Linear models: Recap

Linear models:

- Perceptron

$$\text{score}(y, \boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y)$$

- Naïve Bayes:

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

- 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 representation: $\boldsymbol{f}(\boldsymbol{x}, y)$

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

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

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

- 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 $K \times V$ matrix

Assignment Project Exam Help

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

https://powcoder.com
Add WeChat powcoder

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

Scores for each class

- Verify that $\psi_1, \psi_2, \ldots, \psi_K$ correspond to the scores for each class

$$\boldsymbol{\Psi} = \boldsymbol{\Theta} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{\theta}_1 \cdot \boldsymbol{x} = \psi_1 \\ \boldsymbol{\theta}_2 \cdot \boldsymbol{x} = \psi_2 \\ \cdots \\ \boldsymbol{\theta}_3 \cdot \boldsymbol{x} = \psi_K \end{bmatrix}$$

Implementation in Pytorch


```
In [48]: weights = torch.randn(3,9)
         print(weights)
         input = torch.randn(9)
         print(input)
         output = torch.matmul(weights, input)
         print(output)
         softmax = nn.Softmax(dim=0)
         probs = softmax(output)
         print(probs)

tensor([[-0.3518, -0.3291,  1.6937, -0.9947, -0.1390, -0.7788, -0.0252, -0.1557,
         -1.2138],
        [-1.2000,  1.3527,  1.9529, -1.3182,  0.1101, -0.7105, -0.4409,  0.9753,
         -0.0082],
        [-0.1561, -0.9144,  0.4833,  0.9997,  0.6840, -1.4633,  1.1353,  0.3069,
         -0.0584]])
tensor([ 0.0148,  0.0565, -0.6462, -0.0155, -0.5532, -0.8514, -0.1339,  0.5056,
         0.6025])
tensor([-1.1695, -0.1363,  0.5428])
tensor([0.1069, 0.3005, 0.5926])
```

Digression: Matrix multiplication

- Matrix with m rows and n columns:

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

where $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$

- Example:

$$\begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 7 \\ 1 & 2 & 4 \end{bmatrix}$$

Digression: 3-D matrix multiplication

```
In [27]: import torch
         input = torch.randint(5, (2, 3, 4))
         print(input)
         mat2 = torch.randint(5, (2, 4, 3))
         print(mat2)
         out = torch.bmm(input,mat2)
         print(out)

tensor([[[0, 1, 3, 2],
         [3, 1, 4, 1],
         [0, 1, 2, 0]],

        [[3, 2, 3, 4],
         [4, 3, 2, 4],
         [3, 0, 2, 3]]])
tensor([[[0, 2, 0],
         [4, 4, 2],
         [2, 2, 4],
         [1, 1, 4]],

        [[3, 0, 4],
         [3, 0, 1],
         [0, 0, 4],
         [2, 4, 2]]])
tensor([[[12, 12, 22],
         [13, 19, 22],
         [ 8,  8, 10]],

        [[23, 16, 34],
         [29, 16, 35],
         [15, 12, 26]]])
```

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

SoftMax

- SoftMax, also known as normalized exponential function.

$$\text{SoftMax}_i(\boldsymbol{\psi}) = \frac{\exp \psi_i}{\sum_j^K \exp \psi_j}$$

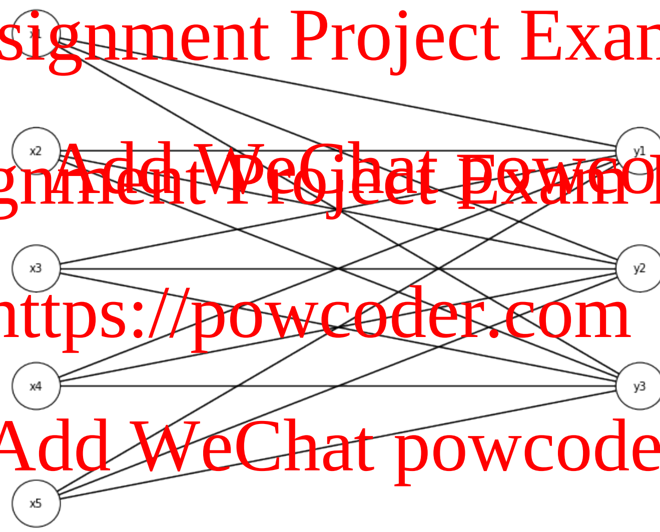
  for $i = 1, 2, \cdots, K$

- Applying SoftMax turns the scores into a probabilistic distribution:

