OxML 2024
Practical Session: Optimization + DNN

Ziyan Wang

King's College London

Link and Slide

Link for Part I Optimization: Link
Link for Part II DNN: Link

Check the slide in the Slack.

Remark: Gradient Descent

What is the Gradient Descent (GD)?

Gradient Descent is a commonly used optimization algorithm used to find the minimum value of a function.

How do we use GD in ML?

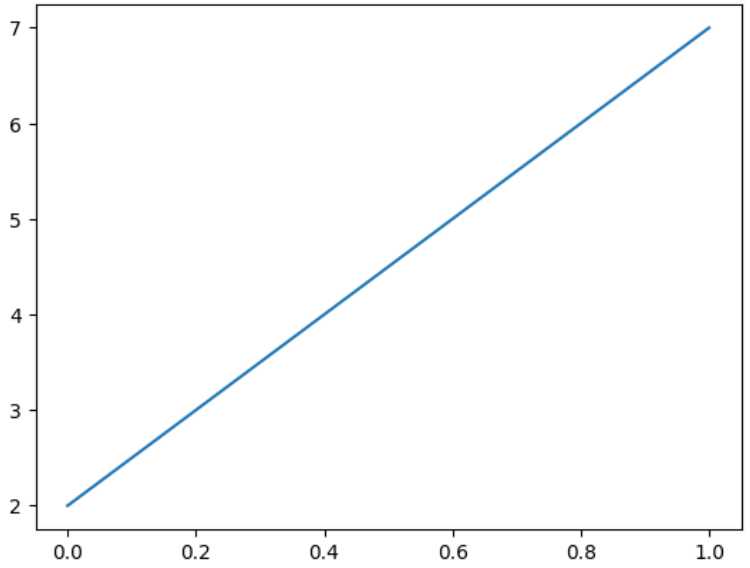
In machine learning, we often use GD to minimize the loss function to find the optimal model parameters.

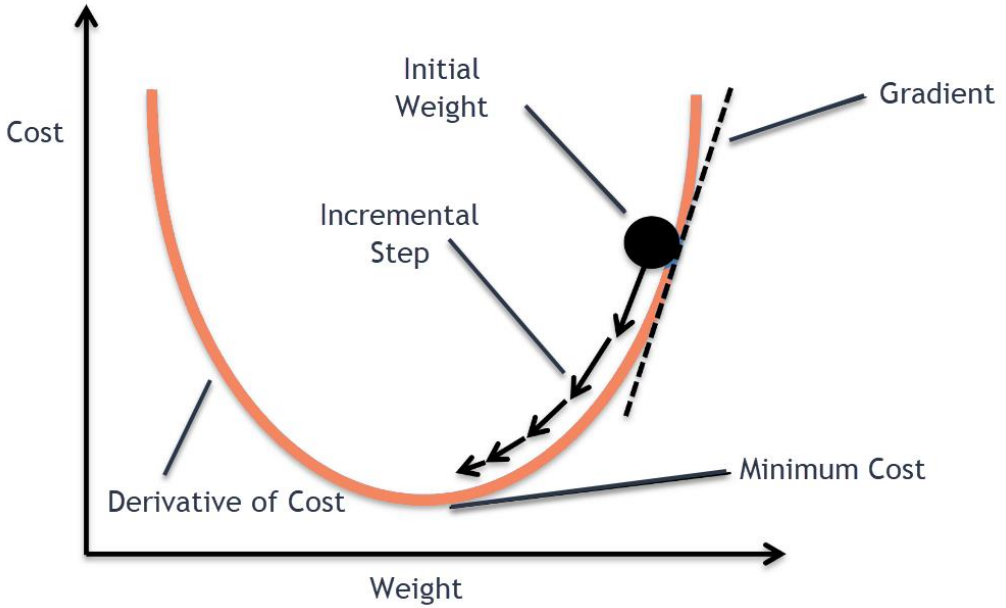
The algorithm moves along the negative gradient direction of the loss function by iteratively adjusting the parameters until it reaches the minimum value.

