제공해주신 PDF 파일들의 내용을 바탕으로, 개념 설명은 최소화하고 실제 구현 코드**(Python/NumPy)** 위주로 핵심을 정리했습니다.

신경망 구현 핵심 코드 정리 **(5장 & 6장)**

1. 역전파를 위한 기본 계층 **(Layers)** 구현

오차역전파법을 위해 각 노드(계층)는 forward(순전파)와 backward(역전파) 메서드를 가집니다.

1.1 곱셈 및 덧셈 계층 **(**단순 연산**)**

Python

```
[cite_start]# [cite: 961, 970] 곱셈 계층
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        return x * y

    def backward(self, dout):
        dx = dout * self.y  # x와 y를 바꾼 값을 곱함
        dy = dout * self.x
        return dx, dy

[cite_start]# [cite: 1014, 1020] 덧셈 계층
class AddLayer:
    def forward(self, x, y):
        return x + y

    def backward(self, dout):
        dx = dout * 1
```

```
        dy = dout * 1
        return dx, dy
```

1.2 활성화 함수 계층 (ReLU, Sigmoid)

Python

```
[cite_start]# [cite: 767, 773, 1340, 1348] ReLU: 0 이하는 0, 0 초과는 그대로
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0
        return out

    def backward(self, dout):
        dout[self.mask] = 0 # 순전파 때 0 이하였던 곳은 역전파 때도 0
        dx = dout
        return dx

[cite_start]# [cite: 793, 799, 1441] Sigmoid: y = 1 / (1 + exp(-x))
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = 1 / (1 + np.exp(-x))
        self.out = out
        return out

    def backward(self, dout):
        # 역전파 공식: dout * y * (1 - y)
        dx = dout * (1.0 - self.out) * self.out
        return dx
```

1.3 Affine (행렬 내적) 및 Softmax 계층

Python

```
[cite_start]# [cite: 810, 822, 1445, 1450] Affine: 행렬의 곱 (Batch 처리 대응)
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.x = x
        out = np.dot(x, self.W) + self.b
        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0) # 배치 단위의 합
        return dx

[cite_start]# [cite: 874, 894, 1465] SoftmaxWithLoss: 출력 정규화 + 교차 엔트로피 오차
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y = None # softmax 출력
        self.t = None # 정답 레이블

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        # 역전파 공식: (출력 - 정답) / 배치크기
        dx = (self.y - self.t) / batch_size
        return dx
```

---

2. 신경망 전체 구조 (TwoLayerNet) 구현

계층을 조립하여 전체 네트워크를 구성합니다. OrderedDict를 사용하여 계층 순서를 관리합니다.

Python

```
[cite_start]# [cite: 912, 913, 1096]
from collections import OrderedDict

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 1. 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

        # 2. 계층 생성 (순서 중요)
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

        self.lastLayer = SoftmaxWithLoss()

    [cite_start]# [cite: 436, 914] 순전파 (추론)
    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x

    [cite_start]# [cite: 914] 손실 함수 값 계산
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)
```

```
    [cite_start]# [cite: 439, 919] 오차역전파를 통한 기울기 계산 (핵심)
    def gradient(self, x, t):
        # 1. 순전파
        self.loss(x, t)

        # 2. 역전파
        dout = 1
        dout = self.lastLayer.backward(dout) # 마지막 층

        layers = list(self.layers.values())
        layers.reverse() # 계층 순서 반대로
        for layer in layers:
            dout = layer.backward(dout)

        # 3. 결과 저장
        grads = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        return grads
```

---

3. 매개변수 갱신 (Optimizer) 구현

구해진 기울기(grads)를 이용해 가중치(params)를 갱신하는 알고리즘들입니다.[1]

3.1 SGD (확률적 경사 하강법)[2]

Python

```
[cite_start]# [cite: 280, 290] 단순한 기울기 갱신
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
```

```
        for key in params.keys():
            params[key] -= self.lr * grads[key]
```

3.2 M³omentum (관성)

Python

```
[cite_start]# [cite: 366, 372] 이전 속도(v)를 누적하여 갱신
class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum * self.v[key] - self.lr * grads[key]
            params[key] += self.v[key]
```

3.3 AdaGrad (학습률 감소)

Python

```
[cite_start]# [cite: 402] 많이 갱신된 변수는 학습률을 줄임 (h에 기울기 제곱 누적)
class AdaGrad:
    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

    def update(self, params, grads):
        if self.h is None:
            self.h = {}
```

```
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

        for key in params.keys():
            self.h[key] += grads[key] * grads[key]
            # 1e-7은 0으로 나누는 것 방지
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

3.4 Adam[4]

Python

```
[cite_start]# [cite: 406] Momentum + AdaGrad. 코드 생략 (복잡도 고려),
# 핵심 파라미터: beta1(일차 모멘텀), beta2(이차 모멘텀)
```

---

4. 학습 최적화 및 오버피팅 억제 코드

4.1 가중치 초기화 (Weight Initialization)

초기값 설정에 따라 학습 속도와 수렴 여부가 결정됩니다.

Python

```
[cite_start]# [cite: 597, 636, 1117]
node_num = 100 # 앞 층의 노드 수

# 1. 표준편차 0.01 (비권장)
w = 0.01 * np.random.randn(node_num, node_num)

# 2. Xavier 초깃값 (Sigmoid, Tanh 사용 시 권장)
w = np.random.randn(node_num, node_num) / np.sqrt(node_num)

# [cite_start]3. He 초깃값 (ReLU 사용 시 권장) [cite: 661]
w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)
```

4.2 드롭아웃 (Dropout) 구현

Python

```
[cite_start]# [cite: 205] 훈련 시 랜덤하게 뉴런 삭제, 시험 시에는 비율 곱해서 출력
class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg=True):
        if train_flg:
            # mask 생성: 삭제할 뉴런은 False
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask
```

4.3 하이퍼파라미터 최적화 (Random Search)

Python

```
[cite_start]# [cite: 228, 230] 대략적인 범위 설정 후 랜덤 샘플링
weight_decay = 10 ** np.random.uniform(-8, -4)
lr = 10 ** np.random.uniform(-6, -2)

# 위 파라미터로 학습 수행 및 검증 데이터(val) 정확도 확인
# val_acc_list = train(lr, weight_decay)
```

출처
1. https://amber-chaeeunk.tistory.com/21
2. http://everyday-deeplearning.tistory.com/category/PYTHON%EC%9C%BC%EB%A1%9C%20%EB%94%A5%EB%9F%AC%EB%8B%9D%ED%95%98%EA%B8%B0
3. https://hul980.tistory.com/23

4. https://hul980.tistory.com/23
5. https://hul980.tistory.com/23
6. https://hul980.tistory.com/23