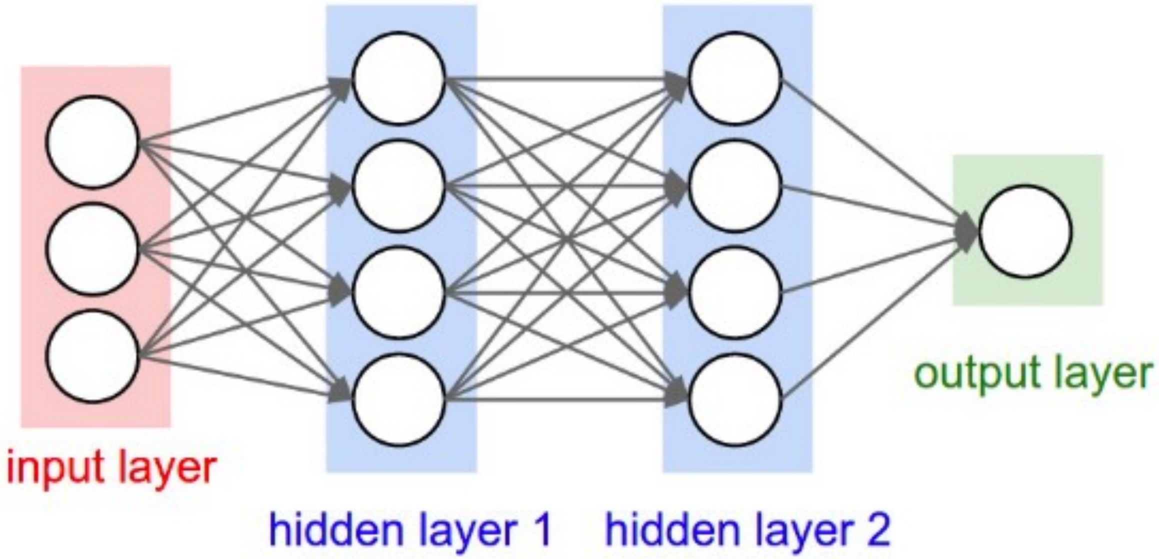
Fully-connected Neural Network

Hongchang Gao

Spring 2024

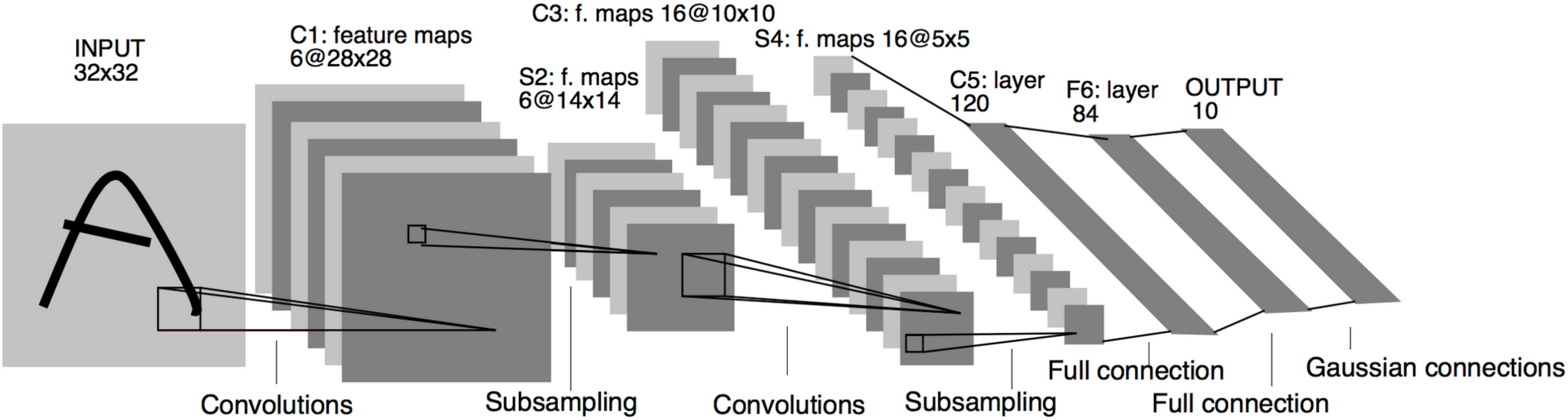
Overview

- Fully connected neural network
  - Multi-layer perceptron (MLP)