Start with a Line.

Problem: Given a line, $y = kx + b$, for illustration purposes, how to use gradient descent to find those two coefficients k and b?

MSE loss $(\hat{y} - y)^2$. Its derivative is just $2(\hat{y} - y)$.



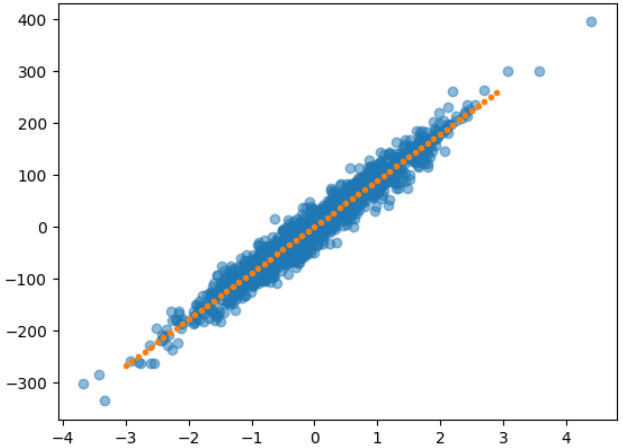


Data Points, Linear Regression.

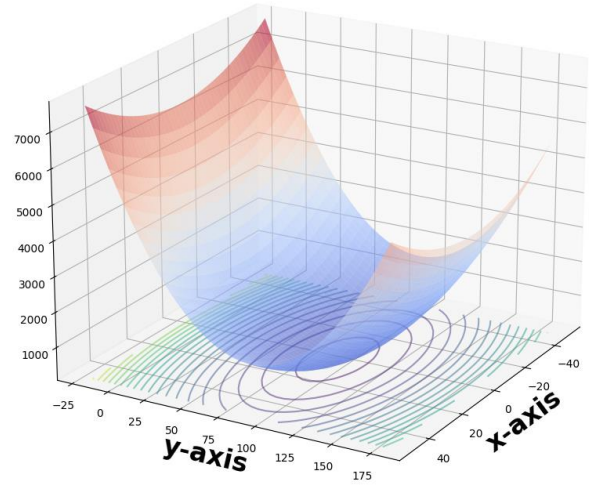
Problem: Given data points $\mathbf{x}^1, \ldots, \mathbf{x}^n \in \mathbb{R}^d$ and $y^1, \ldots, y^n \in \mathbb{R}$, find the best fit line. Consider a simple linear regression model with $\hat{y} = \mathbf{w}^T\mathbf{x} + b$.The formulation can be simplified to $\hat{y} = \boldsymbol{\theta}^T\tilde{\mathbf{x}}$, in which $\boldsymbol{\theta} = [\mathbf{w}^T, b]^T, \tilde{\mathbf{x}} = [\mathbf{x}^T, 1]^T$.

Objective (MSE):

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{2n}\sum_{i=1}^{n}\left(\boldsymbol{\theta}^T\mathbf{x}_i' - y_i\right)^2$$



Data Distribution



Loss Surface

Remark: Different Types of GD.

Gradient Descent (GD):

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \gamma \cdot \frac{1}{n} \sum_{i=1}^{n} \tilde{\mathbf{x}}_i^T \left(\boldsymbol{\theta}_t^T \tilde{\mathbf{x}}_i - y_i \right)$$
$$t = 0,1,2, \ldots$$

Stochastic Gradient Descent (SGD):

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \gamma \tilde{\mathbf{x}}_i^T \left(\boldsymbol{\theta}_t^T \tilde{\mathbf{x}}_i - y_i \right)$$
$$i \sim \text{uniform } (0, n)$$

Mini-batch Stochastic Gradient Descent (Mini-SGD):

