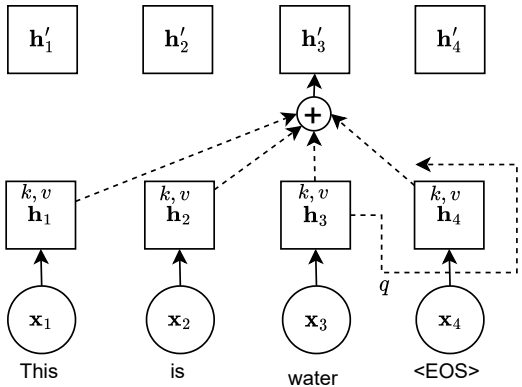
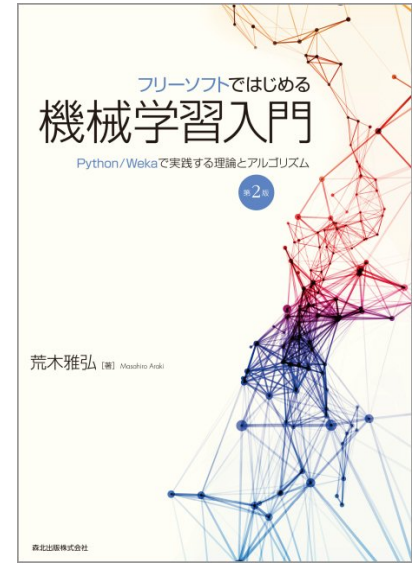
9. 深層学習



- 9.1 深層学習とは
- 9.2 DNN のモデル
- 9.3 多階層ニューラルネットワーク
- 9.4 畳み込みネットワーク
- 9.5 リカレントネットワーク



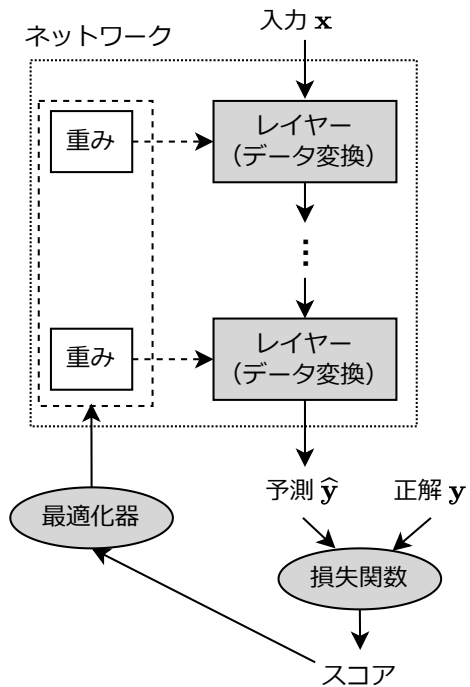
- 荒木雅弘:『フリーソフトではじめる機械学習入門(第2版)』(森北出版, 2018年)
- スライドとJupyter notebook
- サポートページ

9.1 深層学習とは

- 深層学習とは
  - 多階層のニューラルネットワークを用いた機械学習
    - 基本的な学習手段は誤差逆伝播法
  - 特徴抽出段階も学習対象とすることで大幅に性能が向上
  - 入力の種類に応じたネットワーク構造が提案された
    - 画像認識に適した畳み込みネットワーク
    - 自然言語処理に適したリカレントネットワーク
    - マルチモーダルに適用された Transformer アーキテクチャ

9.2 DNN のモデル

- 深層学習ライブラリ keras の枠組み(スライド第8章の図 再掲)



