E資格試験対策講座修了テスト

合計点 87/106 点

こちらはAidemy Premium Plan E資格試験対策講座の修了テストです。
問題は合計で106問の4択です。64問以上正解で合格です。
前のセクションに戻ることはできません。よく見直してから次のセクションに進んでください。

E資格試験の試験時間は120分となっていますが、このテスト上では時間は測定していません。
ご自身の復習の意味も込めて、時間を気にせずに入念に進めていただいても問題ありません。

メールアドレス *

jinlin.007@hotmail.com

0 / 0 ポイント

氏名（受験申し込みコードの発行申請の際に参照するため、フルネームで入力してください） *

林　勁

氏名（フリガナ） *

リン　ケイ

所属企業・学校名（受験申し込みコードの発行申請の際に参照いたします） *

大和証券

現在、学生の方はチェックを入れてください

☐ 学生

受講者番号 *

i1121

1 基礎数学　6 / 7 ポイント

線形代数

1.1 以下の行列**A**を固有値分解してみましょう

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 2 & -1 \\ 0 & 0 & 3 \end{pmatrix}$$

✓ 1.1.1 まず、固有値を計算してみると、λ=(a), 2, 3の3つが求まります。 * 1/1

- ○ (a) = -1
- ◉ (a) = 1 ✓
- ○ (a) = 1/2
- ○ (a) = 4

✕ 1.1.2 λ=2についての固有ベクトルは以下になります。 * 0/1

$$t\begin{pmatrix} 1 \\ (b) \\ 0 \end{pmatrix}$$

- ○ (b) = 0
- ◉ (b) = -1 ✕
- ○ (b) = 2
- ○ (b) = 1

正解

- ◉ (b) = 1

1.2 次の行列**A**を特異値分解してみましょう

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

✓ 1.2.1 以下の固有値を計算するとλ=(c), 1を求めることができます。 * 1/1

$$AA^T = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

- ○ (c) = -1
- ○ (c) = -2
- ◉ (c) = 2 ✓
- ○ (c) = 1/2

✓ 1.2.2 そして、Aの左特異ベクトルとして以下のUを求めることができます。 * 1/1

$$U = \begin{pmatrix} 0 & (d) \\ 1 & 0 \end{pmatrix}$$

- (d) = 1 ✓
- (d) = -1
- (d) = 2
- (d) = 1/2

最終的に、特異値分解の結果の一つとして以下のように分解できます。

$$A = \begin{pmatrix} 0 & (d) \\ 1 & 0 \end{pmatrix} \begin{pmatrix} (e) & 0 & 0 \\ 0 & (f) & 0 \end{pmatrix} \begin{pmatrix} 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \\ 1 & 0 & 0 \\ 0 & \sqrt{\frac{1}{2}} & (g) \end{pmatrix}$$

✓ 1.2.3 上記の(e)に当てはまる数値を選択肢から選んでください。 * 1/1

$$A = \begin{pmatrix} 0 & (d) \\ 1 & 0 \end{pmatrix} \begin{pmatrix} (e) & 0 & 0 \\ 0 & (f) & 0 \end{pmatrix} \begin{pmatrix} 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \\ 1 & 0 & 0 \\ 0 & \sqrt{\frac{1}{2}} & (g) \end{pmatrix}$$

- (e) = 1
- (e) = -1
- (e) = 1/2
- (e) = √2 ✓

✓ 1.2.4 上記の(f)に当てはまる数値を選択肢から選んでください。 * 1/1

$$A=\begin{pmatrix}0 & (d)\\ 1 & 0\end{pmatrix}\begin{pmatrix}(e) & 0 & 0\\ 0 & (f) & 0\end{pmatrix}\begin{pmatrix}0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}}\\ 1 & 0 & 0\\ 0 & \sqrt{\frac{1}{2}} & (g)\end{pmatrix}$$

- (f) = 1 ✓
- (f) = -1
- (f) = 1/2
- (f) = √2

✓ 1.2.5 上記の(g)に当てはまる数値を選択肢から選んでください。 * 1/1

$$A=\begin{pmatrix}0 & (d)\\ 1 & 0\end{pmatrix}\begin{pmatrix}(e) & 0 & 0\\ 0 & (f) & 0\end{pmatrix}\begin{pmatrix}0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}}\\ 1 & 0 & 0\\ 0 & \sqrt{\frac{1}{2}} & (g)\end{pmatrix}$$

- (g) = 1
- (g) = -1
- (g) = - √(1/2) ✓
- (g) = √2

2 応用数学 10 / 12 ポイント

確率・統計
情報理論

2.1 確率

✓ 2.1.1 離散変数Xに対する確率分布P(x)の分散はVar[X] = (a)となります。 * 1/1

- (a) = E[x]^2-E[x^2]
- (a) = E[x^2]-E[x]^2 ✓
- (a) = E[f(x)]^2-E[f(x)^2]
- (a) = E[f(x)^2]-E[f(x)]^2

✓ 2.1.2 関数f(x)の離散的な確率分布P(x)のもとでの期待値はE[f] = (b)となります。 * 1/1

- (b) = ∫P(x)f(x)
- (b) = ∫P(x)/f(x)
- (b) = ∑P(x)f(x) ✓
- (b) = ∑P(x)/f(x)

✓ 2.1.3 yが起こったもとでxが起こる条件付き確率を以下のように定義することができます。 * 1/1

$$p(x \mid y) = \frac{(c)}{p(y)}$$

- (c) = P(x)
- (c) = P(x, y) ✓
- (c) = P(xy)
- (c) = P(y | x)

✓ 2.1.4 この条件付き確率よりベイズの定理を導出することができます。 * 1/1

$$p(x \mid y) = \frac{(d)}{p(y)} = \frac{(d)}{\int p(x, y)dx}$$

- (d) = P(y | x)P(y)
- (d) = P(y | x)
- (d) = P(x)
- (d) = P(y | x)P(x) ✓

2.2 ベルヌーイ分布

0と1の2つの事象があると仮定します。1の起きる確率をP(X=1) = θと定義します。

✓ 2.2.1 すると、余事象より0の起きる確率はP(X=0) = (e)となります。 * 1/1

- (e) = θ
- (e) = - θ
- (e) = θ - 1
- (e) = 1 - θ ✓

✓ 2.2.2 よって、これを連続的に行うベルヌーイ分布の生起確率を以下のように定義することができます。 * 1/1

$$P(x \mid \theta) = (f)^x (g)^{1-x}$$

- (f) = θ, (g) = 1 - θ ✓
- (f) = 1 - θ, (g) = θ
- (f) = θ, (g) = θ
- (f) = 1 - θ, (g) = 1 - θ

