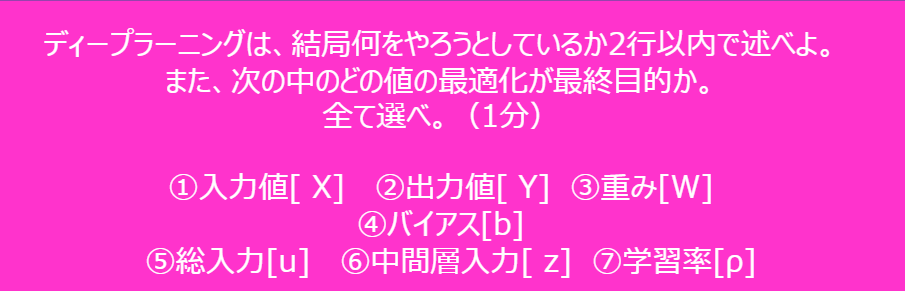
**まとめ要点１：**

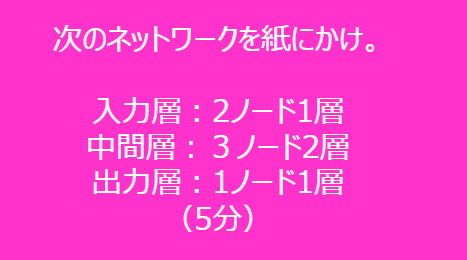
　入力層～中関層

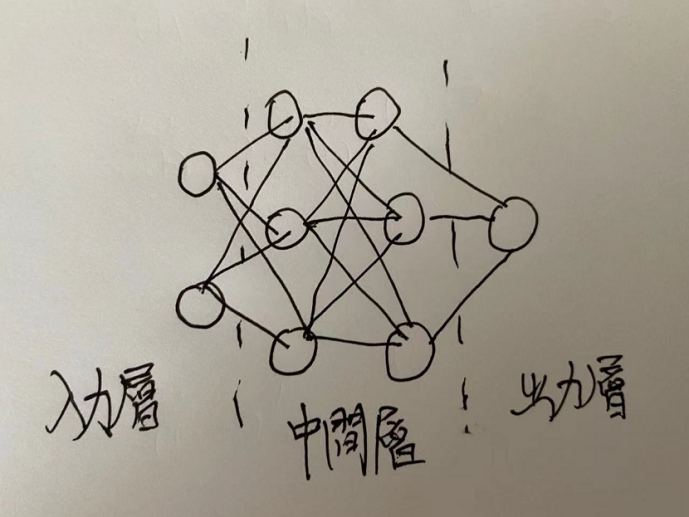
入力値層はデータが初めてNWに入力する時、入れたNW層である。一般的にはデータの種類により変わる可能性が高い、例えば人間の情報を入力する時、身長、ひげの本数、耳の大きさなどの情報を一つずつ受け取って、計算して中関層に転送する。入力層と出力層の中、全ての層は中間層と呼びます。

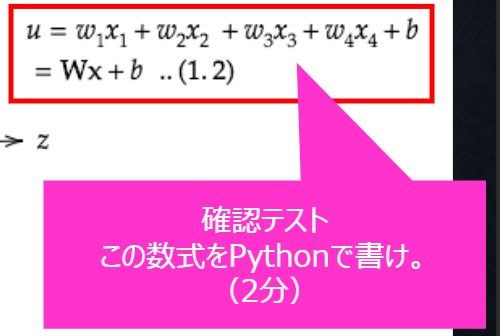
**確認テスト：**

****

**答え：**③と④、重みとバイアスなどのパラメータを最適化した時点、モデルはトレニンーグ完了とする。



**答え：**



**答え：u1 = np.dot(x, W1) + b1**

**実装演習中1625977642(1)**

**実装演習確認：**

****

****

**サマリーと考察：**

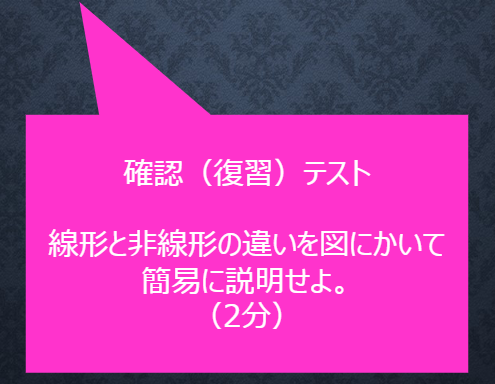
**通常のニューラルネットワークは入力層、中間層(隠れ層)、出力層、三種類に分けられいる。データの流れとして、入力層はデータを入力して、入力したデータを重みとバイアスをつけて、次の層(中間層の第一層)の全ノードと連結する(Fully Connected Layer)、そして中間層同士の間に同じ手法で出力層までデータを送ります。**

