第10章

翻訳モデル

13T4054L　根岸　睦

Encoder-Decoder 翻訳モデル

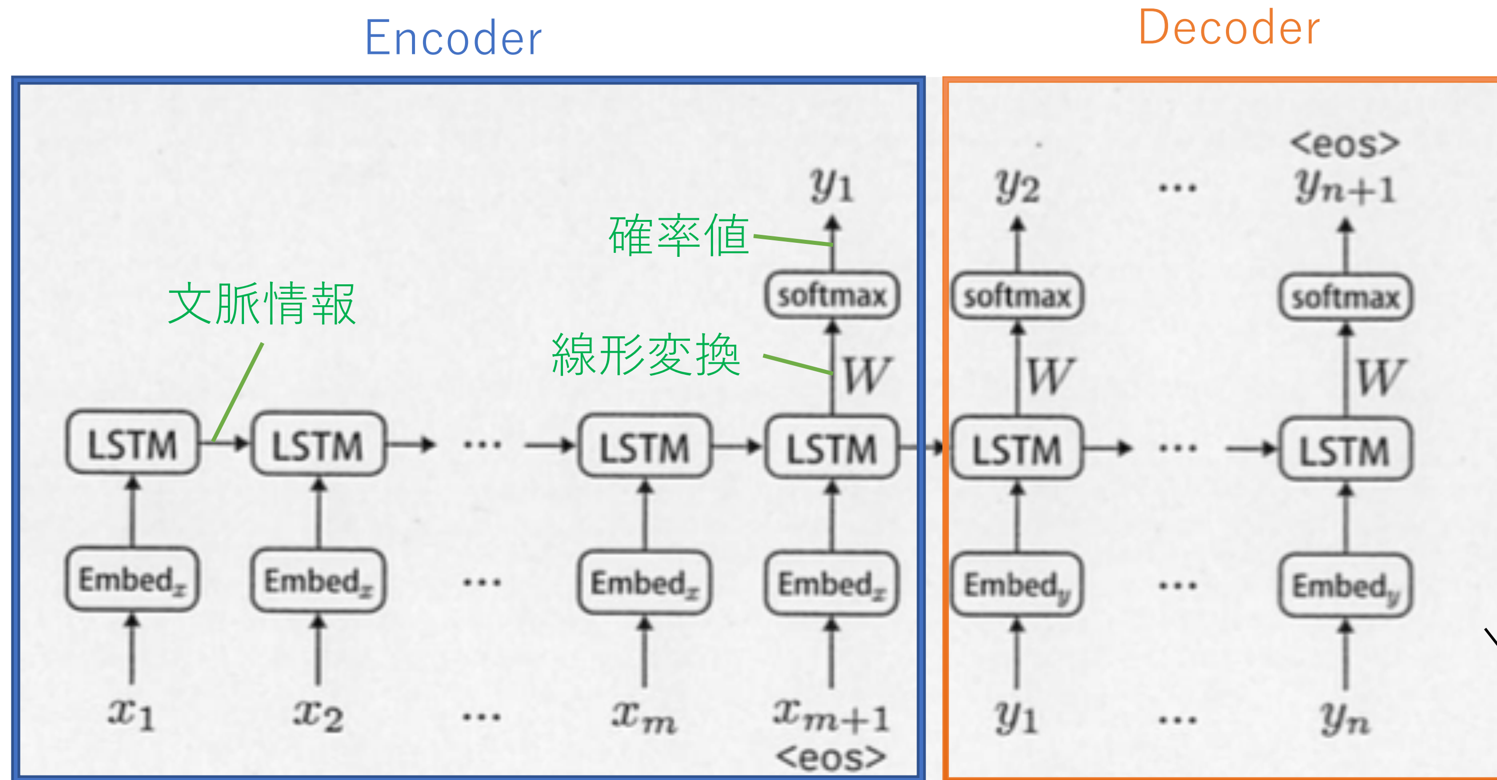


図 10.3 Encoder-Decoder 翻訳モデルの詳細図

目的

原言語 $x_1, x_2, \ldots, x_m$

変換

目的言語 $y_1, y_2, \ldots, y_n$

学習の際は正解の単語ｔと出力の単語ｙとの損失を累積してゆき誤差逆伝播を行う

対訳データの準備

```
#  単語 -> IDの辞書（日本語）
jvocab = {}
jlines = open('jp.txt').read().split('\n')
for i in range(len(jlines)):
    lt = jlines[i].split()
    for w in lt:
        if w not in jvocab:
            jvocab[w] = len(jvocab)
jvocab['<eos>'] = len(jvocab)
jv = len(jvocab)

#  単語 <--> IDの辞書（英語）
evocab = {}
id2wd = {}
elines = open('eng.txt').read().split('\n')
for i in range(len(elines)):
    lt = elines[i].split()
    for w in lt:
        if w not in evocab:
            id = len(evocab)
            evocab[w] = id
            id2wd[id] = w
id = len(evocab)
evocab['<eos>'] = id
id2wd[id] = '<eos>'
ev = len(evocab)
```