✓ 2.2.3 すると、ベルヌーイ分布の期待値はE[x] = (h)となります。 * 1/1

- (h) = θ ✓
- (h) = 1 - θ
- (h) = θ^2
- (h) = √θ

✓ 2.2.4 すると、ベルヌーイ分布の分散VarはVar(x) = (i)となります。 * 1/1

- (i) = (1 - θ)^2
- (i) = 1 - θ
- (i) = θ
- (i) = θ(1 - θ) ✓

2.3 マルチヌーイ分布

✗ 2.3.1 k個の異なる状態をとるマルチヌーイ分布のパラメータはp ∈ (j)と定義することができます。 * 0/1

- (j) = [0, k]^k
- (j) = [0, 1]^k ✗
- (j) = [0, 1]^(k - 1)
- (j) = [0, k]^(k - 1)

正解

- (j) = [0, 1]^(k - 1)

2.4 情報理論

✗ 2.4.1 自己情報量はI(x) = (k)と定義することができます。 * 0/1

- (k) = -logx
- (k) = logP(x) ✗
- (k) = -logP(x)
- (k) = 2^P(x)

正解

- (k) = -logP(x)

✓ 2.4.2 エントロピーはH[P(x)] = (I)と定義することができます。 * 1/1

- (I) = - ∫P(x)logP(x)dx ✓
- (I) = ∫P(x)logP(x)dx
- (I) = - P(x)logP(x)
- (I) = P(x)logP(x)

✓ 2.4.3 KLダイバージェンスはKL[q(x)][p(x)] = (m)と定義することができます。 * 1/1

- (m) = - ∫q(x)log{p(x)/q(x)} ✓
- (m) = - ∫q(x)log{p(x)q(x)}
- (m) = - ∫p(x)/q(x)
- (m) = - ∫p(x)q(x)

3 機械学習アルゴリズム 15 / 18 ポイント

機械学習の基礎
実用的な方法論

3.1 学習アルゴリズム

✓ 3.1.1 機械学習をしていく上で学習が不足して適切な学習ができていない場合を(a)といいます。 * 1/1

- (a) = アンサンブル学習
- (a) = 過少適合 ✓
- (a) = 過剰適合
- (a) = 転移学習

✓ 3.1.2 機械学習をしていく上で学習をしすぎて適切な学習ができていない場合を(b)といいます。 * 1/1

- (b) = アンサンブル学習
- (b) = 過少適合
- (b) = 過剰適合 ✓
- (b) = 転移学習

コンピュータプログラムは、**(c)**で測定される**(d)**における性能が**(e)**により改善される場合、その**(d)**のクラス及び**(c)**に関して**(e)**が学習する。

✓ 3.1.3 上記の機械学習アルゴリズムの定義より(c)に当てはまる選択肢を選んでください。 * 1/1

- (c) = 経験E
- (c) = 性能指標P ✓
- (c) = タスクT
- (c) = 精度A

✓ 3.1.4 上記の機械学習アルゴリズムの定義より(d)に当てはまる選択肢を選んでください。 * 1/1

- (d) = 経験E
- (d) = 性能指標P
- (d) = タスクT ✓
- (d) = 精度A

✓ 3.1.5 上記の機械学習アルゴリズムの定義より(e)に当てはまる選択肢を選んでください。 * 1/1

- (e) = 経験E ✓
- (e) = 性能指標P
- (e) = タスクT
- (e) = 精度A

✗ 3.1.6 「コスト関数の極値を探索するあらゆるアルゴリズムは、全ての可能なコスト関数に適用した結果を平均すると同じ性能となる」という定理を(f)といいます。 * 0/1

- (f) = ハップス・ギュルダンの定理 ✗
- (f) = ノーフリーランチ定理
- (f) = 中心極限定理
- (f) = ド・モルガンの定理

正解

- (f) = ノーフリーランチ定理

✓ 3.1.7 最尤推定において、未知のデータ生成分布P_data(x)から生成された集合Xとパラメトリックな以下の集合P_model(x; θ)があったとき、θの推定量はθ_ML = (g)となります。 * 1/1

- (g) = maxP_model(X; θ)
- (g) = argmaxP_model(X; θ) ✓
- (g) = minP_model(X; θ)
- (g) = argminP_model(X; θ)

✓ 3.1.8 この分布間の誤差をKLダイバージェンスで測定すると以下のようになります。 * 1/1

$$D_{KL}(\hat{P}_{data} \,||\, P_{model}) = E_{x\ \hat{p}_{data}}[log\hat{P}_{data}(x) - (h)]$$

- (h) = logP_model(x) ✓
- (h) = logP_data(x)
- (h) = P_model(x)
- (h) = P_data(x)

✓ 3.1.9 データをk個の重複しない集合に分割し、そのうちの一つをテストデータ、残りを訓練データとして訓練・精度計算をおこない、これをk回繰り返して平均を取ることで精度評価をおこなう手法を(i)といいます。 * 1/1

- ホールドアウト法
- PCA法
- サブマリン投法
- k-分割交差検証法 ✓

3.2 ロジスティック回帰

✓ 3.2.1 ロジスティック回帰の出力に使われるシグモイド関数の実装は以下になります。 * 1/1

```
1  import numpy as np
2
3
4  def sigmoid(x):
5      return 1 / (j)
6
```

- (j) = (1 - np.exp(x))
- (j) = (1 + np.exp(x))
- (j) = (1 - np.exp(-x))
- (j) = (1 + np.exp(-x)) ✓

3.3 サポートベクターマシン

✓ 3.3.1 2クラス分類が可能な重みwとバイアスbの条件は以下になります。 * 1/1

$$(\mathbf{w}^T\mathbf{x}^{(\mathbf{N})} + b)y^{(N)} \geqq (k)$$

- (k) = 1 ✓
- (k) = -1
- (k) = 0
- (k) = ±1

✓ 3.3.2 その上で、以下の最適化問題を解くことでマージンを最大化することができます。 * 1/1

$$\frac{1}{2}||(\mathbf{l})||^2$$

- (l) = b
- (l) = w ✓
- (l) = x
- (l) = y

✓ 3.3.3 非線形SVMで用いられるrbfカーネル関数は以下のように定義することができます。 * 1/1

$$K(\mathbf{x^{(i)}}, \mathbf{x^{(j)}}) = exp(-\frac{||\mathbf{x^{(i)}} - \mathbf{x^{(j)}}||}{(m)})$$

- (m) = σ
- (m) = 2σ
- (m) = σ^2
- (m) = 2σ^2 ✓

3.4 勾配降下法

✓ 3.4.1 勾配降下法によって重みを更新していく関数learning() を実装してください。 * 1/1

```python
import numpy as np


def learning(self, X, y):
    self.eta = 0.03
    self.epoch = 100

    # Xに合わせた重みを生成
    rgen = np.random.RandomState(42)
    # バイアス項w_0x_0も生成するため size は 1+X.shape[1]にしている
    self.w_ = rgen.normal(size=1 + X.shape[1])
    # コストを格納する空リストを用意
    self.cost_ = []

    # 学習をepoch回繰り返す
    for i in range(self.epoch):
        u = self.calculate_net_input(X)
        output = self.sigmoid(u)
        errors = (y - output)
        self.w_[1:] += (n)
        self.w_[0] += self.eta * errors.sum()
        # 損失関数の計算
        cost = np.dot(-y, np.log(output)) - np.dot((1 - y), np.log(1 - output))
        # この回でのコストを格納
        self.cost_.append(cost)
    return self