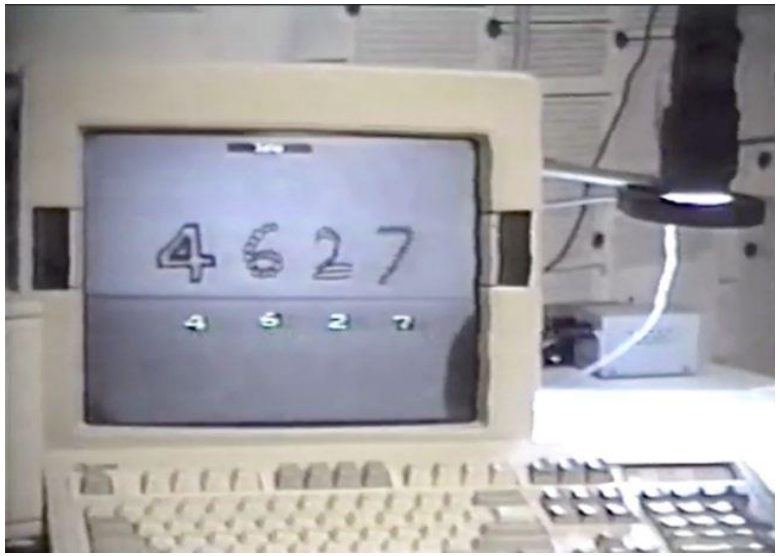
$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \gamma \cdot \frac{1}{m} \sum_{i=1}^{m} \tilde{\mathbf{x}}_i^T \left(\boldsymbol{\theta}_t^T \tilde{\mathbf{x}}_i - y_i \right)$$
$$\{\tilde{\mathbf{x}}_i, y_i\}_{i=1}^{m} \sim \text{ uniform } (\mathcal{D})$$

Remark: Adam

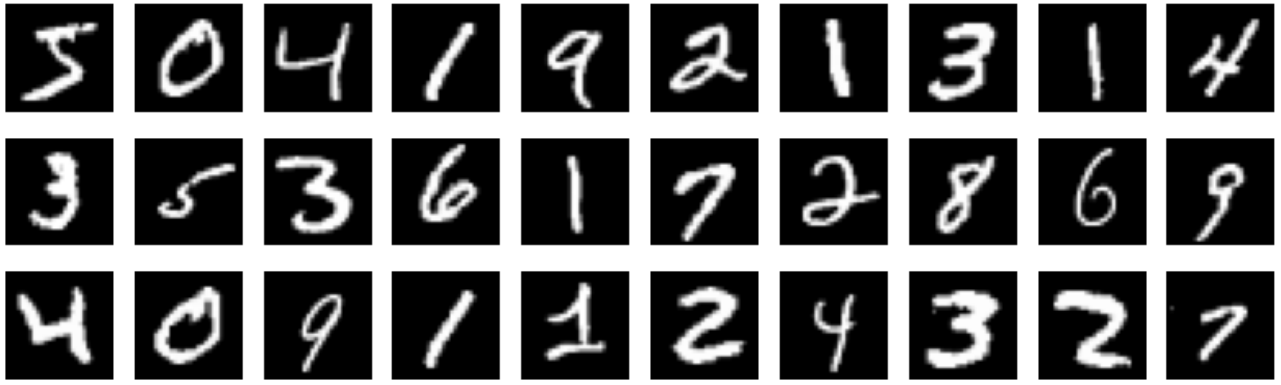
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