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

- Verify this is exactly logistic regression

Logistic regression as a neural network



$$\boldsymbol{y} = \text{SoftMax}(\boldsymbol{\Theta x})$$

$$V = 5 \; K = 3$$

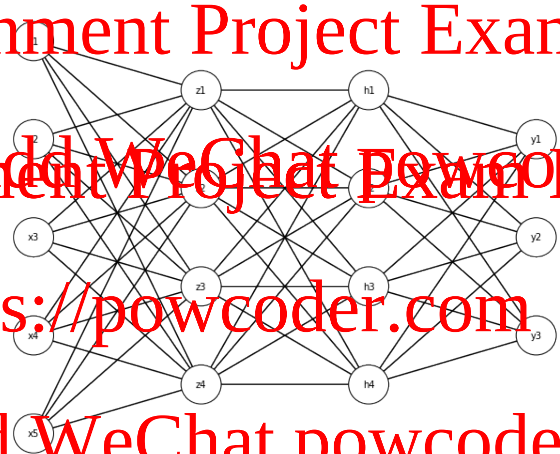
Going deep

- There is no reason why we can't add layers in the middle

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

Going even deeper

- There is no reason why we can't add layers in the middle



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

- But why?

Non-linear classification

Linear models like Logistic regression can map data into a 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?

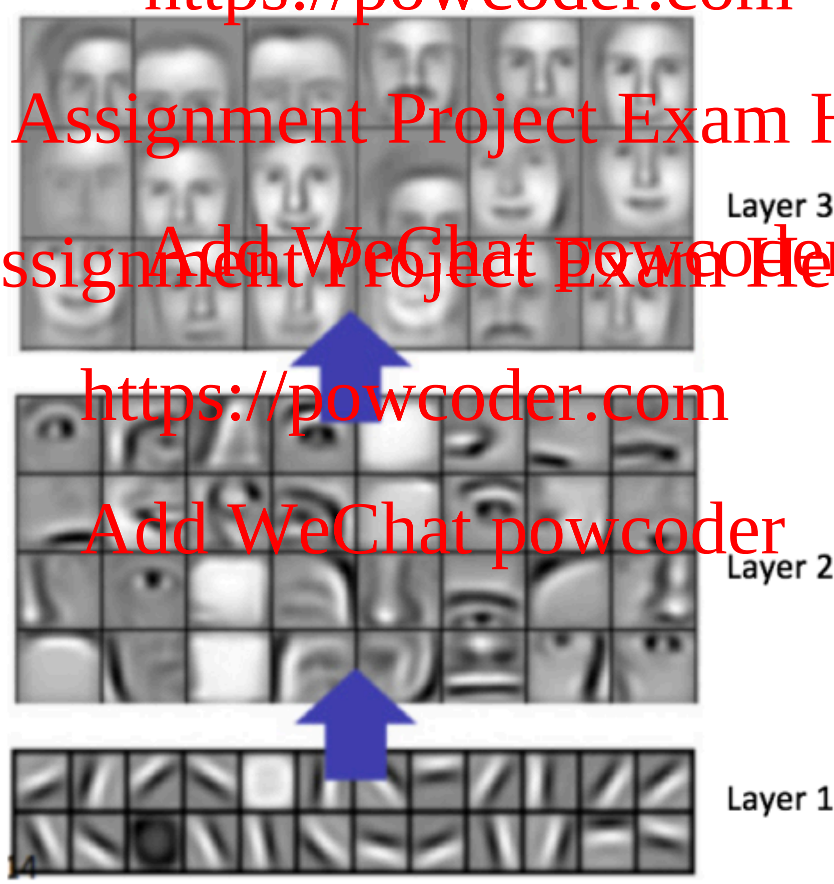
- There have been rapid advances in **deep learning**, a family of nonlinear methods that learn complex functions of the input through multiple layers of computation.
- Deep learning facilitates the incorporation of **word embeddings**, which are dense vector representations of words, that can be learned from massive amounts of unlabeled data.
  - 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 imagine a scenario that you can predict the shapes of parts of an image, mouth, hand, ear.

