EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

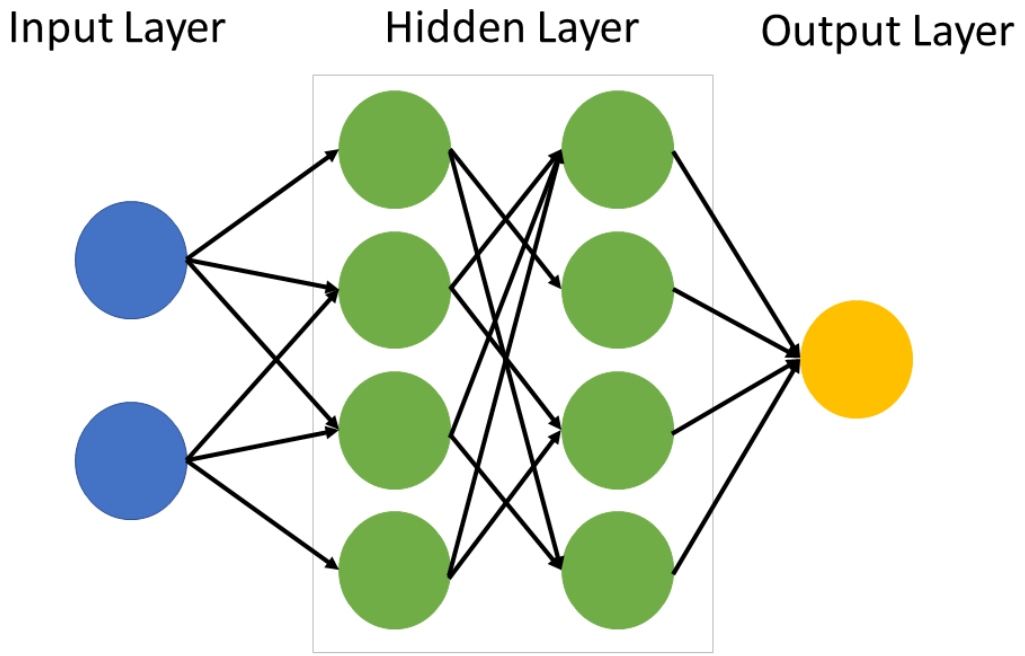
Image Analysis II

Radovan Fusek

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Deep learning - basic principles

1. Guess some solution
2. Compute the error of this solution
3. **Modify the parameters based on this error – We need to find the minimum of the error "landscape" - Gradient Descent**