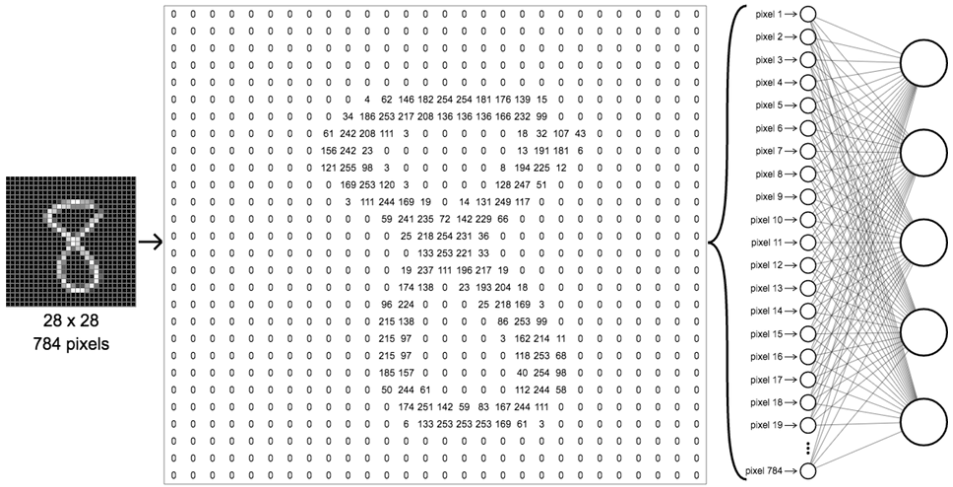
```
elif method == 'Adam':
    idx = random.sample(range(0, m), batch_size)
    batch_gradient = x[idx].T.dot(error[idx])/ batch_size

    gred_m = beta1 * gred_m + (1-beta1) * batch_gradient
    gred_n = beta2 * gred_n + (1-beta2) * batch_gradient**2

    gred_m_adaptive = gred_m/(1-beta1**counter)
    gred_n_adaptive = gred_n/(1-beta2**counter)

    gradient = gred_m_adaptive/ (np.sqrt(gred_n_adaptive) + epsilon)
```

MNIST. Train a Classifier I



Problem: a real-world example of classification using neural networks, the task of recognizing and labeling images of handwritten digits. We are going to use the MNIST dataset, which contains **60,000** labeled images of handwritten digits sized 28x28 pixels, whose classification accuracy serves as a common benchmark in machine learning research.

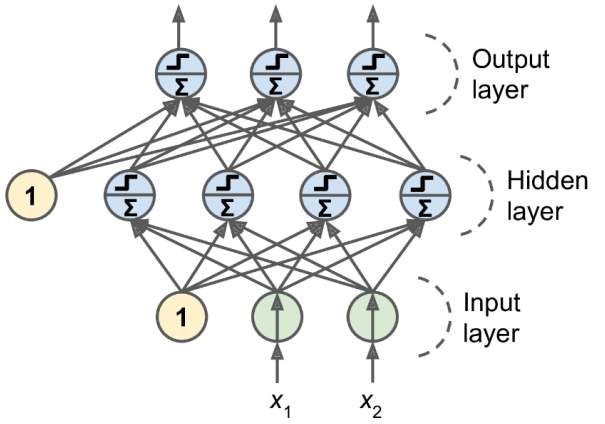




Remark: Deep Neural Network / MLP

What is the Deep Neural Network (DNN)?

A DNN is an artificial neural network with multiple layers between the input and output layers. It consists of an input layer, several hidden layers, and an output layer. Each layer is composed of interconnected nodes (neurons) with activation functions.



When do we use DNN?

Solving complex, non-linear problems, Handling large amounts of high-dimensional data

Components in DNN?

