주요 개념

1. 활성화 함수 (Activation Function)

신경망의 각 뉴런에서 입력 신호의 총합을 비선형 출력 신호로 변환하는 함수입니다. 비선형성을 도입하여 신경망이 복잡한 패턴을 학습할 수 있도록 합니다.

- **ReLU (Rectified Linear Unit)**: `max(0, x)`. 구현이 간단하고 계산 비용이 적으며, 기울기 소실(Vanishing Gradient) 문제를 완화합니다. 가장 널리 사용됩니다.
- **Sigmoid**: `1 / (1 + exp(-x))`. 0과 1 사이의 값을 출력하여 이진 분류의 출력층이나 게이트 메커니즘에 사용됩니다. 기울기 소실 문제가 있습니다.
- **Tanh (Hyperbolic Tangent)**: `(exp(x) - exp(-x)) / (exp(x) + exp(-x))`. -1과 1 사이의 값을 출력하며, Sigmoid보다 중심이 0에 가까워 학습에 유리합니다. 기울기 소실 문제가 있습니다.
- **Softmax**: 다중 분류 문제의 출력층에서 사용되며, 각 클래스에 속할 확률 분포를 출력합니다. 모든 출력 값의 합은 1이 됩니다.

2. 손실 함수 (Loss Function)

모델의 예측 값과 실제 값 사이의 오차를 측정하는 함수입니다. 모델 학습 시 이 손실 함수 값을 최소화하는 방향으로 가중치를 업데이트합니다.

- **MSE (Mean Squared Error)**: 회귀 문제에서 주로 사용되며, 예측 값과 실제 값 차이의 제곱 평균입니다.
- **MAE (Mean Absolute Error)**: 회귀 문제에서 사용되며, 예측 값과 실제 값 차이의 절댓값 평균입니다.
- **Binary Cross-Entropy**: 이중 분류 문제에서 사용됩니다.
- **Categorical Cross-Entropy**: 다중 분류 문제에서 사용되며, 원-핫 인코딩된 타겟 레이블에 사용됩니다.
- **Sparse Categorical Cross-Entropy**: 다중 분류 문제에서 사용되며, 정수 인코딩된 타겟 레이블에 사용됩니다.

3. 옵티마이저 (Optimizer)

손실 함수를 최소화하기 위해 모델의 가중치(Weight)와 편향(Bias)을 업데이트하는 알고리즘입니다.

- **SGD (Stochastic Gradient Descent)**: 가장 기본적인 옵티마이저로, 배치(Batch) 단위로 기울기를 계산하여 가중치를 업데이트합니다.
- **Adam (Adaptive Moment Estimation)**: 학습률을 자동으로 조절하며, 모멘텀(Momentum)과 RMSprop의 장점을 결합하여 빠르고 안정적인 수렴을 돕습니다. 가장 널리 사용됩니다.
- **RMSprop (Root Mean Square Propagation)**: 학습률을 적응적으로 조절하여 진동을 줄이고 수렴 속도를 높입니다.

4. 규제 (Regularization)

모델의 과적합을 방지하고 일반화 성능을 향상시키기 위한 기법입니다.

- **L1 규제 (Lasso)**: 가중치의 절댓값 합에 비례하는 항을 손실 함수에 추가합니다. 일부 가중치를 0으로 만들어 피처 선택 효과가 있습니다.
- **L2 규제 (Ridge, Weight Decay)**: 가중치의 제곱 합에 비례하는 항을 손실 함수에 추가합니다. 가중치 값을 작게 만들어 과적합을 방지합니다.

5. Dropout

신경망 학습 시 무작위로 일부 뉴런을 비활성화(드롭아웃)시켜 모델의 과적합을 방지하는 기법입니다. 각 학습 단계마다 다른 신경망 구조를 사용하는 효과를 내어 앙상블 학습과 유사한 효과를 얻습니다.

6. 파인튜닝 관련 개념

사전 학습된 모델(Pre-trained Model)을 기반으로, 새로운 작업(Task)에 맞게 모델을 미세 조정하는 기법입니다. 대규모 데이터셋으로 학습된 모델의 가중치를 가져와 우리가 가진 더 작은 데이터셋에 맞게 조정함으로써, 학습 시간을 단축하고 성능을 향상시킬 수 있습니다.

