CE6146
Introduction to Deep Learning
Introduction to AWS for Deep Learning

Chia-Ru Chung

Department of Computer Science and Information Engineering

National Central University

2023/11/9

20231102 Exercise

1. **D**	2. **B**	3. **B**	4. **A**	5. **B**
6. **C**	7. **D**	8. **A**	9. **C**	10. **B**
11. **C**	12. **B**	13. **C**	14. **C**	15. **B**
16. **C**	17. **D**	18. **A**	19. **A**	20. **C**

20231026 Exercise #20

You are eligible for the credit if you chose B. Contact the TA to have the points added.

- Which gating unit in an LSTM controls what information will be stored in the cell state?

(A) Input gate (B) Forget gate

(C) Memory gate (D) Output gate

- Given the wording of the question, the most direct and common answer would be (A) Input gate because it is specifically responsible for updating the cell state with information from the current input. The forget gate's role is more about filtering the past information, which indirectly affects what the future state of the cell will be by deciding what to "forget" from the existing state.
- However, the forget gate's function of retaining or removing information from the previous state could be viewed as a method of “controlling” what gets stored (not forgetting ultimately results in storing). (B) Forget gate could also be considered a valid answer. This is particularly true if the question includes the retention of past information as a form of “storage”.

20231106 Exercise #2

- What is the purpose of the encoder in an autoencoder?

  (A) To classify the input data (B) To compress the input into a latent-space representation

  (C) To reconstruct the input data (D) To separate the input data into clusters

- The answer provided as “C) To reconstruct the input data” describes the ultimate goal of the entire autoencoder system rather than the specific function of the encoder within it. The encoder's role is specifically to transform the input data into a lower-dimensional space, which is a compressed representation called the latent space. It is the decoder's job to take this representation and reconstruct the input data from it.

20231106 Exercise #6

- In an RNN, what is passed from one step of the sequence to the next?
  (A) Input data (B) Output data (C) Hidden state (D) Error gradient

- In the context of RNN the hidden state is the memory of the network. It contains information about all previous inputs in the sequence that the network has seen up to that point. At each step in the sequence, the RNN takes the current input and the previous hidden state to produce the new hidden state, which will then be passed to the next step in the sequence. This process allows the RNN to maintain a form of memory across the inputs it processes.
- The meaning of the question is to identify what element within the RNN architecture is responsible for carrying information across time steps or sequence elements. This is a fundamental characteristic of RNNs that differentiates them from other neural network architectures that do not have a temporal component.

Outline

- Review
- Introduction to AWS for Deep Learning

Review

- Review All Methods So Far
- Generative Adversarial Networks

Neural Networks (NN)

- Definition: Algorithms inspired by the human brain to recognize patterns. Basic NNs, often referred to as fully connected or dense networks, are used for a wide range of prediction tasks where the input data is structured tabular data.
- Structure: Input Layer -> Hidden Layers -> Output Layer.
- Function: Non-linear transformation of input data to predictions.
- Use Cases: Stock price prediction, customer churn analysis.

Convolutional Neural Networks (CNNs)

- CNNs employ a mathematical operation called convolution which processes data with a grid-like topology. Features are learned and used hierarchically, with higher-level, more abstract features built upon lower-level, simpler features.
- Specialty: Efficient for grid-like data (e.g., images).
- Components: Convolutional layers, pooling layers, fully connected layers.
- Feature Hierarchy: Learns from general to specific patterns.
- Use Cases: Image classification, face recognition, medical image analysis.

Recurrent Neural Networks (RNNs)

- RNNs have the ability to retain information over time because of their internal memory. RNNs process sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far.
- Design: Processes sequences & retains information over time.
- Challenges: Difficulty with long-range dependencies due to vanishing gradients.
- Use Cases: Text generation, speech recognition.

Long Short-Term Memory (LSTM)

- LSTM is a special kind of RNN, capable of learning long-term dependencies. Instead of having a single neural network layer, there are four, interacting in a very special way.
- Enhancement over RNN: Solves long-term dependency problems.
- Structure: Complex cell state with input, output, and forget gates.
- Use Cases: Machine translation, time series forecasting.

Gated Recurrent Units (GRUs)

- GRUs combine the forget and input gates into a single “update gate”. They also merge the cell state and hidden state, and make a few other changes. The result is a simpler and faster RNN model.
- Simplification of LSTM: Combines gates for efficiency.
- Structure: Update and reset gates.
- Use Cases: Language modeling, sequence learning with fewer parameters.