```

- ○ (n) = self.eta * np.dot(X, errors)
- ○ (n) = self.eta * np.linarg(X, errors)
- ◉ (n) = self.eta * np.dot(errors, X) ✓
- ○ (n) = self.eta * np.dot(errors, X.T)

3.5 主成分分析

✗ 3.5.1 主成分分析の有名なPCA法のアルゴリズムに当てはまらないものを以下の選択肢から選んでください。 * 0/1

- ○ データを標準化する
- ◉ k個の固有ベクトルから特徴変換行列Wを生成する ✗
- ○ 特徴変換行列Wを転置する
- ○ 特徴同士の相関行列を計算する

正解

- ◉ 特徴変換行列Wを転置する

3.6 k-meansクラスタリング

✓ 3.6.1 k-means法のアルゴリズムについて当てはまらないものを以下の選択肢から選んでください。 * 1/1

- ○ セントロイドから最も近い重心を持つ群に各入力データを割り当てる
- ○ 全入力データに対してkの中心座標を再計算する
- ○ 入力データ空間にランダムにk個の点を配置する
- ◉ 均等に離れた位置になるまでセントロイドをランダムに生成する ✓

3.7 最近傍法, k近傍法

✓ 3.7.1 k近傍法は、機械学習を全くおこなっていないので(o)と呼ばれる。 * 1/1

- ○ 自然学習
- ○ 人間学習
- ◉ 怠惰学習 ✓
- ○ 無学習

3.8 深層学習の発展を促す課題

✗ 3.8.1 次の説明のうち間違っているものを選択肢から選んでください。 * 0/1

- ○ データの次元が増えると、格子状のセルで訓練データを説明するのが難しくなるため、機械学習が難しくなる。
- ○ 単純なアルゴリズムの多くは良好な汎化のために平滑化事前分布または局所一様分布に依存している。
- ○ カーネルマシンのカーネルの重要なカーネルは局所カーネル群であり、ここでk(u, v)はu=vのときに小さな値となり、uとvが互いに離れて増大していくにつれて増大する。
- ◉ モデルを非線形に変えた場合、ほとんどのコスト関数が閉形式では最適化できなくなり、その場合勾配降下法のような繰り返しの数値的な最適化処理を選択する必要がある。 ✗

正解

- ◉ カーネルマシンのカーネルの重要なカーネルは局所カーネル群であり、ここでk(u, v)はu=vのときに小さな値となり、uとvが互いに離れて増大していくにつれて増大する。

4 深層学習 27 / 33 ポイント

順伝播型ネットワーク
深層モデルのための正則化
深層モデルのための最適化
生成モデル

4.1 順伝播型ネットワーク

✓ 4.1.1 活性化関数ReLUを実装してください。 * 1/1

```
import numpy as np


def relu(x):
    return (a)

```

- (a) = np.max(0, x) ✓
- (a) = np.sum(x)
- (a) = 1 / 1 + np.exp(-x)
- (a) = np.tanh(x)

✓ 4.1.2 分類モデルのコスト関数として用いられることの多いクロスエントロピー誤差を実装してください。 * 1/1

```
import numpy as np


def cross_entropy_error(y_pred, y_true):
    error = 1e-7
    loss = (b)
    return loss

```

- (b) = - np.sum(y_pred * np.log(y_true + error))
- (b) = - np.sum(y_true * np.log(y_pred + error)) ✓
- (b) = np.sum(y_pred * np.log(y_true + error))
- (b) = np.sum(y_true * np.log(y_pred + error))

✓ 4.1.3 分類モデルの出力に用いられることの多いソフトマックス関数を実装してください。 * 1/1

```
1  import numpy as np
2
3
4  def softmax(x):
5      return (c)
6
```

- (c) = np.exp(x) / x
- (c) = np.exp(x) / np.sum(np.exp(x)) ✓
- (c) = np.exp(x) * np.sum(np.exp(x))
- (c) = np.sum(np.exp(x)) / np.exp(x)

✗ 4.1.4 出力ユニットに線形ユニットを用いることで(d)を防ぎ、勾配に基づく最適化手法を適切に使うことができます。 * 0/1

- (d) = 最大化
- (d) = 不足
- (d) = 消失 ✗
- (d) = 飽和

正解

- (d) = 飽和

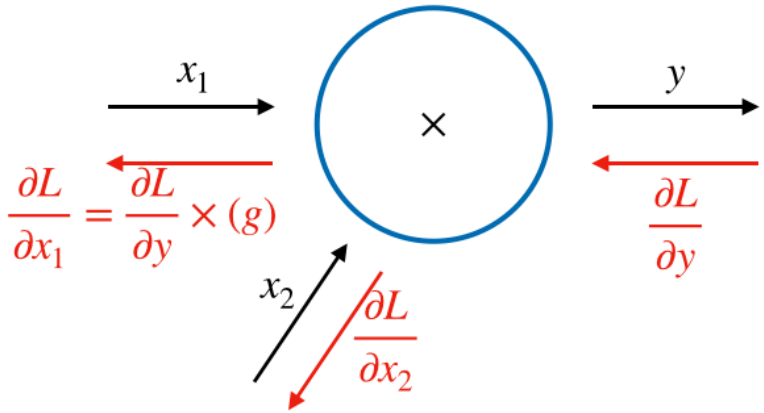
✓ 4.1.5 シグモイド関数をσ(x)とするとσ(0)=(e)となります。 1/1

- (e) = 0.5 ✓
- (e) = 0
- (e) = 1
- (e) = -∞

✓ 4.1.6 マルチヌーイ分布の出力ユニットにソフトマックスユニットを用いることで合計が(f)となり、各ラベルの確率を出力することができます。 * 1/1

- (f) = 1 ✓
- (f) = 0
- (f) = -1
- (f) = ∞

✓ 4.1.7 スカラー積のオペレーションを計算グラフで表すと以下のようになります。青印は順伝播、赤印は逆伝播を示しています。 * 1/1



- (g) = x_1
- (g) = x_2 ✓
- (g) = L
- (g) = y

✓ 4.1.8 MLPの誤差逆伝播法を実装すると以下のようになります。 * 1/1

```python
import numpy as np


def sigmoid(u):...


def softmax(u):...


def forward(x):...


x = np.array([[1, 0.5]])
y, z1 = forward(x)


# 誤差逆伝播法
def back_propagation(x, z1, y, d):
    w1 = np.array([[0.3, 0.5], [0.4, 0.6]])
    w2 = np.array([[0.3, 0.5], [0.4, 0.6]])

    # 順伝播型ネットワークで計算した出力と教師データとの誤差を計算してください。
    error_range = y - d
    grad_w2 = z1.T.dot(error_range)

    # 活性化関数の微分
    sigmoid_diff = z1 * (1 - z1)
    delta = error_range.dot(w2.T) * sigmoid_diff
    grad_w1 = x.T.dot(delta)

    w2 -= (h) * grad_w2
    w1 -= (h) * grad_w1
    return w1, w2

