Linear models: Recap

Linear models:

- Perceptron

$$\mathrm{score}(y, \boldsymbol{x}; \boldsymbol{\theta}) =$$

- Naïve Bay

$$\log P(y|\boldsymbol{x};\boldsymbol{\theta}) = \log P(\boldsymbol{x}|y;\phi) + \mathrm{lo} \quad B(\boldsymbol{x}) + \boldsymbol{\theta}\cdot\boldsymbol{f}(\boldsymbol{x}, y)$$

- Logistic Regression

$$\log P(y|\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}\cdot\boldsymbol{f}(\boldsymbol{x}, y) - \log \sum_{y'\in\mathcal{Y}} \exp \boldsymbol{\theta}\cdot\boldsymbol{f}(\boldsymbol{x}, y')$$

Features and weights in linear models: Recap

- Feature represe

$$\boldsymbol{f}(\boldsymbol{x}, y = 1) = [\boldsymbol{x}; \underbrace{0; 0; \cdots ; 0}_{(K-1}]$$

$$\boldsymbol{f}(\boldsymbol{x}, y = 2) = [0; 0; \cdots \underbrace{}_{2)\times V}]$$

$$\underbrace{}_{(K}$$

- Weights: $\boldsymbol{\theta}$

$$\boldsymbol{\theta} = [\underbrace{\theta_1; \theta_2; \cdots ; \theta_V}_{y=1}; \underbrace{\theta_1; \theta_2; \cdots ; \theta_V}_{y=2}; \cdots ; \underbrace{\theta_1; \theta_2; \cdots ; \theta_V}_{y=K}]$$

Rearranging the features and weights

- Represent the features $\boldsymbol{x}$ as a *column* vector of length V, and represent the weights as a $\boldsymbol{\Theta}$ as K

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_V \end{bmatrix} \qquad \boldsymbol{\Theta} = \begin{array}{r c} & \begin{array}{cccc} x_1 & x_2 & \cdots & x_V \end{array} \\ \begin{array}{r} y = 1 \\ y = 2 \\ \\ y = K \end{array} & \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \cdots & \theta_{1,V} \\ \theta_{2,1} & \theta_{2,2} & \cdots & \theta_{2,V} \\ \cdots & \cdots & \cdots & \cdots \\ \theta_{K,1} & \theta_{K,2} & \cdots & \theta_{K,V} \end{bmatrix} \end{array}$$

- What is $\boldsymbol{\Theta}\boldsymbol{x}$?

Scores for each class

- Verify that $\psi_1, \psi_2, \cdots, \psi_K$ corres ach class

$$\mathbf{\Psi} = \mathbf{\Theta}\boldsymbol{x} =$$

Implementation in Pytorch

Digression: Matrix multiplication

- Matrix with m rows and n columns

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p},$$

where C_{ij}

- Example:

$$\begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} & & \\ 1 & 2 & 4 \end{bmatrix}$$

Digression: 3-D matrix multiplication

Tensor shape: (batch-size, sentence-length, embedding size)

SoftMax

- SoftMax, also kn

$$\mathrm{SoftMax}_i(\boldsymbol{\psi}) = \frac{\exp \psi_i}{\;}$$

for $i = 1, 2, \quad , K$

- Applying Sof
distribution

$$\mathrm{SoftMax}(\boldsymbol{\Psi}) = \begin{bmatrix} P(y=1) \\ P(y=2) \\ \cdots \\ P(y=K) \end{bmatrix}$$

- Verify this is exactly logistic regression

Logistic regression as a neural network

$$\boldsymbol{y} = \mathrm{SoftMax}(\boldsymbol{\Theta}\boldsymbol{x})$$
$$V = 5 \; K = 3$$

Going deep

- There is no reason w

https://eduassistpro.github.io/
Add WeChat edu_assist_pro

$$
\begin{aligned}
\boldsymbol{z} &= \sigma(\boldsymbol{\Theta}_1 \boldsymbol{x}) \\
\boldsymbol{y} &= \mathsf{SoftMax}(\boldsymbol{\Theta}_2 \boldsymbol{z})
\end{aligned}
$$

