AMATH 840: Advanced Numerical Methods for Computational and Data Science

Giang Tran
Department of Applied Mathematics, University of Waterloo

Winter 2024

Part 2: Neural Networks

2.1: Feed-Forward Neural Networks - Function Representations

Winter 2024

Outline

Fully Connected Neural Networks

Convolutional Neural Networks

ResNets

Feed-Forward Neural Networks as Function Approximations

- A feed-forward neural network = a composition of linear and nonlinear (activation) functions, alternatively:

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{W}_L\sigma\Big(\mathbf{W}_{L-1}\sigma\big(\dots\sigma(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)\dots\big)+\mathbf{b}_{L-1}\Big)+\mathbf{b}_L, \quad (1)$$

  where

  - L: number of layers; $L-1$: number of hidden layers
  - $\mathbf{x} \in \mathbb{R}^{n^{(0)}}$: input data
  - $\sigma(\cdot)$: nonlinear activation function (applied point-wise)
  - $\boldsymbol{\theta} = \{(\mathbf{W}_\ell, \mathbf{b}_\ell)\}_{\ell=1}^{L}$ are trainable weights and biases, with $\mathbf{W}_\ell \in \mathbb{R}^{n^{(\ell)} \times n^{(\ell-1)}}, \mathbf{b}_\ell \in \mathbb{R}^{n^{(\ell)}}, n^{(\ell)} \in \mathbb{Z}^+$, and $\ell \in [L]$.

- A shallow network, also called a one hidden layer neural network, $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ can be written as:

$$f(\mathbf{x};\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}_1,\mathbf{b}_2) = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)+\mathbf{b}_2.$$

With or Without Bias

- Definition: A fully connected operator is given by

$$\mathrm{fc} : \mathbb{R}^n \to \mathbb{R}^m, \quad \mathrm{fc}(\mathbf{x}) := \mathbf{W}\mathbf{x} + \mathbf{b},$$

  where $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ are trainable weight and bias, respectively.

- Remark:

$$\mathbf{W}\mathbf{x} + \mathbf{b} = \begin{bmatrix} \mathbf{W} & \mathbf{b} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix} = \widehat{\mathbf{W}}\,\widehat{\mathbf{x}}.$$

Nonlinear (Activation) Functions

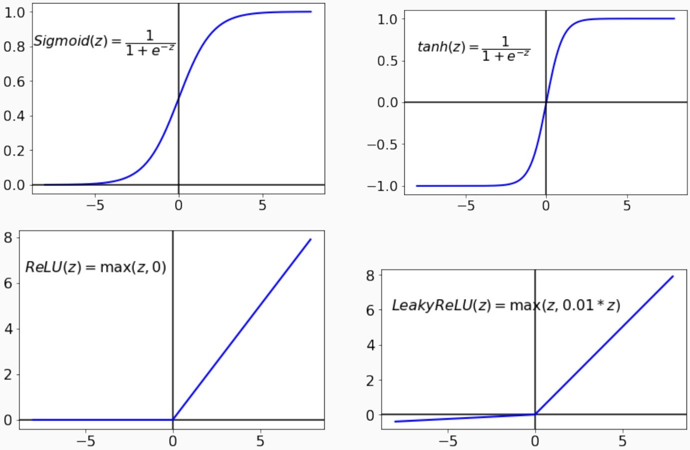


Figure 1: Examples of some popular nonlinear activation functions

1

[1] "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function", by Leshno et al, Neural Networks, 1993.