```

- (h) = learning_rate ✓
- (h) = delta
- (h) = np.sum(x)
- (h) = np.max(x)

4.2 深層モデルのための正則化

✓ 4.2.1 L2正則化の実装は以下のようになります。 * 1/1

```python
import numpy as np


def ridge_(X, y, alpha):
    # np.eye()で単位行列を作っています。
    # X.TはXの転置を表しています。
    # alphaはパラメータです。
    C = (i)
    # np.linalg.inv()は逆行列を求めています、
    return np.linalg.inv(C).dot(X.T.dot(y))

```

- (i) = X.dot(X.T) + alpha * np.eye(X.shape[1])
- (i) = X.T.dot(X) + alpha * np.eye(X.shape[1]) ✓
- (i) = X.dot(X.T) + alpha * np.eye(X.shape[0])
- (i) = X.T.dot(X) + alpha * np.eye(X.shape[0])

✓ 4.2.2 ペアのデータに加えて予測対象が存在しない予測材料のみのデータからも学習をおこなうことを(j)といいます。 * 1/1

- (j) = ダミー学習
- (j) = アンサンブル学習
- (j) = 半教師あり学習 ✓
- (j) = 睡眠学習

✗ 4.2.3 値の多くを0にし、計算を高速化させることを(k)といいます。 * 0/1

- (k) = スパース表現
- (k) = スパム表現
- (k) = one-hotベクトル表現 ✗
- (k) = 近似表現

正解

- (k) = スパース表現

✓ 4.2.4 いくつかのモデルを組み合わせることによって汎化誤差を減少させ、予測結果のバリアンスを低くする手法を(l)といいます。 * 1/1

- (l) = バギング ✓
- (l) = ブースティング
- (l) = クリーピング
- (l) = ネスティング

✓ 4.2.5 前の学習結果を次の学習に使用する手法を(m)といいます。 * 1/1

- (m) = バギング
- (m) = ブースティング ✓
- (m) = クリーピング
- (m) = ネスティング

✓ 4.2.6 ドロップアウト法を実装すると以下のようになります。 * 1/1

```
import numpy as np


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:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return (n)
        else:
            return x * (1.0 - self.dropout_ratio)

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

```

- (n) = x * self.mask ✓
- (n) = x / self.mask
- (n) = self.mask
- (n) = np.exp(x) * self.mask

✗ 4.2.7 L1・L2正則化について誤っている文を以下の選択肢から選んでください。 * 0/1

- ○ L1正則化は次元削減に使われる。
- ◉ L1正則化は外れデータを0にする手法である。 ✗
- ○ L2正則化はデータを滑らかにする手法である。
- ○ L2正則化は次元削減に使われる。

正解

- ◉ L2正則化は次元削減に使われる。

4.3 深層モデルのための最適化

✓ 4.3.1 経験損失最小化を以下のように定義することができます。 * 1/1

$$\mathbb{E}_{x,y}[L(f(x;\theta),y)] = (o)\sum_{i=1}^{m} L(f(x^{(i)};\theta),y^{(i)})$$

- ○ (o) = 1
- ○ (o) = m
- ○ (o) = θ
- ◉ (o) = 1/m ✓

✗ 4.3.2 ランダムに1~n個のバッチごとに学習をする手法を(p)といいます。 * 0/1

- ○ (p) = バッチ学習
- ◉ (p) = ランダムバッチ学習 ✗
- ○ (p) = ミニバッチ学習
- ○ (p) = オンライン学習

正解

- ◉ (p) = ミニバッチ学習

✓ 4.3.3 ニューラルネットワークの重みが大きすぎると、(q)という現象が起こります。 * 1/1

- ( ) (q) = 勾配消失
- ( ) (q) = 重み消失
- (●) (q) = 勾配爆発 ✓
- ( ) (q) = 重み爆発

✓ 4.3.4 確率的勾配降下法の実装は以下のようになります。 * 1/1

```
class SGD:
    # 初期化
    def __init__(self, learning_rate=0.01):  # learning_rate: 学習率
        self.learning_rate = learning_rate

    # 更新
    def update(self, params, grads):  # params:重みパラメータ grads:勾配
        for key in params.keys():
            params[key] -= (r)

```

- ( ) (r) = grads[key]
- ( ) (r) = params[key] * grads[key]
- ( ) (r) = self.learning_rate * params[key]
- (●) (r) = self.learning_rate * grads[key] ✓

✓ 4.3.5 モメンタム法を実装すると以下のようになります。 * 1/1

```
import numpy as np


class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    # 更新
    def update(self, params, grads):
        # vになにもない時、配列を0で初期化
        if self.v is None:
            self.v = {}
            for key, value in params.items():
                self.v[key] = np.zeros_like(value)

        # 数式の実装
        for key in params.keys():
            self.v[key] = (s)
            params[key] += self.v[key]

```

- ( ) (s) = self.v[key] - self.lr * grads[key]
- ( ) (s) = self.v[key] - * grads[key]
- (●) (s) = self.momentum * self.v[key] - self.lr * grads[key] ✓
- ( ) (s) = self.momentum * self.v[key] + self.lr * grads[key]

✓ 4.3.6 AdaGrad法を実装すると以下のようになります。 * 1/1

```python
import numpy as np


class AdaGrad:
    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

    # 更新
    def update(self, params, grads):

        # hがNoneなら0で初期化
        if self.h is None:
            self.h = {}
            for key, value in params.items():
                self.h[key] = np.zeros_like(value)

        # 計算
        for key in params.keys():
            self.h[key] += grads[key] * grads[key]
            params[key] -= (t)

```

- ( ) (t) = self.lr * grads[key] / self.h[key]
- ( ) (t) = self.lr * grads[key] / np.exp(self.h[key] + 1e-7)
- ( ) (t) = self.lr * grads[key] / np.tanh(self.h[key] + 1e-7)
- (•) (t) = self.lr * grads[key] / np.sqrt(self.h[key] + 1e-7) ✓

✓ 4.3.7 RMSProp法を実装すると以下のようになります。 * 1/1

```python
import numpy as np


# 実装
class RMSprop:
    # 初期化
    def __init__(self, lr=0.01, decay_rate=0.99):
        self.lr = lr
        self.decay_rate = decay_rate
        self.h = None

    # 更新
    def update(self, params, grads):
        # hがNoneの場合0で初期化
        if self.h is None:
            self.h = {}
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

        # 計算
        for key in params.keys():
            self.h[key] *= self.decay_rate
            self.h[key] += (1 - self.decay_rate) * grads[key] * grads[key]
            params[key] -= (u)

