

日常记录

冯哲*

计算机科学与技术学院

2020 年 9 月 16 日 更新

目录

1	命题时序逻辑	4
1.1	语法	4
1.2	语义	4
2	PTL练习题	5
	第一部分 paper	6
3	A decision procedure for propositional projection temporal logic with infinite models	7
3.1	PPTL	7
3.1.1	语法	7
3.1.2	语义	7
3.1.3	运算法则	8
	第二部分 Weekly conversation	9

*电子邮件: 1194585271@qq.com

4	2019/7/16 问题	9
4.1	练习	11
5	2020/8/20 问题	12
	第三部分 神经网络形式化	16
6	全连接神经网络	16
6.1	数学推导	16
6.1.1	作用过程示例	16
6.1.2	作用过程推广	18
	神经网络向前传播	21
	神经网络反向传播	22
	全连接神经网络结构图	22
	优化算法	23
6.2	MSVL 语言框架	26
6.2.1	辅助函数	26
6.2.2	矩阵操作函数	26
6.2.3	矩阵激活函数	27
6.2.4	矩阵激活函数导数函数	27
6.2.5	损失函数	27
6.2.6	损失函数的导函数	27
6.2.7	神经网络的初始化	27
6.2.8	神经网络用户自定义参数	28
6.2.9	神经网络的运算所需空间变量及创建所需空间函数	28
6.2.10	初始化神经网络	29

6.2.11	神经网络前向传播	30
6.2.12	神经网络反向传播	30
6.2.13	优化算法	31
6.3	MSVL 语言框架重构	32
6.3.1	对象结构体	32
6.3.2	开发流程	34
	神经网络用户自定义参数及总数据录入	34
	数据集构建	34
	神经网络初始化	35
	神经网络前向传播求损失	35
	神经网络反向传播求梯度	35
	Adam算法构建	36
6.4	FCNN应用	37
6.4.1	Minst 手写字体识别	37

1 命题时序逻辑

1.1 语法

① notation:

\mathbb{Z} : 整数;

\mathbb{N} : 正整数;

N_0 : 非负整数;

Prop : 可数原子命题;

operate: $\neg, \Box, \bigcirc, \Diamond, ;, \vee, \wedge, +, *$

② 归纳定义命题公式:

1. $p \in \text{Prop}, p$ is a formula.

2. $p, q \in \text{Prop},$ so are the constructs.

$p, q, p \wedge q, p; q, \bigcirc p, \Box q, p^+, q^*$ and so on.

1.2 语义

① 状态 S , $B = \{\text{true}, \text{false}\}$

$S : \text{Prop} \rightarrow B$ $\{\text{原子命题} \rightarrow B\}$

状态是原子命题到真假集合的映射。

② 区间

区间 σ : 非空状态序列。

区间长度: 有限区间 **finite** $|\sigma|$, $|\sigma| = \text{status} - 1$

无限区间 **infinite** w

统一★, $N_w = N_0 \vee \{w\}$

operator $=, <, \leq, \preceq$

对于 N_w , $w = w$

对于 $i \in N_0$, $i < w$

\preceq as $\leq -\{w, w\}$

定义:

1. σ as $< s_0, \dots, s_{|\sigma|} >$, 若 σ is infinite. $s_{|\sigma|}$ 无定义。

2. $\sigma_{(i, \dots, j)}$ as $(0 \leq i \preceq j \leq |\sigma|) < s_i, \dots, s_j >$

3. $\sigma^{(k)}$ as $(0 \leq k \preceq |\sigma|) < s_k, \dots, s_{|\sigma|} >$

2 PTL练习题

常用基本符号优先级:

1. \neg
2. $\bigcirc, \Diamond, \Box, +, *$
3. \wedge, \vee
4. $\rightarrow, \leftrightarrow$
5. $;$

2018/7/22

题 1 : 存在原子命题A,B, 将下列命题用PTL公式表示:

- ① A发生, 那么之后B总会发生 (包括当前状态)。
- ② A发生, 那么之前B总会发生 (包括当前状态)。

答:

- ① $\Box(A \rightarrow \Diamond B)$
- ② $\neg(\Box\neg B; A)$

理解: **chop** 分为两个区间, 两个区间上分别真假共四种情况, 而此时否定一种, 剩三种, 与①的蕴含效果一样, 具体情况为 $\neg(P; Q)$ 被这样的区间满足: 对于该区间的任意一种两段划分, 要么第一段不满足P, 要么第二段不满足Q (可以第一段不满足P 同时第二段不满足Q)。

2018/7/23

题 2 : 存在原子命题A,B, 假设有个词语"严格之后"表示之后的状态, 但不包括当前状态。 "严格之前"同理, 将下列命题用PTL公式表示:

- ① A发生, 那么严格之后B总会发生 (不包括当前状态)。
- ② A发生, 那么严格之前B总会发生 (不包括当前状态)。

答:

- ① $\Box(A \rightarrow \bigcirc\Diamond B)$
- ② $\neg((\Box(\text{more} \wedge \neg B)); A)$ \times \Box 和 **more**连用表示无穷区间
- ② $\neg((\Box(\ominus\neg B)); A)$ \times \ominus 不能和**always**直接连用, 因为第一个状态没有上一状态。
- ② $(\neg A) \wedge ((\Diamond B; \bigcirc A) \vee (\Box\neg A))$?
- ② $(\neg A) \wedge \neg(\Box\neg B; \text{skip}; A)$ \checkmark 题目的隐含意思有A不可能发生在起始状态
- ② $(\neg A) \wedge \neg(\Box\neg B; \bigcirc A)$ \checkmark 题目的隐含意思有A不可能发生在起始状态

时序逻辑性质表达 -2018/5/21

是朋友才能发消息，如何表示？

是朋友： f ；发消息： s 。

① 经典逻辑表达：

- a). $s \rightarrow f$ 发消息必是朋友。
- b). $\neg f \rightarrow \neg s$ 不是朋友不能发消息
- c). $\neg(\neg f \wedge s)$ 不是朋友且发消息是不对的 （与b)等价）
- d). $f \vee \neg s$ 要么是朋友要么不发消息 （与b),c)等价）

② 时序逻辑表示：

- a). 将来能发消息，现在一定是朋友。

$$\Diamond(\Diamond s \rightarrow f) \text{ weak } \quad \Box(s \rightarrow f)$$

- b). 现在不是朋友，将来不能发消息。

$$\Box(\neg f \rightarrow \neg s)$$

- c). 现在不是朋友，将来能发消息是不对的。

$$\neg \Diamond(\Box \neg f; s) \quad \Box \neg(\neg f \wedge s)$$

- d). 任何时刻，要么是朋友，要么不发消息。

$$\Box(f \vee \neg s)$$

- e). 在是朋友之前不能发消息。

$$\Diamond(s \rightarrow \Box f)$$

Runtime verification for LTL and TLTL :

$$\neg s \text{ until } f \equiv \Box \neg s; \text{skip}; f$$

第一部分 paper

3 A decision procedure for propositional projection temporal logic with infinite models

无限模型下命题投影时间逻辑 (PPTL) 的决策过程

- (1) 无限模型下命题投影时间逻辑 (PPTL) 的可满足性;
- (2) PPTL 公式的决策过程;
 - (a) PPTL 的范式;
 - (b) PPTL 的范式图;
 - (c) 根据公式转换为范式;
 - (d) 构造给定公式的范式图;
- (3) 扩展 PPTL 的决策过程;

Question ① ITL, PTL, PPTL?

ITL(Interval Temporal Logic) 区间时序逻辑, 规范和验证并发系统。

PTL 投影时序逻辑, 相对于 ITL 扩展了无限模型和投影操作。

PPTL 命题投影时序逻辑。

决策过程是为了检查 PPTL 公式的可满足性条件。

3.1 PPTL

3.1.1 语法

$$P ::= p \mid \bigcirc P \mid \neg P \mid P_1 \vee P_2 \mid (P_1, \dots, P_m) \text{ } prj \text{ } P$$

\bigcirc , prj , 基本的时间运算符。

Question ② PPTL 的投影操作?

