WestlakeNLP

Natural Language Processing

Yue Zhang
Westlake University

西湖大學
WESTLAKE UNIVERSITY

Chapter 13

Neural Networks

Contents

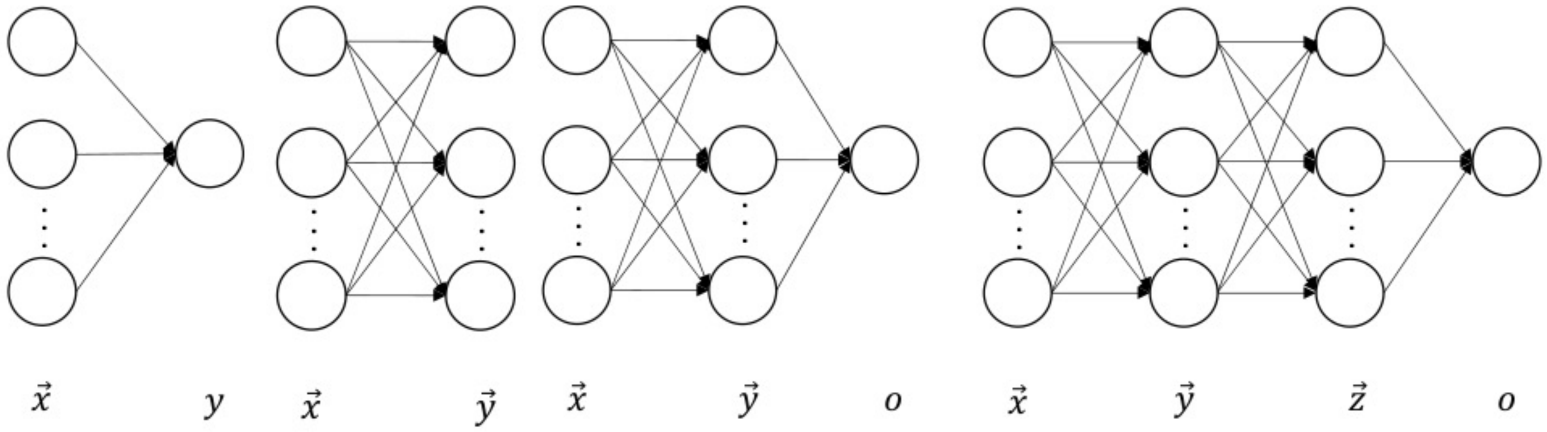
WestlakeNLP

Contents

WestlakeNLP

Multi-layer perceptron

- From a single layer to multiple layers



$y = f(\vec{\theta} \cdot \vec{x})$

$y_i = f(\vec{\theta_i} \cdot \vec{x})$

$$\begin{cases} y_i = f\left(\vec{\theta_i} \cdot \vec{x}\right); \\ o = g(\vec{\theta^o} \cdot \vec{y}) \end{cases}$$

$$\begin{cases} y_j = f\left(\vec{\theta_j^y} \cdot \vec{x}\right); \\ z_i = g\left(\vec{\theta_i^z} \cdot \vec{y}\right); \\ o = h(\vec{\theta^o} \cdot \vec{z}) \end{cases}$$

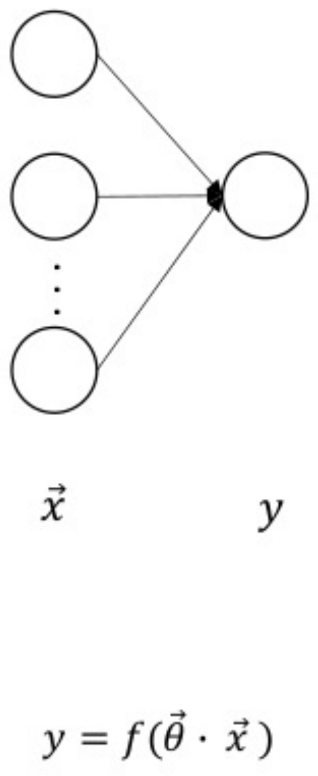
- MLP model can learn non-linear mappings between the input $\vec{x}$ and the output o

Single-layer perceptron

WestlakeNLP

Generalized linear model in Chapter 4

- **Input layer**: $\vec{x}$ - receives input data and represents them using vectors
- **Output unit**: y - makes predictions according to the features extracted from the input layer.
- **Mapping function**: $y = f(\vec{\theta} \cdot \vec{x})$
- **Task**: text classification ($y = +1/-1$)



Multi-outputs

WestlakeNLP

- **Tasks:**

$y_1 = f(\overrightarrow{\theta_1} \cdot \vec{x})$ sentiment

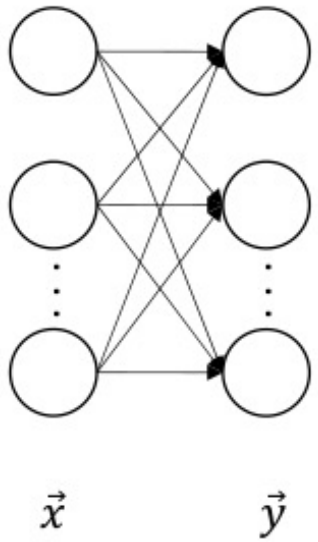
positive/negative

$y_2 = f(\overrightarrow{\theta_2} \cdot \vec{x})$ document class

sports/politics/$\cdots$

...

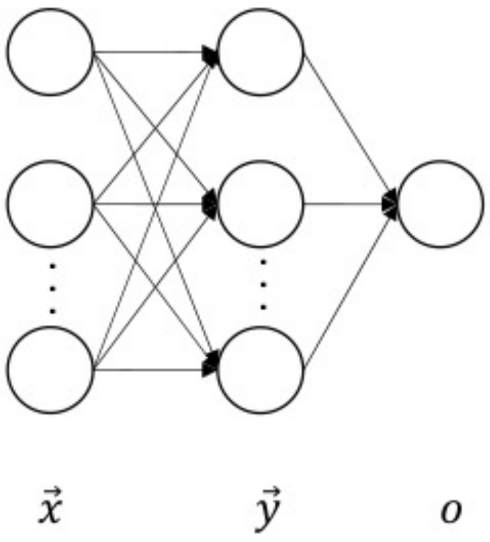
$y_i = f(\overrightarrow{\theta_i} \cdot \vec{x})$ ...