9.3 多階層ニューラルネットワーク

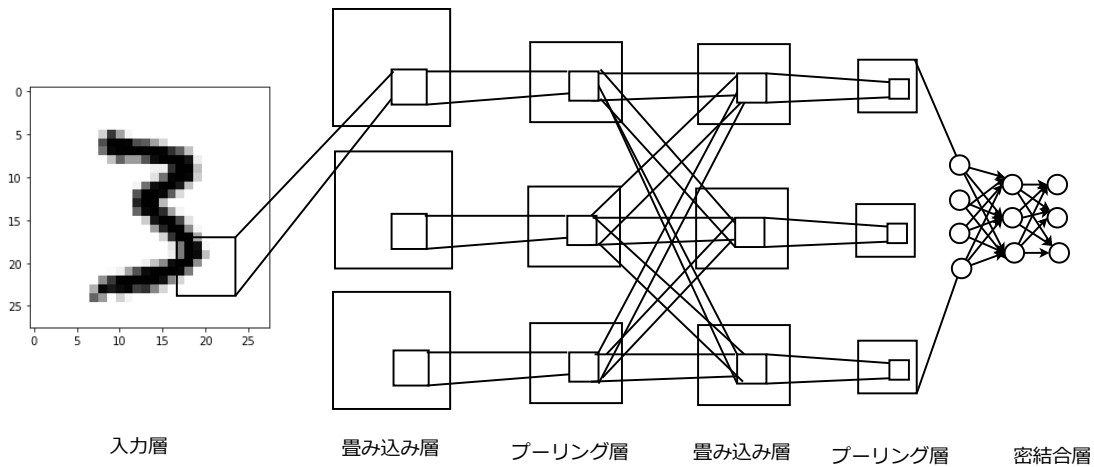
- 多階層ニューラルネットワークの学習
  - 単純な多階層化においては勾配消失問題があった
    - 初期はオートエンコーダなどを用いた事前学習が行われた
    - 現在では ReLU などの活性化関数の導入により勾配消失問題は緩和された
- 過学習の回避
  - ドロップアウト
    - 学習時は予め定めた確率で隠れ層のユニットを無効化する
    - 推論時は各ユニットの出力にその確率を掛ける

9.4 畳み込みネットワーク (1/7)

- 畳み込みネットワーク(CNN: Convolutional Neural Network)とは
  - 画像認識や音声認識に適したニューラルネットワーク
    - 空間あるいは時間軸上に並んだ信号から特定のパターンを見つけ出す
  - 畳み込み層とプーリング層を交互に重ねて特徴抽出
    - 畳み込み層:フィルタを使って特定のパターンを見つける
    - プーリング層:位置の変動を吸収するダウンサンプリング
    - これらを交互に重ねることで、複雑な特徴を表現可能
  - 正規化層で入力値を調整することもある
  - スキップ接続を入れることもある
  - 最後は数段の密結合層(ReLU+Softmax)

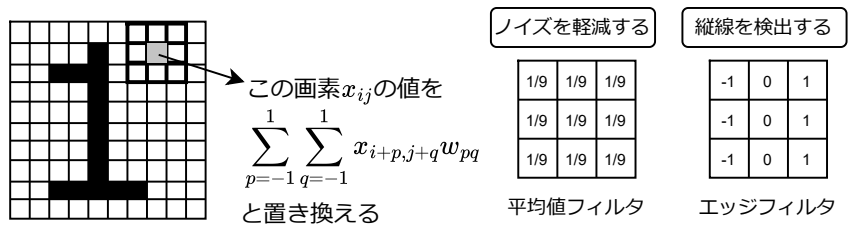
9.4 畳み込みネットワーク（2/7）

- 畳み込みネットワークの構造
  - 畳み込み層は各チャネルでそれぞれ異なるフィルタを学習
    - 重みはチャネル内では共通なので、学習対象であるパラメータの数は少ない
  - プーリング層は最大値または平均値の計算なので、その重みは学習対象外
  - 密結合層は特徴を入力として識別を行う

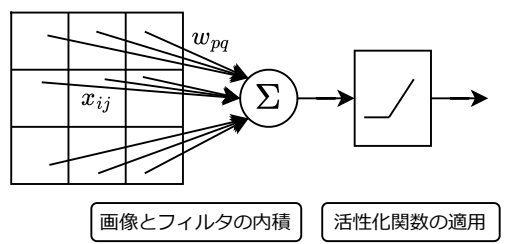


9.4 畳み込みネットワーク（3/7）

- 畳み込み層
  - 畳み込み：フィルタを一定画素（ストライド）ずつずらして画像との内積を計算



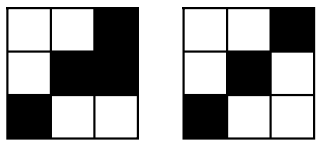
  - 畳み込み層の演算：畳み込み結果に対して活性化関数を適用



    - 結果としてフィルタに対応するパターンの出現を示す**特徴マップ**が得られる

9.4 畳み込みネットワーク (4/7)