3.1.2 语义

1. state (状态):

命题公式到真假集的映射。 $B = \{true, false\}$, $s : \text{Prop} \longrightarrow B$

2. interval (区间):

有限无限区间的统一定义;

区间长度比较操作: $=, <, \leq, \preceq$

3. interpretation (解释):

三元组: $\mathcal{I} = (\sigma, k, j)$

σ , 区间。

$k \preceq j \leq |\sigma|$

3.1.3 运算法则

L_1	$\Box(P \wedge Q) \equiv \Box P \wedge \Box Q$
L_2	$\Diamond(P \vee Q) \equiv \Diamond P \vee \Diamond Q$
L_3	$\bigcirc(P \vee Q) \equiv \bigcirc P \vee \bigcirc Q$
L_4	$\bigcirc(P \wedge Q) \equiv \bigcirc P \wedge \bigcirc Q$
L_5	$R; (P \vee Q) \equiv (R; P) \vee (R; Q)$
L_6	$(P \vee Q); R \equiv (P; R) \vee (Q; R)$
L_7	$\Diamond P \equiv P \vee \bigcirc \Diamond P$
L_8	$\Box P \equiv P \wedge \bigcirc \Box P$
L_9	$\bar{\varepsilon} \wedge \neg \bigcirc P \equiv \bar{\varepsilon} \wedge \bigcirc \neg P$
L_{10}	$\neg \bigcirc P \equiv \bigcirc \neg P$
L_{11}	$\bigcirc P; Q \equiv \bigcirc(P; Q)$
L_{12}	$w \wedge (P; Q) \equiv (w \wedge P); Q$
L_{13}	$Q \text{ prj } \varepsilon \equiv Q$
L_{14}	$\varepsilon \text{ prj } Q \equiv Q$
L_{15}	$(P_1, \dots, P_m) \text{ prj } \varepsilon \equiv P_1; \dots; P_m$
L_{16}	$(P, \varepsilon) \text{ prj } Q \equiv (P \wedge \Diamond \varepsilon) \text{ prj } Q$
L_{17}	$(P_1, \dots, P_t, w \wedge \varepsilon, P_{t+1}, \dots, P_m) \text{ prj } Q \equiv (P_1, \dots, P_t, w \wedge P_{t+1}, \dots, P_m) \text{ prj } Q$
L_{18}	$(P_1, \dots, (P_i \vee P'_i), \dots, P_m) \text{ prj } Q \equiv ((P_1, \dots, P_i, \dots, P_m) \text{ prj } Q) \vee ((P_1, \dots, P'_i, \dots, P_m) \text{ prj } Q)$
L_{19}	$(P_1, \dots, P_m) \text{ prj } (P \vee Q) \equiv ((P_1, \dots, P_m) \text{ prj } P) \vee ((P_1, \dots, P_m) \text{ prj } Q)$
L_{20}	$(P_1, \dots, P_m) \text{ prj } \bigcirc Q \equiv (P_1 \wedge \bar{\varepsilon}; (P_2, \dots, P_m) \text{ prj } Q) \vee (P_1 \wedge \varepsilon; (P_2, \dots, P_m) \text{ prj } \bigcirc Q)$
L_{21}	$(\bigcirc P_1, \dots, P_m) \text{ prj } \bigcirc Q \equiv \bigcirc(P_1; (P_2, \dots, P_m) \text{ prj } Q)$
L_{22}	$(w \wedge P_1, \dots, P_m) \text{ prj } Q \equiv w \wedge ((P_1, \dots, P_m) \text{ prj } Q)$
L_{23}	$(P_1, \dots, P_m) \text{ prj } (w \wedge Q) \equiv w \wedge ((P_1, \dots, P_m) \text{ prj } Q)$

第二部分 Weekly conversation

4 2019/7/16 问题

1. 范式:将 PPTL 公式化成规范形式。

$$Q \equiv \bigvee_{j=1}^{n_0} (Q_{ej} \wedge \varepsilon) \vee \bigvee_{i=1}^n (Q_{ci} \wedge \bigcirc Q'_i)$$