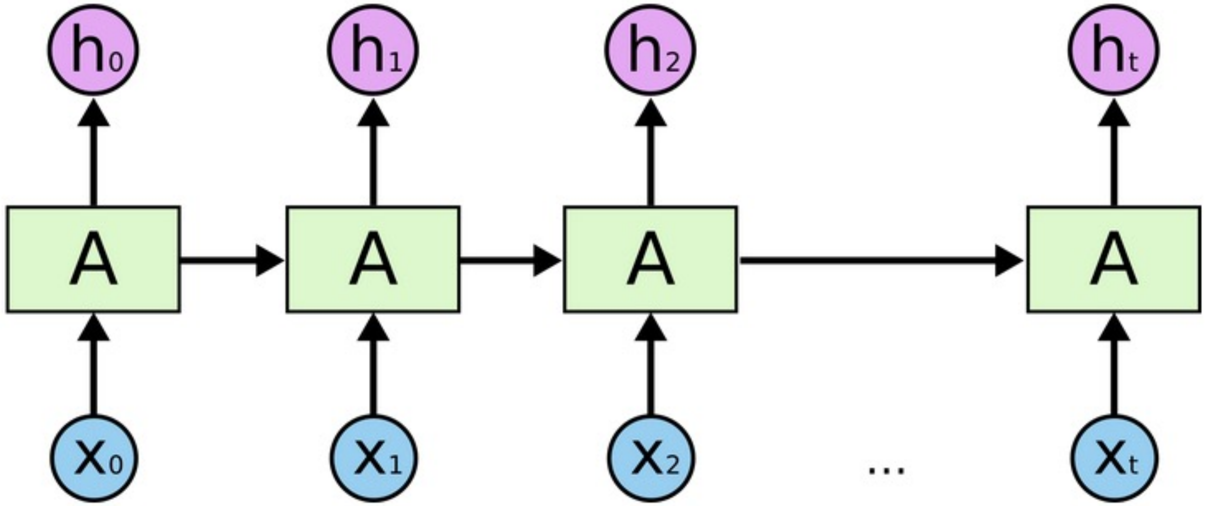
Overview

- Convolutional Neural Network



Overview

- Recurrent Neural network

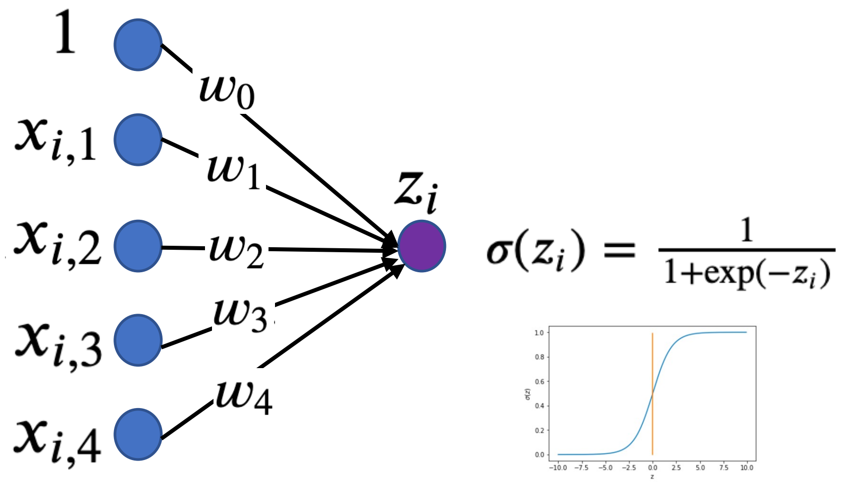


Logistic Regression

- Logistic regression
  - Linear model
  - Single layer

Given n samples: $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$

$$\begin{cases} z_i = \mathbf{w}^T \mathbf{x}_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + w_3 x_{i,3} + w_4 x_{i,4} \\ \sigma(z_i) = \frac{1}{1+\exp(-z_i)} \end{cases}$$



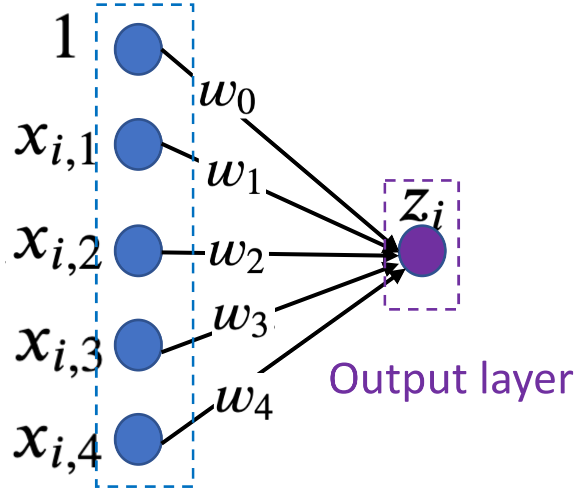
From Single Layer to Multiple Layers

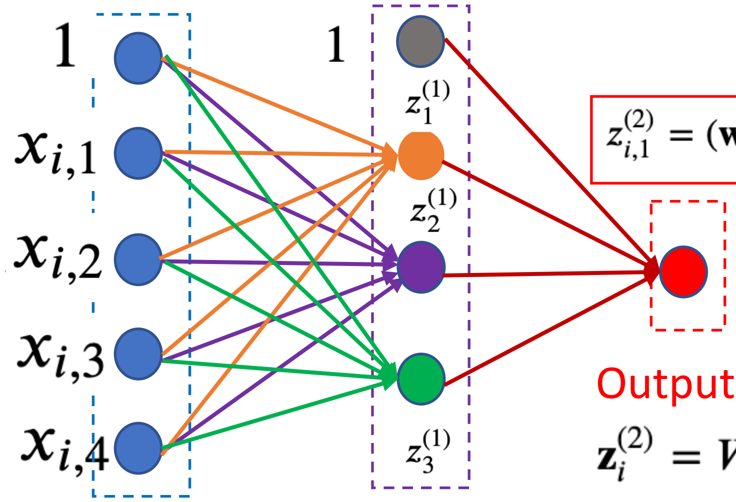
- Stack multiple layers together

Neuron-1
$$z_{i,1}^{(1)} = (\mathbf{w}_1^{(0)})^T \mathbf{x}_i = w_{1,0}^{(0)} + w_{1,1}^{(0)} x_{i,1} + w_{1,2}^{(0)} x_{i,2} + w_{1,3}^{(0)} x_{i,3} + w_{1,4}^{(0)} x_{i,4}$$

Neuron-2
$$z_{i,2}^{(1)} = (\mathbf{w}_2^{(0)})^T \mathbf{x}_i = w_{2,0}^{(0)} + w_{2,1}^{(1)} x_{i,1} + w_{2,2}^{(0)} x_{i,2} + w_{2,3}^{(0)} x_{i,3} + w_{2,4}^{(0)} x_{i,4}$$

Neuron-3
$$z_{i,3}^{(1)} = (\mathbf{w}_3^{(0)})^T \mathbf{x}_i = w_{3,0}^{(0)} + w_{3,1}^{(0)} x_{i,1} + w_{3,2}^{(0)} x_{i,2} + w_{3,3}^{(0)} x_{i,3} + w_{3,4}^{(0)} x_{i,4}$$





$$z_{i,1}^{(2)} = (\mathbf{w}_1^{(1)})^T \mathbf{z}_i^{(1)} = w_{1,0}^{(1)} + w_{1,1}^{(1)} z_{i,1}^{(1)} + w_{1,2}^{(1)} z_{i,2}^{(1)} + w_{1,3}^{(1)} z_{i,3}^{(1)}$$

