RISTEK Data Science & Analytics

Internal Class: Deep Learning

Neural Network Introduction

Elements of Machine Learning

$$h_\theta(x)$$

Hypothesis Class

Function that can be learned by a learning algorithm. Defined by the architecture of the model.

$$l(h_\theta(x), y) = -h_y(x) + \log \sum_{j=1}^{k} \exp(h_j(x))$$

Loss Function

Measures how well the model learns the data. Defines the objective of what to learn.

$$\theta \coloneqq \theta - \frac{\alpha}{B} \sum_{i}^{B} \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

Optimization Method

How the model will adjust the weight based on the calculation of loss function.

What makes a NN framework?

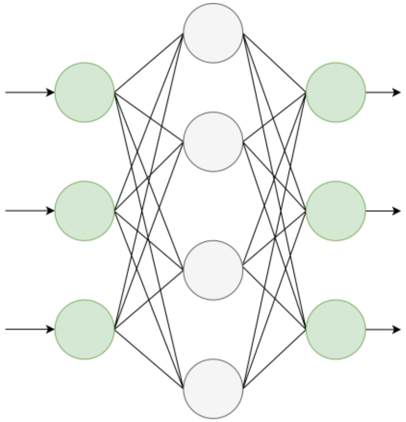
$$h_\theta(x)$$

Hypothesis Class

Function that can be learned by a learning algorithm. Defined by the architecture of the model.

Neural network frameworks are built around constructing deep learning models. Actually, the underlying concept is REALLY EASY compared to optimizing existing algorithms.

How does NN learn then?



$$l(h_\theta(x), y) = -h_y(x) + \log \sum_{j=1}^{k} \exp(h_j(x))$$

$$\frac{\partial}{\partial x_i}$$

Forward pass, calculate the prediction output.

Compute loss function from the prediction output.

Do backpropagation to calculate gradient in respect to the model inputs.

Adjust the weight of each parameter from calculated gradients.

Optimization

Differentiations???

Analytical

$$\frac{d}{dx}\left[\left(f(x)\right)^n\right] = n\left(f(x)\right)^{n-1} \cdot f'(x)$$

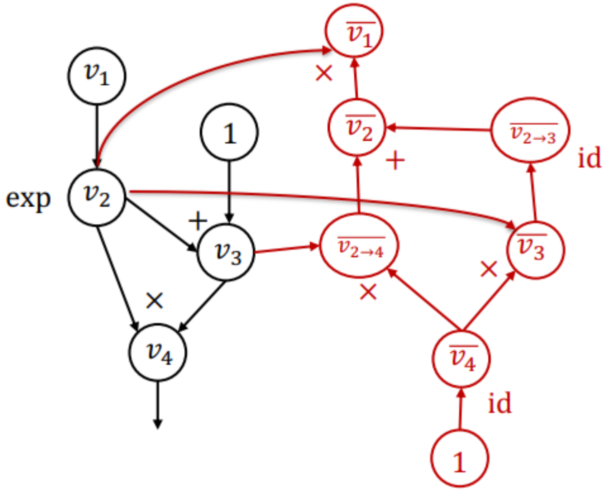
$$\frac{d}{dx}\left[f\left(g(x)\right)\right] = f'\left(g(x)\right)g'(x)$$

Calcworkshop.com

Numeric

$$\frac{\partial f(\theta)}{\partial \theta_i} = \frac{f(\theta + \epsilon e_i) - f(\theta - \epsilon e_i)}{2\epsilon} + o(\epsilon^2)$$

