【要点まとめ】
過去の情報だけではなく、未来の情報もネットワークに加味することで精度の向上を図ったモデル。
（中間層の出力を、未来への順伝播と過去への逆伝播の両方に使用する）
利用例として、文章の推敲や機械翻訳がある。
※機械翻訳であれば、一文から文章全体の誤字脱字を検出することに利用されている。

【実装演習】

```
def bidirectional_rnn_net(xs, W_f, U_f, W_b, U_b, V):
    xs_f = np.zeros_like(xs)
    xs_b = np.zeros_like(xs)
    for i, x enumerate(xs):
        xs_f[i] = x
        xs_b[i] = x[::-1]

    hs_f = _rnn(xs_f, W_f, U_f)
    hs_b = _rnn(xs_b, W_b, U_b)
    hs = np.concatenate([h_f, h_b[::-1]], axis=1)
    ys = hs.dot(V.T)
    return ys
```

【実装演習考察】
過去と未来を合わせたものが特徴量となるため、concatenate(axis=1で水平結合)するのはしっくりきました。
※単なる四則演算では特徴が消失してしまうためです。

【自己学習】
※参考資料：ゼロから作るDeepLearning②
最初から未来も含めた全ての時系列データ（翻訳すべき文章）が与えられるため、双方向RNNは特に機械翻訳で有効である。
例えば翻訳時に、あるワードの周囲の情報をバランスよく含ませることができる。（過去・未来双方からそのワードに干渉できるため）