Autoencoders

- Autoencoders are a type of neural network used to learn efficient representations of the input data, called encodings, typically for the purpose of dimensionality reduction or feature learning.
- Purpose: Efficient representation and dimensionality reduction.
- Mechanism: Encoding input into a latent space, then reconstruction.
- Use Cases: Anomaly detection, noise reduction, data compression.

Optimization Techniques

- The process of optimizing a NN involves adjusting the weights of the network to minimize a loss function. Common optimizers include Gradient Descent, Stochastic Gradient Descent (SGD), Adam, and RMSprop. These optimizers differ in how they manage the learning rate and momentum, which can significantly impact the convergence speed and quality of the training process.
- Objective: Minimize loss function, improve model predictions.
- Strategy: Adjust learning rate, use momentum.

Regularization Techniques

- Regularization methods are used to prevent overfitting, where the model performs well on the training data but poorly on unseen data.
- Goal: Prevent overfitting, improve generalization.
- Methods: L1/L2 Regularization, Dropout, Early Stopping.
- Implementation: Penalty on weight size, randomly ignore nodes, halt training on validation loss increase.

Choosing the Right Method

- Criteria: Data type, problem complexity, performance requirements.
- Use NNs for structured data prediction.
- Use CNNs for spatial data processing like image and video analysis.
- Use RNNs, LSTMs, or GRUs for sequential data such as text and time series, with LSTMs preferred for longer sequences and GRUs when computational efficiency is key.
- Use Autoencoders for dimensionality reduction and feature learning without labeled data.

Generative Adversarial Networks (1/3)

- Generative Adversarial Networks (GANs) are a class of artificial intelligence algorithms used in unsupervised machine learning
- A GAN consists of two neural networks, namely the Generator and the Discriminator, which are trained simultaneously.
- The Generator aims to produce data that mimics some distribution (often the distribution of the training data), while the Discriminator aims to distinguish between genuine and generated data.

Generative Adversarial Networks (2/3)

- Generator:
  - This network aims to produce data that is indistinguishable from some real data.
  - It starts with a random noise and gradually refines its output as the training process advances.
- Discriminator:
  - This network tries to differentiate between real and fake (generated) data.
  - It takes in both real and fake samples and assigns a probability that a given sample is real.

Generative Adversarial Networks (3/3)

- The Generator and Discriminator are in essence co-adversaries, engaged in a kind of cat-and-mouse game, often described mathematically as a minimax game or a zero-sum game.
- In a zero-sum game, one player's gain or loss is exactly balanced by the losses or gains of another player.
- In the context of GANs, the Generator and Discriminator are essentially playing a zero-sum game against each other.

Data Flow in GAN

1. Random noise vectors are sampled from the latent space.
2. These vectors are fed into the Generator.
3. Upsampling layers in the Generator transform these vectors into high-dimensional data.
4. This generated data and real data are fed into the Discriminator.
5. Downsampling layers in the Discriminator reduce the data dimensions.
6. The Discriminator then classifies the downsampled data as real or fake.

Latent Space in GANs

- The latent space is a high-dimensional space from which the random noise vectors are sampled as input to the Generator.
- It serves as a compact representation of the data's inherent features.
- In most GANs, you randomly sample from a predefined distribution (usually a Gaussian distribution) to generate noise vectors.
- The latent space evolves implicitly as the Generator learns to map these random vectors to realistic data.

How to Train a GAN (1/4)

1. Initialize Networks: Initialize the Generator (G) and Discriminator (D) with random weights.
2. Preprocess Data: Prepare your real data samples, often involving normalization and data augmentation.

How to Train a GAN (2/4)

3. Training Loop: Typically involves the following steps:

Step 1: Train the Discriminator

1.1 Sample a mini-batch of real data x from the data distribution.

1.2 Generate a mini-batch of fake data G(z), where z is a random noise vector.

1.3 Compute the Discriminator's loss on real and fake data. Commonly used loss is Binary Cross-Entropy: $L_D = -\log\big(D(x)\big) - \log\big(1 - D(G(z))\big)$

1.4 Update the Discriminator's weights using backpropagation to minimize L_D.

How to Train a GAN (3/4)

3. Training Loop: Typically involves the following steps:

   Step 2: Train the Generator

   2.1 Generate a new mini-batch of fake data G(z).

   2.2 Compute the Generator's loss. The aim is to fool the Discriminator into thinking the fake samples are real: $L_G = -\log(D(G(z)))$