Fully Connected Neural Networks

- If the weight matrices are dense, the neural networks are called fully connected neural networks.
- An example of a fully connected NN with two hidden layers and σ is a nonlinear activation function:

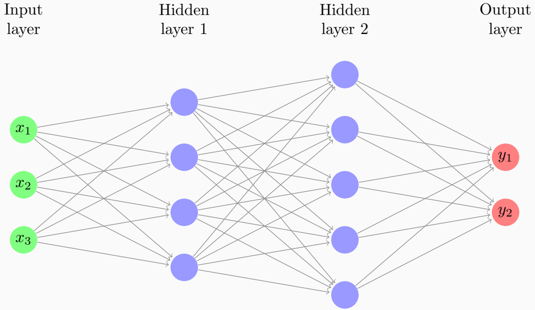


Figure 2: Input is $\mathbf{x} \in \mathbb{R}^3$. Hidden layers are $\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \in \mathbb{R}^4$ and $\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \in \mathbb{R}^5$. Output is $\mathbf{y} = \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 \in \mathbb{R}^2$. Number of parameters is $(4 \times 3 + 4) + (5 \times 4 + 5) + (2 \times 5 + 2)$.

Outline

Convolutional Neural Networks

- A convolutional neural network = a feed-forward neural network where the linear functions are either fully-connected or convolution operators.

- A 2D convolution operator with stride s and padding p is a linear operator:

$$\text{conv} : \mathbb{R}^{n^2} \to \mathbb{R}^{\left(\frac{n+2p-k}{s}+1\right)^2}, \quad \text{conv}(\mathbf{z}) := \mathbf{z} * \mathbf{K} = \mathbf{M}\mathbf{z},$$

  where the kernel $\mathbf{K} \in \mathbb{R}^{k \times k}$ ($k \ll n$) and $\mathbf{M} \in \mathbb{R}^{\left(\frac{n+2p-k}{s}+1\right)^2 \times n^2}$ is a structured sparse matrix associated with the kernel $\mathbf{K}$ (see the explanation in the next two slides).

Example of a 2D Convolution Operator

$$\begin{array}{|c|c|c|}\hline a & b & c \\ \hline d & e & f \\ \hline u & v & w \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 3 & -1 & 4 \\ \hline 1 & 5 & 9 \\ \hline 2 & 6 & -5 \\ \hline \end{array} := 3a - b + 4c + d + 5e + 9f + 2u + 6v - 5w.$$

Example of a 2D Convolution Operator (cont'd)

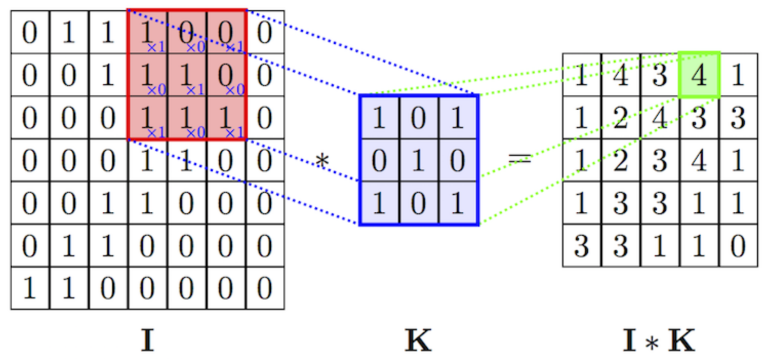


Figure 3: Here $n = 7$, $k = 3$, $s = 1$, $p = 0$, conv : $\mathbb{R}^{7^2} \to \mathbb{R}^{\left(\frac{n+2P-K}{S}+1\right)^2} = \mathbb{R}^{5^2}$.

2

[2] http://perso.mines-paristech.fr/fabien.moutarde/ES_MachineLearning/TP_convNets/convnet-notebook.html

Convolution Neural Networks (cont'd)

- Group local neurons as inputs to next layer $\rightarrow$ Reduces numbers of weights to learn
- General framework

$$\mathbf{x} \rightarrow (\text{conv} \rightarrow \text{ReLU})^N \rightarrow (\text{fc} \rightarrow \text{ReLU})^K \rightarrow \text{fc} \rightarrow f(\mathbf{x})$$

$$f(\mathbf{x}) = \text{fc} \circ (\text{ReLU} \circ \text{fc})^K \circ (\text{ReLU} \circ \text{conv})^N(\mathbf{x})$$