$$\mathbf{z}_i^{(2)} = W^{(1)} \mathbf{z}_i^{(1)}$$

$$\mathbf{z}_i^{(1)} = W^{(0)} \mathbf{x}_i$$

From Single Layer to Multiple Layers

- Multiple Linear Layers

Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$

…

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$

$W^{(0)} \in \mathbf{R}^{d_1 \times d_0}$

$W^{(1)} \in \mathbf{R}^{d_2 \times d_1}$

$W^{(2)} \in \mathbf{R}^{d_3 \times d_2}$

…

$W^{(L-1)} \in \mathbf{R}^{d_L \times d_{L-1}}$

$\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$

$\downarrow$

$\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$

$\downarrow$

$\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$

$\downarrow$

…

$\downarrow$

$\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$

$\downarrow$

From Single Layer to Multiple Layers

- Regression task

$$f(\mathbf{x}_i) = \mathbf{z}_i^{(L)}$$

- Classification task

$$f(\mathbf{x}_i) = \sigma(\mathbf{z}_i^{(L)})$$

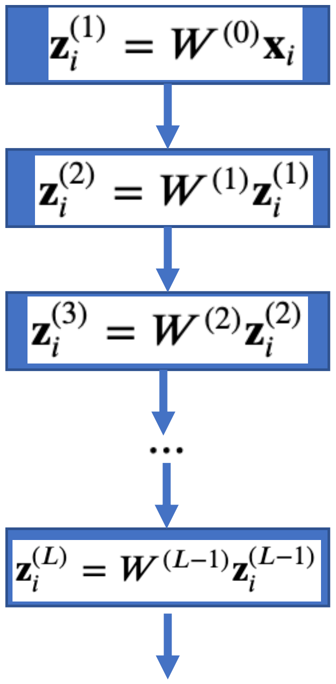
Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$

…

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$



From Linear Model to Non-linear Model

- Stack multiple linear model is still a LINEAR model

$$\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$$

$$\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$$

$$\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$$

$$\cdots$$

$$\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$$

$$\mathbf{z}_i^{(L)} = W^{(L-1)} \cdots W^{(2)}W^{(1)}W^{(0)}\mathbf{x}_i$$

Linear model

Deep linear networks are no more expressive than linear model!

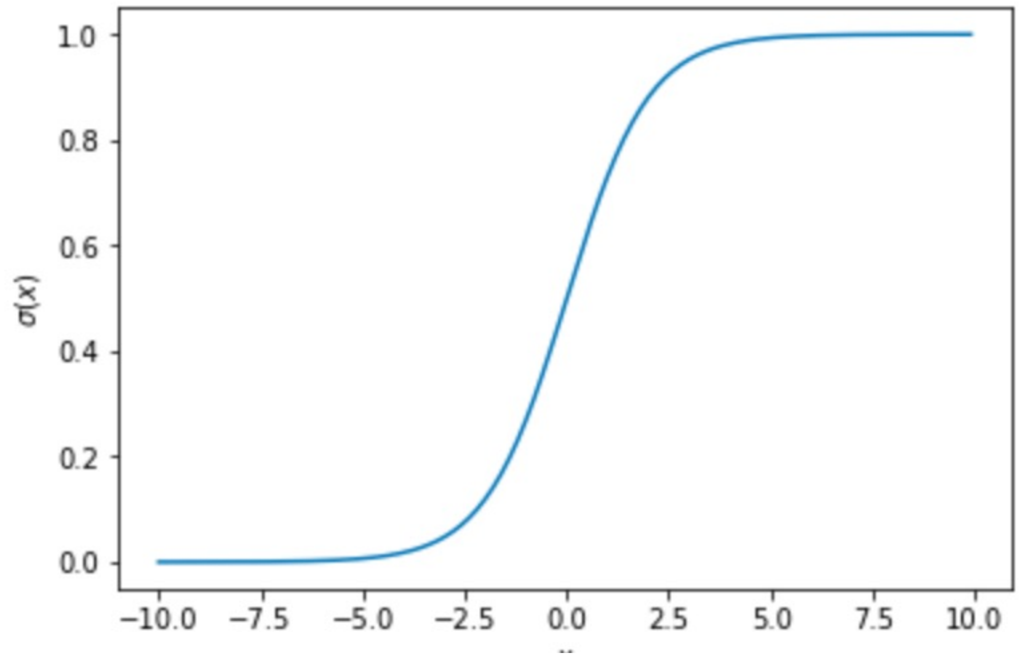
From Linear Model to Non-linear Model

- Activation function
  - Add (non-linear) activation functions to hidden layers
  - Multilayer fully-connected neural nets with nonlinear activation functions are universal approximators: they can approximate any function arbitrarily well.
- Examples:
  - Sigmoid function
  - Tanh function
  - ReLu function
  - LeakyReLu function
  - …

Activation Function

- Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Activation Function

- Sigmoid

Linear model

Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$

…

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$

$$\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$$
$$\downarrow$$
$$\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$$
$$\downarrow$$
$$\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$$
$$\downarrow$$
$$\cdots$$
$$\downarrow$$
$$\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$$
$$\downarrow$$

Non-linear model

$$\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i,\ \mathbf{h}_i^{(1)} = \sigma(\mathbf{z}_i^{(1)})$$
$$\downarrow$$
$$\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{h}_i^{(1)},\ \mathbf{h}_i^{(2)} = \sigma(\mathbf{z}_i^{(2)})$$
$$\downarrow$$
$$\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{h}_i^{(2)},\ \mathbf{h}_i^{(3)} = \sigma(\mathbf{z}_i^{(3)})$$
$$\downarrow$$
$$\cdots$$
$$\downarrow$$
$$\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{h}_i^{(L-1)},\ \mathbf{h}_i^{(L)} = \sigma(\mathbf{z}_i^{(L)})$$
$$\downarrow$$

Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i,\ \mathbf{h}_i^{(1)} = \sigma(\mathbf{z}_i^{(1)})$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{h}_i^{(1)},\ \mathbf{h}_i^{(2)} = \sigma(\mathbf{z}_i^{(2)})$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{h}_i^{(2)},\ \mathbf{h}_i^{(3)} = \sigma(\mathbf{z}_i^{(3)})$