- In text processing, we can imagine a similar scenario. Let's say we want to classify movie reviews (or movies themselves) into a label set of $\{Good, Bad, OK\}$. Instead predicting these labels directly, we first predict a set of composite features such as the story, acting, soundtrack, cinematography, etc. from raw input (words in the text).

Face Recognition



Feedforward neural networks

Formally, this is what we do:

- **Use the text $\boldsymbol{x}$ to predict the features $\boldsymbol{z}$**. Specifically, train a logistic regression classifier to compute $P(z_k|\boldsymbol{x})$, for each $k \in \{1, 2, \cdots, K_z\}$
- **Use the features $\boldsymbol{z}$ to predict the label y**. Train a logistic regression classifier to compute $P(y|\boldsymbol{z})$. $\boldsymbol{z}$ is unknown or hidden, so we will use the $P(\boldsymbol{z}|\boldsymbol{x})$ as the features.

Caveat: it's easy to demonstrate what this is what the model does 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:

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

- The weight matrix $\boldsymbol{\Theta}^{(x\to z)} \in \mathbb{R}^{k_z \times V}$ is constructed by stacking (not concatenating, as in linear models) the weight vectors for each z_k, $\boldsymbol{\Theta}^{(x\to z)} = \left[\boldsymbol{\theta}_1^{x\to z}, \boldsymbol{\theta}_2^{x\to z}, \cdots, \boldsymbol{\theta}_{K_z}^{x\to z}\right]^T$
- 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

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

- The weight matrix $\boldsymbol{\Theta}^{(z \to y)} \in \mathbb{R}^{k_y \times k_z}$ again is constructed by stacking weight vectors for each y_k:
$\boldsymbol{\Theta}^{(z \to y)} = \left[\boldsymbol{\theta}_1^{z \to y}, \boldsymbol{\theta}_2^{z \to y}, \cdots, \boldsymbol{\theta}_{K_y}^{z \to y}\right]^\top$
- The vector of probabilities over each possible value of y is denoted:
$P(\boldsymbol{y} | \boldsymbol{z}; \boldsymbol{\Theta}^{(z \to y)}, \boldsymbol{b}) = \mathrm{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$

Activation functions

- Sigmoid: The range of sigmoid function is $(0, 1)$.

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

- Tanh: The range of the tanh activation function is $(-1, 1)$

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- ReLU: The **rectified linear unit (ReLU)** is zero for 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.randn(4)
sigmoid = nn.Sigmoid()
output = sigmoid(input)

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

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

Output and loss functions

In a multi-class classification setting, a softmax output produces a probabilistic distribution over possible labels. It works well together with negative conditional likelihood (just like logistic regression)

$$-\mathcal{L} = -\sum_{i=1}^{N} \log P(y^{(i)} | \boldsymbol{x}^{(i)}; \Theta)$$

or **cross entropy loss**:

$$\tilde{y}_j \triangleq Pr(y = j | \boldsymbol{x}^{(i)}; \Theta)$$

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

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 (remember perceptron) with a margin loss:

$$\boldsymbol{\Psi}(y; \boldsymbol{x}^{(i)}, \Theta) = \theta_y^{(z \to y)} \cdot \boldsymbol{z} + b_y$$

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

Inputs and Lookup layers

- Assuming a bag-of-words model, when the input $\boldsymbol{x}$ is the count of each word x_j (This can be generalized to feature count).
- To compute the hidden unit z_k:

$$z_k = \sum_{j=1}^{V} \theta_{j,k}^{x \to z} x_j$$

- This text representation is particularly suited for feedforward 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 Networks**, or **ConvNets**, which processes text as a sequence.
- Each word token w_m is represented as a one-hot vector $\boldsymbol{e}_{w_m}$, with dimension V. The complete document can be represented by the horizontal concatenation of these one-hot vectors: $\boldsymbol{W} = [\boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \cdots, \boldsymbol{e}_{w_m}] \in R^{V \times M}$.
- To show that this is equivalent to the bag-of-words model, we can recover the word count from the matrix-vector product $\boldsymbol{W}[1, 1, \cdots, 1]^\top \in R^V$.
- The matrix product $\boldsymbol{\Theta}^{\boldsymbol{x} \to \boldsymbol{z}} \boldsymbol{W} \in R^{k_z \times M}$ contains horizontally concatenated embeddings of each word in the document.