Q , PPTL 范式; Q_p , 原子命题公式; Q'_i , 普通 PPTL 公式。

那么 PPTL 公式怎么转为范式的四种情况: (最后一种为转为完全范式)。

- (1) PPTL 公式转 N_F 。
- (2) project 操作转 N_F 。
- (3) chop 操作转 N_F 。
- (4) $N_F \rightarrow CN_F$

例 N_F 转 CN_F

$$\begin{aligned} (p \wedge \bigcirc P') \vee (q \wedge \bigcirc Q') &\equiv ((p \wedge q) \wedge \bigcirc (P' \vee Q')) \vee ((p \wedge \neg q) \wedge \bigcirc P') \\ &\vee ((\neg p \wedge q) \wedge \bigcirc Q') \vee ((\neg p \wedge \neg q) \wedge \bigcirc false) \end{aligned}$$

$$(p \wedge \bigcirc P') \vee (q \wedge \bigcirc Q') = false \wedge \epsilon \vee (p \wedge \bigcirc P') \vee (q \wedge \bigcirc Q')$$

2. 完全范式(作用: PPTL 完全范式否定简易)

$$Q \equiv (Q_e \wedge \varepsilon) \vee \bigvee_{i=1}^n (Q_i \wedge \bigcirc Q'_i)$$

其中 $\bigvee_i Q_i \equiv true$ and $\bigvee_{i \neq j} (Q_i \wedge Q_j) \equiv false$ 。

3. 完全范式的否定

例 CN_F 的否定

$$\begin{aligned} \neg((p \wedge \bigcirc P') \vee (q \wedge \bigcirc Q')) &\equiv \neg\left(((p \wedge q) \wedge \bigcirc (P' \vee Q')) \vee ((p \wedge \neg q) \wedge \bigcirc P') \right. \\ &\quad \left. \vee ((\neg p \wedge q) \wedge \bigcirc Q') \vee ((\neg p \wedge \neg q) \wedge \bigcirc false) \right) \\ &\equiv ((p \wedge q) \wedge \bigcirc \neg(P' \vee Q')) \vee ((p \wedge \neg q) \wedge \bigcirc \neg P') \\ &\quad \vee ((\neg p \wedge q) \wedge \bigcirc \neg Q') \vee ((\neg p \wedge \neg q) \wedge \bigcirc \neg false) \end{aligned}$$

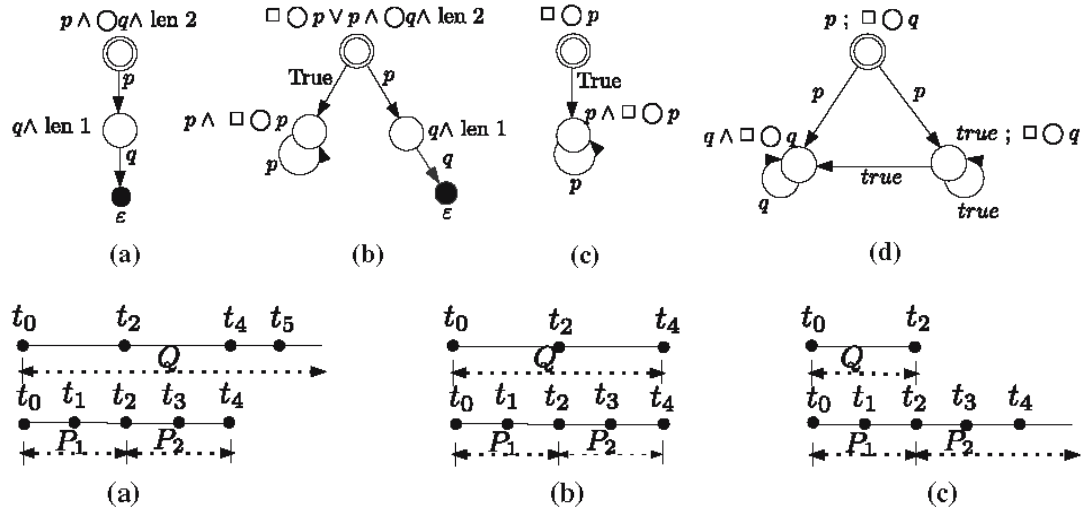
4. 范式图 (重点)