- 畳み込み演算の意味
  - 切り出した画像とフィルタパターンの類似度を求めている

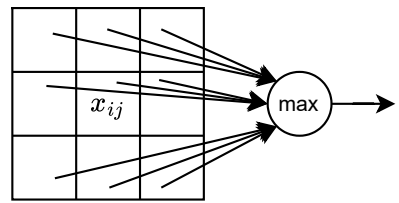


画像 $\mathbf{x}$　　フィルタ $\mathbf{w}$

  - $\boldsymbol{x}$ と $\boldsymbol{w}$ が類似しているとは、$\|\boldsymbol{x}-\boldsymbol{w}\|^2$ が小さな値となるとき
    - $\|\boldsymbol{x}-\boldsymbol{w}\|^2 = \|\boldsymbol{x}\|^2 - 2\boldsymbol{x}^T\boldsymbol{w} + \|\boldsymbol{w}\|^2 \Rightarrow$ 内積 $\boldsymbol{x}^T\boldsymbol{w}$ が大きな値となるとき
- チャネルの意味
  - 幅 H、高さ W のカラー画像の場合、色は RGB の3チャネルで表されるので、入力は (H, W, 3) のテンソルとなる
  - 隠れ層の場合、フィルタの数がその層の出力チャネル数となる

9.4 畳み込みネットワーク（5/7）

- プーリング層
  - 一定範囲の最大値あるいは平均値を計算
  - 畳み込み層の出力である特徴マップの解像度を低くする役割



- 正規化層
  - 入力値からミニバッチ毎の平均値を引いて標準偏差で割る
  - 多階層の演算による値の大きな変動やミニバッチ毎の分布の違いを吸収
- スキップ接続
  - 学習時に誤差をそのまま伝えるので、CNN の多階層化が可能になった

9.4 畳み込みネットワーク（6/7）

- コーディング例
  - ライブラリの読み込みとデータの前処理

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
X_train = X_train.reshape(X_train.shape[0] , 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train = X_train / 255.0
X_test = X_test / 255.0
```

9.4 畳み込みネットワーク（7/7）

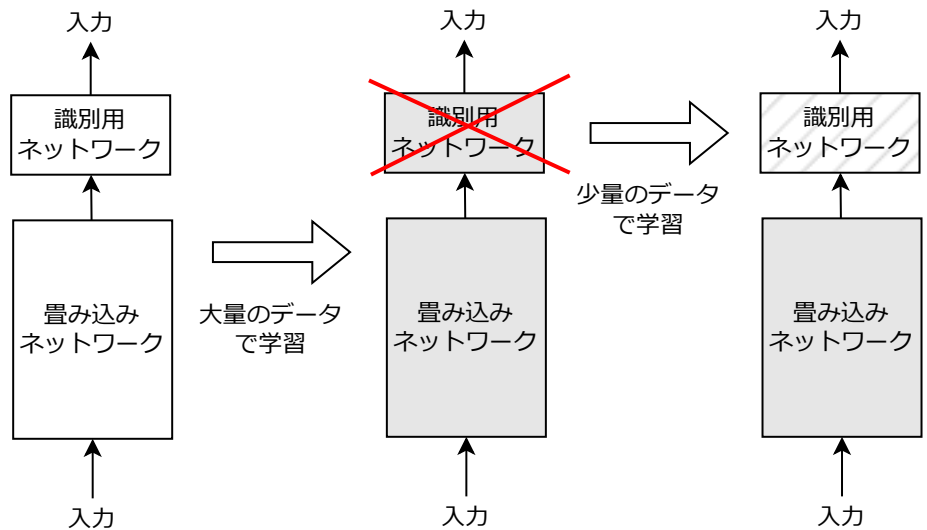
- ネットワークの構成と学習・評価

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model2.fit(X_train, y_train, epochs=5, batch_size=200)
test_loss, test_acc = model2.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc:.4}')
```

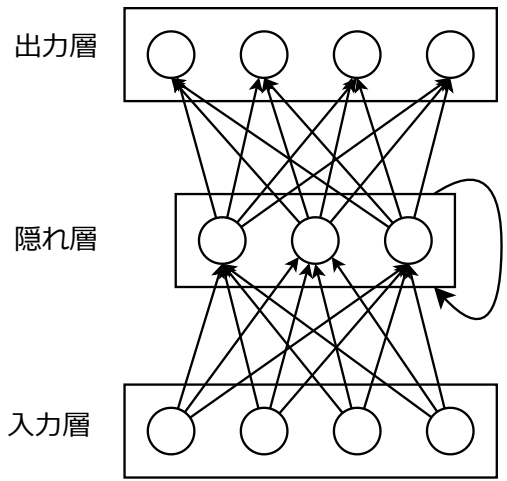
転移学習

- 大量のデータで学習済みの CNN の特徴抽出部分を利用
- 対象タスクの少量のデータで識別用ネットワークのみ学習
  - 特徴抽出部分を追加学習する場合もある