$y_i = f(\overrightarrow{\theta_i} \cdot \vec{x})$

Two-layers

- **Input layer**: $\vec{x}$ - receives input data and represents them using vectors
- **Hidden layers**: $\vec{y}$ - induces useful non-linear features from the input vectors
- **Output layer**: o - makes predictions according to the features extracted from the hidden layers.
- **Task**: o ______ *is liked by John*

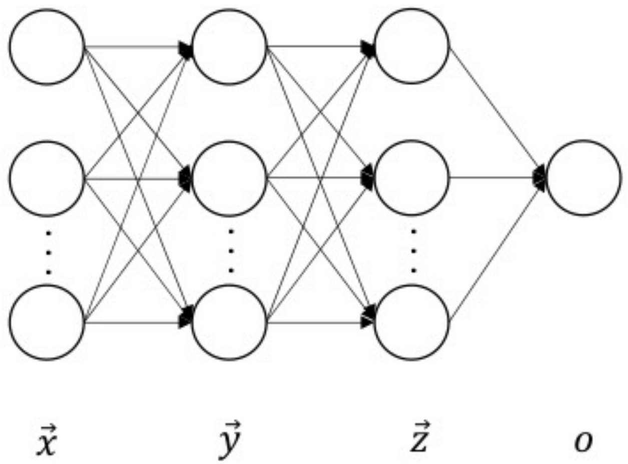


$$\begin{cases} y_i = f\left(\overrightarrow{\theta_i} \cdot \vec{x}\right); \\ o = g(\overrightarrow{\theta^o} \cdot \vec{y}) \end{cases}$$

Three-layers

WestlakeNLP

- **Input layer**: $\vec{x}$ - receives input data and represents them using vectors
- **Hidden layers**: $\vec{y}, \vec{z}$ - induces useful non-linear features from the input vectors
- **Output layer**: o - makes predictions according to the features extracted from the hidden layers.



$$
\begin{cases}
y_j = f\left(\overrightarrow{\theta_j^y} \cdot \vec{x}\right); \\
z_i = g\left(\overrightarrow{\theta_i^z} \cdot \vec{y}\right); \\
o = h(\overrightarrow{\theta^o} \cdot \vec{z})
\end{cases}
$$

Activation function

WestlakeNLP

- Non-linear activation functions

Name	Function
identity	$identity(x) = x$
rectify	$ReLU(x) = \max(x, 0)$
tanh	$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
softmax	$softmax([x_1, x_2, \ldots, x_n]) = [\frac{e^{x_1}}{\sum_{k=1}^{n} e^{x_k}}, \frac{e^{x_2}}{\sum_{k=1}^{n} e^{x_k}}, \ldots, \frac{e^{x_n}}{\sum_{k=1}^{n} e^{x_k}}]$
ELU	$ELU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0. \end{cases}$
softplus	$softplus(x) = \log(1 + e^x)$

Contents

WestlakeNLP

Neural network notation

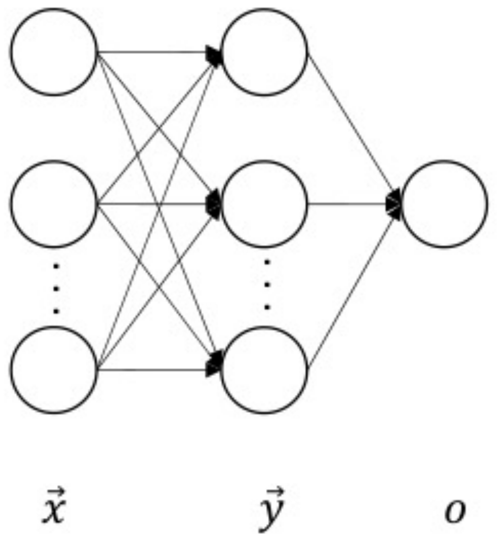
Matrix-vector notation

- Concatenation of *column* vectors

$$\mathbf{W}^y = [\vec{\theta}_1; \vec{\theta}_2; \ldots; \vec{\theta}_m]^T,$$

- Single layer perceptron

$$\mathbf{y} = f(\mathbf{W}^y \mathbf{x}),$$



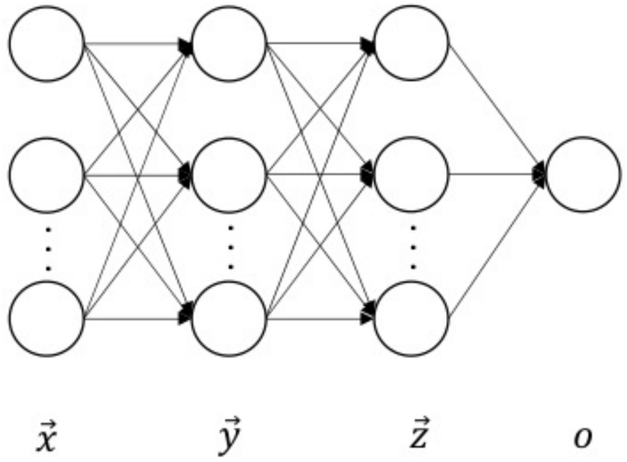
$$\begin{cases} y_i = f\left(\vec{\theta_i} \cdot \vec{x}\right); \\ o = g(\vec{\theta^o} \cdot \vec{y}) \end{cases}$$

Matrix Vector Notation

- Multi-layer perceptron, we use **h** to denote hidden layers as:

$$\mathbf{h}^1 = f(\mathbf{W}^y\mathbf{x})$$
$$\mathbf{h}^2 = g(\mathbf{W}^z\mathbf{h}^1)$$

$$o = h(\mathbf{v}^T\mathbf{h}^2)$$



$$\begin{cases} y_j = f\left(\overrightarrow{\theta_j^y} \cdot \vec{x}\right); \\ z_i = g\left(\overrightarrow{\theta_i^z} \cdot \vec{y}\right); \\ o = h(\overrightarrow{\theta^o} \cdot \vec{z}\,) \end{cases}$$

Matrix Vector Notation

WestlakeNLP

- Multi-class classifier:

$$o = \langle o_1, o_2, \cdots, o_m \rangle$$

$$\mathbf{W}^o = [v_1; v_2; \cdots; v_m]^T$$

- As a result,

$$o = \mathbf{W}^o \mathbf{h}$$

- Applying softmax function:

$$\mathbf{p} = softmax(\mathbf{o})$$

Correlation with linear classifier

WestlakeNLP

- For binary classification, MLP differs from linear perceptron only in the use of hidden layers.
- For multi-class classification
  - Single layer perceptron extends feature vector (Chapter 3)
  - Multi-layer perceptron extends output layer W^o (Chapter 13)
- Duplicating the input feature vector m times equals the duplication of the model parameter vector m times.

Correlation with linear classifier

$$score(c_1) = \vec{\theta} \cdot \vec{\phi}(x, c_1) \qquad score(c_1) = \overrightarrow{\theta_1} \cdot \vec{\phi}(x)$$

$$score(c_2) = \vec{\theta} \cdot \vec{\phi}(x, c_2) \qquad score(c_2) = \overrightarrow{\theta_2} \cdot \vec{\phi}(x)$$

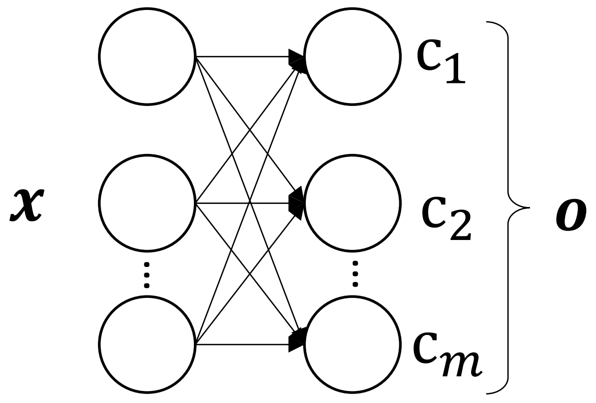
$$\dots \qquad \Rightarrow \qquad \dots$$

$$score(c_m) = \vec{\theta} \cdot \vec{\phi}(x, c_m) \qquad score(c_m) = \overrightarrow{\theta_m} \cdot \vec{\phi}(x)$$