```

- (•) (u) = self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7) ✓
- ( ) (u) = self.lr * grads[key] * (np.sqrt(self.h[key]) + 1e-7)
- ( ) (u) = self.lr * grads[key] / (np.exp(self.h[key]) + 1e-7)
- ( ) (u) = self.lr * grads[key] / (self.h[key] + 1e-7)

✗ 4.3.8 Adam法を実装すると以下のようになります。 * 0/1

```python
import numpy as np


class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    # 更新
    def update(self, params, grads):
        # mとvを0で初期化
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t = [redacted]

        # 計算
        for key in params.keys():
            # self.m[key] = self.beta1*self.m[key] + (1-self.beta1)*grads[key]
            # self.v[key] = self.beta2*self.v[key] + (1-self.beta2)*(grads[key]**2)
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key] ** 2 - self.v[key])

            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

```

- (v) = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) / (1.0 - self.beta1 ** self.iter)
- (v) = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) * (1.0 - self.beta1 ** self.iter) ✗
- (v) = self.lr * np.sqrt(1.0 - self.beta2 * self.iter) / (1.0 - self.beta1 * self.iter)
- (v) = self.lr * np.sqrt(1.0 - self.beta2 * self.iter) * (1.0 - self.beta1 * self.iter)

正解

- (v) = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) / (1.0 - self.beta1 ** self.iter)

✓ 4.3.9 バッチ正則化を実装すると以下のようになります。 * 1/1

```python
import numpy as np


# 入力を作成
np.random.rand(42)

# 5行10列
data = np.random.rand(5, 10)

# 10行5列
data_t = data.T

minibatch = []

# この係数にグローバル学習率が乗算されて、層のオフセットの学習率が決定されます。
# たとえば、OffsetLearnRateFactor が 2 の場合、層のオフセットの学習率は現在のグローバル学習率の 2 倍になります。
offset_learn_rate_factor = 1

# 上記と同様に、この係数にグローバル学習率が乗算されて、層のオフセットの学習率が決定されます。
# たとえば、scale_learn_rate_factor が 2 の場合、層のオフセットの学習率は現在のグローバル学習率の 2 倍になります。
scale_learn_rate_factor = 1

for i in range(data_t.shape[0]):
    # ミニバッチ平均を計算
    batch_mean = data_t[i].mean()
    dispersion = 0
    for j in range(data_t.shape[1]):
        # ミニバッチ分散を計算
        dispersion += (data_t[i][j] - batch_mean) ** 2
        if j == 4:
            dispersion = dispersion / data_t.shape[1]
    # バッチ正規化を計算
    # 1e-8は、ミニバッチの分散に加算する定数
    batch_n = [redacted]
            [redacted]
    minibatch.append(batch_n)

```

- (w) = offset_learn_rate_factor * (data_t[i][j] - batch_mean) / np.sqrt(dispersion + 1e-8) + scale_learn_rate_factor ✓
- (w) = offset_learn_rate_factor * (data_t[i][j] - batch_mean) / np.exp(dispersion + 1e-8) + scale_learn_rate_factor
- (w) = offset_learn_rate_factor * (data_t[i][j] - batch_mean) / dispersion + scale_learn_rate_factor
- (w) = offset_learn_rate_factor * (data_t[i][j] - batch_mean) / np.tanh(dispersion + 1e-8) + scale_learn_rate_factor

✓ 4.3.10 ニュートン法について説明している文から間違っているものを選択してください。 * 1/1

- 二次でない表面についてはヘッセ行列が正定値である限り、ニュートン法を反復的に適用できる。
- イパン的に近似的関数の表面は鞍点のような多くの特徴を持つ非凸面であり、それがニュートン法による更新の妨げとなる。
- 勾配方向dを求め、f(x - αd)を最小にするαを求めることで解を近似することができる。
- ◉ ニュートン法は1次の勾配のみしか考慮されていない。 ✓

4.4 生成モデル

✓ 4.4.1 出力データをそのまま入力データとして再現するニューラルネットを(x)といいます。 * 1/1

- ◉ (x) = オートエンコーダー ✓
- (x) = オートデコーダー
- (x) = GoogLeNet
- (x) = SegNet

✓ 4.4.2 訓練データから新しいデータを生成するネットワークと与えられたデータが訓練データであるか生成データであるかを判別するネットワークの2つを交互に学習させていくようなモデルを(y)といいます。 * 1/1

- ◉ (y) = Generative Adversarial Nets(GAN) ✓
- (y) = Support Vector Machine(SVM)
- (y) = Residual Network
- (y) = Restricted Boltzmann Machine(RBM)

✓ 4.4.3 (y)は学習が進み、生成器と識別器のそれぞれが理想的な性能を有するようになった場合、生成器の生成するデータ分布と訓練データの分布が一致するため、D(x)=(z)に収束します。 * 1/1

- (z) = 1
- ◉ (z) = 1/2 ✓
- (z) = 0
- (z) = 1/3

✓ 4.4.4 (y)の価値関数を以下のように定義することができます。 * 1/1

$$(aa)V(G,D) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))]$$

- (aa) = min_G min_D
- (aa) = max_G min_D
- (aa) = min_G max_D ✓
- (aa) = max_G max_D

✗ 4.4.5 (y)の手法にRNNを適用し、さらにプーリング層を畳み込みで書き換えているDCGANでは生成器の各層にReLU, 識別器の全層に以下で定義されるLeaky ReLUを適用しています。 * 0/1

$$f = \begin{cases} x(x > 0) \\ (ab)(x \leq 0) \end{cases}$$

- (ab) = e^x - 1
- (ab) = ax ✗
- (ab) = log(1 + x)
- (ab) = 0.01x

正解

- (ab) = 0.01x

✓ 4.4.6 識別モデルは識別関数と異なり(ac)というメリットがあります。 * 1/1

- (ac) = モデルの出力の解釈や学習とは別に確率に基づいた識別境界を設定する ✓
- (ac) = 入力データの生成過程やクラスごとのデータ分布などを知ることができる
- (ac) = 入力データ空間に識別超平面を描くことができ、入力データ空間内のクラス分布が直感的に把握できる
- (ac) = 数学的な操作がたやすく、データ量が少ない場合でも機能する

✓ 4.4.7 生成モデルは識別モデルと異なり(ad)というメリットがあります。 * 1/1

- (ad) = モデルの出力の解釈や学習とは別に確率に基づいた識別境界を設定する
- (ad) = 入力データの生成過程やクラスごとのデータ分布などを知ることができる ✓
- (ad) = 入力データ空間に識別超平面を描くことができ、入力データ空間内のクラス分布が直感的に把握できる
- (ad) = 数学的な操作がたやすく、データ量が少ない場合でも機能する