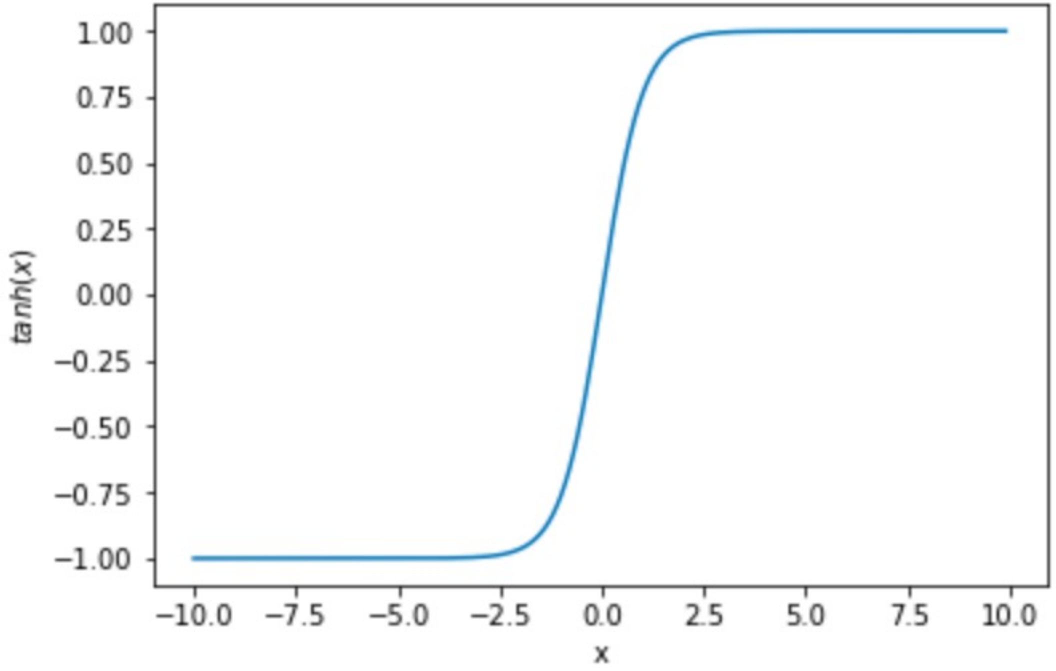
…

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{h}_i^{(L-1)},\ \mathbf{h}_i^{(L)} = \sigma(\mathbf{z}_i^{(L)})$

Activation Function

- Tanh function

$$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



Activation Function

- Tanh function

Linear model

Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$

...

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$

Non-linear activation function $\Rightarrow$

Non-linear model

Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$, $\mathbf{h}_i^{(1)} = \tanh(\mathbf{z}_i^{(1)})$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{h}_i^{(1)}$, $\mathbf{h}_i^{(2)} = \tanh(\mathbf{z}_i^{(2)})$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{h}_i^{(2)}$, $\mathbf{h}_i^{(3)} = \tanh(\mathbf{z}_i^{(3)})$

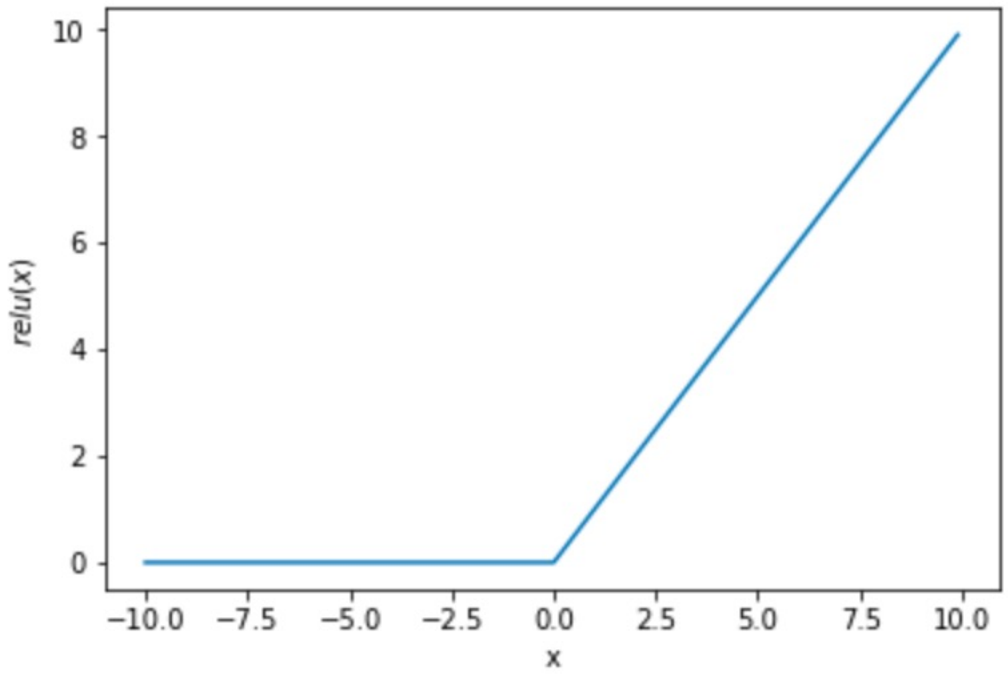
...

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{h}_i^{(L-1)}$, $\mathbf{h}_i^{(L)} = \tanh(\mathbf{z}_i^{(L)})$

Activation Function

- ReLu function

$$\mathrm{ReLu}(x) = \max(0, x)$$



Activation Function

- ReLu function

Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{z}_i^{(1)}$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{z}_i^{(2)}$

...

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{z}_i^{(L-1)}$

Non-linear activation function

Layer 1: $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i$, $\mathbf{h}_i^{(1)} = \mathrm{relu}(\mathbf{z}_i^{(1)})$

Layer 2: $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{h}_i^{(1)}$, $\mathbf{h}_i^{(2)} = \mathrm{relu}(\mathbf{z}_i^{(2)})$

Layer 3: $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{h}_i^{(2)}$, $\mathbf{h}_i^{(3)} = \mathrm{relu}(\mathbf{z}_i^{(3)})$

...