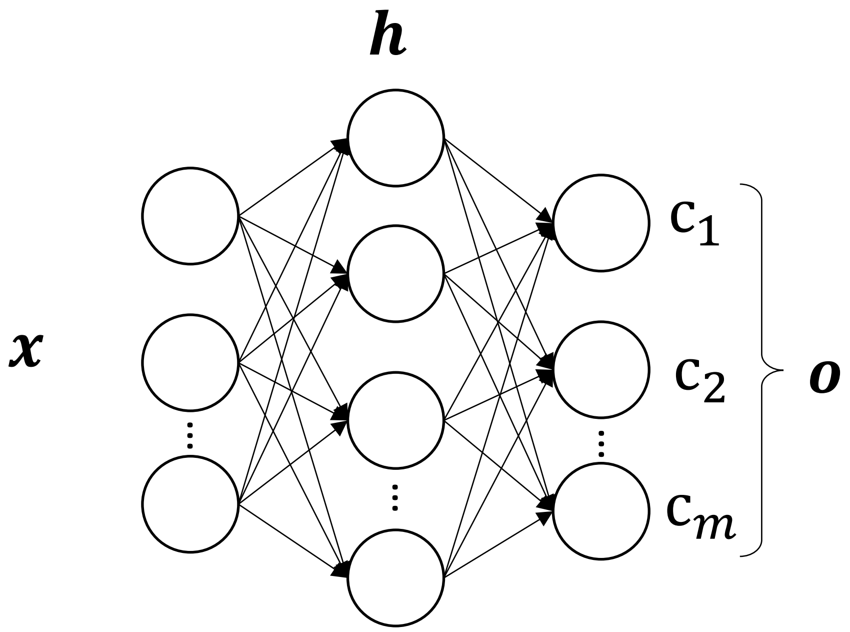
- Where $\vec{\phi}(x)$ denotes the input feature representation without combining the class label, and $\overrightarrow{\theta_i}$ denotes the corresponding weight vector for $\vec{\phi}(x, c_i), i \in [1, \dots, m]$.

Correlation with linear classifier

As a result, no matter for binary or multi-class classification, MLP differs from linear perceptron only in the use of hidden layers.



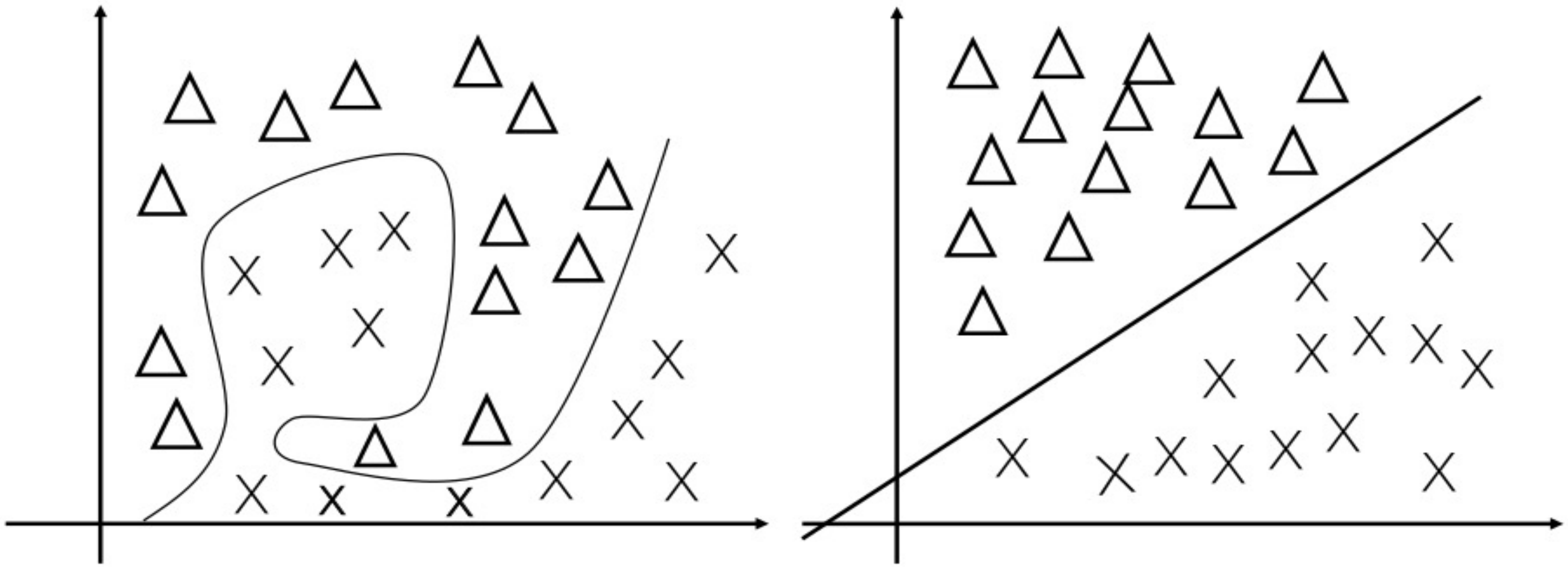
Single-layer perceptron for multiclass classification



Multi-layer perceptron for multiclass classification

Characteristics of neural hidden layers and their representation power

- Low dimensional
- Dense, with nodes in real numbers
- Dynamically calculated



(a) input vector space (b) hidden vector space

The effect of hidden layer representation

Contents

Training multi-layer perceptrons

The principles of training the generalized perceptron model can be applied for the training of multi-layer perceptrons.

- Training set: $D = \{(x_i, c_i)\}|_{i=1}^{N}$
- Input feature vector: $\mathbf{x}_i$
- Gold-standard output label: c_i
- Model target $P(c|\mathbf{x})$
- Parameterization: MLP
- Log-likelihood loss with L_2 regularization:

$$L = -\log P(D) + \lambda||\Theta||^2 = -\sum_{i=1}^{N} \log P(c_i|\mathbf{x}_i) + \lambda||\Theta||^2$$

Training multi-layer perceptrons using SGD

The principle of SGD

- Given a training set D
- The algorithm goes through all the training instances for multiple iterations
- For each training instance, calculate the gradient of a local loss with respect to each model parameter
- Update the model parameters with their respective gradients, possibly with a learning rate factor.