✓ 4.4.8 VAEにおける隠れ変数zの近似として適切なものを次の選択肢から選んでください。なおN(0, 1)は、平均値0、分散値1の正規分布、σ^2は分散値、μは平均値とする。 * 1/1

- z = εσ (ε~N(0,I))
- z = μ
- z = μ+εσ (ε~N(0,I)) ✓
- z = σ

5 畳み込みネットワーク(CNN) 14 / 17 ポイント

畳み込み処理
プーリング
構造出力
データの種類
効率的な畳み込みアルゴリズム
ランダムあるいは教師なし特徴量
画像認識の有名なモデル(VGG, AlexNet, GoogLeNet, ResNet)
特徴量の転移
画像の局在化、検知、セグメンテーション

5.1 畳み込みとプーリング

✓ 5.1.1 画像サイズW×W, フィルタサイズH×H, パディングP, ストライドSの畳み込みをした場合、画像のサイズは以下のようになります。 * 1/1

$$\left(\left\lfloor\frac{(a)}{S}\right\rfloor+1\right)\times\left(\left\lfloor\frac{(a)}{S}\right\rfloor+1\right)$$

- (a) = W - H - P
- (a) = W + H - 2P
- (a) = W - H - 2P
- (a) = W - H + 2P ✓

✓ 5.2.1 2012年、はじめてDeep Learningの手法をとり、ご認識率を大幅に下げたモデルを(b)といいます。 * 1/1

- (b) = AlexNet ✓
- (b) = VGGNet
- (b) = GoogLeNet
- (b) = ResNet

✓ 5.2.2 (b)はすべての活性化関数に(c)を採用しています。 * 1/1

- (c) = ステップ関数
- (c) = シグモイド関数
- (c) = ReLU関数 ✓
- (c) = tanh関数

✓ 5.2.3 (b)のプーリング層では(d)が使用されています。 * 1/1

- (d) = maxプーリング ✓
- (d) = meanプーリング
- (d) = zeroプーリング
- (d) = minプーリング

✓ 5.2.4 2014年に開発された、19層まで重ねた一般的なCNNをおこなっているモデルを(e)といいます。 * 1/1

- (e) = AlexNet
- (e) = VGGNet ✓
- (e) = GoogLeNet
- (b) = ResNet

✓ 5.2.5 (e)では、3×3のフィルタによる(f)を連続して行っています。 * 1/1

- (f) = 畳み込み層 ✓
- (f) = プーリング層
- (f) = 標準化層
- (f) = 全結合層

✓ 5.2.6 2014年に開発された、ネットワークを縦方向だけでなく横方向にまで拡張したモデルを(g)といいます。 * 1/1

- (g) = AlexNet
- (g) = VGGNet
- (g) = GoogLeNet ✓
- (g) = ResNet

✗ 5.2.7 (g)で用いられているInception moduleと呼ばれる複数のフィルタ群により構成されたブロックに関する記述として誤っているものを次の選択肢から選んでください。 * 0/1

- Inception moduleは通常のK×Kの畳み込みフィルタと比較すると非零のパラメータが増えているとみなせるため、相対的にdenseな演算であると言える。
- Inception moduleは大きな畳み込みフィルタを小さな畳み込みフィルタのグループで近似することで、モデルの表現力とパラメータ数のトレードオフを改善していると言える。
- Inception moduleには1×1の畳み込みフィルタが使われているが、このフィルタは次元削減と等価な効果がある。 ✗
- Inception moduleは小さなネットワークを一つのモジュールとして定義し、モジュールの積み重ねでネットワークを構築する。

正解

- Inception moduleは通常のK×Kの畳み込みフィルタと比較すると非零のパラメータが増えているとみなせるため、相対的にdenseな演算であると言える。

✓ 5.2.8 (g)で用いられているGlobal Average Pooling(GAP)では、全結合層を利用しないことでパラメータ数を(h)し、過学習を防ぐことができます。 * 1/1

- (h) = 増幅
- (h) = 拡張
- (h) = 削減 ✓
- (h) = 水増し

✓ 5.2.9 (g)で用いられているAuxililary Lossで、中間層に直接誤差を伝播させることによって(i)を防ぐとともにネットワークの正則化の実現が可能となりました。 * 1/1

- (i) = 過学習
- (i) = 勾配消失 ✓
- (i) = 勾配爆発
- (i) = 計算量の増加

✓ 5.2.10 2015年に開発された、VGGをベースとしてスキップ構造を取り入れたモデルを(j)といいます。 * 1/1

- (j) = AlexNet
- (j) = VGGNet
- (j) = GoogLeNet
- (j) = ResNet ✓

✓ 5.2.11 (j)では、層をまたがる結合として(k)を用いることで勾配消失問題を解消しています。 * 1/1

- (k) = max pooling
- (k) = Identity mapping ✓
- (k) = Global Average Pooling(GAP)
- (k) = Global mapping

5.3 物体検出

✓ 5.3.1 R-CNNでは(I)という検索アルゴリズムを用いて画像から物体候補を探し出し、CNNにかけて特徴マップを出力し、SVMによって回帰分析をおこない領域を決定します。 * 1/1

- (I) = Random Search
- (I) = Grid Search
- (I) = Binaly Search
- (I) = Selective Search ✓

✗ 5.3.2 Faster R-CNNの説明として誤っているものを選択肢から選んでください。 * 0/1

- Fast R-CNNとFaster R-CNNの違いの一つに矩形領域回帰の誤差関数が挙げられる。
- Faster R-CNNでは提案領域に対してCNNで特徴マップを生成し、RoI Poolingという手法で固定サイズに変形する。 ✗
- Faster R-CNNではRegion Proposal Networkと呼ばれるCNNで物体の候補領域を探し出す。
- Faster R-CNNはend-to-end学習が可能なネットワークアーキテクチャである。

正解

- Fast R-CNNとFaster R-CNNの違いの一つに矩形領域回帰の誤差関数が挙げられる。

✓ 5.3.3 一般物体検出アルゴリズムのその他のアルゴリズムについて以下に述べた4つの記述内から正しいものを一つ選んでください。 * 1/1

- YOLOと呼ばれるネットワークアーキテクチャは検出速度は高速だが、画像内のオブジェクト同士が複数重なっている場合にうまく検出できないという欠点がある。 ✓
- YOLOでは画像をある固定長で分割したセルごとに最大2つの候補領域が推定される。
- Single Shot MultiBox Detectorと呼ばれるネットワークモデルの誤差はオブジェクト領域の位置特定誤差と各クラスの確信度に対する確信度誤差の2つの誤差の単純和で表現される。
- YOLOでは画像をある固定長で分解したせるごとにそれぞれどのカテゴリの物体なのかもしくはただの背景なのか、のそれぞれの確率が出力される。

