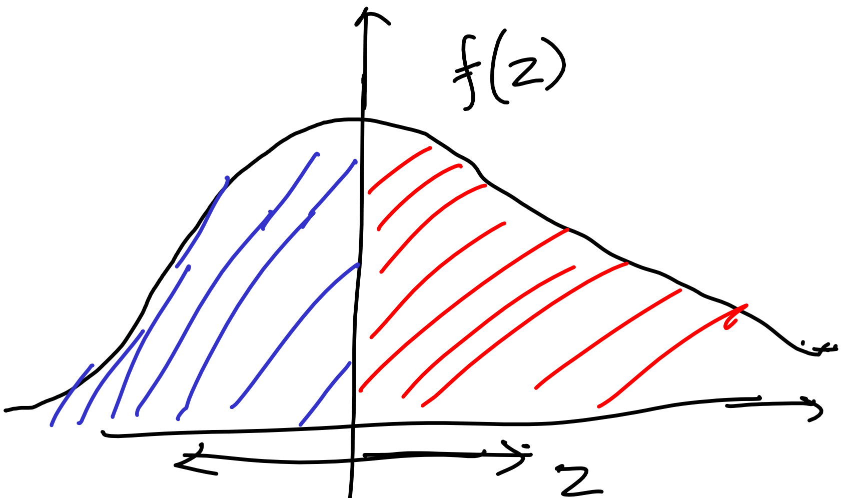
He initialization or KAIMING initialization

② GLOROT or XAVIER initialization

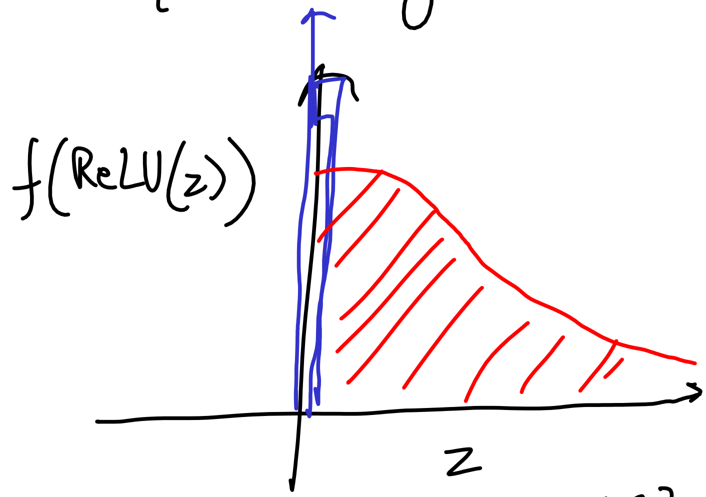
$$w_{ij} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n+m}}\right)$$

## ReLU layer

$$\mathrm{ReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \le 0 \end{cases}$$




$\mathbb{E}[z] = 0 \quad Var(z) = \sigma^2$




$\mathbb{E}[\mathrm{ReLU}(z)] > 0$

$var(\mathrm{ReLU}(z)) = \frac{\sigma^2}{2}$

$\rightarrow$ (gain factor)$^2$

gain factor for ReLU activation is $\sqrt{2}$

mean

STD $\sigma$

for KAIMING $\quad w_{ij} \sim \mathcal{N}\left(0, \frac{\text{gain factor}}{\sqrt{m}}\right)$

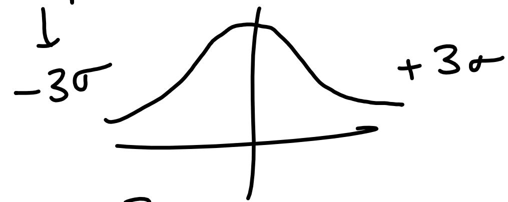
input dim

or $\quad \sim \mathcal{N}\left(0, \frac{\text{gain factor}}{\sqrt{n}}\right)$

output dim

XAVIER $\quad w_{ij} \sim \mathcal{N}\left(0, \frac{\sqrt{2} \times \text{gain factor}}{\sqrt{n+m}}\right)$

Uniform distribution instead of Gaussian




XAVIER $w_{ij} \sim U\left[-\frac{\sqrt{2} \times \text{gain factor} \times 3}{\sqrt{n+m}} \quad +\frac{\sqrt{2} \times \text{gain factor} \times 3}{\sqrt{n+m}}\right]$

Techniques to avoid Vanishing and exploding gradients

1. Normalize the input
2. Normalize the weights to keep the activations normalized

What happens to activation distribution through a linear layer?