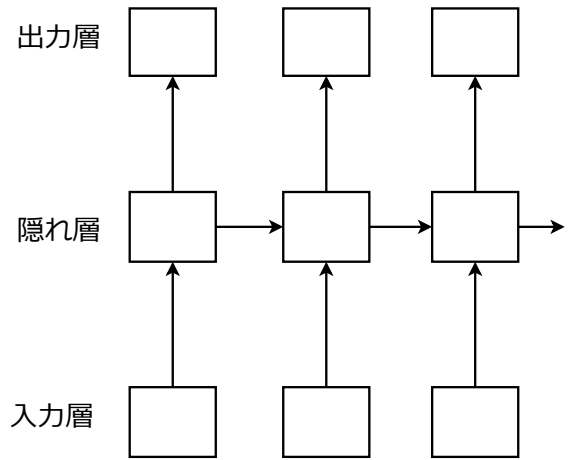


9.5 リカレントネットワーク (1/13)

- リカレントネットワーク(RNN: Recurrent Neural Network)とは
  - 時系列信号の認識や自然言語処理に適したニューラルネットワーク
  - 一時刻前の隠れ層の出力を次の入力と結合して隠れ層に入力



(a) リカレントネットワーク



(b) ループを時間方向に展開

9.5 リカレントネットワーク（2/13）

- 隠れ層の計算
  - 入力を $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ としたとき、内部状態 $\boldsymbol{h}$ が時刻毎に更新されてゆくと考える

$$\boldsymbol{h}_t = f(\boldsymbol{W}[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b})$$

    - $\boldsymbol{W}$：隠れ層のループ、および入力層から隠れ層への重みベクトルを結合した行列
    - $\boldsymbol{b}$：各隠れ層のバイアス項を結合したベクトル
- 出力層
  - 入力系列の識別：$\boldsymbol{h}_N$ から重み付き和の softmax 等で識別結果 y を出力
  - 入力に対するラベルの付与：各$\boldsymbol{h}_t$ から上と同様の手順で y_t を出力

9.5 リカレントネットワーク（3/13）

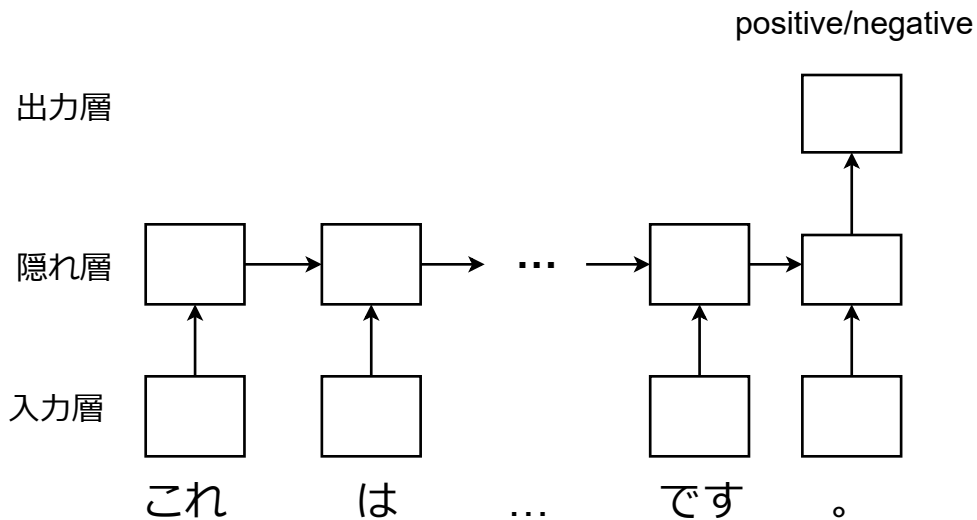
- リカレントネットワークの学習
  - 通常の誤差逆伝播法の更新式に対して、時間を遡った更新が必要
  - 時刻 t において、k 個過去に遡った更新式