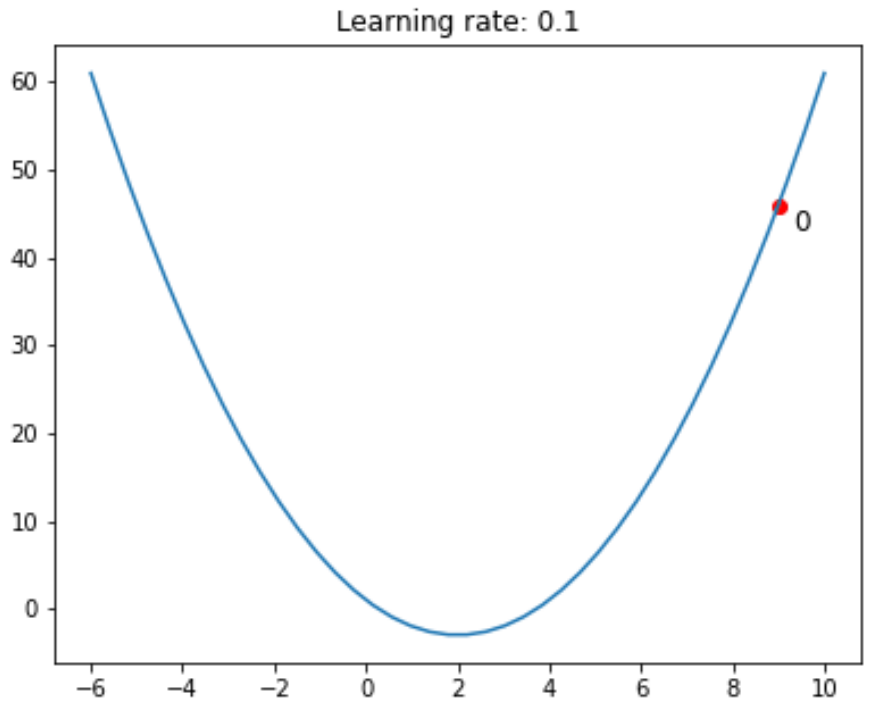
Artificial Neural Networks

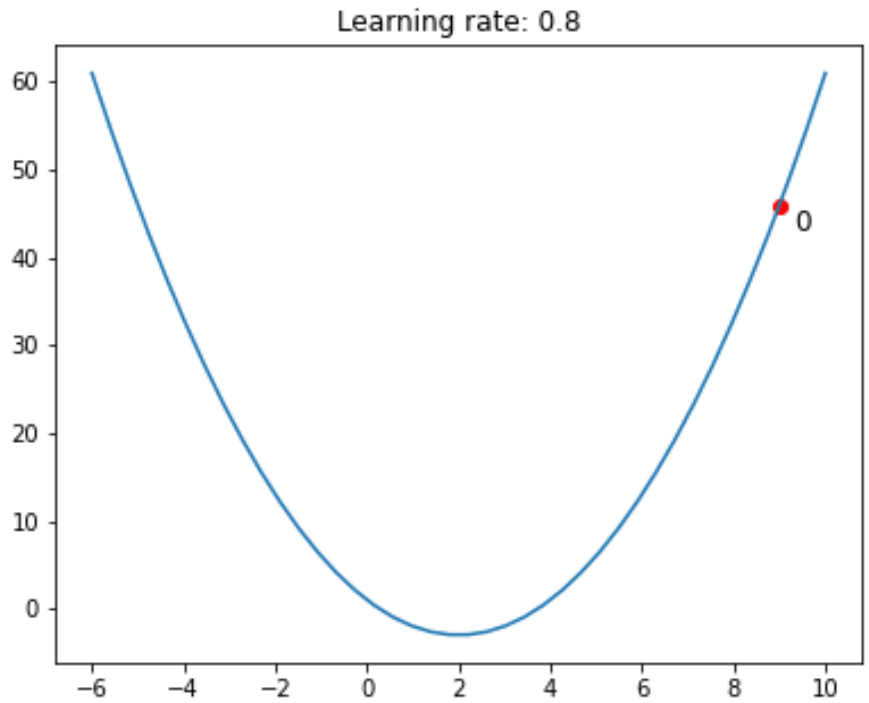
https://en.m.wikipedia.org/wiki/Machine_learning_in_earth_sciences

https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

Gradient Descent

1. Random initialization of starting point
2. Based on gradient (derivation) decided the step way
3. Learning rate – scaling factor for step sizes
4. Repeat the process (number of iterations/epochs)