Layer L: $\mathbf{z}_i^{(L)} = W^{(L-1)}\mathbf{h}_i^{(L-1)}$, $\mathbf{h}_i^{(L)} = \mathrm{relu}(\mathbf{z}_i^{(L)})$

Example

- Image classification

MNIST

- 60,000 training samples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{60,000}$
- Each image $\mathbf{x}_i$ has 28×28 pixels
- Each label $\mathbf{y}_i$ is a 10-dim vector (one-hot encoding)

Example

- Image classification with logistic regression

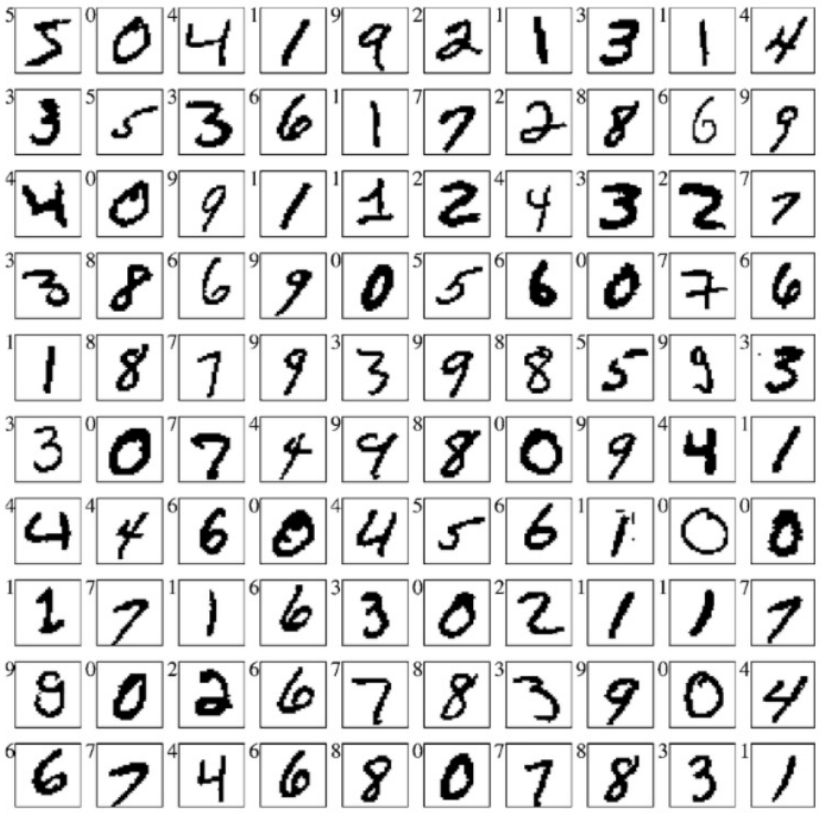
Linear Model: multi-class logistic regression

- Vectorize each 28×28 image to a 784-dim vector, $\mathbf{x}_i \in \mathbb{R}^{784}$
- Add the constant 1 to $\mathbf{x}_i$ (Introduce the bias term). Then, $\mathbf{x}_i \in \mathbb{R}^{785}$
- Denote the model parameter $W \in \mathbb{R}^{10 \times 785}$
- Then, $\mathbf{z}_i = W\mathbf{x}_i$
- Output the prediction using the softmax function

$$f(\mathbf{x}_i) = \mathrm{Softmax}(\mathbf{z}_i)$$

Example

- Image classification with MLP



- Input image $\mathbf{x}_i \in \mathbb{R}^{785}$

- $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i \in \mathbb{R}^{256}$
- $\mathbf{h}_i^{(1)} = \text{relu}(\mathbf{z}_i^{(1)}) \in \mathbb{R}^{256}$

Hidden Layer 1

- $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{h}_i^{(1)} \in \mathbb{R}^{128}$
- $\mathbf{h}_i^{(2)} = \text{relu}(\mathbf{z}_i^{(2)}) \in \mathbb{R}^{128}$

Hidden Layer 2

- $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{h}_i^{(2)} \in \mathbb{R}^{10}$
- $\hat{\mathbf{y}}_i = \text{Softmax}(\mathbf{z}_i^{(3)}) \in \mathbb{R}^{10}$

Output Layer

Example

- Image classification with MLP

  - Input image $\mathbf{x}_i \in \mathbb{R}^{785}$

  - $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i \in \mathbb{R}^{256}$
  - $\mathbf{h}_i^{(1)} = \text{relu}(\mathbf{z}_i^{(1)}) \in \mathbb{R}^{256}$

  - $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{h}_i^{(1)} \in \mathbb{R}^{128}$
  - $\mathbf{h}_i^{(2)} = \text{relu}(\mathbf{z}_i^{(2)}) \in \mathbb{R}^{128}$

  - $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{h}_i^{(2)} \in \mathbb{R}^{10}$
  - $\hat{\mathbf{y}}_i = \text{Softmax}(\mathbf{z}_i^{(3)}) \in \mathbb{R}^{10}$

  - Training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{60,000}$
  - Loss function: $L = -\sum_{i=1}^{60000} \sum_{j=1}^{10} \mathbf{y}_{ij} \log(\hat{\mathbf{y}}_{ij})$

Example

- How many model parameters?

- Input image $\mathbf{x}_i \in \mathbb{R}^{785}$

- $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i \in \mathbb{R}^{256}$
- $\mathbf{h}_i^{(1)} = \text{relu}(\mathbf{z}_i^{(1)}) \in \mathbb{R}^{256}$

- $\mathbf{z}_i^{(2)} = W^{(1)}\mathbf{h}_i^{(1)} \in \mathbb{R}^{128}$
- $\mathbf{h}_i^{(2)} = \text{relu}(\mathbf{z}_i^{(2)}) \in \mathbb{R}^{128}$

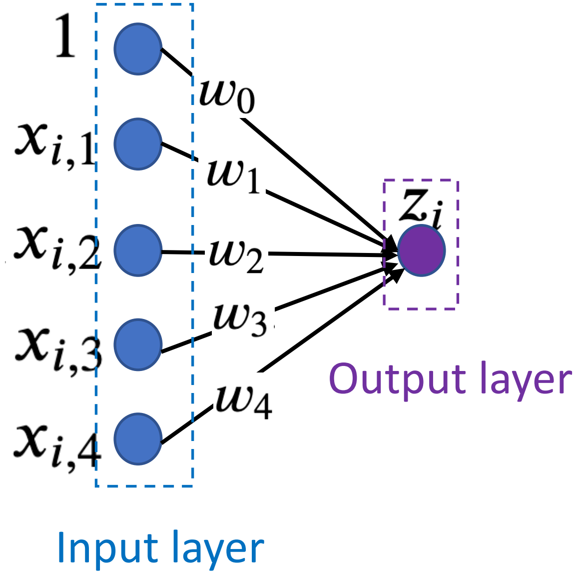
- $\mathbf{z}_i^{(3)} = W^{(2)}\mathbf{h}_i^{(2)} \in \mathbb{R}^{10}$
- $\hat{\mathbf{y}}_i = \text{Softmax}(\mathbf{z}_i^{(3)}) \in \mathbb{R}^{10}$