Going even deeper

- There is no reason w

$$\boldsymbol{z}_1 = \sigma(\boldsymbol{\Theta}_1 \boldsymbol{x})$$
$$\boldsymbol{z}_2 = \sigma(\boldsymbol{\Theta}_2 \boldsymbol{z}_1)$$
$$\boldsymbol{y} = \mathrm{SoftMax}(\boldsymbol{\Theta}_3 \boldsymbol{z}_2)$$

- But why?

Non-linear classification

Linear models like Logis high-dimensional vector space, and they are expressive enough and work well for many NLP problems, why do we need more complex non-linear models?

- There have been rapid advances in , a family of nonlinear me put through mu

- Deep learning facilitates the incorpora **ord embeddings**, which are dense vector r ords, that can be learned from massive amount
  - It has evolved from early static embeddings (e.g., Word2vec, Glove) to recent dynamtic embeddings (ELMO, BERT, XLNet)

- Rapid advances in specialized hardware called graphic processing units (GPUs). Many deep learning models can be implemented efficiently on GPUs.

Feedforward Neural networks: an intuitive justification

- In image classification, instead of using the input (pixels) to predict the image type directly, you can imagi that you can predict the shapes of parts of an ima hand, ear.

- In text process say we want to classify movie reviews (or m into a label set of {*Good*, *Bad*, tead predicting these labels directly, we first predict a set of comp such as the story, acting, soundtrack, cinematography, etc. from raw input (words in the text).

Face Recognition

https://eduassistpro.github.io/
Add WeChat edu_assist_pro

Feedforward neural networks

Formally, this is what we do:

- **Use the text x to predict the featur** ecifically, train a logistic regression classifier to compute or each $k \in \{1, 2, \cdot$ $, K_z$

- **Use the featu** Train a logistic regression cl nknown or hidden, so we will use the $P(z|$ eatures.

Caveat: it's easy to demonstrate what this is wha for image processing, but it's hard to show this is what's actually going on in language processing. Interpretability is a major issue in neural models for language processing.

The hidden layer: computing the composite features

- If we assume each z_k is binary, that is, $z_k \in \{0, 1\}$, then $P(z_k|\boldsymbol{x})$ can be modeled with binary logistic regression:

$$P(z_k = 1|\boldsymbol{x}; \boldsymbol{\Theta}^{(x\to z)}) = \sigma(\boldsymbol{\theta}$$

$$= (1 \quad _k \quad))^{-1}$$

- The weight m

cted by stacking (not concatenating, as in linea vectors for each z_k,

$\boldsymbol{\Theta}^{(x\to z)} = \left[\boldsymbol{\theta}_1^{x\to z}, \boldsymbol{\theta}_2^{x\to z}, \cdots, \boldsymbol{\theta}_{K_z}^{x\to}\right.$
- We assume an offset/bias term is included in $\boldsymbol{x}$ and its parameter is included in each $\boldsymbol{\theta}_k^{x\to z}$

Notations: $\boldsymbol{\Theta}^{(x\to z)} \in \mathbb{R}^{k_z \times V}$ is a real number matrix with a dimension of k_z rows and V columns

The output layer

- The output layer is computed by the multiclass logistic regression probability

$$P(y = j \mid \boldsymbol{z}; \boldsymbol{\Theta}^{(z\to y)}, \boldsymbol{b}) = \frac{\exp(\boldsymbol{\theta}_j^{(z\to y)} \cdot \boldsymbol{z} + b_j)}{\sum_{j'} \exp(\boldsymbol{\theta}_{j'}^{(z\to y)} \cdot \boldsymbol{z} + b_{j'})}$$