Input layer/Hidden layers/Output layer/Neurons/ Activation Functions / Weights and Biases / Loss Function /Optimizer

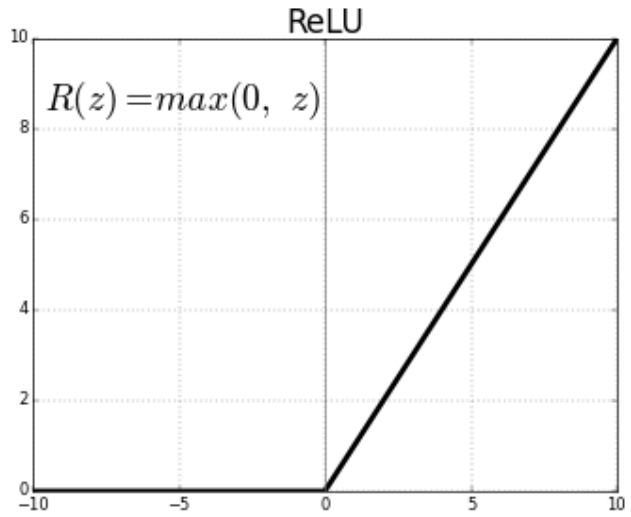
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28 * 28, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Remark: Activation Function

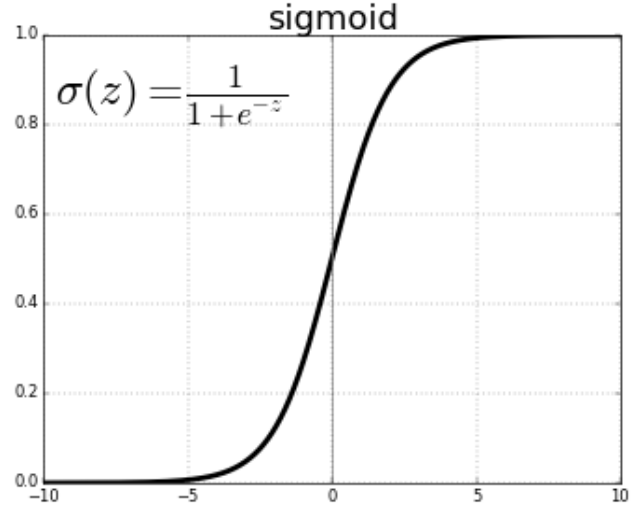
The activation function takes the same weighted sum input from before, $z = b + \sum_i w_i x_i$, and then transforms it once more before finally outputting it.

Advantages: Introducing Non-linearity / Increasing Model's Representational Power / Introducing Sparsity and Feature Selection / Providing Gradients for Backpropagation.

```
x = F.relu(self.fc1(x))
x = F.relu(self.fc2(x))
```




```
x = F.sigmoid(self.fc1(x))
x = F.sigmoid(self.fc2(x))
```



Remark: Cross Entropy Loss and Softmax Activation

For a single sample:

$$L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

For a batch of samples:

$$L = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{C} y_{n,i}\log(\hat{y}_{n,i})$$

The cross-entropy loss encourages the model to:

Assign high probabilities to the correct classes

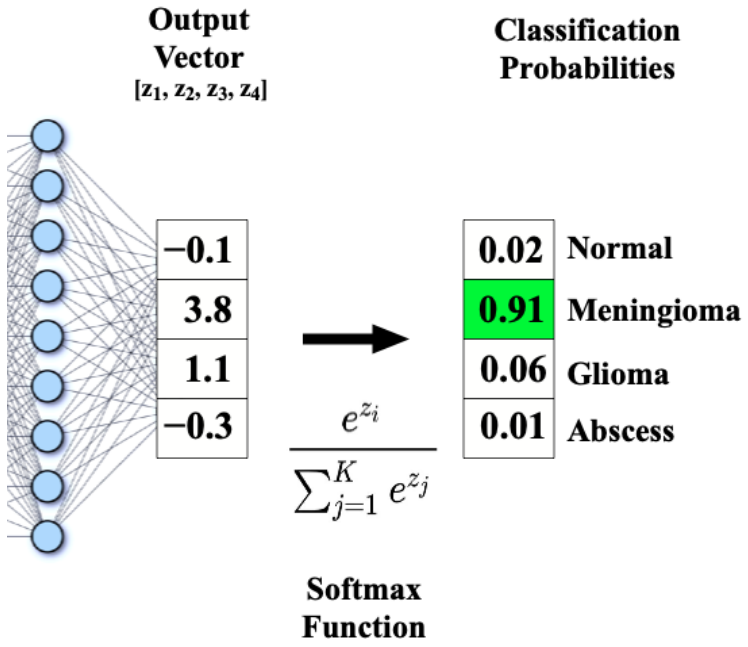
Assign low probabilities to the incorrect classes