**まとめ要点２：**

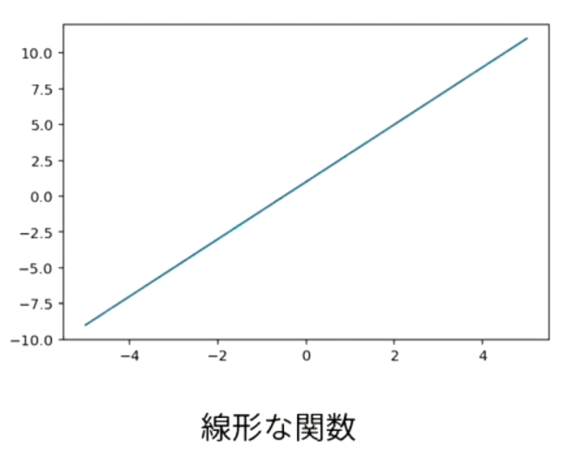
　活性化関数

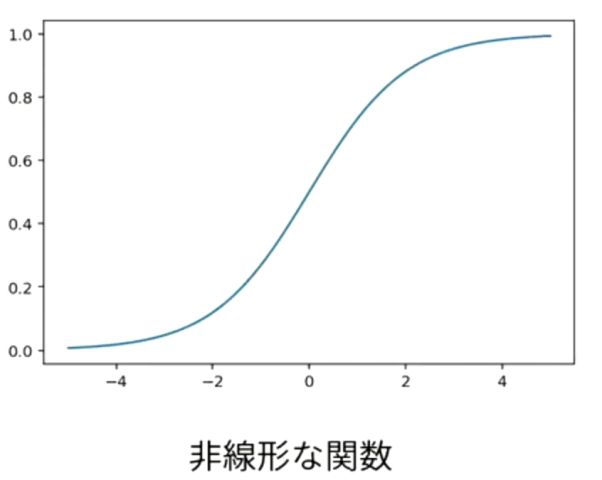
活性化関数は、ューラルネットワークにおいて、次の層への出力の大きさを決める非線形の関数のこと。活性化関数は主に中間層用の活性化関数と出力層用の活性化関数に種類に分かれている。よく使う活性化関数は、RELU、sigmoid、step、SoftMaxなど。活性化関数は入力値の値によって、次の層への信号のON/OFFや強弱を定める働きをもつ。