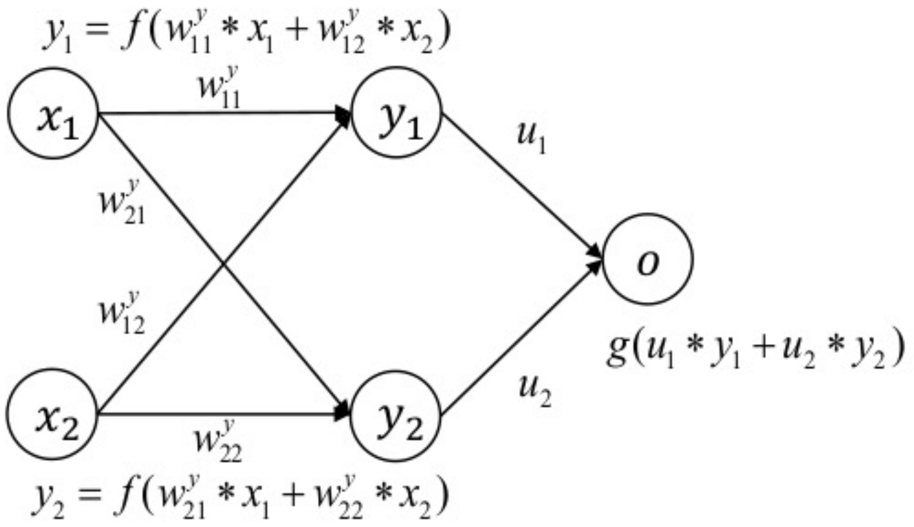
Training a neural network

- Key issue: feed gradient for every model parameter
- Take a simple network for example.

$$\boldsymbol{y} = (\boldsymbol{W}^y \boldsymbol{x})^2$$

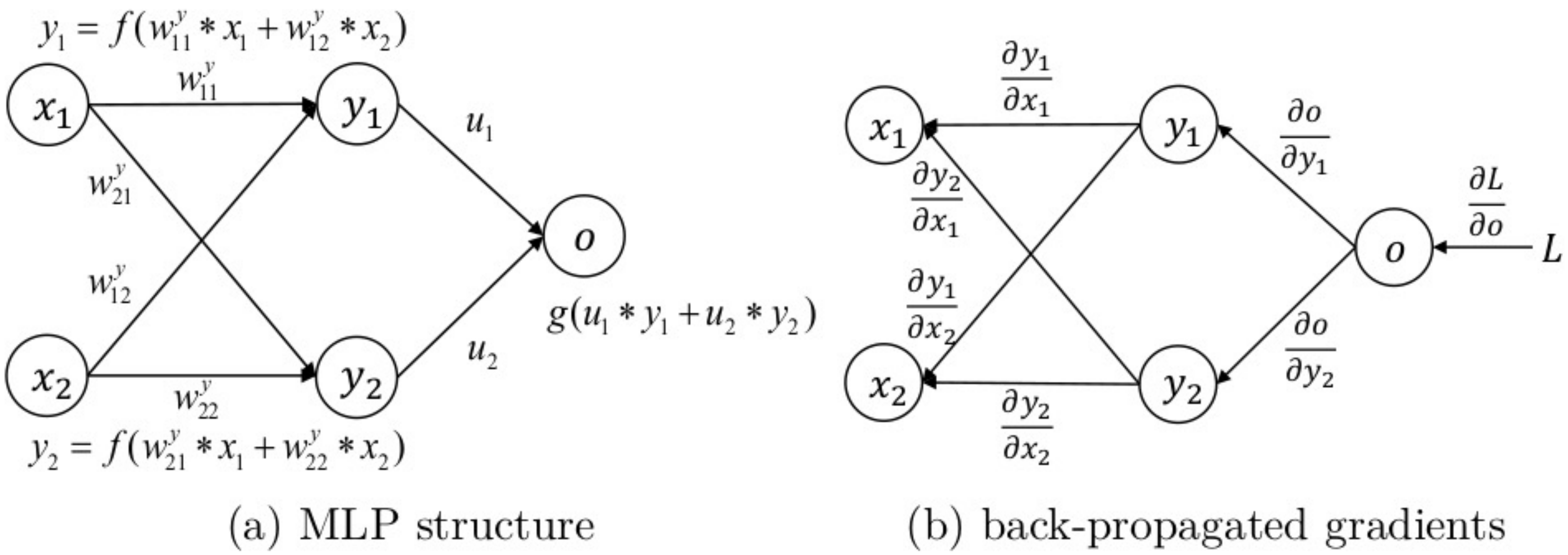
$$\boldsymbol{o} = \sigma(\boldsymbol{u}\boldsymbol{y})$$

$$\boldsymbol{W}^y = \begin{pmatrix} W_{11}^y & W_{12}^y \\ W_{11}^y & W_{12}^y \end{pmatrix} \qquad \boldsymbol{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$



Computation graph for a neural network

Now calculate gradients



Loss function

WestlakeNLP

Given a training instance $(\mathbf{x}_i, c_i)$, the loss is

$$L(\mathbf{x}_i, c_i, \Theta) = -\log P(c_i|\mathbf{x}_i) + \lambda\|\Theta\|^2$$

$$= -\log\sigma(u_1 y_1 + u_2 y_2) + \lambda\|\Theta\|^2$$

$$= -\log\sigma\left(u_1(w_{11}^y x_1 + w_{12}^y x_2)^2 + u_2(w_{21}^y x_1 + w_{22}^y x_2)^2\right)$$

$$+\lambda\left((w_{11}^y)^2 + (w_{12}^y)^2 + (w_{21}^y)^2 + (w_{22}^y)^2 + (u_1)^2 + (u_2)^2\right)$$

Gradients

WestlakeNLP

The local gradients are

$$\frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial u_1} = \frac{\partial -\log o}{\partial u_1} + \frac{\partial \|\Theta\|^2}{\partial u_1}$$

$$= -\frac{\partial \left((u_1 y_1 + u_2 y_2) - \log\left(1 + \exp(u_1 y_1 + u_2 y_2)\right) \right)}{\partial u_1} + 2\lambda u_1$$

$$= -\left(y_1 - \frac{\exp(u_1 y_1 + u_2 y_2)}{1 + \exp(u_1 y_1 + u_2 y_2)} y_1 \right) + 2\lambda u_1$$

$$= -(1 - o) y_1 + 2\lambda u_1$$

$$\frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial u_2} = -(1 - o) \cdot y_2 + 2\lambda u_2$$

Gradients

$$\frac{\partial L(\mathbf{x}_i,c_i,\Theta)}{\partial w_{11}^{y}} = -(1-o)\cdot(u_1\cdot 2(w_{11}^{y}x_1 + w_{12}^{y}x_2)\cdot x_1) + 2\lambda w_{11}^{y}$$

$$\frac{\partial L(\mathbf{x}_i,c_i,\Theta)}{\partial w_{11}^{y}} = -(1-o)\cdot(u_1\cdot 2(w_{11}^{y}x_1 + w_{12}^{y}x_2)\cdot x_1) + 2\lambda w_{11}^{y}$$

$$= -2(1-o)(u_1(w_{11}^{y}x_1 + w_{12}^{y}x_2)\cdot x_1) + 2\lambda w_{11}^{y}$$

$$\frac{\partial L(\mathbf{x}_i,c_i,\Theta)}{\partial w_{12}^{y}} = -2(1-o)(u_1(w_{11}^{y}x_1 + w_{12}^{y}x_2)\cdot x_2) + 2\lambda w_{12}^{y}$$

$$\frac{\partial L(\mathbf{x}_i,c_i,\Theta)}{\partial w_{21}^{y}} = -2(1-o)(u_2(w_{21}^{y}x_1 + w_{22}^{y}x_2)\cdot x_1) + 2\lambda w_{21}^{y}$$

$$\frac{\partial L(\mathbf{x}_i,c_i,\Theta)}{\partial w_{22}^{y}} = -2(1-o)(u_2(w_{21}^{y}x_1 + w_{22}^{y}x_2)\cdot x_2) + 2\lambda w_{22}^{y}$$