- Some popular CNNs: LeNet, AlexNet, VGG16, GoogLeNet, ResNets, FractalNet, ResNext, ...
- To count # parameters and to debug, always check the dimensions of inputs and outputs

Implement a Fully Connected Neural Network in Pytorch

The following codes are taken from

`https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html`

Implement a Fully Connected Neural Network in Pytorch

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
net = Net()
print(net)
```

Outline

Fully Connected Neural Networks

Convolutional Neural Networks

ResNets

ResNets

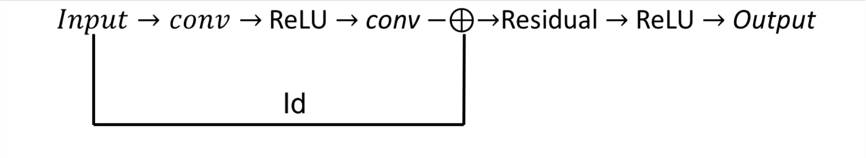
- State-of-the-art performance for image classification, object detection, and semantic segmentation.
- Idea: Introduce a skip connection (identity function) to learn residuals $\rightarrow$ More stable, enable to have very deep networks

[4]

[4] *Deep Residual Learning for Image Recognition* by K. He et al, Proceedings of the IEEE conference on computer vision and pattern recognitio, 2016.

A Basic Block of ResNets

- Example of a simplified (no batch normalization) basic block for ResNet with 2 convolutions of size 3×3 and no biases:



$$\mathbf{x} \xmapsto{\text{conv}\rightarrow\text{ReLU}} \mathbf{h}_1 = \text{ReLU}(\mathbf{W}_0 * \mathbf{x}) \xmapsto{\text{conv}} \mathbf{z}_1 = \mathbf{W}_1 * \mathbf{h}_1 = \mathbf{W}_1(\text{ReLU}(\mathbf{W}_0 * x))$$

$$\mathbf{z}_1 \xmapsto{+\,\text{id}(x)} \mathbf{r}_1 = \mathbf{W}_{proj}\mathbf{x} + \mathbf{z}_1 \xmapsto{\text{ReLU}} \mathbf{y} = \text{ReLU}(\mathbf{r}_1)$$

A Basic Block of ResNets (cont'd)

```
def conv3x3(in_planes: int, out_planes: int, stride: int = 1, groups: int = 1,
dilation: int = 1) -> nn.Conv2d:
    """3x3 convolution with padding"""
    return nn.Conv2d(
        in_planes,
        out_planes,
        kernel_size=3,
        stride=stride,
        padding=dilation,
        groups=groups,
        bias=False,
        dilation=dilation,
    )
```

[5]

[5] https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

A Basic Block of ResNets (cont'd)

```
class BasicBlock(nn.Module):
    expansion: int = 1

    def __init__(
        self,...
    ) -> None:
        super().__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        if groups != 1 or base_width != 64:
            raise ValueError("BasicBlock only supports groups=1 and base_width=64")
        if dilation > 1:
            raise NotImplementedError("Dilation > 1 not supported in BasicBlock")
        # Both self.conv1 and self.downsample layers downsample the input when stride != 1
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = norm_layer(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = norm_layer(planes)
        self.downsample = downsample
        self.stride = stride
```

6

[6] https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

A Basic Block of ResNets (cont'd)

```
def forward(self, x: Tensor) -> Tensor:
    identity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)

    if self.downsample is not None:
        identity = self.downsample(x)

    out += identity
    out = self.relu(out)