What happens to weights through an activation layer, say ReLU?

## 7.3 Parameter initialization

We have discussed both stochastic gradient descent, and how to compute the derivatives that it requires. We now address how to initialize the parameters before we start training. To see why this is important, consider that during the forward pass, each hidden layer $\mathbf{h}_k$ is computed as:

$$\mathbf{h}_{k+1} = \mathbf{a}[\boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{h}_k],$$

where $\mathbf{a}[\bullet]$ applies the ReLU functions and $\boldsymbol{\Omega}_k$ and $\boldsymbol{\beta}_k$ are the weights and biases, respectively. Imagine that we initialize all the biases to zero, and the elements of $\boldsymbol{\Omega}_k$ according to a normal distribution with mean zero and variance $\sigma^2$. Ctonsider two scenarios:

- If the variance $\sigma^2$ is very small (e.g., $10^{-5}$), then each element of $\boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{h}_k$ will be a weighted sum of $\mathbf{h}_k$ where the weights are very small; the result is likely to have a smaller magnitude than the input. After passing through the ReLU function, values less than zero will be clipped and the range of outputs will halve.

  Consequently, the magnitudes of the activations at the hidden layers will tend to get smaller and smaller as we progress through the network.

- If the variance $\sigma^2$ is very large (e.g., $10^5$), then each element of $\boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{h}_k$ will be a weighted sum of $\mathbf{h}_k$ where the weights are very large; the result is likely to have a much larger magnitude than the input. After passing through the ReLU function, any values less than zero will be clipped and so the range of outputs will be halved; however, even after this, their magnitude might still be larger on average. Consequently, the magnitudes of the activations at the hidden layers will tend to get larger and larger as we progress through the network.

$$\begin{aligned} \mathbf{f} &= \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h} \\ \mathbf{h}' &= \mathbf{a}[\mathbf{f}], \end{aligned}$$

$$\begin{aligned} \mathbb{E}[f_i] &= \mathbb{E}\left[\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right] \\ &= \mathbb{E}[\beta_i] + \sum_{j=1}^{D_h} \mathbb{E}[\Omega_{ij} h_j] \\ &= \mathbb{E}[\beta_i] + \sum_{j=1}^{D_h} \mathbb{E}[\Omega_{ij}]\,\mathbb{E}[h_j] \\ &= 0 + \sum_{j=1}^{D_h} 0 \cdot \mathbb{E}[h_j] = 0, \end{aligned}$$

$$\begin{aligned} \sigma_f^2 &= \mathbb{E}[f_i^2] - \mathbb{E}[f_i]^2 \\ &= \mathbb{E}\left[\left(\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right] - 0 \\ &= \mathbb{E}\left[\left(\sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right] \\ &= \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij}^2\right] \mathbb{E}\left[h_j^2\right] \\ &= \sum_{j=1}^{D_h} (\sigma_\Omega^2 \sigma_h^2) = D_h \sigma_\Omega^2 \sigma_h^2, \end{aligned} \tag{7.16}$$