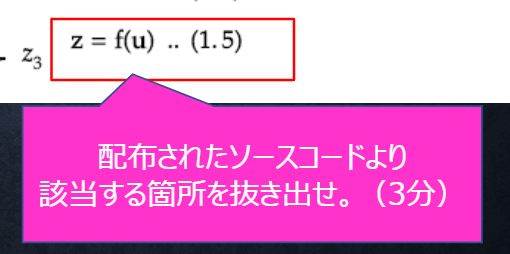
**演習問題：**

****

**答え：**

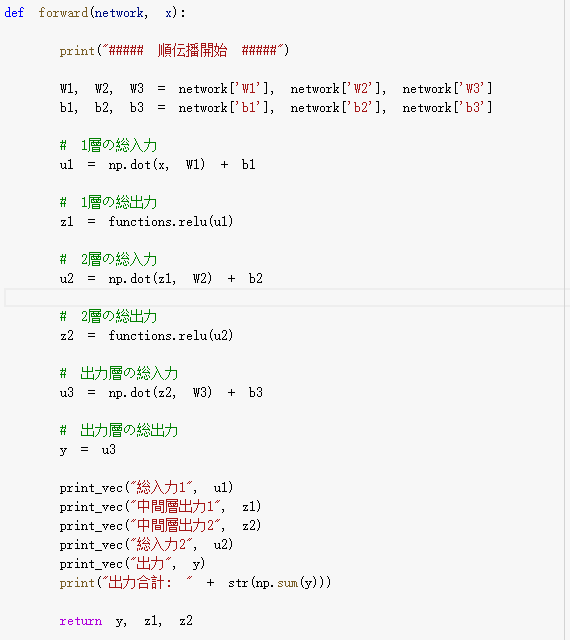






答え：

実装確認：



以上のソースの中、入力層と中関層はRELU活性化関数が使用されていた。

**サマリーと考察：**

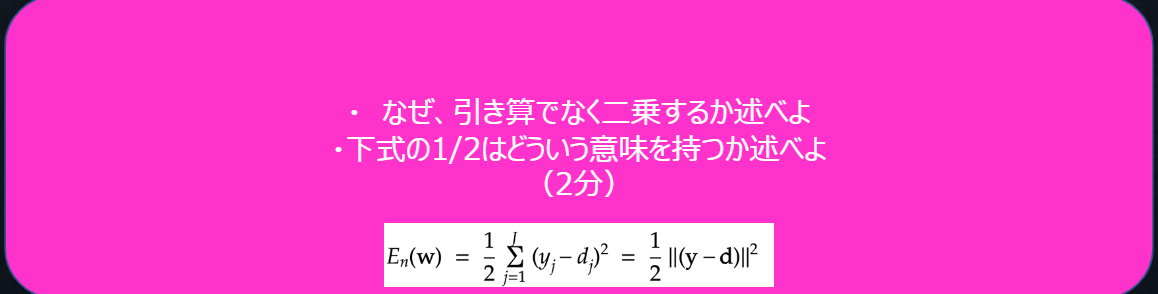
**なぜ、ニューラルネットワークは活性化関数を必要なのか、その原因は現実中のデータは非線性ですが、ニューラルネットワークの計算はほどんど線性的、非線性の活性化関数を使うこと、ニューラルネットワークの学習能力を上昇出来るので。だから、活性化関数最大の特徴は非線性的だと思います。それにしても、種類が違う活性化関数は全部同じ特徴を持つではない、例えばsigmoid関数やtanh関数の特徴は出力を(0，1)と(-1，1)の範囲内に納めることができます、RELUはそれを出来ない。**

**まとめ要点３：**

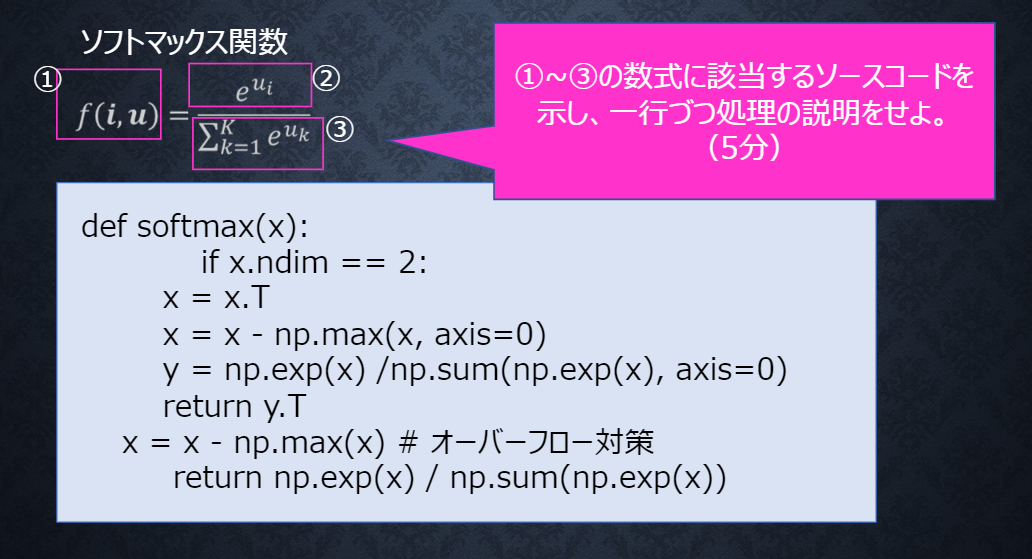
　出力層

出力層は理解しやすいと思います。最終の出力する時、一番最後の層である。出力層が誤差関数を決めないといけないである。よく見られる二乗誤差や分類問題で使てるcross\_entropyなどの誤差関数は記憶した方がいいと思います。誤差関数と出力層が使う活性化関数と区別して、別物だから注意した方がいい。出力層が使う活性化関数は分類問題の場合、出力層の出力は0 ~ 1 の範囲に限定し、総和を１とする必要があるので、中間層と違います。