Linear Model: multi-class logistic regression

- Vectorize each 28×28 image to a 784-dim vector, $\mathbf{x}_i \in \mathbb{R}^{784}$
- Add the constant 1 to $\mathbf{x}_i$ (Introduce the bias term). Then, $\mathbf{x}_i \in \mathbb{R}^{785}$
- Denote the model parameter $W \in \mathbb{R}^{10 \times 785}$
- Then, $\mathbf{z}_i = W\mathbf{x}_i$
- Output the prediction using the softmax function

Optimization

- Use Stochastic Gradient Descent method to learn model parameters
  - Single layer



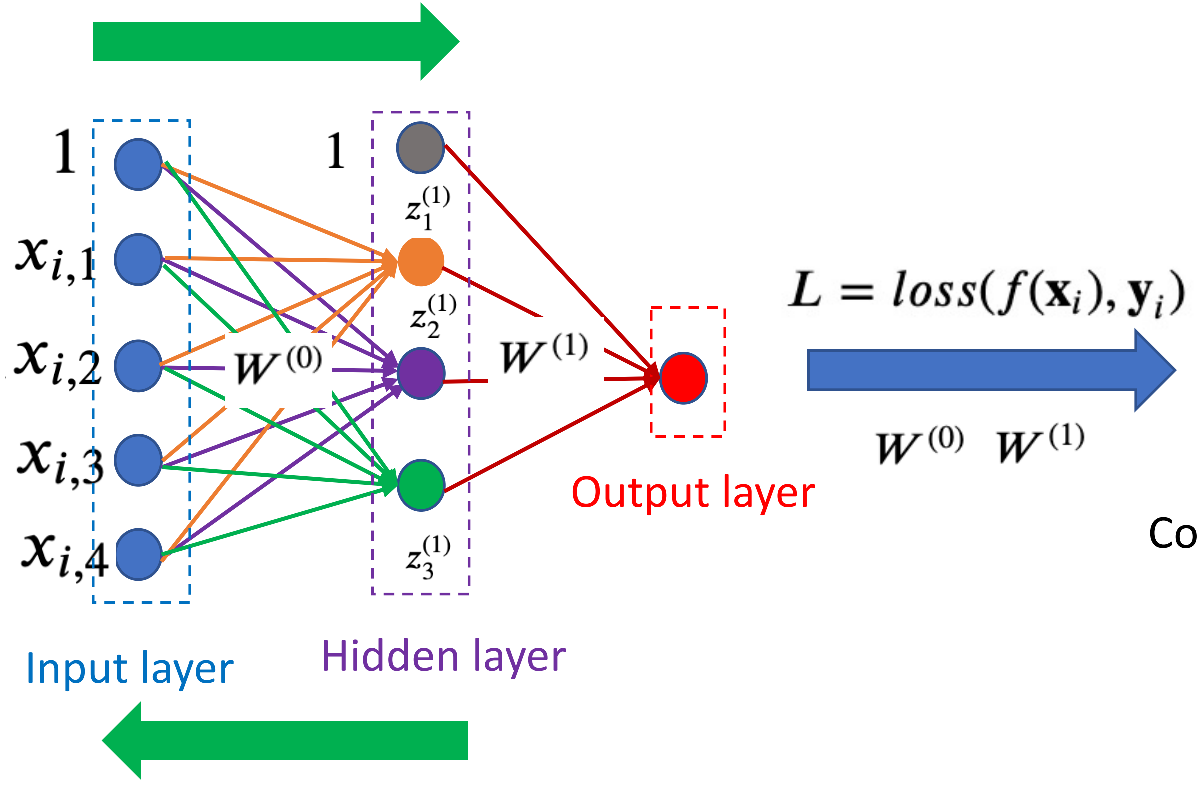
$$L = loss(f(\mathbf{x}_i), \mathbf{y}_i)$$

$$\xrightarrow[\boldsymbol{W}]{}$$

$$W_{t+1} = W_t - \eta \frac{\partial L}{\partial W}|_{W=W_t}$$

Optimization

- Use Stochastic Gradient Descent method to learn model parameters
  - Multiple layer: how to compute gradients?



It is easy to compute gradients regarding $W^{(1)}$

How to compute the gradient regarding $W^{(0)}$

Backpropagation!

Compute gradients from the last layer to the input layer!

Optimization

- Chain rule
  - For the composite function

$$h(x) = f(g(x))$$

  - The gradient is

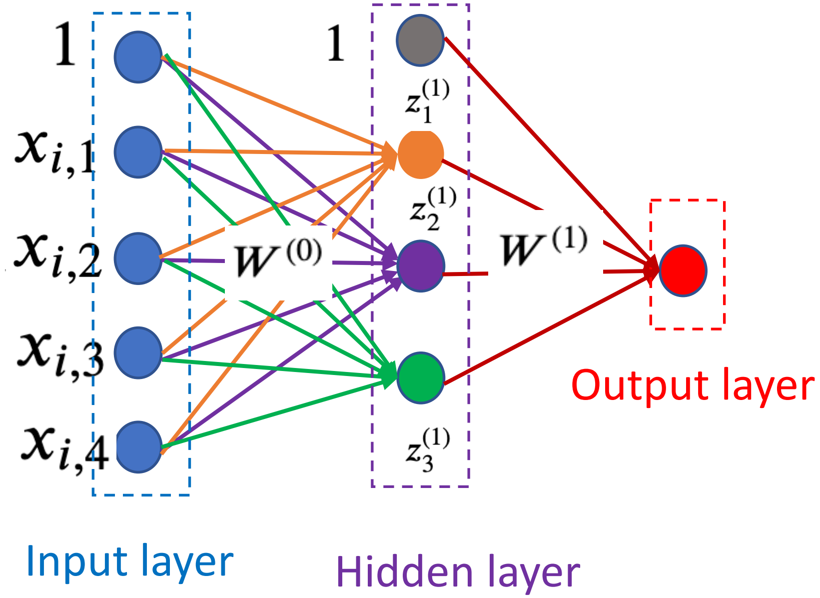
$$\frac{\partial h(x)}{\partial x} = \frac{\partial f(g)}{\partial g} \frac{\partial g(x)}{\partial x}$$