https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Gradient Descent

Python Example

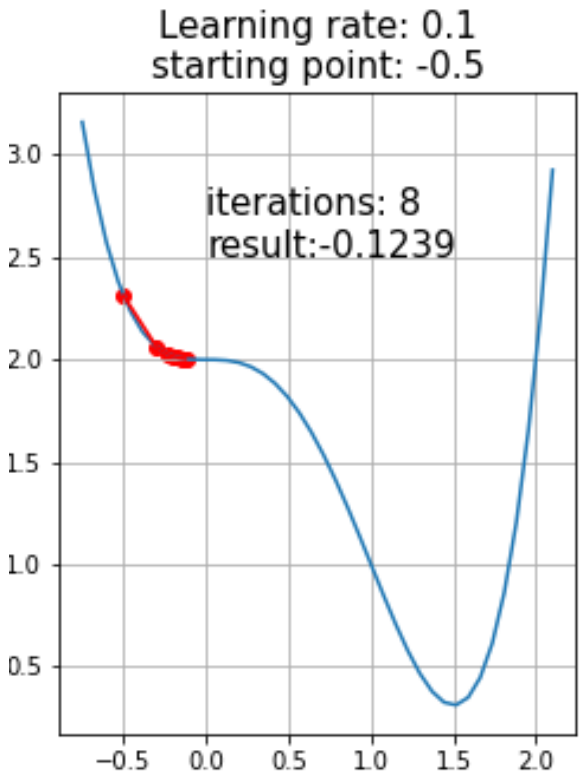
```
import numpy as np
import matplotlib.pyplot as plt

def fx(x):
    return x**2 - 6*x + 1

def deriv(x):
    return ???

x = np.linspace(-14, 20, 2000)

localmin = 18 #np.random.choice(x,1)
grad = deriv(localmin)
print("localmin", localmin)
print("grad", grad)
plt.plot(x, fx(x))
plt.plot(localmin, fx(localmin), 'ro')
plt.show()
```

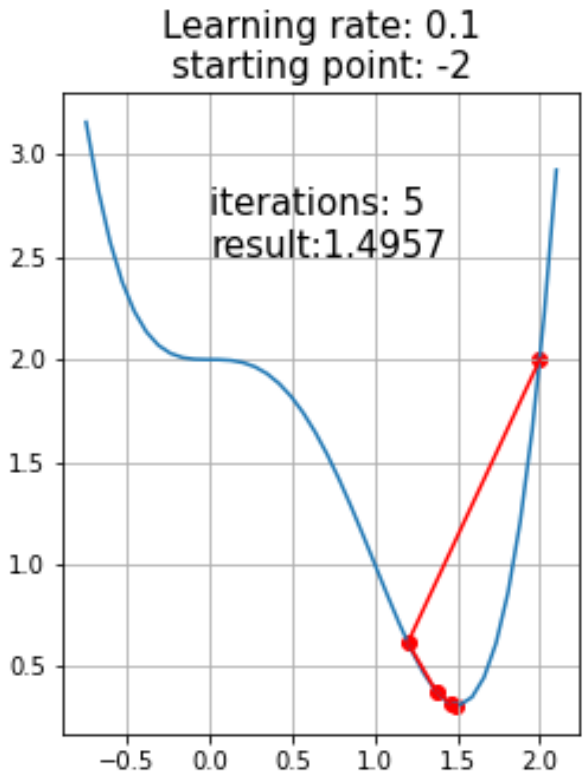
```
learning_rate = 0.05
training_epochs = 1000

for i in range(training_epochs):
    grad = deriv(localmin)
    move = learning_rate*grad
    localmin = localmin - move
print(localmin)
```

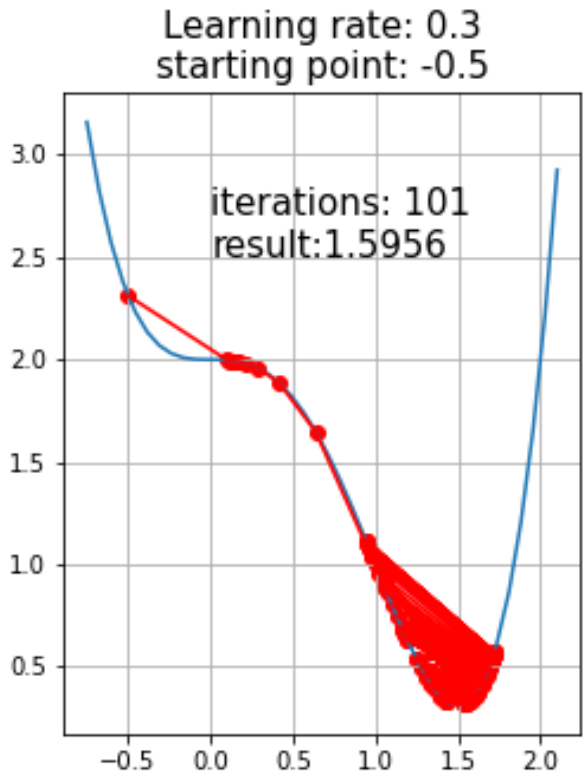
https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