$$w(t) \leftarrow w(t-1) - \sum_{z=0}^{k} \eta \epsilon(t-z) g(t-z-1)$$

  - 勾配消失を避けるため、$k = 10$〜100 程度とする

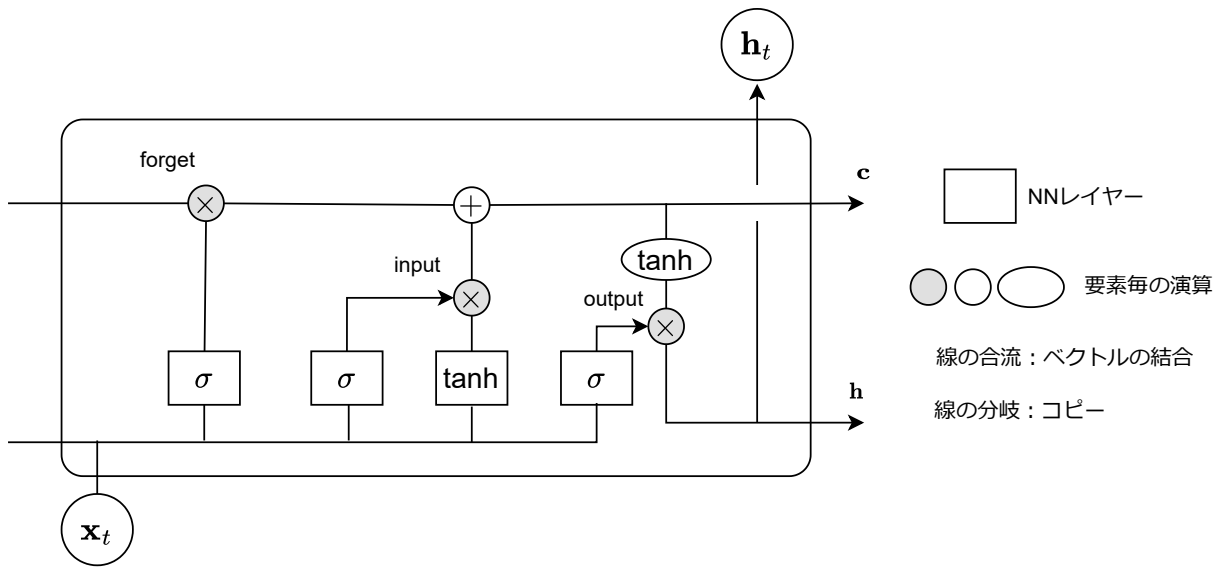
9.5 リカレントネットワーク（4/13）

- 単純なリカレントネットワークの問題点
  - 前方の情報が後方まで伝わりにくい
  - 学習時にループの重み行列が何度も掛けられ、勾配消失／爆発が生じやすい



9.5 リカレントネットワーク（5/13）

- LSTM（long short-term memory）の構造
  - 隠れ層の情報$\boldsymbol{h}$以外に、いくつかのゲートを用いて作成したセル状態$\boldsymbol{c}$を管理するユニット
    - ゲート：選択的に情報を通すメカニズム



9.5 リカレントネットワーク（6/13）

- LSTMのゲート
  - forgetゲート：セル状態のどの部分を忘れるか

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f)$$

  - inputゲート：入力のどの部分をセル状態の計算に用いるか

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i)$$

  - outputゲート：出力のどの部分をセル状態に組み込むか

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o)$$

9.5 リカレントネットワーク（7/13）

- 隠れ層の計算

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c)$$

- セル状態の計算

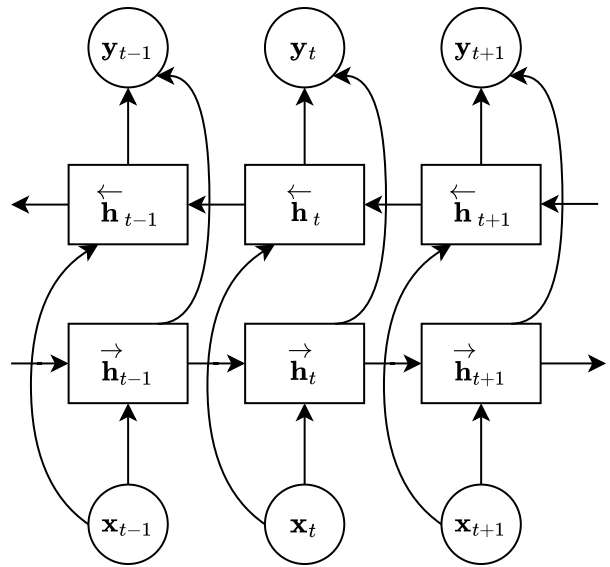
$$\boldsymbol{c}_t = \boldsymbol{f}_t * \boldsymbol{c}_{t-1} + \boldsymbol{i}_t * \tilde{\boldsymbol{c}}_t$$