Matrix-vector notation of gradients

WestlakeNLP

In matrix vector notation

$$\frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial \mathbf{u}} = \langle \frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial u_1}, \frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial u_2} \rangle$$
$$= \langle -(1-o)y_1 + 2\lambda u_1, -(1-o)y_2 + 2\lambda u_2 \rangle$$
$$= -(1-o)\mathbf{y} + 2\lambda \mathbf{u}$$

$$\frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial \mathbf{W}^y} = \begin{pmatrix} \frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial w_{11}^y}, & \frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial w_{12}^y} \\ \frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial w_{21}^y}, & \frac{\partial L(\mathbf{x}_i, c_i, \Theta)}{\partial w_{22}^y} \end{pmatrix}$$

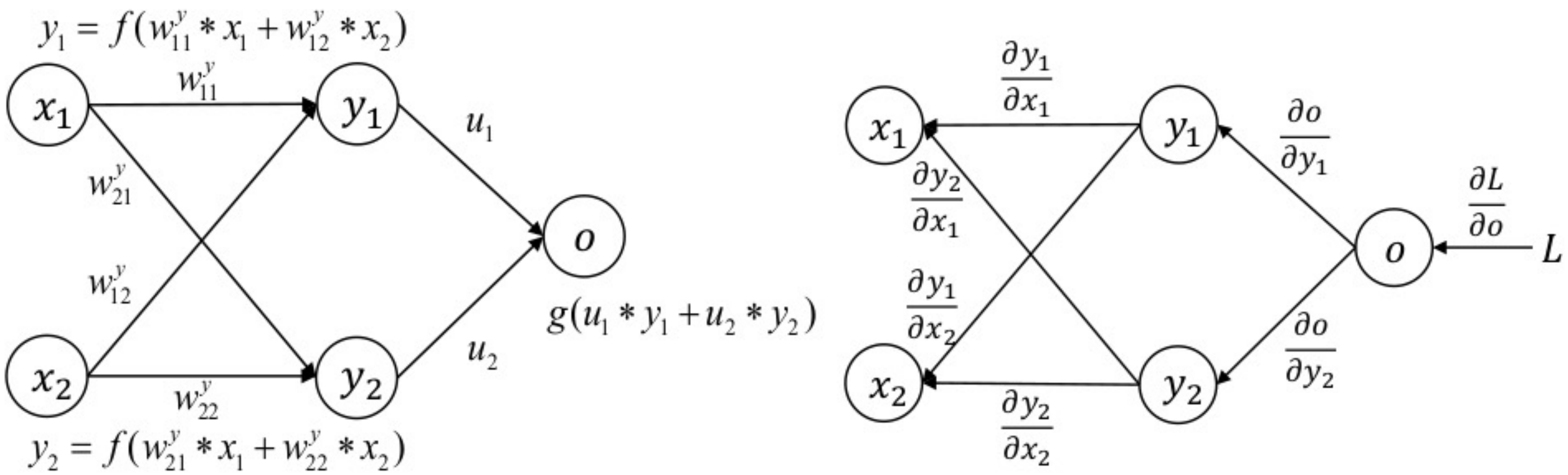
$$= -2(1-o)\boldsymbol{u} \otimes (\boldsymbol{W}^y \boldsymbol{x})\boldsymbol{x}^T$$

Contents

WestlakeNLP

Computation graph for a neural network

Now calculate gradients



(a) MLP structure (b) back-propagated gradients

Back-propagation

WestlakeNLP

- The above process is *tedious* for large neural nets
- Solution: perform **modularized** and incremental gradient calculation
- Back-propagation allows modularization of neural network components in deep networks
  - the forward computation
  - the back-propagation rule
    - the partial derivative of the loss with respect to the **model parameters**
    - the partial derivative of the loss with respect to the **input** layer

Back-propagation

WestlakeNLP

- For each layer
  - the structure – input to output
  - the input -- gradient on output nodes
  - the computation
    - the partial derivative with respect to the **model parameters**
    - the partial derivative with respect to the **input** nodes

Back-propagation

WestlakeNLP

For the MLP

$$\mathbf{y} = (\mathbf{W}^y\mathbf{x})^2, \quad o = \sigma(\mathbf{u}^T \cdot \mathbf{y})$$

For SGD, the local loss is

$$L(\mathbf{x}, c, \Theta) = L^o + \|\Theta\|^2$$

For the layer $\mathbf{y} \to o$, input is $\frac{\partial L^o}{\partial o}$

$$\frac{\partial L^o}{\partial \mathbf{u}} = \frac{\partial L^o}{\partial o} \cdot o(1-o)\mathbf{y}$$

$$\frac{\partial L^o}{\partial \mathbf{y}} = \frac{\partial L^o}{\partial o} \cdot o(1-o)\mathbf{u}$$

For the layer $\mathbf{x} \to \mathbf{y}$, input is $\frac{\partial L^o}{\partial \mathbf{y}}$

$$\frac{\partial L^o}{\partial \mathbf{W}^y} = \frac{\partial L^o}{\partial \mathbf{y}} \otimes (2\mathbf{W}^y\mathbf{x}) \cdot \mathbf{x}^T$$

Back-propagation for calculating gradients for arbitrary network

WestlakeNLP

Inputs: a network of M layers, each with a ForwardCompute function and a BackPropagate function;
the set of model parameters for the ith layer is Θ_i;
a gold-standard output $\mathbf{y}$ at the output layer;
an input $\mathbf{x}$;

Initialisation: $\mathbf{h}_0 \leftarrow \mathbf{x}$;

for $l \in [1, \ldots, M]$ **do** ▷ `forward computation`

  $\mathbf{h}_l \leftarrow \textsc{ForwardCompute}(\mathbf{h}_{l-1}, \Theta_l)$

$L \leftarrow \textsc{ComputeLoss}(\mathbf{h}_M, \mathbf{y})$;

$\mathbf{g}_M \leftarrow L$;

for $l \in [M, \ldots, 1]$ **do** ▷ `back-propagation`

  $\mathbf{g}_{l-1}, \mathbf{g}_l^{\Theta} \leftarrow \textsc{BackPropagate}(\mathbf{g}_l, \Theta_l)$

Output: $\{\mathbf{g}_l^{\Theta}\}|_{l=1}^{M}$;