   2.3 Update the Generator's weights using backpropagation to minimize L_G.

How to Train a GAN (4/4)

3. Training Loop: Typically involves the following steps:

   Step 3: Check Convergence

   3.1 Monitor the losses L_D and L_G.

   3.2 Optionally, evaluate using metrics like Fréchet Inception Distance (FID) or Inception Score (IS).

   3.3 If the networks have converged or met specific criteria, exit the loop; otherwise, return to Step 1.

Evaluation Metrics for GANs (1/3)

- Fréchet Inception Distance (FID):
  - FID measures the similarity between the generated data and real data distributions in the feature space of a pre-trained Inception network.
  - The FID between two multivariate Gaussians $N(\mu_1, \Sigma_1)$ and $N(\mu_2, \Sigma_2)$ is defined as: $\text{FID} = ||\mu_1 - \mu_2||^2 + \text{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{\frac{1}{2}})$. μ_1, Σ_1 are the mean and covariance of features from the real data, and μ_2, Σ_2 are from the generated data.

Evaluation Metrics for GANs (2/3)

- Inception Score (IS)
  - IS uses the Inception network to classify generated images into different classes and computes a score based on the distribution of these classes.
  - For a set of N generated samples, the IS is given by: IS $=$ $\exp(\frac{1}{N}\sum_{i=1}^{N} KL(p(y|x_i)||p(y))$. $p(y|x_i)$ is the conditional class distribution for sample x_i, and $p(y)$ is the marginal class distribution.

Evaluation Metrics for GANs (3/3)

- Wasserstein Distance (For WGANs)
  - This metric is specific to Wasserstein GANs (WGANs) and measures the Earth Mover's Distance between the real and generated data distributions.
  - In the context of WGANs, the Wasserstein distance is given by the Discriminator‘s loss: $W = \max_D \mathbb{E}_{x \sim P_{\text{real}}}[D(x)] - \mathbb{E}_{x \sim P_{\text{fake}}}[D(x)]$.$\mathbb{E}$ represents the expected value, and P_{real}, P_{fake} are the real and fake data distributions..

A simple GAN using Keras (1/6)

1. Import Required Libraries

```
from keras.models import Sequential, Model
from keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Flatten
from keras.layers import Input, Conv2D, Conv2DTranspose
from keras.optimizers import Adam
from keras.datasets import mnist
import numpy as np
```

2. Load and Preprocess the Data

```
(x_train, _), (_, _) = mnist.load_data()
x_train = (x_train.astype(np.float32) - 127.5) / 127.5
x_train = np.expand_dims(x_train, axis=-1)
```

A simple GAN using Keras (2/6)

3. Define the Generator Model

```
def build_generator(input_shape):
    model = Sequential()
    model.add(Dense(7 * 7 * 128, activation='relu', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Reshape((7, 7, 128)))  # Corresponding reshape
    model.add(Conv2DTranspose(64, kernel_size=3, strides=2, padding='same'))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Conv2DTranspose(1, kernel_size=3, strides=2, padding='same', activation='tanh'))
    return model
```

A simple GAN using Keras (3/6)

4. Define the Discriminator Model

```
def build_discriminator(input_shape):
    model = Sequential()
    model.add(Conv2D(64, kernel_size=3, strides=2, padding='same', input_shape=input_shape))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model
```

A simple GAN using Keras (4/6)

5. Compile the Models

```
def compile_models(generator, discriminator):
    discriminator.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    discriminator.trainable = False
    gan_input = Input(shape=(100,))
    gan_output = discriminator(generator(gan_input))
    gan = Model(gan_input, gan_output)
    gan.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    return gan
```

A simple GAN using Keras (5/6)

6. Train the GAN

```
def train_gan(generator, discriminator, gan, epochs=10000, batch_size=128):
    for epoch in range(epochs):
        # Train Discriminator
        noise = np.random.normal(0, 1, (batch_size, 100))
        fake_images = generator.predict(noise)
        real_images = x_train[np.random.randint(0, x_train.shape[0], batch_size)]
        labels_real = np.ones((batch_size, 1))
        labels_fake = np.zeros((batch_size, 1))
        d_loss_real = discriminator.train_on_batch(real_images, labels_real)
        d_loss_fake = discriminator.train_on_batch(fake_images, labels_fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # Train Generator
        noise = np.random.normal(0, 1, (batch_size, 100))
        labels_gan = np.ones((batch_size, 1))
        g_loss = gan.train_on_batch(noise, labels_gan)

        print(f"{epoch} [D loss: {d_loss[0]}] [G loss: {g_loss}]")
```

A simple GAN using Keras (6/6)

7. Execute the Training

```
# Build and compile the models
generator = build_generator((100,))
discriminator = build_discriminator(x_train[0].shape)
gan = compile_models(generator, discriminator)

# Train the GAN
train_gan(generator, discriminator, gan)
```

Introduction to AWS for Deep Learning

- Introduction to Cloud Computing and AWS
- AWS Core Services for Deep Learning
- Deep Learning Frameworks on AWS

Cloud Computing

- Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing.