  - 学習時に$\boldsymbol{W}_c$を何度も掛けることがなくなったので、勾配消失／爆発が起きにくい
- 出力の計算

$$\boldsymbol{h}_t = \boldsymbol{o}_t * \tanh(\boldsymbol{c}_t)$$

9.5 リカレントネットワーク（8/13）

- 双方向LSTMの構造
  - 過去だけでなく、未来の情報も用いて出力を計算



9.5 リカレントネットワーク（9/13）

- コーディング例
  - ライブラリの読み込みとデータの前処理

```
import numpy as np
import tensorflow as tf

max_features = 10000
maxlen = 50
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.imdb.load_data(num_words=max_features)
X_train = tf.keras.preprocessing.sequence.pad_sequences(X_train, maxlen=maxlen)
X_test = tf.keras.preprocessing.sequence.pad_sequences(X_test, maxlen=maxlen)
```

9.5 リカレントネットワーク (10/13)

- 単純なRNNネットワークの構成と学習・評価

```
model1 = tf.keras.Sequential([
    tf.keras.layers.Embedding(max_features, 128),
    tf.keras.layers.SimpleRNN(64),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model1.fit(X_train, y_train, epochs=2, batch_size=32, validation_split=0.2)
test_loss, test_acc = model1.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc:.4}')
```

9.5 リカレントネットワーク (11/13)