Parameter Initialization

Randomly initialize the parameters with different values

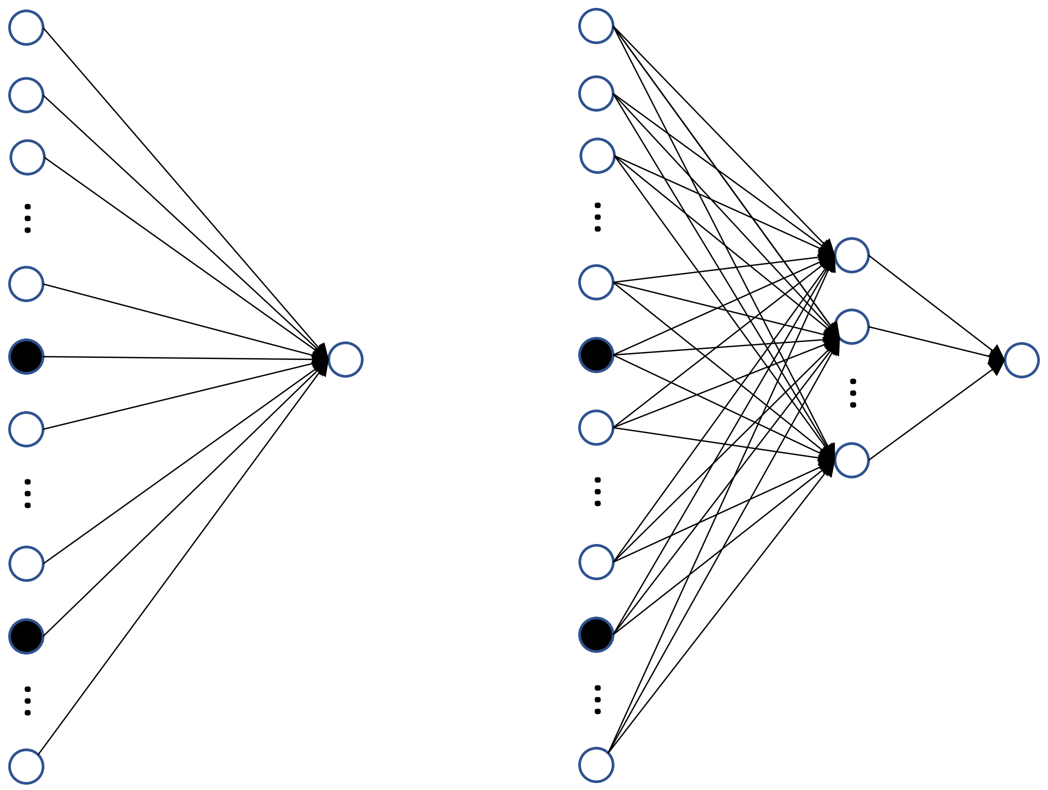
Given a model parameter **W** at the first layer, initialization of each element in **W** include

1. Xavier Uniform Initialization. $\mathbf{w} \sim \mathcal{U}\left(-\sqrt{\frac{6}{d_l + d_{l-1}}}, \sqrt{\frac{6}{d_l + d_{l-1}}}\right)$
2. Xavier Normal Initialization. $\mathbf{w} \sim \mathcal{N}\left(0, \frac{2}{d_l + d_{l-1}}\right)$
3. Kaiming Uniform Initialization. $\mathbf{w} \sim \mathcal{U}\left(-\sqrt{\frac{6}{d_{l-1}}}, \sqrt{\frac{6}{d_{l-1}}}\right)$
4. Kaiming Normal Initialization. $\mathbf{w} \sim \mathcal{N}\left(0, \frac{2}{d_{l-1}}\right)$

Contents

Neural Text Classification Structure

- Neural hidden layers are dense low-dimensional vectors
- Input still discrete sparse high-dimensional

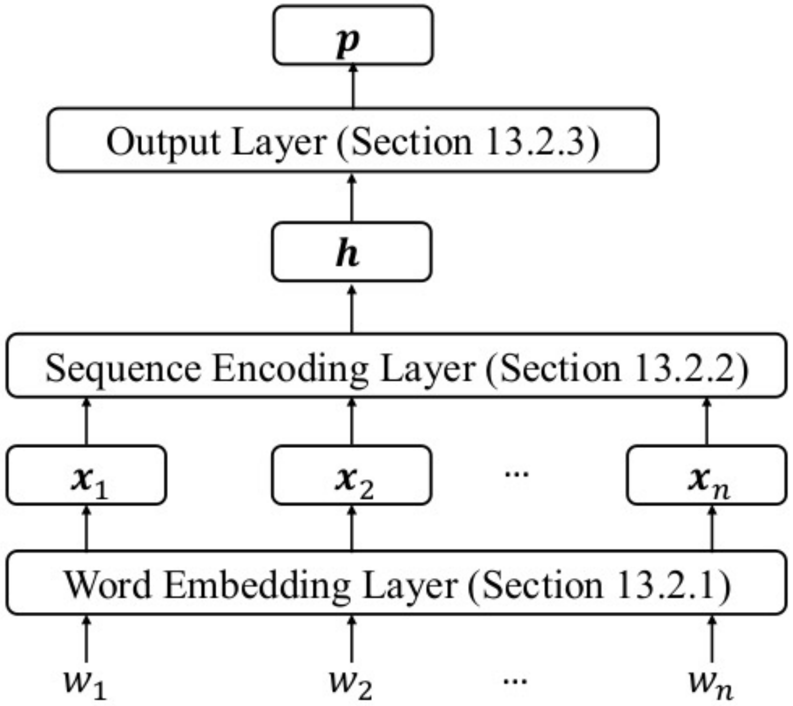


Neural Text Classification Structure

WestlakeNLP

Represent each word in the sentence also using a dense low-dimensional vector, called word embedding.

Use a sequence encoding network to extract hidden features automatically.



Contents

WestlakeNLP

Embedding layer

WestlakeNLP

- Dense embeddings offer a better semantic similarity measure correspond with sparse vectors (Chapter 5)
  - One-hot column vector, distributional vector, PMI vector: $\mathbf{x} \in \mathbb{R}^{|V|}$
  - Word embedding matrix (embedding lookup table): $\mathbf{W} \in \mathbb{R}^{d \times |V|}$
  - The embedding vector of x can be defined by

$$emb(x) = \mathbf{Wx}$$

  - For neural network, $emb(x)$ can be low-dimensional (500-2000)
- Pre-training

  Word embedding values can be separately trained over large raw texts before model training.

Contents

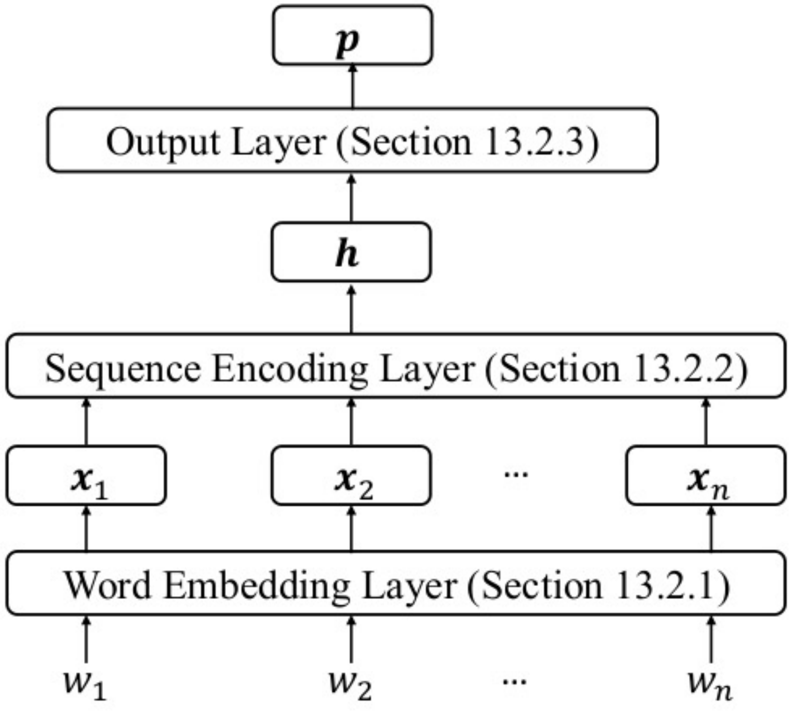
WestlakeNLP

Sequence encoder

WestlakeNLP

A subnetwork that transforms a sequence of dense vectors into a single dense vector that represents features over the whole sequence.