- $f(y) = y^2$
- $g(x) = 3x + 5$
- $h(x) = f(g(x))$

- $\frac{\partial f(g)}{\partial g} = 2g$
- $\frac{\partial g(x)}{\partial x} = 3$
- $\frac{\partial h(x)}{\partial x} = \frac{\partial f(g)}{\partial g} \frac{\partial g(x)}{\partial x} = (2 * (3x + 5)) * 3$

Backpropagation

- 1. Compute gradients of the last layer



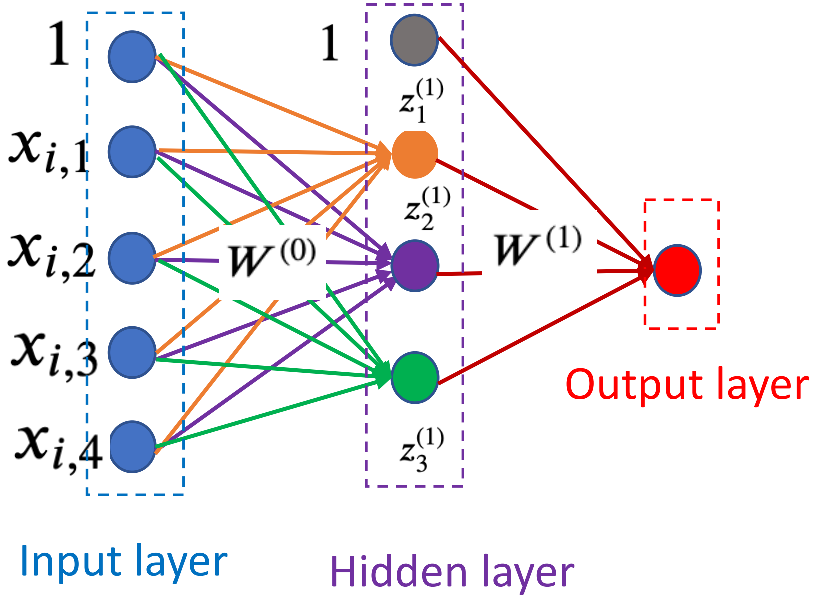
Loss function is the function of $W^{(1)}$ and $\mathbf{z}^{(1)}$

Compute their gradients as regular models

$$\frac{\partial L}{\partial W^{(1)}} \qquad \frac{\partial L}{\partial \mathbf{z}^{(1)}}$$

Backpropagation

- 2. Compute gradients of hidden layers based on the chain rule



How to compute $\frac{\partial L}{\partial W^{(0)}}$

$\mathbf{z}^{(1)}$ is a function of $W^{(0)}$: $\mathbf{z}^{(1)} = W^{(0)}\mathbf{x}$

Based on the chain rule:

$$\frac{\partial L}{\partial W^{(0)}} = \boxed{\frac{\partial L}{\partial \mathbf{z}^{(1)}}} \boxed{\frac{\partial \mathbf{z}^{(1)}}{\partial W^{(0)}}}$$

$\mathbf{x}$

known

Backpropagation

- $\mathbf{z}_i^{(1)} = W^{(0)}\mathbf{x}_i \in \mathbb{R}^{256}$
- $\mathbf{z}_i^{(2)} = W^{(1)}\text{relu}(\mathbf{z}_i^{(1)}) \in \mathbb{R}^{128}$
- $\mathbf{z}_i^{(3)} = W^{(2)}\text{relu}(\mathbf{z}_i^{(2)}) \in \mathbb{R}^{10}$
- $\hat{\mathbf{y}}_i = \text{Softmax}(\mathbf{z}_i^{(3)}) \in \mathbb{R}^{10}$

Forward pass

- $\frac{\partial L}{\partial W^{(0)}} = \frac{\partial L}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial W^{(0)}}$
- $\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial W^{(1)}}$
- $\frac{\partial L}{\partial \mathbf{z}^{(1)}} = \frac{\partial L}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}}$
- $\frac{\partial L}{\partial W^{(2)}}$
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}}$

