MACHINE LEARNING

MEI/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2024/2025

Machine Learning

[07a]

Syllabus

- Example CNN Classification

Argument Parsing

```
ap = argparse.ArgumentParser()
ap.add_argument('-d', '--dataset', required=True, help='CSV learning dataset file')
ap.add_argument('-o', '--output_folder', required=True, help='Output folder')
ap.add_argument('-b', '--batch_size', type=int, default=100, help='Learning batch size')
ap.add_argument('-iw', '--image_width', type=int, default=512, help='Image width')
ap.add_argument('-ih', '--image_height', type=int, default=128, help='Image height')
ap.add_argument('-l', '--learning_rate', type=float, default=1e-3, help='Learning rate')
ap.add_argument('-de', '--decay_rate', type=float, default=1e-2, help='Decay rate')
ap.add_argument('-dr', '--dropout_rate', type=float, default=0.25, help='Dropout rate')
ap.add_argument('-e', '--epochs', type=int, default=1000000, help='Tot. epochs')
ap.add_argument('-pl', '--probability_learn', type=float, default=0.7, help='Probability Learning set')
ap.add_argument('-pv', '--probability_validation', type=float, default=0.15, help='Probability Validation set')
args = ap.parse_args()
```

The script is then executed by: “python3 script.py –d ‘data.csv’,...

Large Dataset Loading

```
def read_csv(dataset):
    # ############################
    # Load Data in '.csv' format: [ [filename_1, label_1], [filename_2, label_2],...]
    samp = []
    with open(dataset) as f:
        csv_file = csv.reader(f, delimiter=',')
        for row in csv_file:
            samp.append(row)

    random.shuffle(samp)
    return samp
```

The “csv” file should be in the format:
/path/image_1.jpg 1
/path/image_2.jpg 0
/path/image_3.jpg 2

Dataset Splitting

```
def split_dataset(dt):
    dt_l = []
    dt_v = []
    dt_t = []
    onehot_encoder = OneHotEncoder(sparse=False)
    onehot_encoder.fit(np.asarray([x[-1] for x in dt]).reshape(-1, 1))
    out = onehot_encoder.transform(np.asarray([x[-1] for x in dt]).reshape(-1, 1))
    dt = list(zip(dt, out))

    for el in dt:
        x = random.random()
        if x < args.probability_learn:
            dt_l.append([el[0][0], el[1]])
        elif x < args.probability_learn + args.probability_validation:
            dt_v.append([el[0][0], el[1]])
        else:
            dt_t.append([el[0][0], el[1]])

    return dt_l, dt_v, dt_t
```

Divides the available data into three sub-sets: learning + validation + test

Dataset Splitting

```
def get_input_batch(gt, idx, augm, tot_c):
    tot = min(args.batch_size, len(gt) - idx)

    imgs = np.zeros((tot, args.image_height, args.image_width, 1)).astype('float')
    labels = np.zeros((tot, tot_c)).astype('float')

    for i in range(tot):
        img = cv2.imread(gt[idx + i][0])
        if augm is not None:
            img = augm.augment_image(img)
        img = cv2.resize(img, (args.image_width, args.image_height))

        imgs[i, :, :, 0] = img[:, :, 0] / 255
        labels[i, :] = gt[idx + i][1]

    return imgs, labels
```

Load one batch of (maximum) ”batch_size” images and the corresponding ground truth

Create CNN

```
def create_cnn(tot_c):
    imgs_input = Input((args.image_height, args.image_width, 3))

    conv12 = Conv2D(64, kernel_size=3, strides=2, padding="same")(imgs_input)
    conv12_bn = BatchNormalization(momentum=0.8)(conv12)
    conv12_a = LeakyReLU()(conv12_bn)
    drop12 = Dropout(args.dropout_rate)(conv12_a)

    conv13 = Conv2D(128, kernel_size=3, strides=2, padding="same")(drop12)
    conv13_bn = BatchNormalization(momentum=0.8)(conv13)
    conv13_a = LeakyReLU()(conv13_bn)
    drop13 = Dropout(args.dropout_rate)(conv13_a)

    conv14 = Conv2D(256, kernel_size=3, strides=2, padding="same")(drop13)
    conv14_bn = BatchNormalization(momentum=0.8)(conv14)
    conv14_a = LeakyReLU()(conv14_bn)
    # drop14 = conv14_a
    drop14 = Dropout(args.dropout_rate)(conv14_a)

    conv15 = Conv2D(512 , kernel_size=3, strides=2, padding="same")(drop14)
    conv15_bn = BatchNormalization(momentum=0.8)(conv15)
    conv15_a = LeakyReLU()(conv15_bn)
    drop15 = Dropout(args.dropout_rate)(conv15_a)

    conv16 = Conv2D(512, kernel_size=3, strides=2, padding="same")(drop15)
    conv16_bn = BatchNormalization(momentum=0.8)(conv16)
    conv16_a = LeakyReLU()(conv16_bn)
    drop16 = Dropout(args.dropout_rate)(conv16_a)
    pooled = Flatten()(drop16)

    dense1 = Dense(128, activation='relu', kernel_constraint=None)(pooled)
    drop1 = Dropout(args.dropout_rate)(dense1)

    dense2 = Dense(64, activation='relu', kernel_constraint=None)(drop1)
    drop2 = Dropout(args.dropout_rate)(dense2)

    outp = Dense(tot_c, activation='sigmoid', kernel_constraint=None)(drop2)
    out = Softmax()(outp)

    md = Model(inputs=imgs_input, outputs=out)
    md.compile(optimizer=SGD(lr=args.learning_rate, momentum=0.8),
loss=tf.keras.losses.CategoricalCrossentropy())
    md.summary()
    return md
```

Creates a CNN of 27 layers

Train()

```
i = 0
while i < len(l_s):
        [imgs, gt] = get_input_batch(l_s, i, augmenter, tot_c)
        lo = md.train_on_batch(imgs, gt)
        lo_l.append(lo)
        i += args.batch_size
        print('\r Learn [%d - %d/%d]...' % (epoch, i, len(l_s)), end='')
```

One training epoch

```
i = 0
while i < len(v_s):
        [imgs, gt] = get_input_batch(v_s, i, None, tot_c)
        lo = md.test_on_batch(imgs, gt)
        lo_v.append(lo)
        i += args.batch_size
        print('\r Valid [%d - %d/%d]...' % (epoch, i, len(v_s)), end='')
```

One validation epoch

Train()

```
ep = range(1, epoch + 1)
fig_1 = plt.figure(1, figsize=(18, 8))
plt.clf()
gs = gridspec.GridSpec(2, 2, figure=fig_1)

ax = fig_1.add_subplot(gs[0, 0])
ax.plot(ep, losses_learn, '-g')
ax.plot(ep, losses_valid, '-r')
ax.grid(True)
ax.title.set_text('Losses’)

...
```

Plot intermediate results