PPTL 公式决策过程的可视化

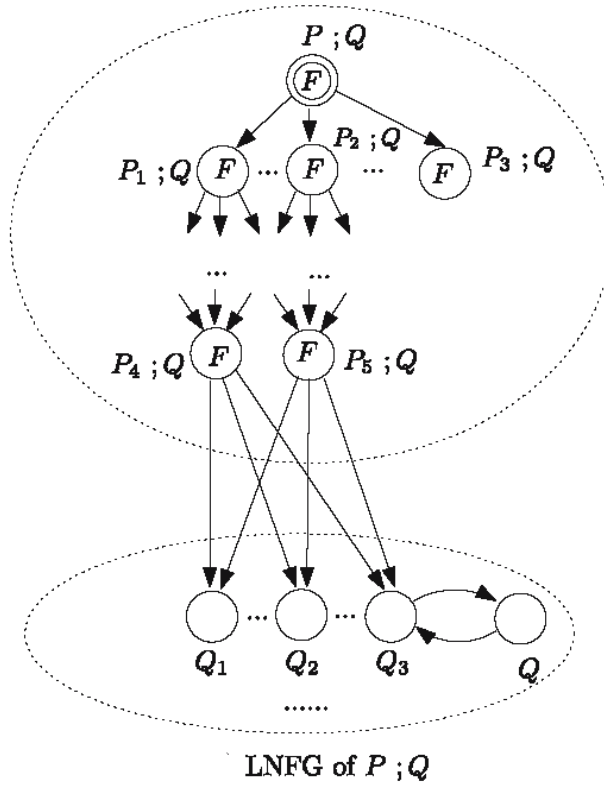
NFG 举例:

5. 带标签的范式图

由于 prj 操作有明确的语义, $(P_1, P_2) prj Q$ 的语义:



即就是 PPTL 公式 P_1 必须是有穷区间上的。但这一点在范式图上无法限制，所以加入标签。



4.1 练习

$$\begin{aligned}
& \text{NF}(\text{O}^2(\Box \bigcirc p; q) \vee (p; \Box \bigcirc q)) \\
& \equiv \text{NF}(\bigcirc^2(\Box \bigcirc p; q)) \vee \text{NF}(p; \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee \text{CHOP}(p; \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee \text{CHOP}(\text{NF}(p); \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee \text{CHOP}(p \wedge \varepsilon \vee p \wedge \mathbf{0}\mathbf{true}; \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee \text{CHOP}(p \wedge \varepsilon; \Box \bigcirc q) \vee \text{CHOP}(p \wedge \mathbf{0}\mathbf{true}; \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee p \wedge \text{CHOP}(\varepsilon; \Box \bigcirc q) \vee p \wedge \text{CHOP}(\bigcirc\mathbf{true}; \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee p \wedge (\text{NF}(\bigcirc q \wedge \varepsilon) \vee \text{NF}(\bigcirc q \wedge \bigcirc \Box \bigcirc q)) \vee p \wedge \bigcirc(\mathbf{true}; \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee p \wedge (\mathbf{false} \vee \bigcirc(q \wedge \Box \bigcirc q)) \vee p \wedge \bigcirc(\mathbf{true}; \Box \bigcirc q) \\
& \equiv \bigcirc(\bigcirc(\Box \bigcirc p; q)) \vee p \wedge \bigcirc(q \wedge \Box \bigcirc q) \vee p \wedge \bigcirc(\mathbf{true}; \Box \bigcirc q)
\end{aligned}$$

5 2020/8/20 问题

1. 卷积与池化中的填充与步幅的问题。

(a) `tf.nn.conv2d`

参数	内容	说明
input	默认NHWC=[batch, height, width, channels]	默认NHWC, 四维张量形式组织data_format参数指定形式组织
filter	[filter_height, filter_width, in_channels, out_channels]	卷积核以层定义, 且同层卷积核尺寸一致。
strides	list length 1(int), 2 or 4	默认NC步幅为1; 1->HW; 2->HW; 4->NCHW 具体形式看data_format组织, 不支持NC维度上的步幅。
padding	string->'VALID' or 'SAME'; list [[0, 0], [pad.top, pad.bottom], [pad.left, pad.right], [0, 0]]	string: VALID->不填充(舍弃); SAME->输入输出等SHAPE填充, 与步长也相关。list: NC维度不填充。
data_format	默认NHWC=[batch, height, width, channels]	CPU内存局部性原理NHWC比NCHW更高效。但对于数据的结构, NCHW更好理解。
dilations	list length 1(int), 2 or 4, add 0 to input	默认NC膨胀系数为1; 1->HW; 2->HW; 4->NCHW 具体形式看data_format组织
name	A name for the operation	optional