- **전이 학습 (Transfer Learning)**: 특정 도메인에서 학습된 지식을 다른 관련 도메인에 적용하는 광범위한 개념입니다. 파인튜닝은 전이 학습의 한 종류입니다.
- **특징 추출 (Feature Extraction)**: 사전 학습된 모델의 일부(주로 합성곱 기반)를 특징 추출기로 사용하고, 그 위에 새로운 분류기(Classifier)를 추가하여 학습합니다. 이 과정에서는 사전 학습된 모델의 가중치는 고정(Freeze)합니다.
- **파인튜닝 (Fine-tuning)**: 특징 추출과 유사하지만, 사전 학습된 모델의 가중치 일부 또는 전체를 새로운 데이터에 맞게 재학습(미세 조정)합니다. 일반적으로 낮은 학습률(Learning Rate)을 사용하여 기존 가중치를 크게 변경하지 않도록 합니다.

각종 기법 예제 코드

PyTorch

```
import torch
import torch.nn as nn
from torchvision import models

# 1. 활성화 함수
relu = nn.ReLU()
sigmoid = nn.Sigmoid()
tanh = nn.Tanh()
softmax = nn.Softmax(dim=1)

# 2. 손실 함수
mse_loss = nn.MSELoss()
mae_loss = nn.L1Loss()
binary_cross_entropy = nn.BCELoss()
categorical_cross_entropy = nn.CrossEntropyLoss() # Softmax 포함

# 3. 옵티마이저
# model.parameters()는 예시이며, 실제 모델의 파라미터를 전달해야 합니다.
# model = YourModel()
# optimizer_sgd = torch.optim.SGD(model.parameters(), lr=0.01)
# optimizer_adam = torch.optim.Adam(model.parameters(), lr=0.001)
# optimizer_rmsprop = torch.optim.RMSprop(model.parameters(), lr=0.001)

# 4. 규제 (옵티마이저에 weight_decay 파라미터로 L2 규제 적용)
# optimizer_adam_l2 = torch.optim.Adam(model.parameters(), lr=0.001,
weight_decay=1e-5)
```

```
# 5. Dropout
dropout = nn.Dropout(p=0.5)

# 6. 파인튜닝 (예: ResNet18 모델 로드 및 수정)
# 사전 학습된 ResNet18 모델 로드
model_ft = models.resnet18(pretrained=True)

# 모든 파라미터를 고정 (가중치 동결)
for param in model_ft.parameters():
    param.requires_grad = False

# 마지막 분류기(fc layer)를 새로운 작업에 맞게 교체
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 10) # 10개 클래스로 분류하는 작업으로 가정

# 교체한 레이어의 파라미터만 학습하도록 설정
# optimizer_ft = torch.optim.Adam(model_ft.fc.parameters(), lr=0.001)
```

Keras

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.applications import ResNet50

# 1. 활성화 함수
relu = layers.Activation('relu')
sigmoid = layers.Activation('sigmoid')
tanh = layers.Activation('tanh')
softmax = layers.Activation('softmax')

# 2. 손실 함수 (compile 시 문자열로 지정)
# model.compile(loss='mean_squared_error')
# model.compile(loss='mean_absolute_error')
# model.compile(loss='binary_crossentropy')
# model.compile(loss='categorical_crossentropy')
# model.compile(loss='sparse_categorical_crossentropy')

# 3. 옵티마이저 (compile 시 문자열 또는 객체로 지정)
# model.compile(optimizer='sgd')
# model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001))
# model.compile(optimizer='rmsprop')

# 4. 규제 (레이어에 kernel_regularizer 인자로 L1/L2 규제 적용)
l1_reg = regularizers.l1(0.01)
l2_reg = regularizers.l2(0.01)
# dense_layer = layers.Dense(64, activation='relu', kernel_regularizer=l2_reg)

# 5. Dropout
dropout_layer = layers.Dropout(0.5)
```

```
# 6. 파인튜닝 (예: ResNet50 모델 로드 및 수정)
# 사전 학습된 ResNet50 모델 로드 (include_top=False로 분류기 제외)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,
224, 3))

# 베이스 모델의 가중치 동결
base_model.trainable = False

# 새로운 분류기 추가
inputs = keras.Input(shape=(224, 224, 3))
x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(10, activation='softmax')(x) # 10개 클래스로 분류
model_ft = keras.Model(inputs, outputs)

# 새로운 분류기만 학습하도록 컴파일
# model_ft.compile(optimizer=keras.optimizers.Adam(),
#                  loss='categorical_crossentropy',
#                  metrics=['accuracy'])

# (선택적) 일부 레이어의 동결 해제 후 미세 조정
# base_model.trainable = True
# for layer in base_model.layers[:-10]: # 마지막 10개 레이어를 제외하고 동결
#     layer.trainable = False
#
# model_ft.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-5), # 낮은 학
습률 사용
#                  loss='categorical_crossentropy',
#                  metrics=['accuracy'])
```