- Pooling
- Convolutional network
- Recurrent neural network
- Attentional neural network



Pooling

WestlakeNLP

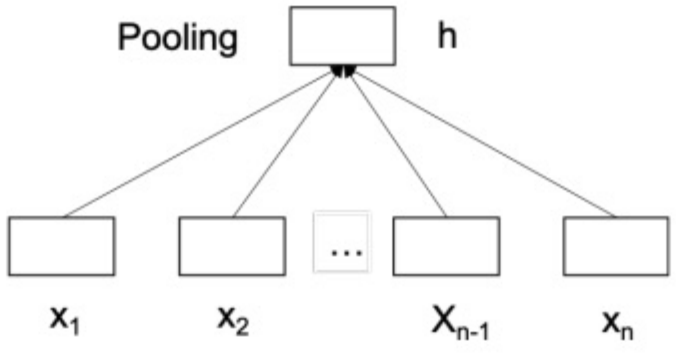
Pooling based sequence representation (deep averaging network)

- Sum pooling

$$\text{sum}(\mathbf{X}_{1:n}) = \sum_{i=1}^{n} \mathbf{x}_i$$

- Average pooling

$$\text{avg}(\mathbf{X}_{1:n}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$



- Max pooling

$$\max(\mathbf{X}_{1:n}) = \langle \max_{i=1}^{n} \mathbf{x}_i[1], \max_{i=1}^{n} \mathbf{x}_i[2], \dots, \max_{i=1}^{n} \mathbf{x}_i[d] \rangle^T$$

- Min pooling

$$\min(\mathbf{X}_{1:n}) = \langle \min_{i=1} \mathbf{x}_i[1], \min_{i=1} \mathbf{x}_i[2], \dots, \min_{i=1} \mathbf{x}_i[d] \rangle^T$$

Pooling

- Back-propagation
  - For sum pooling, $\frac{\partial L}{\partial \mathbf{x}_i} = \frac{\partial L}{\partial \mathbf{h}}$ for all $\mathbf{x}_i (i \in [1, \ldots, n])$
  - For average pooling, $\frac{\partial L}{\partial \mathbf{x}_i} = \frac{1}{n}\frac{\partial L}{\partial \mathbf{h}}$
  - For maximum pooling, $\frac{\partial L}{\partial \mathbf{x}_i[j]}$

$$= \begin{cases} \frac{\partial L}{\partial \mathbf{h}}[j] & if\ i = \text{argmax}_{i' \in [1,\ldots,n]} \mathbf{x}_{i'}[j], (i \in [1, \ldots, n], j \in [1, \ldots, d]) \\ 0 & otherwise \end{cases}$$

  - Pooling can work with a variable-sized set of input vectors, aggregating them into a fix-sized output.

Convolutional neural network (CNN)

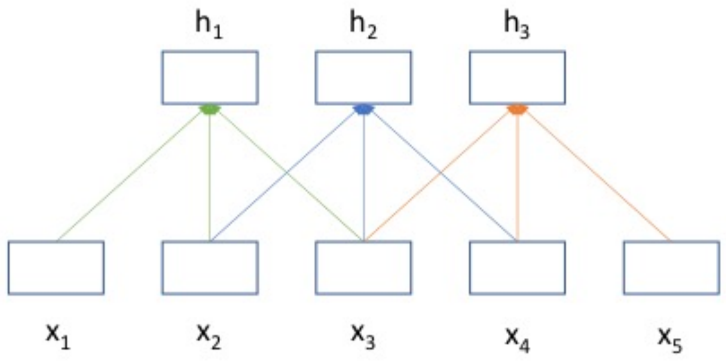
WestlakeNLP

- Pooling extract *uni*gram-level features
- No model parameters
- No n-gram features with $n > 1$.

Convolutional neural network (CNN)

Use convolutional filters to extract n-gram features

- Window-size K filters
  - Input: $\mathbf{X}_{1:n} = \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_n$
  - Output: $\mathbf{H}_{1:m} = \mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_m$
  - Input channel and output channel dimensions: d_I, d_O



$$\mathbf{H}_{1:n-K+1} = \mathrm{CNN}(\mathbf{X}_{1:n}, K, d_O)$$

$$\mathbf{h}_\mathrm{i} = \mathbf{W}\mathbf{X}_{\mathrm{i}:i+K-1} + \mathbf{b}$$

Convolutional neural network (CNN)

WestlakeNLP

Back-propagation

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{i=1}^{n-K+1} \left(\frac{\partial L}{\partial \mathbf{h}_i} (\mathbf{x}_i \oplus \mathbf{x}_{i+1} \oplus \cdots \oplus \mathbf{x}_{i+K-1})^T \right)$$

$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{i=1}^{n-K+1} \frac{\partial L}{\partial \mathbf{h}_i}$$

$$\frac{\partial L}{\partial \mathbf{x}_i} (i \in [1, \ldots, n])$$

Comparison with discrete n-gram features

WestlakeNLP

CNN features are different from Chapter3 feature vectors

- Dense and low-dimensional
- Dynamically computed
- Adjustable during training

Contents

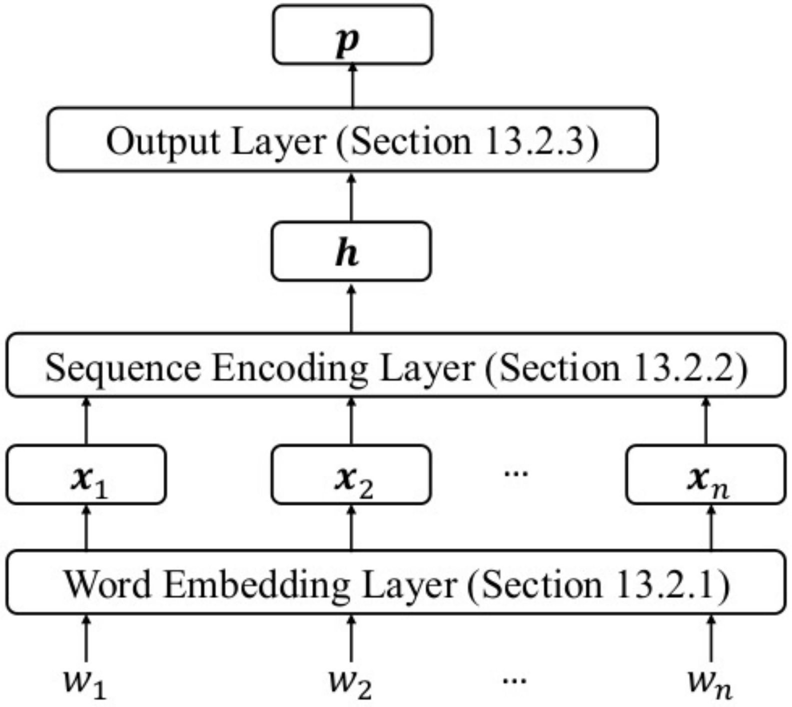
WestlakeNLP

Neural Text Classififcation Structure

WestlakeNLP

Represent each word in the sentence also using a dense low-dimensional vector, called word embedding.