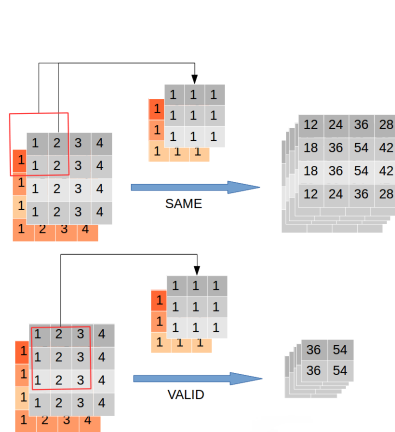


图 1: 正常卷积

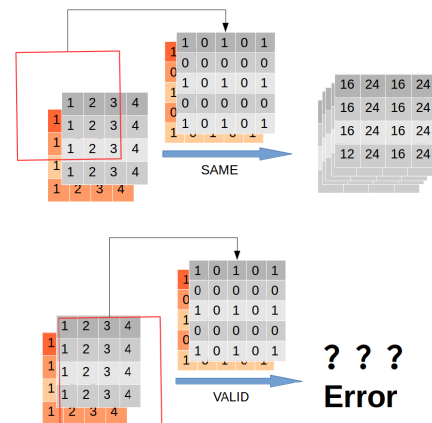


图 2: 空洞卷积

(b) `tf.nn.max_pool`

参数	内容	说明
input	维度 N+2 N为数据单元的维度	默认NHWC, 四维张量形式组织data_format参数指定形式组织
ksize	list length 1, N, or N+2	NC维度为1, 其余自定义。
strides	list length 1, N, or N+2	NC维度为1, 其余自定义, 不支持NC维度的步幅。
padding	string->'VALID' or 'SAME'; list [[0, 0], [pad.top, pad.bottom], [pad.left, pad.right], [0, 0]]	string: VALID->不填充; SAME->输入输出等SHAPE填充, 与步长也相关。list: NC维度不填充。
data_format	string N=1:NWC; N=2:NHWC; N=3:NDHWC	根据数据单元的维度, 进行默认选取。
name	A name for the operation	optional

注:

- Pooling is not yet supported on the batch dimension. [Op:MaxPool]
- Current implementation does not yet support strides in the batch and depth dimensions. [Op:Conv2D]

2. 卷积核尺寸按层定义还是按网络定义。

按层定义, 每层的卷积核shape相同, 不同的卷积核拥有不同的bias。

3. 卷积核对应输入通道与输出通道的关系。

卷积核输入通道对应input数据channel; 卷积核输出通道对应output(featureMap)数据channel。

4. 卷积神经网络一般绘图的方法。

- <http://alexlenail.me/NN-SVG/index.html>

- <https://www.zhihu.com/question/40698990>
- <https://github.com/HarisIqbal88/PlotNeuralNet>
- https://github.com/gwding/draw_convnet
- <https://link.zhihu.com/?target=https%3A//github.com/ajtulloch/dnngraph>

