Lecture 03. Word Window Classification, Neural Networks, and PyTorch

Ⅰ. Classification review/introduction

1. Classification

$$\{x_i, y_i\}_{i=1}^{N}$$

x_i : inputs (d-dimension vector), y_i : C개의 class 중 x_i에 해당하는 class(label)

-Classfication : x값들이 주어졌을 때, y 값에 따라서 영역을 구분할 수 있는 선을 구해야 한다.(Classifier : 영역을 구분하는 선)

-softmax, logistic regression 등의 linear classifier를 사용한다.

-softmax classifier : 입력값을 probability distribution으로 제공하고 각 x에 대해서 p(y|x)를 예측해야 한다. (f_y : x가 class y에 속할 확률)

$$p(y|x) = \frac{exp(W_y x)}{\sum_{c=1}^{C} exp(W_c x)} = \frac{exp(f_y)}{\sum_{c=1}^{C} exp(f_c)}$$

2. Training

-모델의 최종 목표는 정확한 클래스를 예측하는 것(classifier)이다. 그러므로 정확한 클래스에 대해서 가장 높은 확률을 제공해야 한다. 그러므로 $-\log p(y|x)$의 값을 최소화해 한다.($\log p(y|x)$ 최대화, p(y|x) 최대화)

-log probability를 사용하면 손실함수를 적용하고 계산하기 편해짐.

3. Cross-entropy

$$H(p, q) = -\sum_{c=1}^{C} p(c) log q(c)$$

p : true probability distribution, q: computed model probability, q(c) : our estimated model

-classficiation의 경우는 특별하게 p=[0,0,...,0,1,0,...,0]으로 표현된다.

-p(c)는 올바른 클래스의 경우에만 1, 아닌 경우는 모두 0

-전체 데이터에 대한 cross-entropy loss

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -log(\frac{e^{fy_i}}{\sum_{c=1}^{C} e^{fc}})$$

4. Gradient descent

-전체 파라미터 θ가 각 단어들을 나타내는 word vector들로 구성되었다고 정의하면

$$0 - \begin{bmatrix} W_{1\cdot} \\ \vdots \\ W_{C\cdot} \end{bmatrix} - W \in \mathbb{R}^{Cd}$$

-손실함수의 gradient는

$$\nabla J(\theta) = \begin{bmatrix} \nabla W_{1\cdot} \\ \vdots \\ \nabla W_{C\cdot} \end{bmatrix} \in \mathbb{R}^{Cd}$$

-손실함수의 gradient를 사용해서 w값을 수정한다. W를 수정하면, class를 구분하는 선을 움직이는 것과 같다.

-word vectors는 one-hot vectors를 나타내고 아래의 수식을 통해 (linear) softmax classifier로 쉬운 classification을 위해서 중간 layer vector space 사이를 움직인다.

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+Vd}$$

Very large number of parameters!

Ⅱ. Neural networks introduction

선으로만 구분하기 어려운 문제가 발생->조금 더 복잡한 classifier가 필요->multi-layer neural network를 사용

1. Neural computation

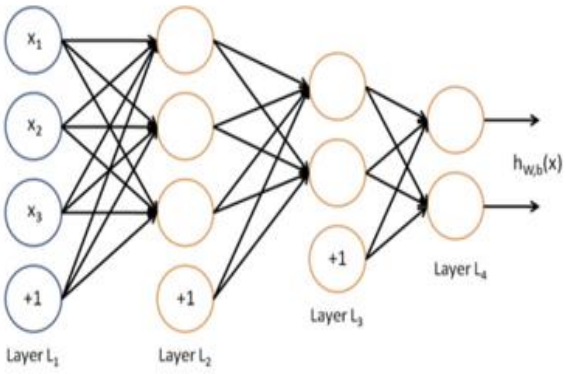
-$x_0, x_1, \ldots, x_n$: 다른 뉴런으로부터의 입력

-$w_0, w_1, \ldots, w_n$: 각 입력에 대한 가중치

-b : 해당 유닛의 bias

-f : activation function (non-linearity를 제공하기 위해 사용함. f를 사용하지 않는 경우, 아무리 많은 층을 거쳐도 최종적으로 linear transform이 된다.)