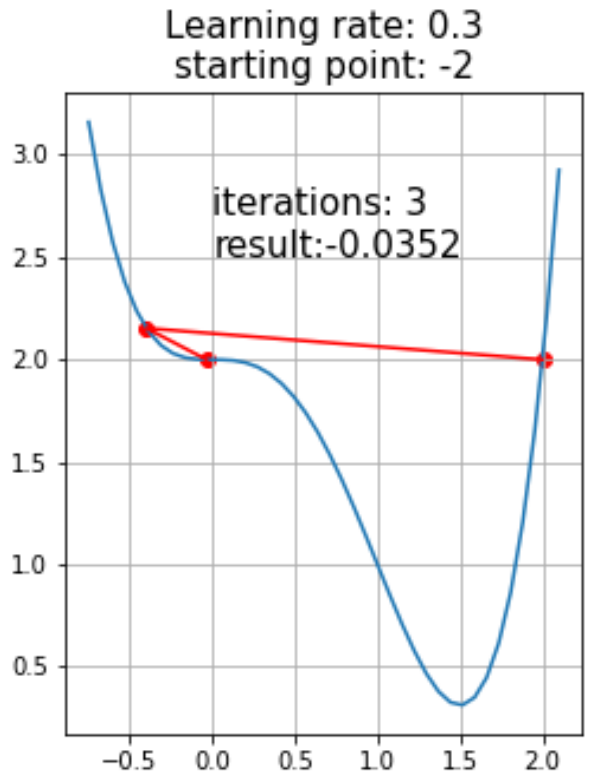
Gradient Descent

- **Problems?**



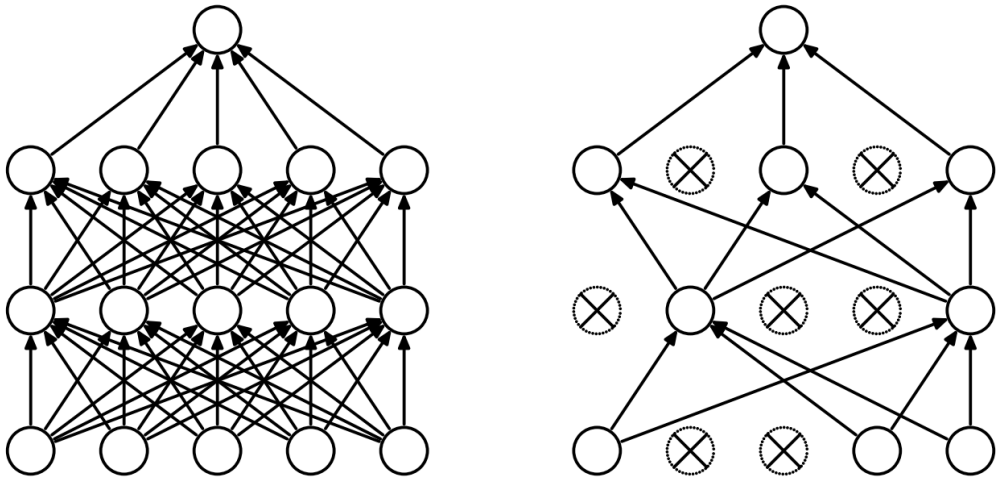






https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

Dropout



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

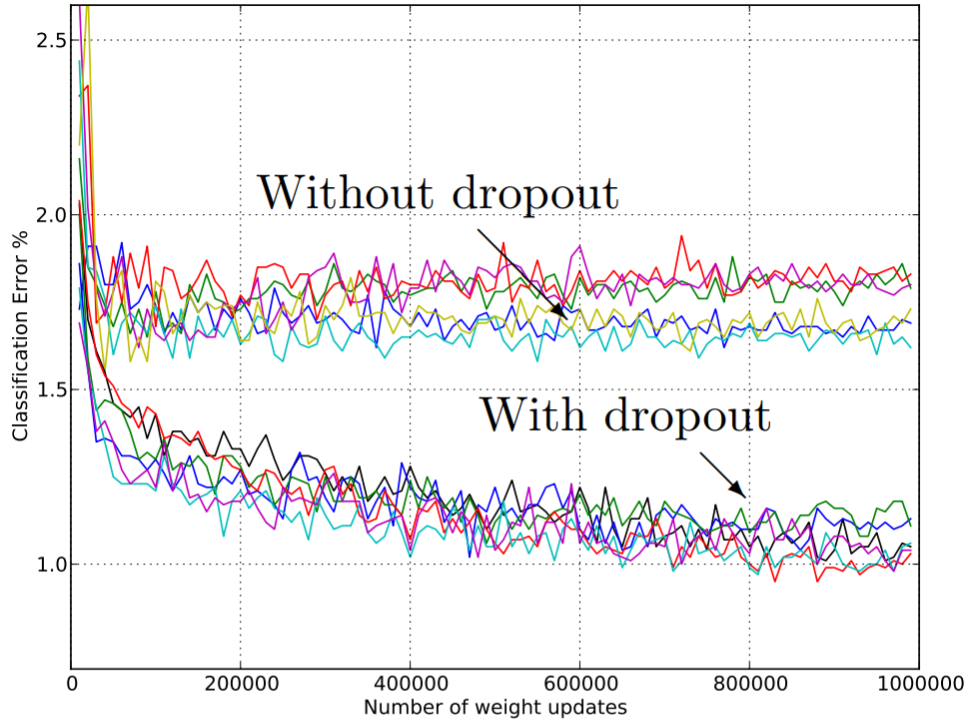


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

https://arxiv.org/pdf/1207.0580.pdf

Dropout

DROPOUT

CLASS torch.nn.Dropout(*p=0.5*, *inplace=False*) [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters:

- **p** (*float*) – probability of an element to be zeroed. Default: 0.5
- **inplace** (*bool*) – If set to `True`, will do this operation in-place. Default: `False`

Shape:

- Input: $(*)$. Input can be of any shape
- Output: $(*)$. Output is of the same shape as input

Examples:

```
>>> m = nn.Dropout(p=0.2)
>>> input = torch.randn(20, 16)
>>> output = m(input)
```

"Remember that you must call **model.eval()** to set dropout and batch normalization layers to evaluation mode before running inference. Failing to do this will yield inconsistent inference results. If you wish to resuming training, call **model.train()** to ensure these layers are in training mode."

https://pytorch.org/tutorials/beginner/saving_loading_models.html#saving-loading-model-for-inference

https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

Data Normalization

Why we need to normalize data?

W1	Sample-1	Sample-2	Sample-3
0.5	0.1	10.5	0.5
0.2	0.3	20.2	200.2
0.1	0.2	15.1	15.1
-0.5	-0.4	20.5	0.5
0.1	0.2	21.1	21.1

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Data Normalization

Why we need to normalize data?

- **Changes during back propagation can be bigger for a particular sample**

W1	Sample-1	Sample-2	Sample-3
0.5	0.1	10.5	0.5
0.2	0.3	20.2	200.2
0.1	0.2	15.1	15.1
-0.5	-0.4	20.5	0.5
0.1	0.2	21.1	21.1

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Data Normalization

Why we need to normalize data?