The loss increases as the predicted probability diverges from the true label

```
criterion = nn.CrossEntropyLoss()
loss = criterion(outputs, labels)
```

For the Softmax activation:

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$



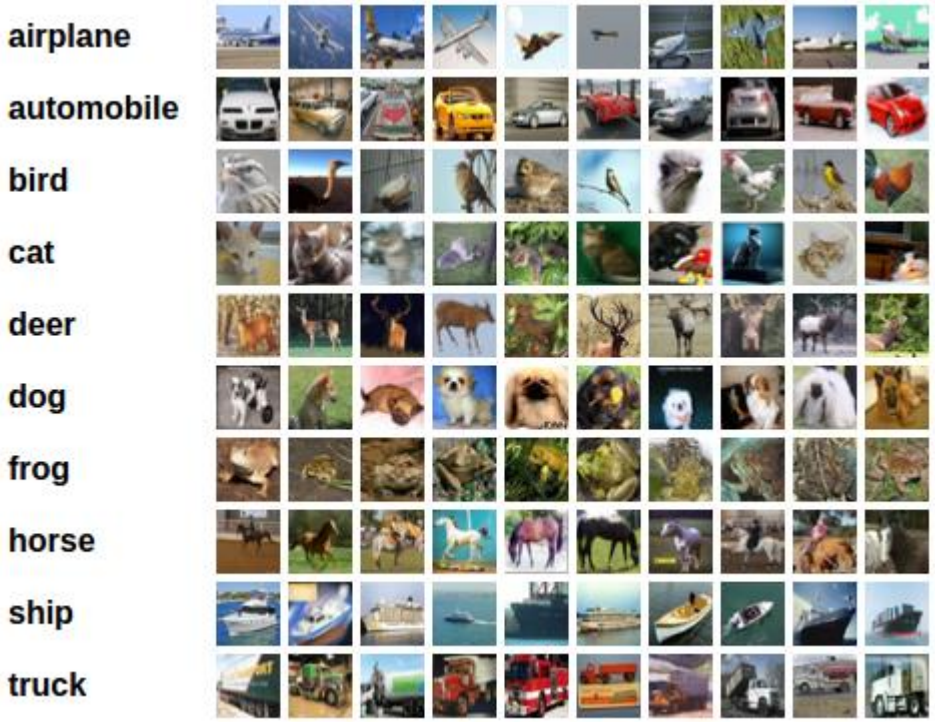
CIFAR10, Train a Classifier II

Problem: The CIFAR10 dataset, which has the classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'.

The images in CIFAR-10 are of size 3x32x32, i.e.

3-channel color images of 32x32 pixels in size.