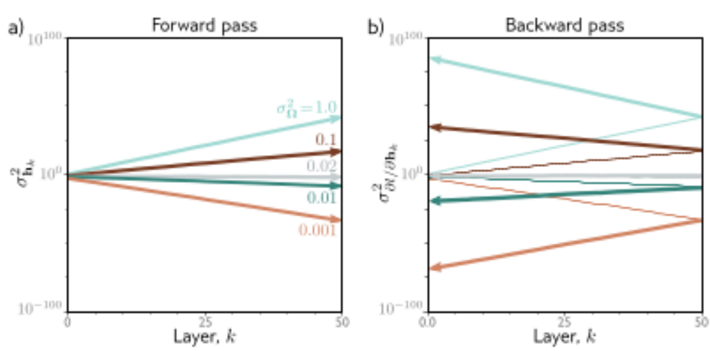



**Figure 7.4** Weight initialization. Consider a deep network with 50 hidden layers and $D_h = 100$ hidden units per layer. The network has a 100 dimensional input $\mathbf{x}$ initialized with values from a standard normal distribution, a single output fixed at $y = 0$, and a least squares loss function. The bias vectors $\boldsymbol{\beta}_k$ are initialized to zero and the weight matrices $\boldsymbol{\Omega}_k$ are initialized with a normal distribution with mean zero and five different variances $\sigma^2_\Omega \in \{0.001, 0.01, 0.02, 0.1, 1.0\}$. a) Variance of hidden unit activations computed in forward pass as a function of the network layer. For He initialization ($\sigma^2_\Omega = 2/D_h = 0.02$), the variance is stable. However, for larger values it increases rapidly, and for smaller values, it decreases rapidly. b) The variance of the gradients in the backward pass (solid lines) continues this trend; if we initialize with a value larger than 0.02, the magnitude of the gradients increases rapidly as we pass back through the network. If we initialize with a value smaller, then the magnitude decreases. These are known as the *exploding gradient* and *vanishing gradient* problems, respectively.

1. He initialization or Kaiming initialization (He et al. 2015)

2. Glorot or Xavier Initialization (Glorot and Bengio. 2010)

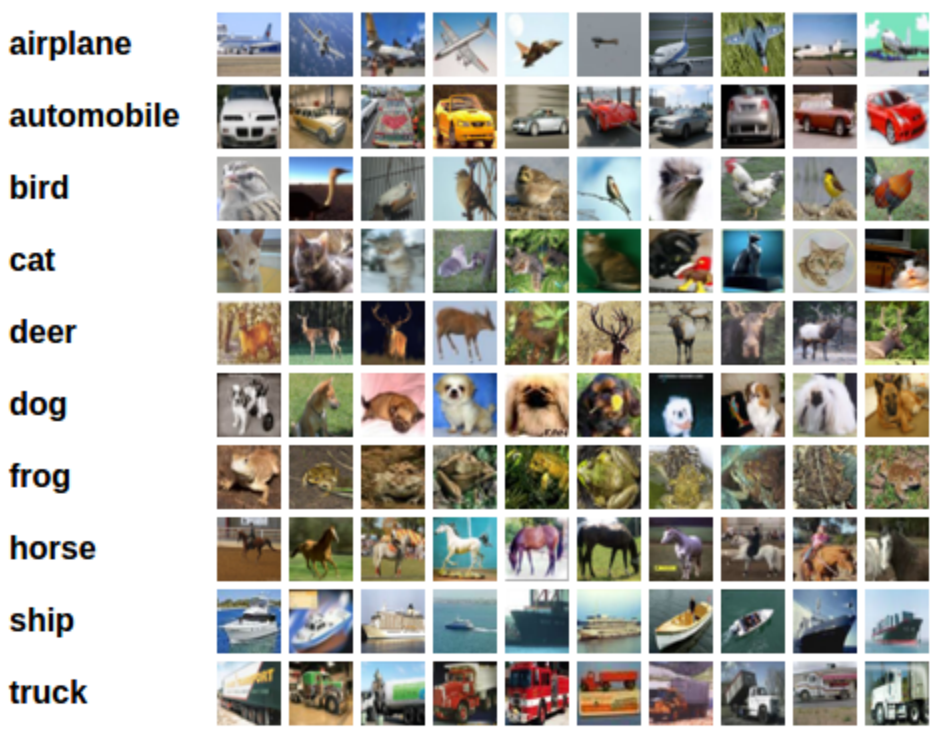
```
# Adapted from: Chapter 7 and 8 of Deep Learning with Pytorch by Eli Stevens (2020)
# References
# 1. 2010-glorot.pdf from milestone papers
# 2. 2015-HeInitialization.pdf from milestone papers
# 3. 2015-BatchNorm.pdf from milestone papers
# 4. Section 11.4 of UDLBook
# 5. Chapter 7 of UDLBook
try:
    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, |
```