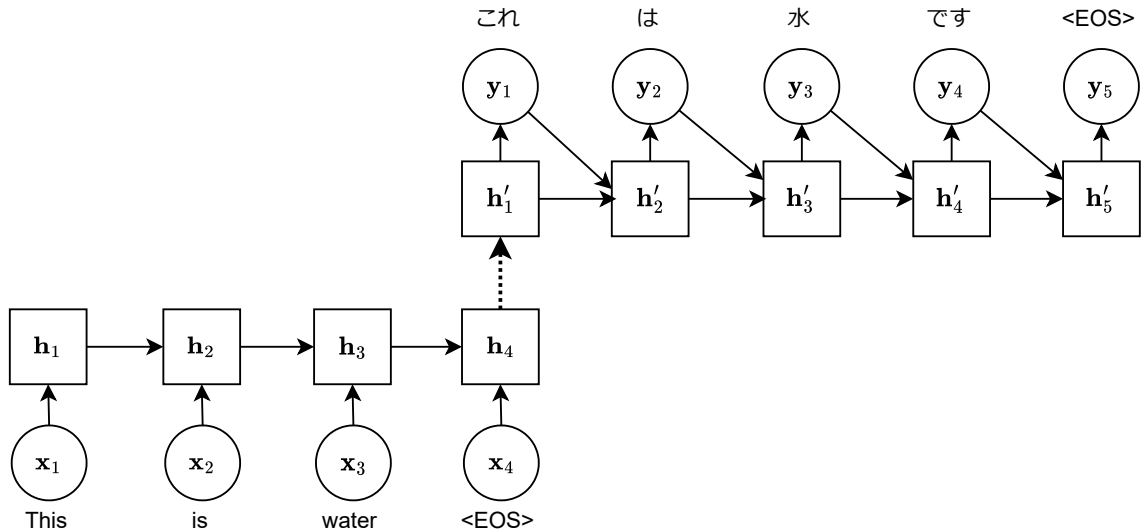
- 双方向LSTMネットワークの構成と学習・評価

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Embedding(max_features, 128),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model2.fit(X_train, y_train, epochs=2, batch_size=32, validation_split=0.2)
test_loss, test_acc = model2.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc:.4}')
```

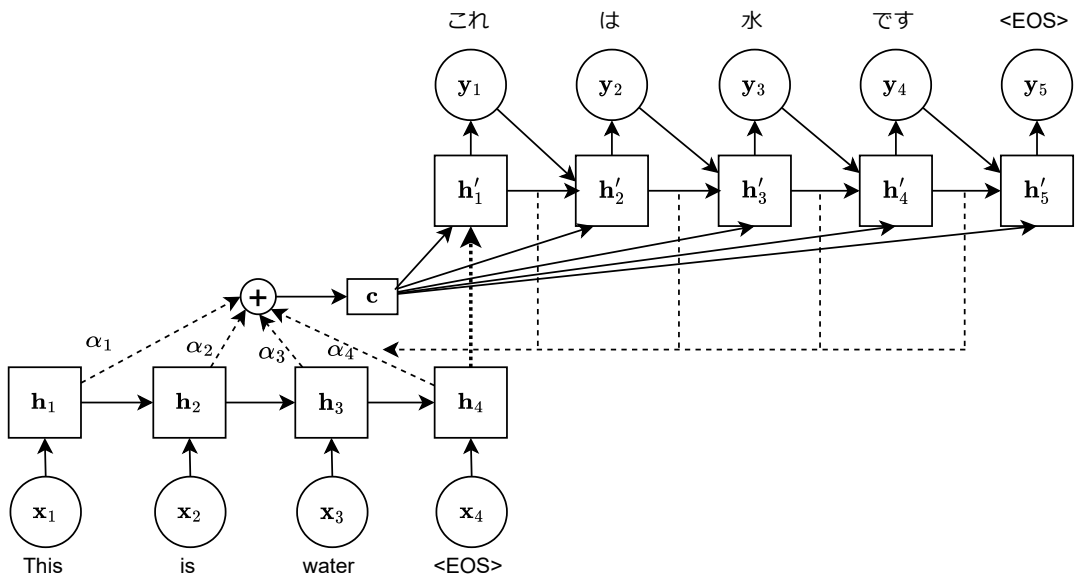
9.5 リカレントネットワーク (12/13)

- Encoder-Decoder
  - 入力の内容をひとつの表現にまとめて、そこから出力を生成
  - 機械翻訳や対話システムにおける応答生成ができることが示された



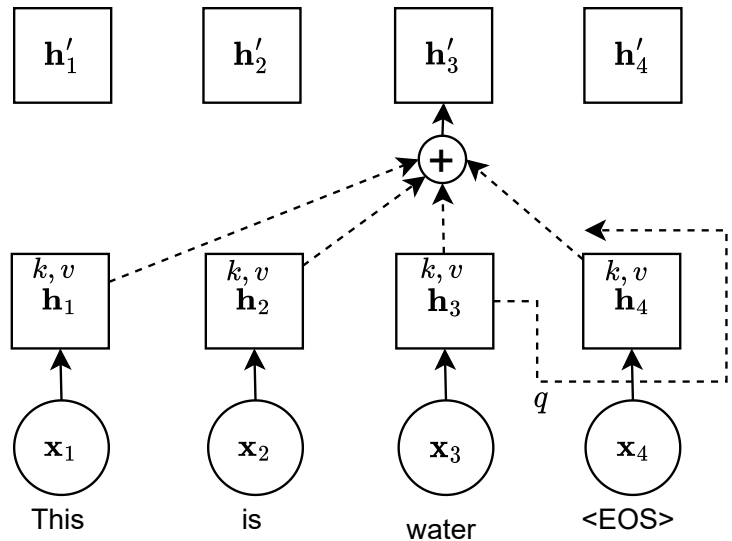
9.5 リカレントネットワーク (13/13)

- Encoder-Decoder + Attention
  - アテンション：Encoder内のすべての隠れ層出力の重み付き和
  - 入力のどの部分を見るかという情報を出力時に利用



Transformer (1/6)

- Self-attention
  - リカレント構造を廃止して入力系列全体を入力とし、各単語のベクトル表現を作成する
  - 隠れ層の出力としてベクトル表現を作るときに、入力の他の部分との関係を計算
  - BERTなどの基盤モデルに使われる



Transformer (2/6)

- Self-attentionの計算
  - 入力 $\boldsymbol{h}$ からクエリ $\boldsymbol{q}$ 、キー $\boldsymbol{k}$、値 $\boldsymbol{v}$ を計算

$$\boldsymbol{q} = \boldsymbol{W}_Q \boldsymbol{h}, \quad \boldsymbol{k} = \boldsymbol{W}_K \boldsymbol{h}, \quad \boldsymbol{v} = \boldsymbol{W}_V \boldsymbol{h}$$

  - 特定の入力 $\boldsymbol{h}$ のクエリ $\boldsymbol{q}$ に対して、全入力 $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N$ の $\boldsymbol{k}$ との内積を計算
    - さらにこれをクエリの次元数のルート $\sqrt{d_k}$ で割って、内積の分散が1となるようにする

$$\alpha_i = \frac{\boldsymbol{q}^T \boldsymbol{k}_i}{\sqrt{d_k}}$$

  - これらのsoftmaxを値 $\boldsymbol{v}_i$ の重みとする重み付き和によって、求めるベクトル表現を得る