Find a single hidden vector for the sequence.



Output layer

Output classes: $C = \{c_1, \dots, c_{|C|}\}$

- Input vector: a sequence of vectors $\mathbf{X}_{1:n}$
- CNN calculates a sequence of vectors $\mathbf{H}_{1:n-K+1}$
- Pooling gives a dense and more abstract vector representation $\mathbf{h}$
- Softmax multi-class output layer calculates the classification probability distribution:

$$\mathbf{o} = \mathbf{W}^o \mathbf{h} + \mathbf{b}^o$$

$$\mathbf{p} = \mathrm{softmax}(\mathbf{o})$$

Contents

WestlakeNLP

Training under the SGD framework

WestlakeNLP

- With log-likelihood loss (cross-entropy loss)
  - Training samples: $\{(\mathbf{X}_i, c_i)\}|_{i=1}^{N}$
  - Cross-entropy loss: $L = -\sum_{i=1}^{N} \log \mathbf{p}[c_i]$
  - Back-backpropagation, SGD
- Compared to max margin loss, cross-entropy loss gives more fine-grained supervision signal.

Contents

WestlakeNLP

Neural network models are difficult to train

WestlakeNLP

- Train arbitrary hyper-surface shapes in a high-dimensional vector space
- Gradient diminishing -- Back-propagated gradients can become negligibly small through layers
- Gradient explosion – Back-propagated gradients become infinitely large causing numerical overflow
- Tendency of overfitting

Contents

WestlakeNLP

Avoid Gradient Explosion

WestlakeNLP

- Gradient clipping

  Prevent gradient being too large by consulting hard threshold values

Residual network

WestlakeNLP

- Add a direct connection between the input layer and the output layer
  - Input vector: $\boldsymbol{x}$
  - Baseline network: g($\boldsymbol{x}$ (nonlinear transformation))
  - Residual network = R$_{ESIDUAL}(x, g)$: $\mathbf{h} = g(\mathbf{x}) + \mathbf{x}$
- Given a local loss L and back-propagated gradients $\frac{\partial L}{\partial \mathbf{h}}$

  Calculate $\frac{\partial L}{\partial \boldsymbol{x}}$ as $\frac{\partial L}{\partial \mathbf{x}}[g] + \frac{\partial L}{\partial \mathbf{h}}$ preventing failure of training
- Residual networks are effective for training very deep neural networks

Contents

WestlakeNLP

Layer Normalization

WestlakeNLP

- **Internal covariate shift**

Slightly changing one parameter of a layer can greatly affect the distribution of the node values in the subsequent layers

- **Layer normalization**

Calculates the mean and variance statistics over **z** for defining a mapping function $LayerNorm: \mathbb{R}^d \rightarrow \mathbb{R}^d$ $LayerNorm(\mathbf{z}; \boldsymbol{\alpha}, \boldsymbol{\beta})$ is given by ($\boldsymbol{\alpha}$: *gains*, $\boldsymbol{\beta}$: *biases*)

$$\mu = \frac{1}{d}\sum_{i=1}^{d} \mathbf{z}[i] \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\mathbf{z}[i] - \mu)}$$

$$\text{LayerNorm}(\mathbf{z}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\mathbf{z} - \mu}{\sigma} \otimes \boldsymbol{\alpha} + \boldsymbol{\beta}$$

Dropout

WestlakeNLP

- A training setting for neural networks to prevent overfitting

Randomly set the values of nodes or node connections to zeroes with a probability

- Given a vector $\mathbf{x} \in \mathbb{R}^d$ and a dropout probability p, $\text{DROPOUT}(\mathbf{x}, p)$ is defined as

  $\mathbf{m} \sim Bernoulli(p)$ (sample from Bernoulli distribution)

  $\hat{\mathbf{m}} = \frac{\mathbf{m}}{1-p}$

  $\text{DROPOUT}(\mathbf{x}, p) = \mathbf{x} \otimes \mathbf{m}$

  Dropout mask: $\mathbf{m}$

  Scaled mask: $\hat{\mathbf{m}}$

Contents

SGD training

WestlakeNLP

- The general updating rules of the time step t for SGD are

$$\mathbf{g}_t = \frac{\partial L(\Theta_{t-1})}{\partial \Theta_{t-1}}$$

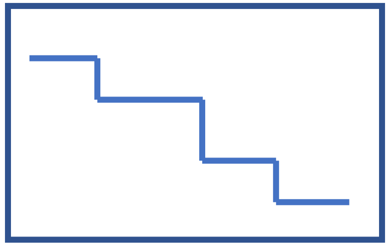
$$\Theta_t = \Theta_{t-1} - \eta \mathbf{g}_t$$

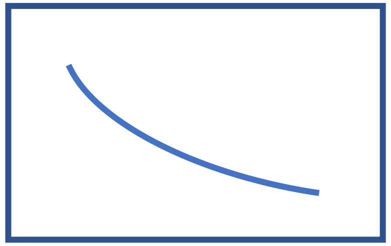
Model parameter: Θ Loss function: $L(\Theta)$

- For training neural networks,
  - g_t can be calculated on a mini-batch of training examples
  - The number of training iterations (epoch) can be selected according to development experiments. (Early stopping)
  - Adjust the learning rate η at different time steps

Several techniques for improving SGD training

- Learning rate decay
  - step decay
  - exponential decay
  - gradient clipping





Prevent gradient being too large by consulting hard threshold values

- SGD with Momentum

A way to soften oscillations, accelerating the converging process

SGD with momentum

- The parameter update considers not only the immediate gradient but also the history gradients
- The update rules for momentum SGD is

$$\mathbf{g}_t = \frac{\partial L(\Theta_{t-1})}{\partial \Theta_{t-1}}$$

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t$$

$$\Theta_t = \Theta_{t-1} - \mathbf{v}_t$$

- Memory vector (velocity vector): $\mathbf{v}_t$
- Momentum hyper-parameter (friction parameter): γ

Contents

WestlakeNLP

Hyper-Parameter Search

WestlakeNLP

- Grid search
  - Specify a set of candidate values for each hyperparameter
  - Build a model for every combination of the specified hyperparameters and evaluate the performance of each model
- Random search
  - Random combinations of hyperparameters

Summary

WestlakeNLP

- Multi-layer perceptrons and deep neural networks
- Convolutional neural networks for text classification
- Dropout, layer normalizations and residual network
- SGD with momentum