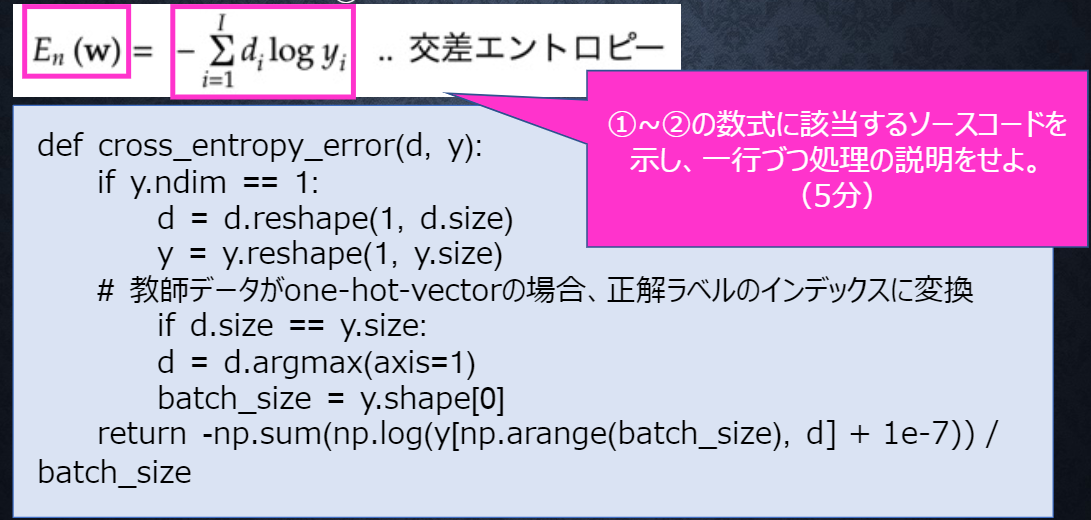
**演習問題：**



答え：2乗は答えの正負両方の値を統一して、全体的の影響を減らすため。2分の1は計算を簡単にするため、本質的の意味はない。

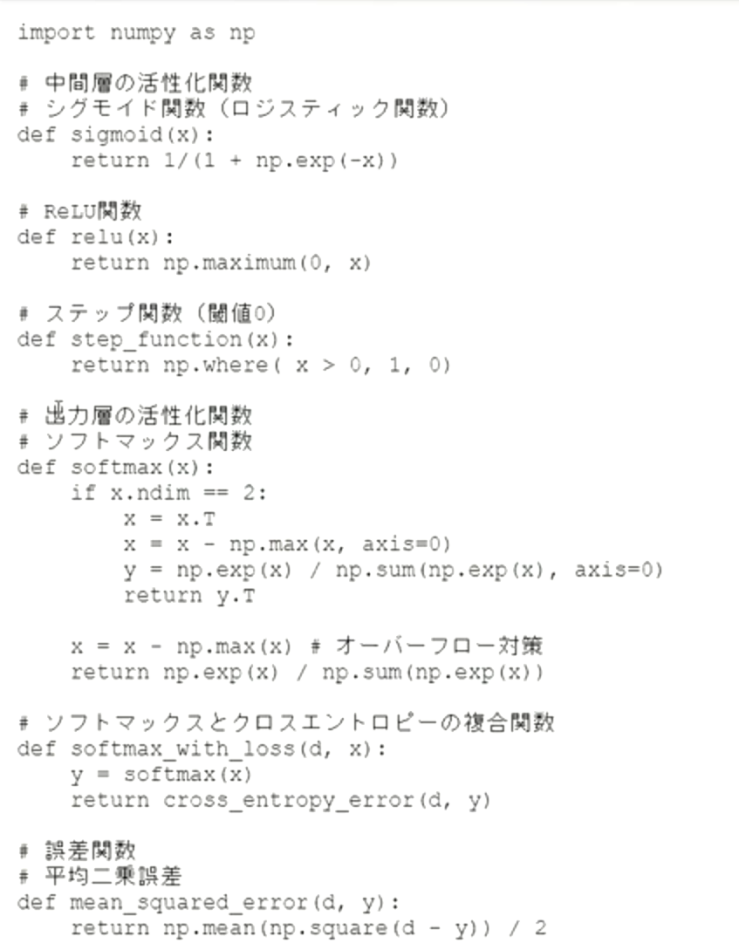
****

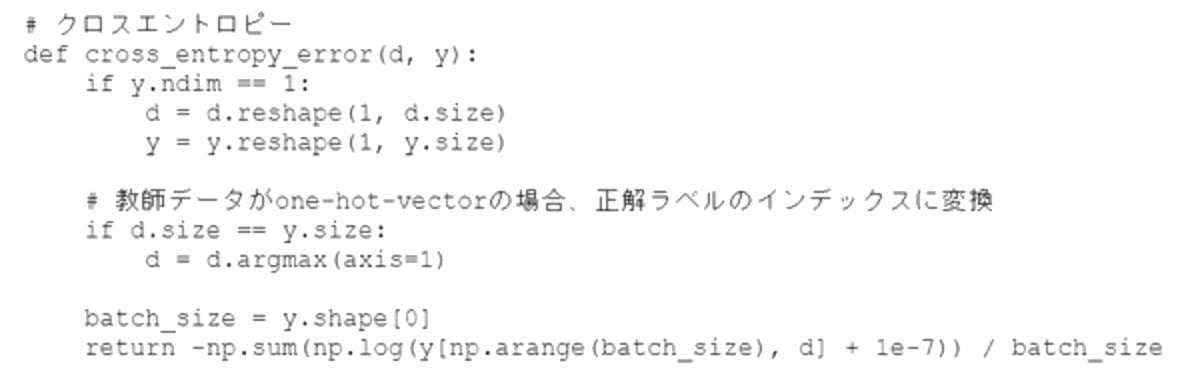
**答え：ミニバッチの場合、上の部分を使うですが、通常の場合、最後一行は本質的計算式である。**



**答え：前問と同じ、最後の一行は計算式である。