・日本語　→　英語　の翻訳

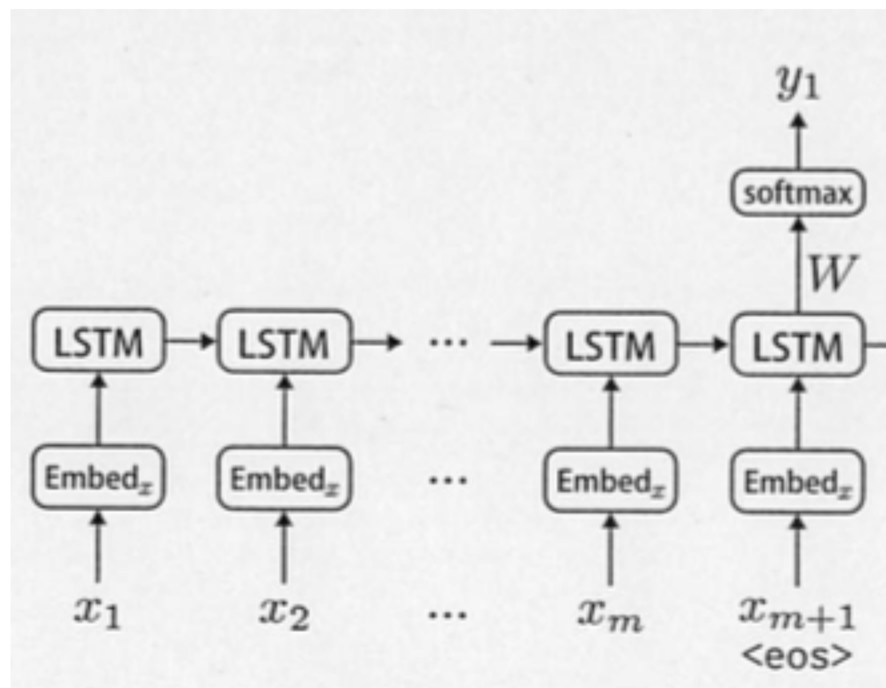
・jlines、elines : 各行のデータが入っているリスト

・jvocab、evocab: 単語をIDに直す辞書

・id2wd: ID から単語に直す辞書（出力側の英語のみ）

Chainのモデル部分

Encoder

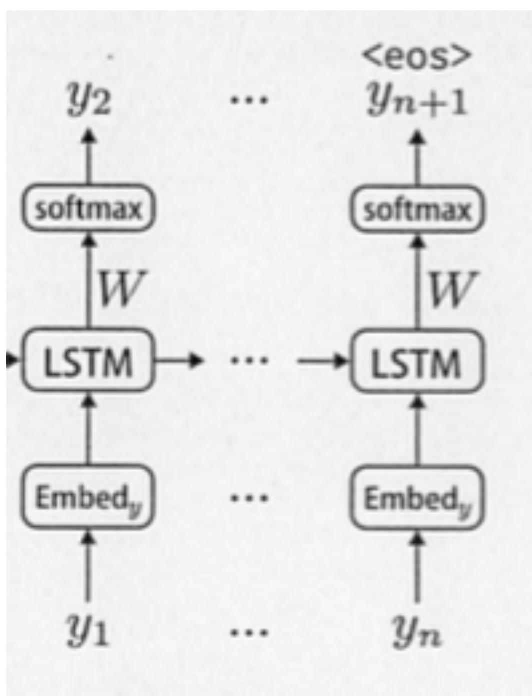



```
# Encoder側
    self.H.reset_state()
    for i in range(len(jline)):
        wid = jvocab[jline[i]]
        # enbedding
        x_k = self.embedx(Variable(\
                    np.array([wid], dtype=np.int32)))
        # 文脈情報保持
        h = self.H(x_k)
    # <eos>embedding
    x_k = self.embedx(Variable(\
                    np.array([jvocab['<eos>']], dtype=np.int32)))
    # 正解
    tx = Variable(np.array([evocab[eline[0]], \
                            dtype=np.int32]))

    h = self.H(x_k)
    # 損失
    accum_loss = F.softmax_cross_entropy(self.W(h), tx)
```

Chainのモデル部分

Decoder




```
# Decoder側
    for i in range(len(eline[i])):
        wid = evocab[eline[i]]
        x_k = self.embedy(Variable(np.array([wid], \
                                    dtype=np.int32)))
        next_wid = evocab['<eos>'] \
            if (i == len(eline) - 1) \
            else evocab[eline[i+1]]
        tx = Variable(np.array([next_wid], \
                                dtype=np.int32))
        h = self.H(x_k)

        # 損失の累積
        loss = F.softmax_cross_entropy(self.W(h), tx)
        accum_loss += loss
```