$$\boldsymbol{h}' = softmax(\boldsymbol{\alpha})^T \boldsymbol{v}$$

Transformer (3/6)

- Transformerアーキテクチャ [Vaswani et al., 2017]

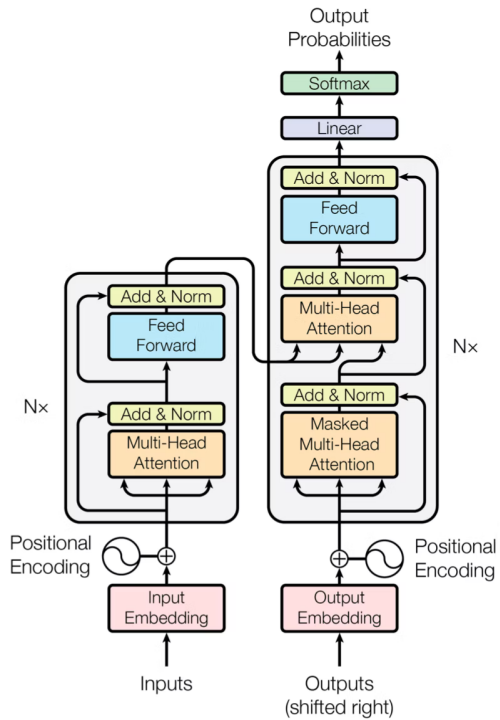


Figure 1: The Transformer - model architecture.

Transformer (4/6)

- Transformerユニット
  - encoder側
    - Self-attentionを多チャネルで行う（multi-headとよぶ）
    - Self-attentionの出力にスキップ接続を加えて、正規化を行う
    - それによって得られたベクトルに対して密結合層で新たなベクトルに変換する
    - この出力に対してもスキップ接続を加えて、正規化を行う
  - decoder側
    - Self-attention後の1段目の正規化の後に、encoderの出力ベクトルも対象に加えたself-attentionを多チャネルで行う処理、およびスキップ接続・正規化を挿入する

Transformer (5/6)

- Transformerの動作
  - 入力の処理 (encoder)
    - 入力の各要素を密ベクトルに変換(embedding)
    - 位置の情報を付与(positional encoding)
    - Transformerユニットの処理をN段階繰り返し、入力系列に対する隠れ層出力系列を得る
  - 出力の処理 (decoder)
    - 先頭記号を密ベクトルに変換したものを入力とする
    - 位置の情報を付与(positional encoding)
    - Self-attentionの後、encoderの出力も加えたsource-target attentionの処理を行うユニットの処理をN段繰り返し、出力系列を得る
    - その出力系列を出力側の入力として、同様の処理を繰り返すことで順次、次の出力を得る

Transformer (6/6)

- Transformerを応用した基盤モデル
  - BERT
    - 入力側(encoder)のみを用いる
    - 事前学習：入力単語系列のうち、一部を [MASK] という記号に置き換え、その単語を当てるという自己教師あり学習を繰り返す
    - fine-tuning: 目的のタスクに応じてネットワーク構造を追加して再学習を行う
  - GPT
    - 出力側(decoder)のみを用いる
    - 事前学習：次単語を予測する問題を繰り返す
    - few-shot learning: 入出力の事例を少数与えて、出力を得る
  - T5
    - 様々なタスクでTransformerを学習
    - タスクの指示と入力を与えて、出力を得る

9.6 まとめ

- 画像認識・自然言語処理などのタスクに適したネットワーク構造が提案されている
- Transformerを応用した基盤モデルの活用が有望
- 参考文献
  - 岡野原 大輔: ディープラーニングを支える技術, 技術評論社, 2021.
  - Francois Chollet: Deep Learning with Python, Second Edition, Manning, 2021.