numpyを使えば、すぐ作れるですが、式の中に1e-7は、対数関数自体はｘは０に近つけば、値は無限に近くなるので、1e-7を加えることでそんな問題を解決出来るため追加された。**

**よく使う活性化関数と誤差関数の実装確認：**





**サマリーと考察：**

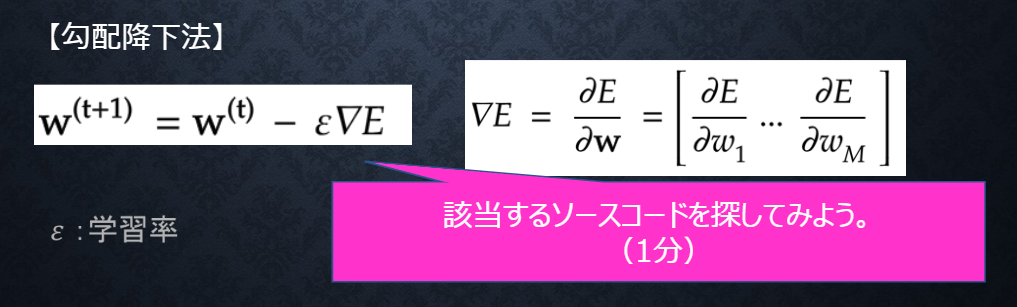
**出力層がどの活性化関数と誤差関数を選択すればいいかについて、問題次第です、例えば二分類問題、sigmoid関数と(binary cross entropy)の組み合わせ、多重分類問題はSoftMaxと(categorical cross entropy)など、最終的値の範囲は制限がないのばあい理論上活性化関数を使わなくでもいいと考えています。**