Backward pass

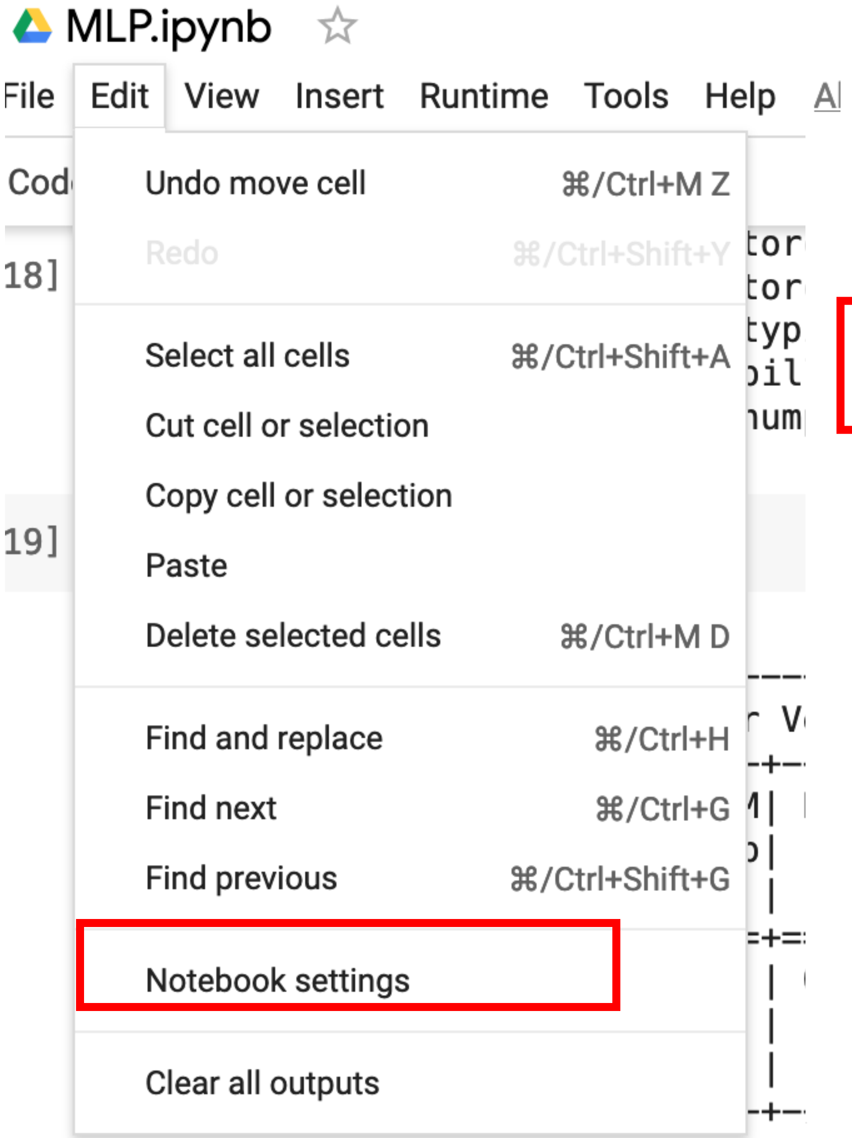
Implementation

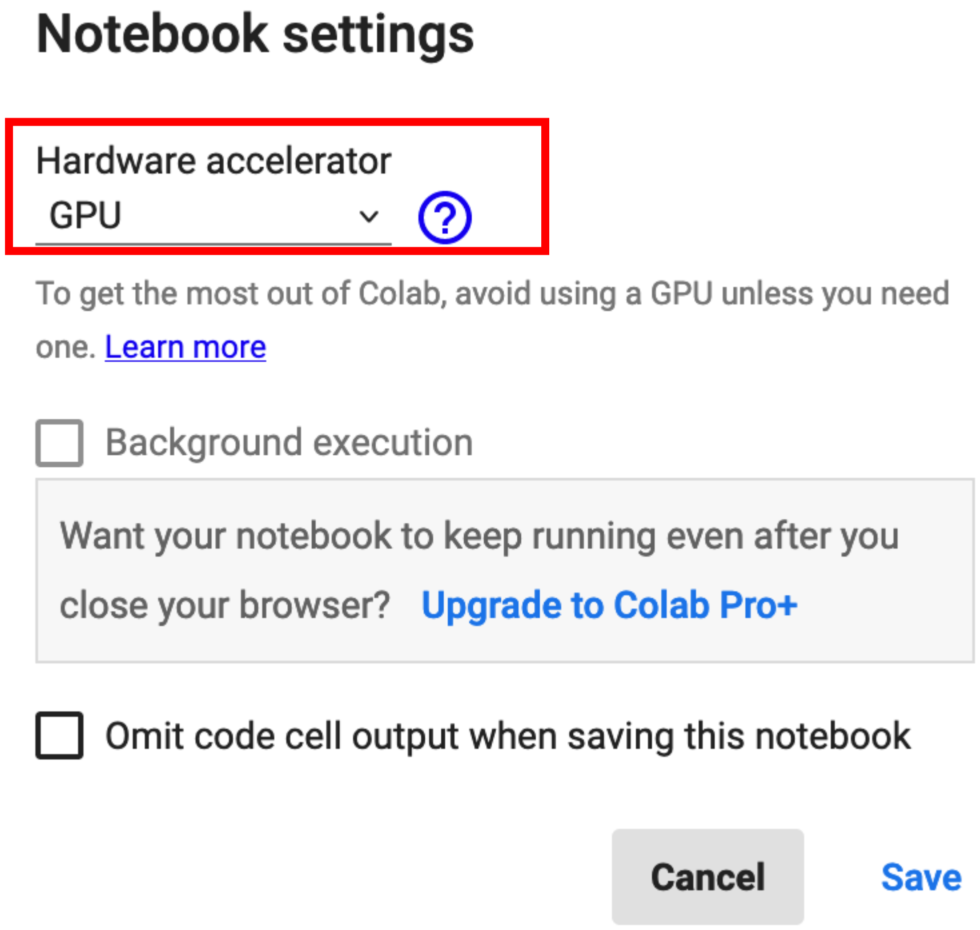
- Deep Learning Toolbox
  - Tensorflow
  - PyTorch

- GPU resources:
  - Google Colab (free)

Implementation

- Google Colab





Implementation

- Build an MLP with PyTorch

```python
# build an mlp
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.fc1 = nn.Linear(28*28, 256)   # linear layer (784 -> 256)
        self.fc2 = nn.Linear(256,128)  # linear layer (256 -> 128)
        self.fc3 = nn.Linear(128,10)  # linear layer (128 -> 10)

    def forward(self, x):
        h0 = x.view(-1,28*28) #input layer
        h1 = F.relu(self.fc1(h0)) # hidden layer 1
        h2 = F.relu(self.fc2(h1)) # hidden layer 2
        h3 = self.fc3(h2) # output layer

        return h3
```

Implementation

- Loss function and optimizer

```
# loss function
criterion = nn.CrossEntropyLoss()

# optimizer
optimizer = torch.optim.SGD(model.parameters(),lr = args['lr'])
```

- Training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{60,000}$
- Loss function: $L = -\sum_{i=1}^{60000} \sum_{j=1}^{10} \mathbf{y}_{ij} \log(\hat{\mathbf{y}}_{ij})$

Implementation

- Train the model

```
for batch_idx, (data, target) in enumerate(train_loader):
    data, target = data.cuda(), target.cuda()

    output = model(data)
    loss = criterion(output, target)

    # compute gradients
    optimizer.zero_grad()
    loss.backward()

    #to do a one-step update on our parameter.
    optimizer.step()

    #Print out the loss periodically.
    if batch_idx % args['log_interval'] == 0:
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))
```

Forward pass

Compute the loss function value

Implementation

- Test the model

```
test_loss = 0
correct = 0
for data, target in test_loader:
    data, target = data.cuda(), target.cuda()

    output = model(data)
    test_loss += criterion(output, target).item() # sum up batch loss
    pred = output.data.max(1, keepdim=True)[1]
    correct += pred.eq(target.data.view_as(pred)).long().cpu().sum()

test_loss /= len(test_loader.dataset)

print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
    test_loss, correct, len(test_loader.dataset),
    100. * correct / len(test_loader.dataset)))
```