5.4 セマンティックセグメンテーション

✓ 5.4.1 FCN以前は(m)により、固定サイズの画像しか扱うことができませんでした。 * 1/1

- (m) = ネットワーク内にMax Pooling層が存在すること。
- (m) = ネットワーク内に全結合層が存在すること。 ✓
- (m) = ネットワーク内に畳み込み層が存在すること。
- (m) = ハードウェアの性能限界

✕ 5.4.2 FCNによる出力ユニット数は(n)となります。 * 0/1

- (n) = 画像サイズ×画像サイズ ✕
- (n) = 画像サイズ
- (n) = 分類クラス数
- (n) = 画像サイズ×分類クラス数

正解

- (n) = 画像サイズ×分類クラス数

6 回帰結合型ニューラルネットワークと再帰的ネットワーク(RNN) 9 / 13 ポイント

回帰結合型のニューラルネットワーク
双方向RNN
Encoder-DecoderとSequence-to-Sequence
深層回帰結合型のネットワーク
再帰型ニューラルネットワーク
長期依存性の課題
エコーステートネットワーク
複数時間スケールのためのLeakyユニットとその他の手法
ゲート付きRNN
長期依存性の最適化
自然言語処理とRNN
メモリネットワーク

6.1 回帰結合型のニューラルネットワーク

✕ 6.1.1 RNNの隠れ層の順伝播はいかのように実装することができます。 * 0/1

```
class Output2hidden(object):
    def __init__(self, n_input, n_hidden, n_output):
        self.n_input = n_input
        self.n_hidden = n_hidden
        self.n_output = n_output
        self.hidden_weight = np.random.randn(n_hidden, n_input + 1)
        self.output_weight = np.random.randn(n_output, n_hidden + 1)
        self.recurrent_weight = np.random.randn(n_hidden, n_output + 1)

    # 中略
    def forward(self, x, y):
        r = self.recurrent_weight.dot(np.hstack((1.0, y)))
        h = sigmoid(self.hidden_weight.dot(np.hstack((1.0, x))) + r)
        y = np.tanh(self.output_weight.dot(np.hstack((1.0, h))))
        return h, y

    def forward_seq_test(self, X):
        y = np.zeros(self.n_output)
        hs, ys = ([], [])
        for x in X:
            h, y = self.forward(x, y)
            hs.append(h)
            ys.append(y)
        return hs, ys

    def forward_seq_training(self, X, T):
        T = (a)
        hs, ys = ([], [])
        for i in range(X.shape[0]):
            h, y = self.forward(X[i], T[i])
            hs.append(h)
            ys.append(y)
        return hs, ys

```

- (a) T = np.zeros(self.n_output) ✕
- (a) T = np.concatenate((np.zeros(self.n_output).reshape(1,-1),T), axis=0)
- (a) T = T[1:]
- (a) y = np.zeros(self.n_output)

正解

- (a) T = np.concatenate((np.zeros(self.n_output).reshape(1,-1),T), axis=0)

6.2 双方向RNN

✗ 6.2.1 双方向RNNを説明したもので間違っているものを選択してください。 * 0/1

- (●) 全方向出力に依存する出力を得たい場合に使用される。 ✗
- ( ) 系列の開始から同じ向きに移動するRNNと、系列の終わりから時間とは逆方向に移動するRNNを組み合わせたものである。
- ( ) RNNを4つ設けることで画像のような二次元データにも拡張可能である。
- ( ) 双方向にRNNを組み合わせることによって長期のデータにも適応可能となった。

正解

- (●) 双方向にRNNを組み合わせることによって長期のデータにも適応可能となった。

6.3 Encoder-Decoder

✓ 6.3.1 Encoder-Decoderの説明として間違っているものを以下の選択肢から選んでください。 * 1/1

- ( ) Encoderと呼ばれるRNNが入力を処理し、文脈Cを出力する。
- ( ) Decoderは出力系列Yを生成するために固定長のベクトルによって条件付けられている。
- (●) 入力と出力配列の長さは同じでなければならない。 ✓
- ( ) すべてのデータのペアについてlogP(Y)の平均を最大化するよう2つのRNNが同時に学習を進める。

6.4 長期依存性の課題

✓ 6.4.1 RNNは、学習を進めていくと重みWの固有値がt乗されていくため、固有値が1に満たない場合は(b)に収束し、1より大きい場合は発散してしまうという問題があります。 * 1/1

- (●) (b) = 0 ✓
- ( ) (b) = 1
- ( ) (b) = 1/2
- ( ) (b) = 1/√2

6.5 エコーステートネットワーク(ESN)

✓ 6.5.1 エコースティックネットワークの説明として間違っているものを選択肢から選んでください。 * 1/1

- 回帰隠れユニットが過去の入力の履歴をうまく補足できるようにしている。
- 重みをスペクトル半径に固定することによって勾配が爆発しないようにした。
- ESNに重みを設定することによって重みを初期化でき、長期依存性の学習に役立っている。
- ◉ ESNでは逆に短期での学習が難しくなっている。 ✓

6.6 ゲート付きRNN

✓ 6.6.1 従来のRNNにあった勾配消失問題をCECとゲートという概念を導入することで解決したモデルがLSTMです。LSTMのゲートの内、適切でないものを以下の選択肢から選んでください。 * 1/1

- 忘却ゲート
- 入力ゲート
- ◉ 変換ゲート ✓
- 出力ゲート

✓ 6.6.2 LSTMの各時刻での中間層の処理は以下のようになります。 * 1/1

```
import numpy as np

n_input = 100
n_hidden = 256

w = (c)
b = np.zeros(n_hidden * 4)


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def lstm(x, h, c):
    inputs = np.concatenate((x, h), axis=1)
    inputs = np.matmul(inputs, w) + b
    z, i, f, o = np.hsplit(inputs, 4)

    input_gate = sigmoid(i)
    forget_gate = sigmoid(f)
    output_gate = sigmoid(o)

```

- (c) = np.random.randn(n_input * n_hidden, n_hidden * 3)
- (c) = np.random.randn(n_input * n_hidden, n_hidden * 4)
- (c) = np.random.randn(n_input + n_hidden, n_hidden * 3)
- ◉ (c) = np.random.randn(n_input + n_hidden, n_hidden * 4) ✓

✓ 6.6.3 LSTMの中間層の出力は以下のように計算することができます。 * 1/1

```
31    next_c = (d)
32    next_h = np.tanh(next_c) * output_gate
33
34    return next_h, next_c
```

- (d) = c * forget_gate + np.tanh(z) * input_gate ✓
- (d) = c * input_gate + np.tanh(z) * forget_gate
- (d) = c * forget_gate + z * input_gate
- (d) = c * input_gate + z * forget_gate