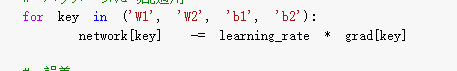
**まとめ要点４：**

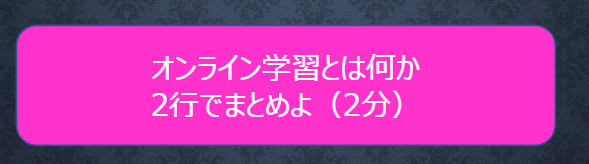
　勾配降下法

簡単に言うと勾配降下法は最適のパラメータを探すための手法である。勾配降下法を計算する時データが重いの場合、時間やメモリ容量などの資源を大量消費するので、通常の勾配降下法を元にはいろんな改良された勾配降下法が存在する、ミニバッチや確率的勾配降下法などはそれぞれののメリットがあります。

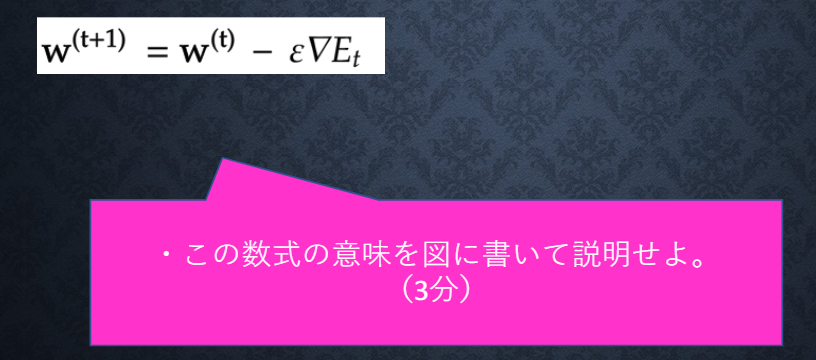
**演習問題：**

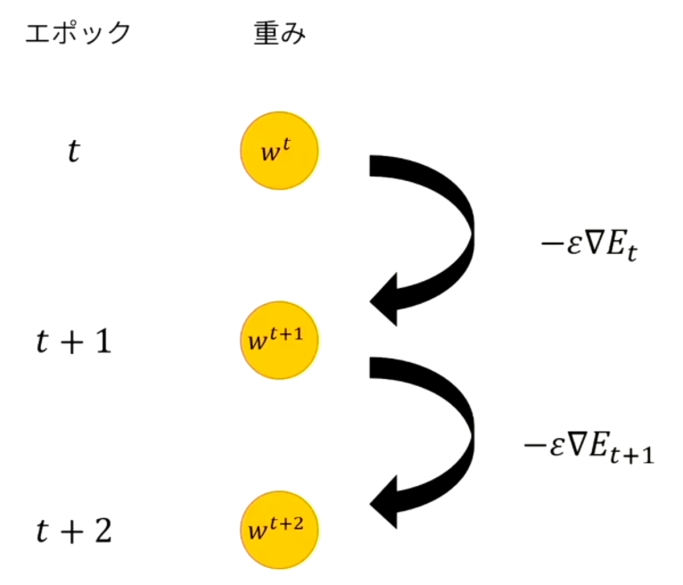


答え：

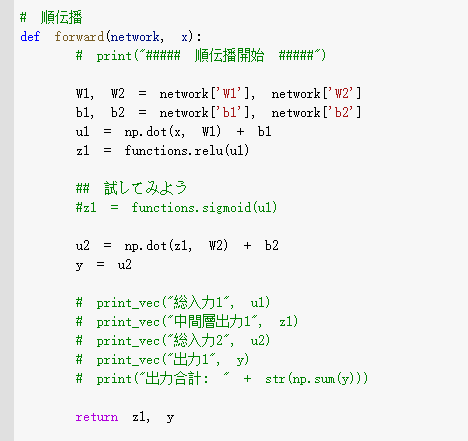


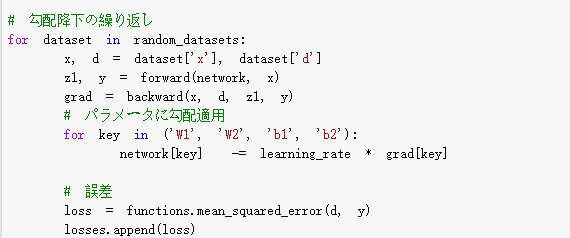
答え：オンライン学習はデータが入力した時点NWのパラメータを更新する方法である。一方、バッチ学習は一度全部のデータを使ってパラメータを学習する方法である。