Why Cloud Computing

- In the context of deep learning and research, cloud computing provides the necessary computational power and storage required to process and analyze large datasets, train complex models, and perform extensive simulations.
- It enables researchers and practitioners to access state-of-the-art GPUs and other computing resources that might be too costly or impractical to maintain on-premises.
- This democratizes access to advanced computational capabilities, allowing for more innovation and exploration in various fields.

How Does Cloud Computing Work (1/2)

- Imagine you need a book to read. You have two options: buy the book and bring it home (which is like owning a computer with software installed) or go to a public library to read it (which is like using cloud computing).
  - Buying a Book: It's yours, but it costs more, and you need space at home. If you want ten books, you need to buy all ten.
  - Using the Library: You can read any book you want, you don't pay for each book, and you don't need space at home for a big bookshelf. You just need a library card, which is like your cloud account.

How Does Cloud Computing Work (2/2)

- User Request: You (the user) send a request to access computing services. This is like asking the librarian for a book.
- Data Center Response: The cloud provider's data center (a massive collection of servers and storage) receives your request and processes it. This is like the librarian checking the book out for you.
- Resource Allocation: The cloud provider allocates resources to meet your request. This can include storage space, CPU, memory, etc., just like the librarian finds you a place to sit and read.
- Service Delivery: The requested service (like a software application or data storage) is delivered to your device over the internet, ready for use. This is like reading the book in the library.
- Pay for Use: You only pay for the cloud services you use, for the time you use them, just as you might pay a fee to the library if you use a special service or rent a book.
- Release Resources: When you're done, the resources you used are freed up for someone else to use. This is like returning the book so the next person can read it.

Types of Cloud Services

- Infrastructure as a Service (IaaS):

  Provides basic infrastructure, compute, networking, and storage resources on-demand.

- Platform as a Service (PaaS):

  Offers a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure.

- Software as a Service (SaaS):

  Delivers software applications over the internet, on-demand, and typically on a subscription basis.

Infrastructure as a Service (IaaS)

- Components of IaaS: Virtual machines, servers, storage, networks, and operating systems.
- Provider Responsibility: The cloud provider manages the infrastructure, while users manage the applications, data, runtime, and middleware.
- User Control: Users have control over the infrastructure without physically managing it.
- Use Cases: Test and development, website hosting, storage and backup, web apps, high-performance computing.
- Examples: AWS EC2, Google Compute Engine, Microsoft Azure.

Platform as a Service (PaaS)

- Components of PaaS: Hosted development kits, database management systems, business analytics.
- Provider Responsibility: Cloud providers manage the infrastructure and platforms that run the applications.
- User Control: Users manage the applications and services they develop, and the cloud provider manages everything else.
- Use Cases: Development framework, analytics or business intelligence, additional services like workflow, directory, security.
- Examples: AWS Elastic Beanstalk, Google App Engine, Microsoft Azure App Services.

Platform as a Service (PaaS)

- Components of SaaS: Software applications accessible from various devices over the internet or a thin client.
- Provider Responsibility: SaaS providers manage the infrastructure, middleware, app software, and app data.
- User Control: Users get to use the software without worrying about installation, maintenance, or coding.
- Use Cases: Email, calendaring, office tools, customer relationship management (CRM).
- Examples: Google Workspace, Salesforce, Microsoft Office 365.

Benefits for Deep Learning

- IaaS Benefits:

  Flexibility to choose the computing resources as per the need of the deep learning models, which can be scaled up as the complexity of the model increases.

- PaaS Benefits:

  Provides a platform with built-in software components and tools specifically designed for application development, which can be beneficial for developing custom deep learning models and managing their lifecycle.

- SaaS Benefits:

  Access to sophisticated applications with AI and machine learning capabilities without the need for a deep understanding of the underlying algorithms or infrastructure.