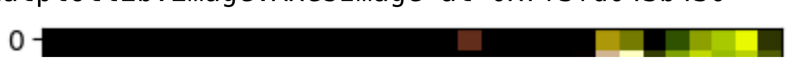


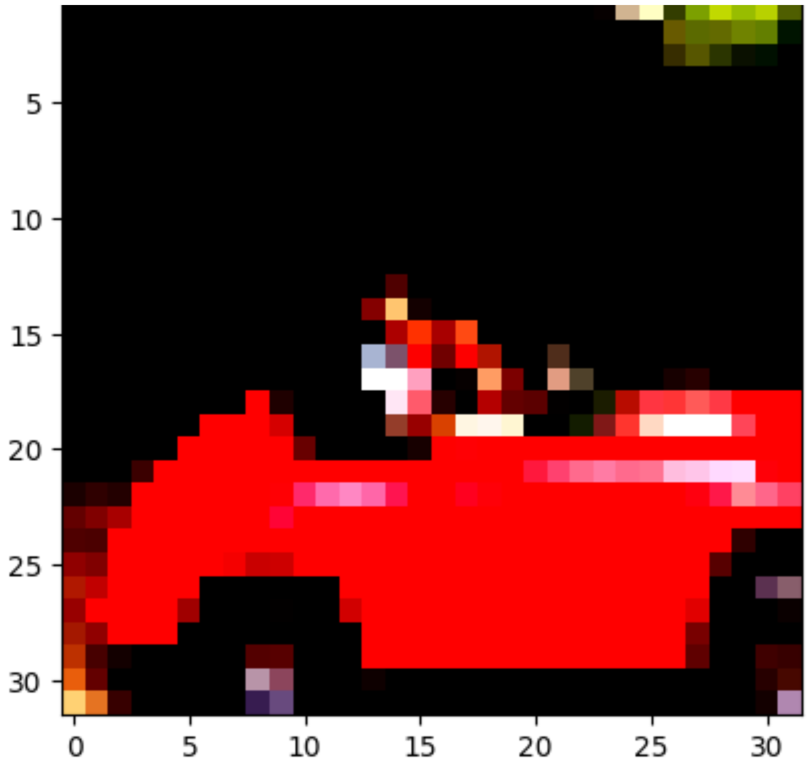

```
img, label = all_training_data[99]
img.shape, label
```

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

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

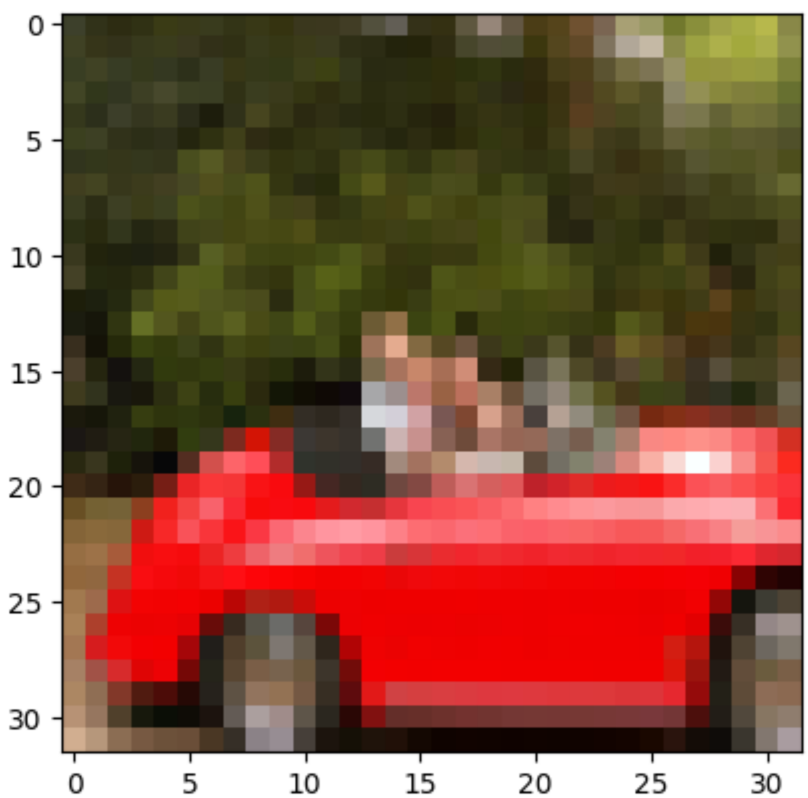
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wi
<matplotlib.image.AxesImage at 0x7f3fd043b430>
```