-여러 개의 unit을 함께 사용하면 hidden layer를 사용해서 각각의 유닛이 스스로 유의미한 요소를 학습할 수 있도록 한다. 최종 classifier가 올바른 결정을 내릴 수 있도록 하는 요소들을 학습한다. Cross-entropy loss를 최소화하는 방향으로 전체 유닛을 학습한다.



multilayer neural network

$$h_{w,b}(x) = f(w^{\mathsf{T}}x + b)$$

$$f(z) = \frac{1}{1+e^{-z}}$$

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$
etc.

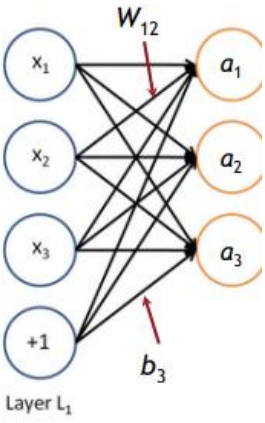
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

Activation f is applied element-wis

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$



Ⅲ. Named Entity Recognition(NER)

주어진 text에 대해서 이름을 찾아서 people, places, organization 등으로 분류.

1. Task 해결방법 : 각 단어에 한 번씩 접근하면서 classifier를 통해 class를 할당

   -여러 개의 단어로 구성된 이름의 경우, 연속적으로 동일한 class를 부여한 경우 하나의 Entity로 인식

   -이 방법은 실생활에서 사용하기에 한계가 있음. (단어를 자르는 곳, ambiguous한 단어들이 많아서 실제로 entity인지 확인하는 과정에도 어려움이 있다.)

2. Context 내에서 동작하는 word classifier를 사용

   -인접한 단어들을 통해서 단어의 ambiguity를 해결한다.

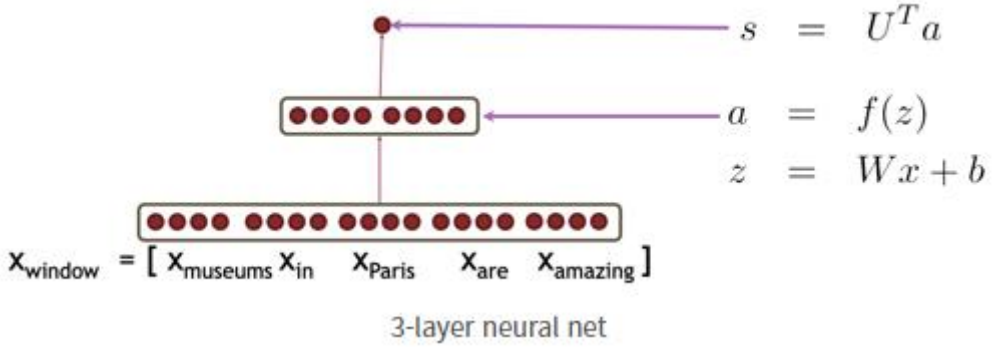
   -한 window내에 위치하는 단어들의 word vector들을 concatenate

   -새로 생성된 벡터를 softmax의 입력으로 제공해서 확률을 계산한다.

   가운데 위치하는 단어가 location entity면 높은 점수, 아닌 경우 낮은 점수를 제공

   Cross-entropy loss, stochastic gradient descent를 적용.

   -3-layer neural net에서 해당 task를 해결해본다. 중간층을 통해서 단어들 사이의 non-Linear한 관계를 학습할 수 있다.



3-layer neural net

3. $\nabla_\theta J(\theta)$ 를 계산하는 방법

-n개의 input, 1개의 output인 경우 : $f(x) = f(x_1, x_2, \ldots, x_n)$

$$\frac{\partial f}{\partial x} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]$$

-n개의 input, m개의 output인 경우 : $f(x) = [f_1(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n)]$

Matrix of partial derivative=Jacobian matrix

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \cdots & & \cdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\left(\frac{\partial f}{\partial x}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

-h=f(z)일 때,

$$\frac{\partial h}{\partial z} = \frac{\partial f'(z_i)}{\partial z} = \begin{pmatrix} f'(z_1) & \cdots & 0 \\ \cdots & & \cdots \\ 0 & \cdots & f'(z_n) \end{pmatrix} = diag(f'(z))$$