- The weight matrix $\boldsymbol{\Theta}^{(z\to y)} \in \mathbb{R}^{K_y \times K_z}$ is constructed by stacking weight vectors for each class
  $\boldsymbol{\Theta}^{(z\to y)} = \left[\boldsymbol{\theta}_1^{z\to y}, \boldsymbol{\theta}_2^{z\to y}, \cdots, \boldsymbol{\theta}_{K_y}^{z\to y}\right]$

- The vector of probabilities over each possible value of y is denoted:
  $P(\boldsymbol{y} \mid \boldsymbol{z}; \boldsymbol{\Theta}^{(z\to y)}, \boldsymbol{b}) = \mathrm{SoftMax}(\boldsymbol{\Theta}^{(z\to y)}\boldsymbol{z} + \boldsymbol{b})$

Activation functions

- Sigmoid: The ran).

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Tanh: The range of the tanh activation functi $-1, 1)$

- ReLU: The **rectified linear unit (R** or negative inputs, and linear for positive inputs

$$ReLU(x) = max(x, 0) = \begin{cases} 0 & x < 0 \\ x & otherwise \end{cases}$$

Sigmoid and tanh are sometimes described as **squashing functions**.

Activation functions in Pytorch


```
from torch import nn
import torch

input = torch.ra         4)
sigmoid = nn.Si
output = sigmoi

tanh = nn.Tanh()
output = tanh(input)

relu = nn.ReLU()
output = relu(input)
```

Output and loss functions

In a multi-class classific

probabilistic distribution over possible labels. It works well together with negative conditional likelihood (just like logistic regression)

$$= \sum^{N} \log P(y^{(i}$$

or **cross entropy l**

$$\tilde{y}_j \triangleq Pr(y =$$

$$-\mathcal{L} = -\sum_{i=1}^{N} \boldsymbol{e}_{y^{(i)}} \cdot \log \tilde{\boldsymbol{y}}$$

where $\boldsymbol{e}_{y^{(i)}}$ is a **one-hot vector** of zeros with a value of one at the position $y^{(i)}$

Output and loss function

- There are alternatives to SoftMax and cross-entropy loss, just as there are alternatives in linear models.

- Pairing an affine transformation (rememb h a margin loss:

$$\mathbf{\Psi}(y; \boldsymbol{x}^{(i)}, \Theta) = \boldsymbol{\theta}_y^{(z \rightarrow y)} \cdot \boldsymbol{z} + b_y$$

$$\ell_{\text{MARGIN}}(\Theta; \boldsymbol{x}^{(i)}, y^{(i)}) = \max_{y \neq y^{(i)}} \left(1 + \Psi(y; \boldsymbol{x}^{(i)}, \Theta) - \Psi(y^{(i)}; \boldsymbol{x}^{(i)}, \Theta)\right)_+$$

Inputs and Lookup layers

- Assuming a bag-of-words model, when the input $\boldsymbol{x}$ is the count of each word x_i (This can be generalized to feature count).

- To compute the hidden unit z_k:

- This text representation is particularly s ard networks.
- The connections from word j to each of the hidden units z_k form a vector $\boldsymbol{\theta}_j^{(x \to z)}$ is sometimes described as the embedding of word j. Word embeddings can be learned from unlabeled data, using techniques such as Word2Vec and GLOVE.

Alternative text representations

- Alternatively, a text can be represented as a sequence of word tokens $w_1, w_2, w_3, \cdots, w_M$. This view is useful for models such as **Convolutional Neural Netw** **vNets**, which processes text as a sequence.

- Each word tok tor $\boldsymbol{e}_{w_m}$, with dimensi represented e-hot vectors: $\boldsymbol{W} = [\boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \cdots, \boldsymbol{e}_{w}$

- To show that this is equivalent to the bag-o can recover the word count from the matrix-vector product $\boldsymbol{W}[1, 1, \cdots, 1]^\top \in R^V$.
- The matrix product $\boldsymbol{\Theta}^{\boldsymbol{x} \rightarrow \boldsymbol{z}} \boldsymbol{W} \in R^{k_z \times M}$ contains horizontally concatenated embeddings of each word in the document.