    return out
```

[7]

[7] https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

Dimension Calculation of a Basic Block of ResNets

- For example, feedforward x of size $56 \times 56 \times 3$ through a ResNet basic block with two convolutions of size 3×3, $P = 1$, $S = 2$ and $S = 1$, respectively.

$$\mathbf{x} \xrightarrow[\text{ReLU}]{\text{conv}, S=2} \mathbf{h}_1 = \text{ReLU}(\mathbf{W}_0 * \mathbf{x}) \xrightarrow{\text{conv}, S=1} \mathbf{z}_1 = \mathbf{W}_1 * \mathbf{h}_1 \xrightarrow{+\,\text{id}(x)} \mathbf{r}_1 = \text{id}(\mathbf{x}) + \mathbf{z}_1 \xrightarrow{\text{ReLU}} \mathbf{y}$$

- Size of $\mathbf{h}_1$ is

$$\left(\frac{56+2-3}{2}+1\right) \times \left(\frac{56+2-3}{2}+1\right) \times 1 = 28 \times 28 \times 1$$

- Size of $\mathbf{z}_1$ is

$$\left(\frac{28+2-3}{1}+1\right) \times \left(\frac{28+2-3}{1}+1\right) \times 1 = 28 \times 28 \times 1$$

Dimension Calculation of a Basic Block of ResNets (cont'd)

$$\mathbf{x} \xrightarrow[\text{ReLU}]{\text{conv}, S=2} \mathbf{h}_1 = \text{ReLU}(\mathbf{W}_0 * \mathbf{x}) \xrightarrow{\text{conv}, S=1} \mathbf{z}_1 = \mathbf{W}_1 * \mathbf{h}_1 \xrightarrow{+\text{id}(\mathbf{x})} \mathbf{r}_1 = \text{id}(\mathbf{x}) + \mathbf{z}_1 \xrightarrow{\text{ReLU}} \mathbf{y}$$

- Note: $\text{id}(\mathbf{x})$ is of size $56 \times 56 \times 1$ and z_1 is of size $28 \times 28 \times 1$.
- Size of $\mathbf{r}_1 = \mathbf{W}_{proj}(\mathbf{x}) + \mathbf{z}_1$ is $28 \times 28 \times 1$, where $\mathbf{W}_{proj}$ is a projection mapping or a convolution with $k = 1, s = 2, p = 0$.

Example of a 18-layers ResNet - ResNet18

- Example of a 18-layers ResNet

$$Input \xrightarrow[k=7]{s=2,p=3} \mathrm{conv}_{7\times7}\, 64 \xrightarrow[k=3,s=2,p=1]{maxpool} (Block(\mathrm{conv}\, 64)) \rightarrow (Block(\mathrm{conv}\, 64)) \rightarrow$$
$$\rightarrow Block(\mathrm{conv}\, 128, s=2) \rightarrow Block(\mathrm{conv}\, 128) \rightarrow Block(\mathrm{conv}\, 256, s=2) \rightarrow$$
$$\rightarrow Block(\mathrm{conv}\, 256) \rightarrow Block(\mathrm{conv}\, 512, s=2) \rightarrow Block(\mathrm{conv}\, 512) \xrightarrow{average}$$
$$\xrightarrow{pool} \mathrm{fc}\, 1000 \rightarrow softmax$$

- All convolutions are 3×3, unless specified otherwise.
- 64, 128, 256, 512 are numbers of filters.
- Periodically, double number of filters and downsample spatially using stride 2

[8]

[8] "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385

Example of a 18-layers ResNet - ResNet18(cont'd)

```
def resnet18(*, weights: Optional[ResNet18_Weights] = None,
progress: bool = True, **kwargs: Any) -> ResNet:
    """ResNet-18 from ‘Deep Residual Learning for Image Recognition
    <https://arxiv.org/abs/1512.03385>‘__.
        ...
    """
    weights = ResNet18_Weights.verify(weights)

    return _resnet(BasicBlock, [2, 2, 2, 2], weights, progress, **kwargs)
```

Remark: See also class ResNet(nn.Module) from the file resnet.py.

9

[9] https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

ResNets Architectures

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4: ResNet Architectures

Exercise: Calculate the number of trainable parameters of ResNet-18, given the size of each input data is 224x224 (gray images).

[10]

[10] *Deep Residual Learning for Image Recognition* by K. He et al, Proceedings of the IEEE conference on computer vision and pattern recognitio, 2016.