✗ 6.6.4 GRUの説明として間違っているものを以下の選択肢から選んでください。 * 0/1

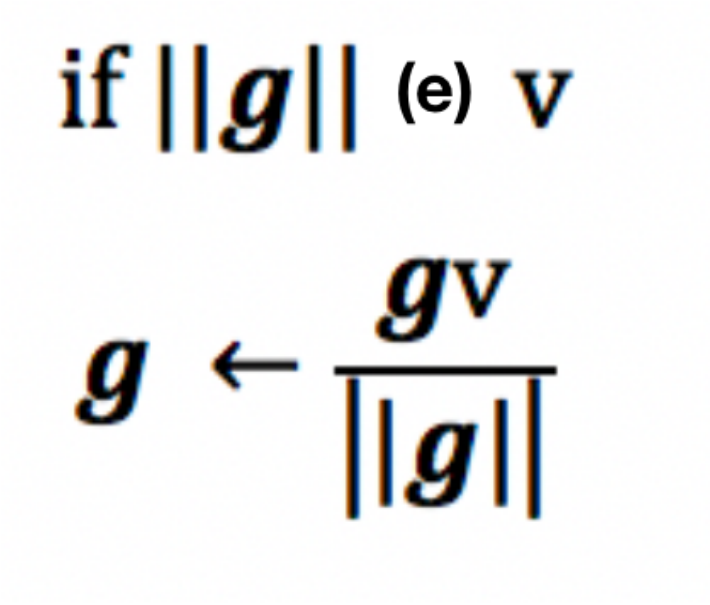
- LSTMをベースに「リセットゲート」と呼ばれるゲートを導入し、隠れ状態の重みを操作する。
- 実装する際はLSTMと同様にmodel.add(GRU())とすればよい。
- LSTM同様、隠れ状態hと履歴Cの2つの時間方向へ伝播する必要がある。
- LSTMをベースに「更新ゲート」と呼ばれるゲートを導入し、忘却と出力の重みを操作する。 ✗

正解

- LSTM同様、隠れ状態hと履歴Cの2つの時間方向へ伝播する必要がある。

6.7 長期依存性の最適化

✓ 6.7.1 勾配をg, 閾値をvとおくと勾配クリッピングは以下のように定義することができます。 * 1/1



- (e) <
- (e) > ✓
- (e) ≤
- (e) ≥

✓ 6.7.2 勾配クリッピングの実装は以下のようになります。 * 1/1

```python
import numpy as np


def clip_gradients(grads, max_norm):
    norm = 0
    for grad in grads:
        norm += np.sum(grad ** 2)
    norm = np.sqrt(norm)

    if norm > max_norm:
        clip = max_norm / (norm + 1e-7)
        for grad in grads:
            grad (f)

```

- (f) += clip
- (f) *= clip ✓
- (f) -= clip
- (f) /= clip

6.8 自然言語処理とRNN

✗ 6.8.1 自然言語処理にニューラルネットワークを使うことによる利点として適切なものを以下の選択肢から選んでください。 * 0/1

- (●) 単語を平等に扱える点 ✗
- ( ) 単語の意味を扱える点
- ( ) 単語のベクトルの次元がとても大きくなる点
- ( ) 単語を定量的に表現できる点

正解

- (●) 単語の意味を扱える点

6.9 メモリネットワーク

✓ 6.9.1 Attentionの説明として間違っているものを以下の選択肢から選んでください * 1/1

- ( ) 参照元の系列sのどこに注意するかを対象の系列tの各時刻において計算することで臨機応変に参照元の系列の情報を考慮しながら対象の系列の情報を抽出することができる。
- ( ) 双方向のRNNであっても適用可能である。
- ( ) sの隠れ状態ベクトルの重み付き平均を用いるSoft Attentionやランダムに1つの隠れ状態ベクトルを選択するHard Attentionなどがある。
- (●) 自然言語処理のみにおいて使われる技術である。 ✓

7 強化学習 6 / 6 ポイント

方策勾配法
価値反復法

7.1 方策勾配法

✓ 7.1.1 ベルマン方程式に従って行動価値関数Q(s, a)をモデル化する価値反復に基づく方法には欠点があります。その問題を解決する方法として開発された方策勾配の定理より、目的関数J(θ)=V(s_0)に関する勾配を以下のように定義することができます。 * 1/1

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[(a)]$$

- (●) (a) = ∇_θ log π_θ(a | s)Q^π(s, a) ✓
- ( ) (a) = ∇_θ π_θ(a | s)Q^π(s, a)
- ( ) (a) = ∇_θ log π_θ(a | s) / Q^π(s, a)
- ( ) (a) = ∇_θ π_θ(a | s) / Q^π(s, a)

7.2 価値反復法(DQN)

✓ 7.2.1 DQNでの改善点について誤っている点を次の選択肢から選んでください。 * 1/1

- ○ 報酬、TD誤差のクリッピング
- ○ Experienced Replay
- ○ Fixed Target Q-network
- ◉ 方策の最適化 ✓

✓ 7.2.2 Q学習による損失関数は以下のように定義することができます。 * 1/1

$$L(\theta) = E[(r + \gamma \ \textbf{(b)}_{a'} \ Q(s', a'; \theta) - Q(s, a; \theta))^2]$$

- ◉ (b) = max ✓
- ○ (b) = min
- ○ (b) = sign
- ○ (b) = lim

✓ 7.2.3 上の損失に(c)を適用すると期待値Eが最適化され、以下のような損失関数になります。 * 1/1

$$L(\theta) = E_{s,a,r,s' \ D}[(r + \gamma \ \textbf{(b)}_{a'} \ Q(s', a'; \theta) - Q(s, a; \theta))^2]$$

- ◉ (c) = Experienced Replay ✓
- ○ (c) = Target Q-network
- ○ (c) = 報酬のクリッピング
- ○ (c) = REINFORCEアルゴリズム

✓ 7.2.4 上の損失に(d)を適用するとθが最適化され、以下のような損失関数になります。 * 1/1

$$L(\theta) = E_{s,a,r,s' \ D}[(r + \gamma \ \textbf{(b)}_{a'} \ Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$

- ○ (d) = Experienced Replay
- ◉ (d) = Target Q-network ✓
- ○ (d) = 報酬のクリッピング
- ○ (d) = REINFORCEアルゴリズム

✓ 7.2.5 報酬の値を[-1, 0, 1]の3値に制限することで学習を安定させる手法を(e)といいます。 * 1/1

- ○ (e) = Experienced Replay
- ○ (e) = Target Q-network
- ◉ (e) = 報酬のクリッピング ✓
- ○ (e) = REINFORCEアルゴリズム

終了

0 / 0 ポイント

以上で修了テストは終了です。お疲れ様でした。
64点以上にて合格です。

24時間以内に、ご入力いただいたメールアドレスに結果通知が送付されます。
結果通知メールが受信できない場合、メールアドレスが誤っている可能性があります。その場合は、受験申し込みコードの発行申請ができませんので、必ずお申し付けください。
（結果通知メールは、迷惑メールフォルダなどに振り分けられる可能性があります。）