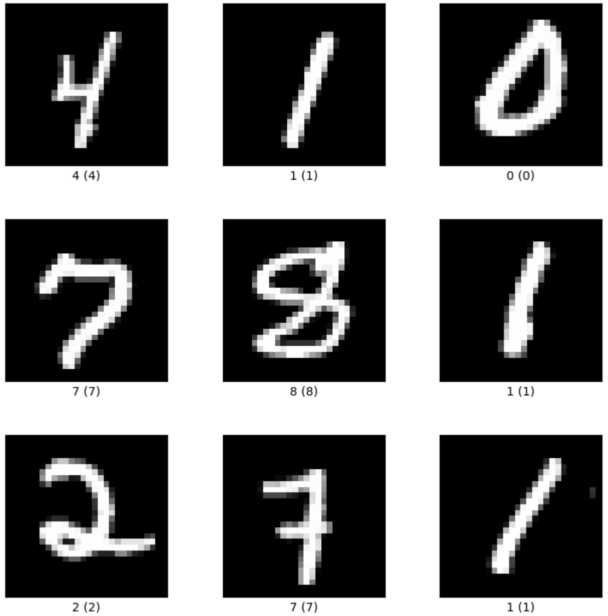
Computation Graph



This is just one of many optimization attempts in order to make NN frameworks more efficient!

Implementation

MNIST: Hello World of Deep Learning



Task Definition

Given a set of 28x28 grayscaled handwritten digits, how can we make a model such that it could predict their respective digit classes correctly?

Dataset Utility Class

notebook.ipynb

```
class CustomDataset(Dataset):
    def __init__(self, features: pd.DataFrame, labels: pd.Series = None):
        self.features = torch.tensor(features.values, dtype=torch.float32)
        self.labels = torch.tensor(labels.values, dtype=torch.int64)\
            if labels is not None else None

    def __len__(self):
        return len(self.features)

    def __getitem__(self, idx):
        if self.labels is None:
            return self.features[idx]
        return self.features[idx], self.labels[idx]
```

Data Processing

notebook.ipynb

```
train_frac = 0.8
train_len = int(train_frac * len(train))
train_data, validation_data = train.iloc[:train_len], train.iloc[train_len:]

train_dataset = CustomDataset(train_data.drop(columns='label'), train_data['label'])
validation_dataset = CustomDataset(train_data.drop(columns='label'), train_data['label'])

train_dataloader = DataLoader(train_dataset, batch_size=64, shuffle=True)
validation_dataloader = DataLoader(validation_dataset, batch_size=64, shuffle=True)
```

Simple Dense Neural Model

notebook.ipynb

```python
class DenseModel(nn.Module):
    def __init__(self, in_features: int, num_classes: int):
        super().__init__()
        self.input = nn.Linear(in_features, 256)
        self.hidden = nn.Linear(256, 64)
        self.output = nn.Linear(64, num_classes)

    def forward(self, x):
        x = F.relu(self.input(x))
        x = F.relu(self.hidden(x))
        logits = self.output(x)

        return logits
```

Minimalistic Training Loop

notebook.ipynb

```python
def training_loop(model, optimizer, epochs, loss_fn, data):
    for t in range(epochs):
        loop = tqdm(data, total=len(data))
        model.train()

        for _, (X, y) in enumerate(loop):
            optimizer.zero_grad()

            pred = model(X)
            loss = loss_fn(pred, y)

            loss.backward()
            optimizer.step()

            loop.set_description(f"Epoch [{t+1}/{epochs}]")
            loop.set_postfix(loss=loss.item())

    print("Training completed.")
```

Minimalistic Validation Loop

notebook.ipynb

```python
def validation_loop(dataloader, model, loss_fn):
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size

    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f}\n")
```

Model Inference

notebook.ipynb

```
def inference(dataloader, model):
    model.eval()
    predictions = []

    with torch.no_grad():
        for X in tqdm(dataloader, desc="Inference"):
            pred = model(X)
            predictions.append(pred.argmax(1).cpu().numpy())

    predictions = np.concatenate(predictions)

    return predictions
```

Tired of these “low level” stuffs?

Pretrained models

PyTorch compatibility

To the rescue!

Utility functions

Goodbye to *boilerplate codes*, spend more time on ~~debugging errors~~ **training the model**!