Cloud Deployment Models

- Public Cloud:

  Services are delivered over the public internet and shared across organizations.

- Private Cloud:

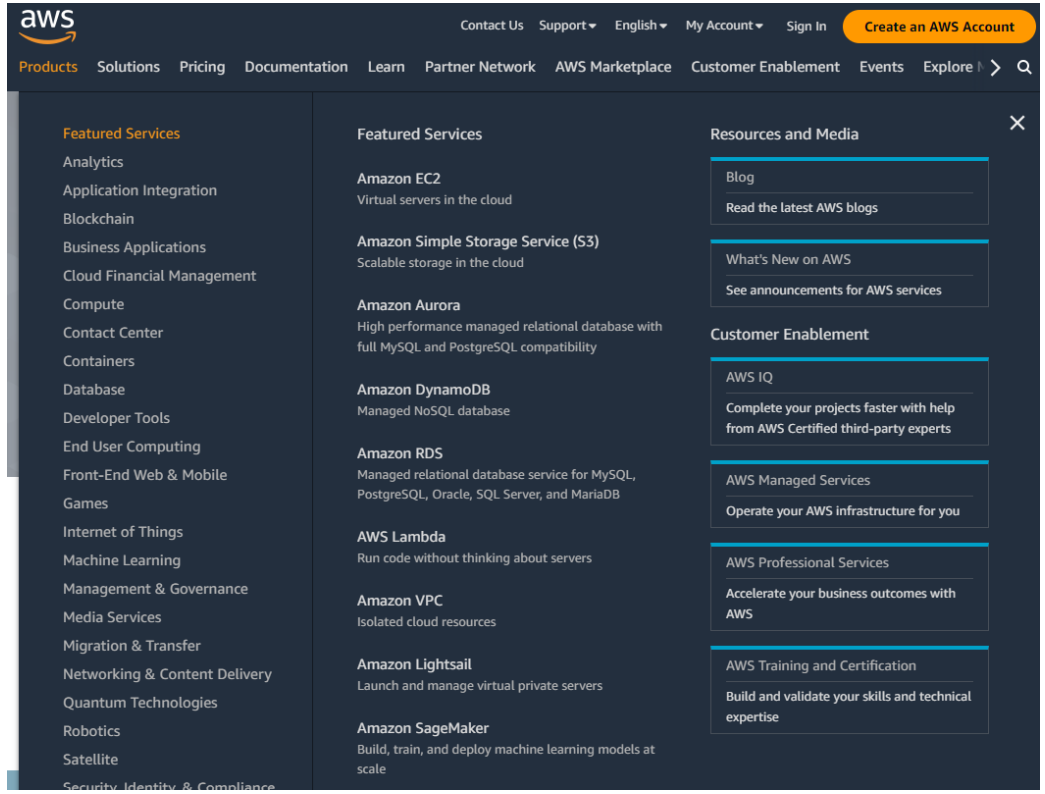
  Services are maintained on a private network for a single organization.

- Hybrid Cloud:

  Combines public and private clouds, bound together by technology that allows data and applications to be shared between them.

Overview of Amazon Web Services

- Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 200 fully featured services from data centers globally.



Source: https://aws.amazon.com/tw/

Core AWS Services for Deep Learning (1/2)

- Amazon EC2: Provides scalable computing capacity in the AWS cloud. It reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. (https://aws.amazon.com/tw/ec2/)
- Amazon S3: Offers secure, durable, and highly-scalable object storage. It's a simple storage service that offers an extremely durable, highly available, and infinitely scalable data storage infrastructure at very low costs. (https://aws.amazon.com/tw/s3/)

Core AWS Services for Deep Learning (2/2)

- AWS Lambda: A serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. (https://aws.amazon.com/tw/lambda/)
- Amazon SageMaker: A fully managed service that provides every developer and data scientist with the ability to build, train, and deploy machine learning models quickly. (https://aws.amazon.com/tw/sagemaker/)

AWS for Machine Learning and Deep Learning

- Deep Learning AMIs: Amazon Machine Images (AMIs) designed for deep learning. Pre-installed with popular frameworks like TensorFlow, PyTorch, Apache MXNet, and others.
- Deep Learning Containers: AWS offers Docker images pre-installed with deep learning frameworks for use on AWS services like Amazon ECS and AWS Fargate.
- Elastic Inference: Allows you to attach low-cost inference acceleration to Amazon EC2 and SageMaker instances or ECS tasks, reducing the cost of inference without compromising performance.

Thank you