- **Changes during back propagation can be bigger for a particular sample**
- **Even the value inside one samples can be in different range**

W1	Sample-1	Sample-2	Sample-3
0.5	0.1	10.5	0.5
0.2	0.3	20.2	200.2
0.1	0.2	15.1	15.1
-0.5	-0.4	20.5	0.5
0.1	0.2	21.1	21.1

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Data Normalization

Why we need to normalize data?

- **Changes during back propagation can be bigger for a particular sample**
- **Even the value inside one samples can be in different range**
- **e.g. Z-transform, min-max scaling (0-1), batch normalization**

W1	Sample-1	Sample-2	Sample-3
0.5	0.1	10.5	0.5
0.2	0.3	20.2	200.2
0.1	0.2	15.1	15.1
-0.5	-0.4	20.5	0.5
0.1	0.2	21.1	21.1

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Data Normalization

"Remember that you must call **model.eval()** to set dropout and batch normalization layers to evaluation mode before running inference. Failing to do this will yield inconsistent inference results. If you wish to resuming training, call **model.train()** to ensure these layers are in training mode."

https://pytorch.org/tutorials/beginner/saving_loading_models.html#saving-loading-model-for-inference

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Model Information

```python
def main(argv):

    n_params = 0
    tensor = torch.randn(1, 3, 28, 28)
    net = NetSimple()
    net(tensor)
    #requires_grad=True we can freeze the layer
    for p in net.named_parameters():
        print(p)

    for p in net.parameters():
        if p.requires_grad:
            print(p.numel())
            n_params = n_params+p.numel()
    print("n_params", n_params)

    from torchsummary import summary
    print(summary(net, (3, 28, 28)))
```

```python
class NetSimple(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.fc1 = nn.LazyLinear(256)
        self.fc2 = nn.LazyLinear(256)
        self.fc3 = nn.LazyLinear(10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all
        print("x shape", x.shape)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Dataset

```
class ParkingData(Dataset):
    def __init__(self, train_imgs, train_labels):
        ???
    def __getitem__(self,index):
        ???
        return tensor, label
    def __len__(self):
        return self.len
```

```
parking_data = ParkingData(train_images, train_labels, transform)
train_dataloader = DataLoader(parking_data, batch_size=8, shuffle=True)
```

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Dataset

```
train_full_imgs = [transform(cv2.imread(name, 1)) for name in glob.glob("train_images/full/*.png")]
train_full_labels = [1 for i in train_full_imgs]
```

```
transform = torchvision.transforms.Compose([
                        torchvision.transforms.ToPILImage(),
                        torchvision.transforms.Resize(size=(40, 40)),
                                                ???
                        torchvision.transforms.ToTensor(),
                        torchvision.transforms.Normalize((0.5,), (0.5,))
                      ])
```

https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

ColorJitter

The `ColorJitter` transform randomly changes the brightness, saturation, and other properties of an image.

```
jitter = T.ColorJitter(brightness=.5, hue=.3)
jitted_imgs = [jitter(orig_img) for _ in range(4)]
plot(jitted_imgs)
```

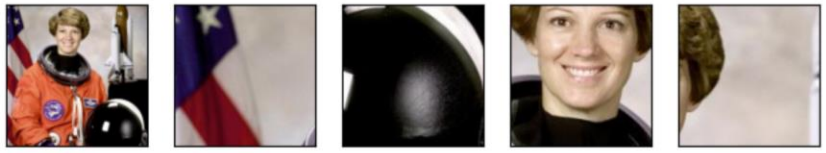
Original image

RandomCrop

The `RandomCrop` transform (see also `crop()`) crops an image at a random location.

```
cropper = T.RandomCrop(size=(128, 128))
crops = [cropper(orig_img) for _ in range(4)]
plot(crops)
```

Original image



Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Data Normalization

Task - Experiment with:

- learning rate
- number of epochs
- normalization
- dropout
- loss function
- data augmentation
- …

Dropout: A Simple Way to Prevent Neural Networks from Overfitting