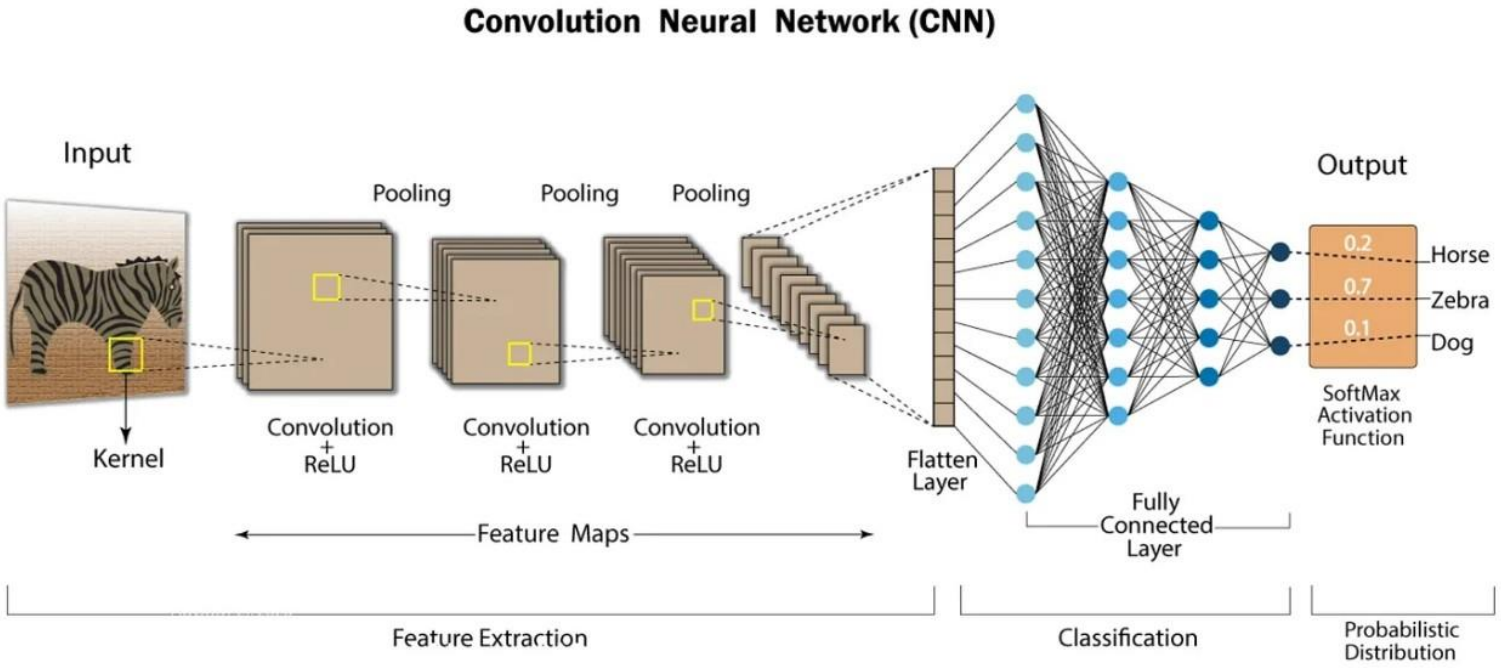
Training an image classifier to identify the type of the input image.



Remark: Convolutional Neural Network

What is the Convolutional Neural Network (CNN)?

CNN are a type of deep learning neural network architecture that is particularly well suited to image classification and object recognition tasks. A CNN works by transforming an input image into a feature map, which is then processed through multiple convolutional and pooling layers to produce a predicted output.




```python
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 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 dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

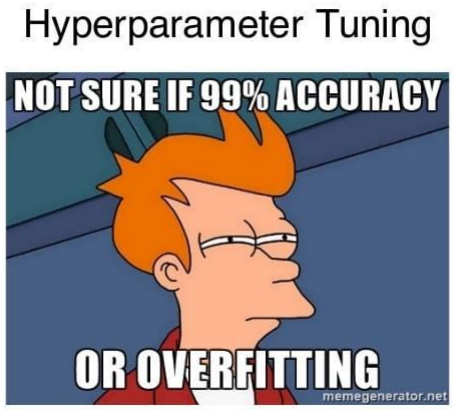
How to design your network?

Determining the optimal parameter values for each layer in a CNN/MLP is a complex problem.

Several factors need to be considered, such as the **characteristics of the task**, **the size** and **complexity of the data set**, and **the limitations of computing resources**.

Maybe you can…

1. Refer to the existing classic network structure (LeNet /ResNet)
2. Experiment and tweak, or using Grid Search or Random Search tech
3. Consider the characteristics of the task
4. Consider computing resource constraints



Thank you!