# 深層学習day1-2　レポート

深層学習day1　確認テスト

1.
入力と出力の間に中間層を設けて、それぞれのパーセプトロンの重みに特徴を学習させることで、モデルを作成しようとしている。

重み　バイアス

2.
（省）

3.
入力層の一つ一つが動物の特徴になる。

4.
u = np.dot(W,x) + b

5.
```
# 2層の総出力
z2 = functions.relu(u2)
```

6.
入力と出力が比例関係にない方が、非線形。

7.
```
functions.relu(u1)
```

8.
二乗することで、誤差の正負を排して評価できる。
1/2は微分した際に形を単純にするための係数である。

9.
```
def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x -np.max(x, axis=0)
        y = np.exp(x) /np.sum(np.exp(x), axis=0)
    return y.T
    x = x -np.max(x) #オーバーフロー対策
    return np.exp(x) / np.sum(np.exp(x))
```

10.
```
def cross_entropy_error(d, y):
    ify.ndim== 1:
        d= d.reshape(1, d.size)
        y= y.reshape(1, y.size)
    # 教師データがone-hot-vectorの場合、正解ラベルのインデックスに変換
    if d.size==y.size:
        d= d.argmax(axis=1)
        batch_size= y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), d] + 1e-7)) / batch_size
```

11.
```
network[key]  -= learning_rate * grad[key]
```

12.
逐次来るデータに対して学習し、モデルを更新していく手法。

13.
重み-誤差関数のグラフに対して、最小値を目指すため、現在地点の重みに対する偏微分の値から関数の傾きを導き、誤差の小さい重みの方へ、学習率の分だけ重みを更新している。

14.
```
delta1 = np.dot(delta2, W2.T) * functions.d_sigmoid(z1)
```

15.
```
delta2 = functions.d_mean_squared_error(d, y)


grad['W2'] = np.dot(z1.T, delta2)
```

# 深層学習day1　要点まとめ

・Section 1　入力層～中間層
　入力層から中間層への変換は特に変わったことはなく、特徴量に対して重みとバイアスを用いて線形に変換を行う。一般にはそれによって得られた出力を活性化関数によって特徴を分かりやすくする処理がおこなわれる。中間層が多いほど説明力が増す。

・Section 2　活性化関数
　活性化関数にも種類があり、得意不得意がある。ロジスティック回帰にはよくシグモイド関数が用いられていたが、NNのように重みの数が多くなってくると処理が重くなったり、学習の効率が悪くなったりすることから、代わりにReLU関数などが使われる。

・Section 3　出力層
　出力層で出た値は、誤差の評価を含め、推論結果は人間が扱う。中間層とは違って、人間が見て正確に理解しやすいデータとするために、ソフトマックス関数のような、確率として返す活性化関数が用いられる場合がある。

・Section 4　勾配降下法
　重みやバイアスの値を更新するために勾配降下法を用いる。大きく分けると、偏微分をして傾きを出すステップと、学習率分値を更新するステップがあるが、それぞれ学習するデータの特性に合わせて工夫される。たとえば、学習の数が多い場合、偏微分をまとめて行わず、バッチごとに行う。また、誤差の最小値にうまくたどり着かない場合は、学習率や、学習の仕方のアルゴリズムに手を加える。

・Section 5　誤差逆伝播法
　NNのように層を重ねるようになると、出力から遠い層の重みを更新する際に工夫が必要になる。具体的には勾配を逆伝播して再帰的な処理を行わずに重みを更新する。この際に、層が厚くなればなるほど、勾配の値が小さくなっていく、勾配消失問題が発生するため、特に活性化関数を工夫する場合が大い。

# 深層学習day2　確認テスト

1.
dz/dx = 2

2.
(2)

3.
順伝播の際にその重みに関する特徴が無視され、結果的に学習されなくなる。

4.
特徴量の大きさを揃えられるので、学習の際に特徴量による有利不利が出ない。
勾配学習の際に、極端な爆発や消失をある程度抑えられる。

5.
モメンタム：勾配学習に加速度の概念を取り入れていて、前のステップの学習の方向に進みやすい。

AdaGrad：徐々に更新の幅が小さくなっていくので、緩やかな傾きの場合最小値にたどり着きやすい。一方、鞍点問題も起こりやすい。

RMSprop：AdaGradを改良したもので、急速な学習率の低下を抑えられる。

6.
右

7.
７×７

# 深層学習day2　要点まとめ

・Section 1　勾配消失問題
　誤差逆伝播をして重みを学習していく際、層が厚くなるほど、誤差が伝播しにくくなる。そのため、誤差関数の偏微分の値が小さくなりすぎないように、ロジスティック回帰などのようにシグモイド関数を使わず、ReLU関数がよく使われる。また、重みがそもそも小さいと、逆伝播する際にそれより前のノードの学習がうまくいかないこともあるので、重みの初期値を工夫する必要がある。

・Section 2　学習率最適化手法
　学習率は一般的には誤差関数の傾きにかけて重みを更新する時に利用し、常に一定であることが多いが、実際に適用すると、学習がうまく収束しないことや、極小値に収束してしまうことがある。それに対し、モメンタムでは、学習率に加速度の概念を導入し、収束を早めたり、極小値を超えられるように工夫されている。AdaGradやRMSpropは最小値付近での収束をよくし、Adamではその両方の利点を生かせるようなアルゴリズムである。

・Section 3　過学習
　過学習とは、準備した訓練用データに特化しすぎたモデルを作ってしまうことによって、汎化性能がおちてしまうことのことである。過学習であることを判定する明確なラインはないが、パラメーターの更新に対して、テスト誤差と訓練誤差の差が開き出したらその可能性がある。これを避けるために、重みによるモデルの表現力を抑えられるように正則化項を誤差関数に導入したり、ノード数をランダムに落としながら学習させる方法がある。

・Section 4　畳み込みニューラルネットワークの概念
　CNNというと、画像の学習に用いられるイメージだったが、実際には、１次元データから３次元データまで、様々な活用が考えられる。層の構成を大きく分けると、コンボリューション層とプーリング層に分かれており、最終的に出力層にて、得たい推論を得る。画像であれば、距離が近いピクセルのデータとの関係が特徴に影響することは明らかであり、そのような特徴をすくいだせるような工夫がなされているアルゴリズムである。

・Section 5　最新のCNN
　AlexNetの説明があった。ILSVRCにて飛躍的な成績を残した学習手法である。3つのコンボリューション層と２つのプーリング層からなる。画像の水増し技術や、ドロップアウトも取り入れられていたそうだ。しかし、翌年には成績は抜かれて、今ではあまり使われていないという。