答え：

**実装確認：**





**サマリーと考察：**

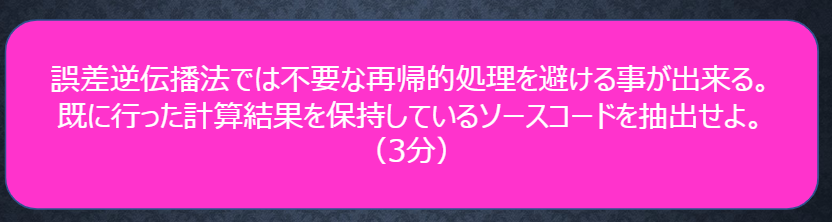
**なぜ勾配降下法は最適のパラメータを見つけることが出来ますかについて簡単に説明しようとします、例えば関数自体(微分可能)は山、今の私たちは山頂にいる、私たちの目標はできるだけ早く山から降りること。一番険しいところから降りるのは、最速の選択。では、どうやって一番険しいところを見つけるのか、それは山の勾配の逆方向なので、だから、何度も何度も山の微分を取って、最後山頂から降りることが出来るの原因。**

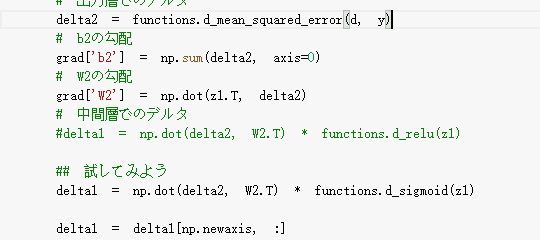
**まとめ要点5：**

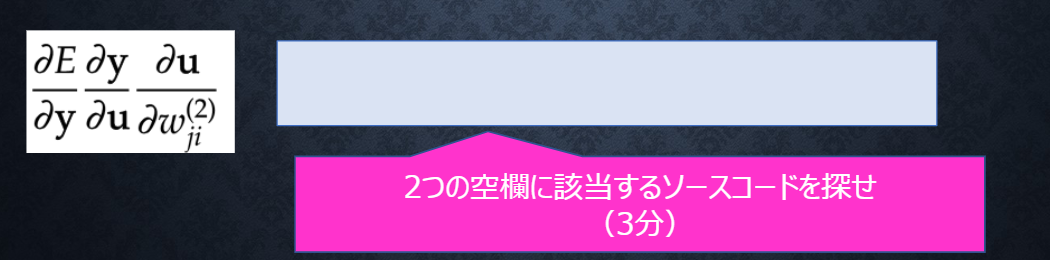
　誤差逆伝播法

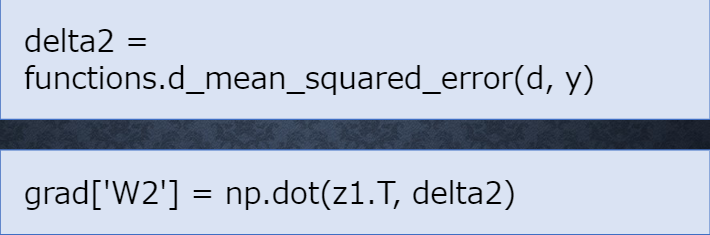
誤差逆伝播法とは算出された誤差を、出力層側から順に微分し、前の層前の層へと伝播。最小限の計算で各パラメータでの微分値を解析的に計算する手法。順伝播の場合、更新量を計算する時各パラメータを再帰的な微分する必要があるので、実装する時計算量が大きいので、誤差逆伝播法を使うことで不必要な微分を避けることで計算効率を上昇出来る。

**演習問題：**



答え：



答え：  
**サマリーと考察：**

**誤差逆伝播法が速いの原因とは、例えば１億の入力値、一個の出力しかないの場合、順伝播はの方が各パラメータに対して微分しないといけない(１億回操作)、誤差逆伝播法の方が誤差を利用して、1回だけ微分すれば各パラメータに対しての偏微分の結果が分かれている。この場合、効率を考えると、誤差逆伝播法は順伝播実行効率の一億倍(理想)。**