```
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"), encoc
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/python3.9
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/python
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyt
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/pyth
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/python
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
```

**TensorBoard** INACTIVE

**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)
            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: {\
        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, -0.01
         -0.0608, -0.0433,  0.0770, -0.0703, -0.0210, -0.0316, -0.0518,  0.04
          0.0136, -0.0489, -0.0238, -0.0347,  0.0809,  0.0455,  0.0984, -0.04
         -0.0562, -0.0729,  0.0985,  0.0218, -0.0347, -0.0804,  0.0060,  0.01
          0.0298, -0.0306,  0.0793,  0.0897,  0.0392, -0.0096,  0.0931,  0.01
         -0.0718, -0.0351, -0.0133,  0.0873, -0.0747, -0.0172, -0.0958,  0.00
         -0.0508, -0.0934,  0.0348, -0.0389,  0.0372, -0.0371,  0.0141, -0.07
         -0.0675,  0.0806, -0.0965, -0.0980,  0.0127,  0.0440, -0.0584,  0.09
          0.0964, -0.0403,  0.0963,  0.0796, -0.0636, -0.0133,  0.0358, -0.01
          0.0373, -0.0487,  0.0901,  0.0995,  0.0008,  0.0702,  0.0146,  0.08
          0.0094,  0.0963,  0.0146,  0.0245,  0.0065, -0.0438, -0.0614,  0.07
          0.0128, -0.0173, -0.0965, -0.0417, -0.0960, -0.0260,  0.0025, -0.08
          0.0284,  0.0480, -0.0144, -0.0521],
        [-0.0851,  0.0805,  0.0900, -0.0173, -0.0005, -0.0925,  0.0612, -0.04
         -0.0946,  0.0524,  0.0226, -0.0501,  0.0109,  0.0450,  0.0653,  0.09
          0.0857, -0.0151, -0.0560,  0.0294, -0.0166,  0.0335,  0.0782,  0.00
          0.0454, -0.0105, -0.0878,  0.0290, -0.0168, -0.0111,  0.0344, -0.02
          0.0367,  0.0931,  0.0323,  0.0160,  0.0651, -0.0514,  0.0038, -0.01
          0.0393,  0.0193,  0.0465, -0.0680, -0.0848,  0.0457, -0.0351,  0.06
          0.0859, -0.0309, -0.0798,  0.0473,  0.0099, -0.0528,  0.0280,  0.06
         -0.0579,  0.0152, -0.0115, -0.0596, -0.0682, -0.0161, -0.0451,  0.09
          0.0425,  0.0418,  0.0512, -0.0760, -0.0100, -0.0437,  0.0616, -0.08
         -0.0113, -0.0864,  0.0253, -0.0600, -0.0555, -0.0045, -0.0403, -0.09
         -0.0446, -0.0178, -0.0258, -0.0181,  0.0706, -0.0967, -0.0053, -0.02
          0.0818,  0.0196,  0.0578, -0.0638, -0.0309, -0.0526, -0.0533,  0.00
          0.0894, -0.0606,  0.0309,  0.0490],
        [ 0.0030,  0.0755,  0.0253,  0.0273, -0.0347, -0.0118,  0.0463, -0.00
          0.0704, -0.0103, -0.0588, -0.0779,  0.0074, -0.0607,  0.0695,  0.07
          0.0119, -0.0048, -0.0443, -0.0292, -0.0643, -0.0809,  0.0356,  0.04
          0.0235,  0.0922,  0.0020, -0.0316, -0.0575, -0.0695,  0.0368,  0.01
         -0.0237,  0.0079, -0.0558, -0.0597,  0.0246, -0.0686,  0.0042, -0.09
          0.0218,  0.0899,  0.0337, -0.0536,  0.0333,  0.0166,  0.0354, -0.00
          0.0558, -0.0413,  0.0592, -0.0987,  0.0463, -0.0291, -0.0359, -0.07
         -0.0924,  0.0776,  0.0422, -0.0312, -0.0908,  0.0573,  0.0588,  0.09
         -0.0928, -0.0131, -0.0091,  0.0777, -0.0899,  0.0634,  0.0807,  0.01
         -0.0523,  0.0134,  0.0306,  0.0451,  0.0528,  0.0679, -0.0788, -0.00
         -0.0083,  0.0255, -0.0343, -0.0378,  0.0787, -0.0797,  0.0828, -0.02
         -0.0939, -0.0582, -0.0016,  0.0378,  0.0073, -0.0678, -0.0449,  0.03
```