学習

```
#  モデルと最適化アルゴリズムの設定
demb = 100
model = MyMT(jv, ev, demb) # モデル生成
optimizer = optimizers.Adam() # 最適化アルゴリズム選択
optimizer.setup(model) # アルゴリズムにモデルをセット

#  学習
for epoch in range(100):
    for i in range(len(jlines)-1):
        jln = jlines[i].split()
        jlnr = jln[::-1]
        eln = elines[i].split()
        model.H.reset_state()
        model.cleargrades()
        loss = model(jlnr, eln) #  誤差の算出
        loss.backward() # 逆方向の計算、勾配の計算
        loss.unchain_backward()
        optimizer.update() # パラメータの更新
    outfile = "mt-" + str(epoch) + ".model"
    serializers.save_npz(outfile, model)
```

日本語の単語列を逆順にしている

少し結果がよくなる

翻訳

```
def mt(model, jline):
    model.H.reset_state()
    for i in range(len(jline)):
        wid = jvocab[jline[i]]
        x_k = model.embedx(\
                    Variable(np.array([wid], dtype=np.int32)))
        h = model.H(x_k)
    x_k = model.embedx(Variable(np.array([jvocab['<eos>']], \
                                            dtype=np.int32)))
    h = model.H(x_k)
    wid = np.argmax(F.softmax(model.W(h)).data[0])
    print id2wd[wid],
    loop = 0
    while (wid != evocab['<eos>']) and (loop <= 30):
        x_k = model.embedy(Variable(\
            np.array([wid], dtype=np.int32)))
        h = model.H(x_k)
        wid = np.argmax(F.softmax(model.W(h)).data[0])
        print id2wd[wid],
        loop += 1
    print
```

Attentionの導入

$\bar{h}$ は大域的に保持

$\bar{h}_1$ $\bar{h}_2$ $\bar{h}_m$ h_t c_t α_t $\tilde{h}_t$

x_1 x_2 … x_m x_{m+1} <eos> y_t

図 10.4　Attention を導入したモデル

$$\alpha_t(i) = \frac{\exp((\bar{h}_i, h_t))}{\sum_{j=1}^{m} \exp((\bar{h}_j, h_t))}$$

$$c_t = \sum_{i=1}^{m} \alpha_t(i)\bar{h}_i$$

$$\tilde{h}_t = \tanh(\boldsymbol{W}_c[c_t; h_t])$$

Wで重みをつけてsoftmaxを通して出力

Attention導入後のモデル

mk_ct関数

```
demb = 100
def mk_ct(gh,ht):
    s = 0.0
    for i in range(len(gh)):
        s += np.exp(ht.dot(gh[i]))
    ct = np.zeros(demb)
    for i in range(len(gh)):
        alpi = np.exp(ht.dot(gh[i]))/s
        ct += alpi * gh[i]
    ct = Variable(np.array([ct]).astype(np.float32))
    return ct
```

$$\alpha_t(i) = \frac{\exp((\bar{h}_i, h_t))}{\sum_{j=1}^{m} \exp((\bar{h}_j, h_t))}$$

$$c_t = \sum_{i=1}^{m} \alpha_t(i) \bar{h}_i$$

Attention導入後のモデル

Encoder

```
# Encoder側
    gh = []
    self.H.reset_state()
    for i in range(len(jline)):
        wid = jvocab[jline[i]]
        x_k = self.embedx(Variable(\
                    np.array([wid], dtype=np.int32)))
        h = self.H(x_k)
        # 文脈情報コピー
        gh.append(np.copy(h.data[0]))
    x_k = self.embedx(Variable(\
                    np.array([jvocab['<eos>']], dtype=np.int32)))
    tx = Variable(np.array([evocab[eline[0]], \
                            dtype=np.int32]))

    h = self.H(x_k)
    # Attentionの処理
    ct = mk_ct(gh, h.data[0])
    h2 = F.tanh(self.Wc1(ct) + self.Wc2(h))
    accum_loss = F.softmax_cross_entropy(self.W(h2), tx)
```

$$\tilde{h}_t = \tanh(\boldsymbol{W}_c[c_t; h_t])$$

Attention導入後のモデル

Decoder

```
# Decoder側
    for i in range(len(eline[i])):
        wid = evocab[eline[i]]
        x_k = self.embedy(Variable(np.array([wid], \
                                    dtype=np.int32)))
        next_wid = evocab['<eos>'] \
            if (i == len(eline) - 1) \
            else evocab[eline[i+1]]
        tx = Variable(np.array([next_wid], \
                                dtype=np.int32))
        h = self.H(x_k)
        #  Attentionの処理
        ct = mk_ct(gh, h.data)
        h2 = F.tanh(self.Wc1(ct) + self.Wc2(h))
        loss = F.softmax_cross_entropy(self.W(h2), tx)
        accum_loss += loss
```

$$\tilde{h}_t = \tanh(\boldsymbol{W}_c[c_t; h_t])$$