tf.nn.conv2d 参数说明

```
'''
conv2d_v2(input, filters, strides, padding, data_format='NHWC',
          dilations=None, name=None)
Computes a 2-D convolution given 4-D 'input' and 'filters' tensors.
Given an input tensor of shape '[batch, in_height, in_width, in_channels]'
and a filter / kernel tensor of shape
'[filter_height, filter_width, in_channels, out_channels]', this op
performs the following:

1. Flattens the filter to a 2-D matrix with shape
   '[filter_height * filter_width * in_channels, output_channels]'.
2. Extracts image patches from the input tensor to form a *virtual*
   tensor of shape '[batch, out_height, out_width,
   filter_height * filter_width * in_channels]'.
3. For each patch, right-multiplies the filter matrix and the image patch
   vector.

In detail, with the default NHWC format,

    output[b, i, j, k] =
        sum_{di, dj, q} input[b, strides[1] * i + di, strides[2] * j + dj, q] *
        filter[di, dj, q, k]

Must have 'strides[0] = strides[3] = 1'. For the most common case of the same
horizontal and vertical strides, 'strides = [1, stride, stride, 1]'.

Args:
input: A 'Tensor'. Must be one of the following types:
    'half', 'bfloat16', 'float32', 'float64'.
    A 4-D tensor. The dimension order is interpreted according to the value
    of 'data_format', see below for details.
filters: A 'Tensor'. Must have the same type as 'input'.
    A 4-D tensor of shape
    '[filter_height, filter_width, in_channels, out_channels]'
```

```

strides: An int or list of 'ints' that has length '1', '2' or '4'. The
        stride of the sliding window for each dimension of 'input'. If a single
        value is given it is replicated in the 'H' and 'W' dimension. By default
        the 'N' and 'C' dimensions are set to 1. The dimension order is determined
        by the value of 'data_format', see below for details.
padding: Either the 'string' "SAME" or "VALID" indicating the type of
        padding algorithm to use, or a list indicating the explicit paddings at
        the start and end of each dimension. When explicit padding is used and
        data_format is "NHWC", this should be in the form '[0, 0], [pad_top,
        pad_bottom], [pad_left, pad_right], [0, 0]'. When explicit padding used
        and data_format is "NCHW", this should be in the form '[0, 0], [0, 0],
        [pad_top, pad_bottom], [pad_left, pad_right]'.
data_format: An optional 'string' from: "NHWC", "NCHW".
        Defaults to "NHWC".

        Specify the data format of the input and output data. With the
        default format "NHWC", the data is stored in the order of:

            [batch, height, width, channels].

        Alternatively, the format could be "NCHW", the data storage order of:

            [batch, channels, height, width].
dilations: An int or list of 'ints' that has length '1', '2' or '4',
        defaults to 1. The dilation factor for each dimension of 'input'. If a
        single value is given it is replicated in the 'H' and 'W' dimension. By
        default the 'N' and 'C' dimensions are set to 1. If set to k > 1, there
        will be k-1 skipped cells between each filter element on that dimension.
        The dimension order is determined by the value of 'data_format', see above
        for details. Dilations in the batch and depth dimensions if a 4-d tensor
        must be 1.
name: A name for the operation (optional).

Returns:
    A 'Tensor'. Has the same type as 'input'.
'''

```

tf.nn.max_pool 参数说明

```

'''
max_pool_v2(input, ksize, strides, padding, data_format=None, name=None)
    Performs the max pooling on the input.

Args:
    input: Tensor of rank N+2, of shape '[batch_size] + input_spatial_shape +

```

```

    [num_channels]' if 'data_format' does not start with "NC" (default), or
    '[batch_size, num_channels] + input_spatial_shape' if data_format starts
    with "NC". Pooling happens over the spatial dimensions only.
    ksize: An int or list of 'ints' that has length '1', 'N' or 'N+2'. The
           size
           of the window for each dimension of the input tensor.
    strides: An int or list of 'ints' that has length '1', 'N' or 'N+2'. The
            stride of the sliding window for each dimension of the input tensor.
    padding: A string, either 'VALID' or 'SAME'. The padding algorithm.
            See
            the "returns" section of 'tf.nn.convolution' for details.
    data_format: A string. Specifies the channel dimension. For N=1 it can be
                either "NWC" (default) or "NCW", for N=2 it can be either "NHWC"
                (default)
                or "NCHW" and for N=3 either "NDHWC" (default) or "NCDHW".
    name: Optional name for the operation.

Returns:
    A 'Tensor' of format specified by 'data_format'.
    The max pooled output tensor.
'''

```

第三部分 神经网络形式化

6 全连接神经网络

6.1 数学推导

6.1.1 作用过程示例

变量说明：

- 以零为下角标的变量为偏置 (bias)。
- x 表示输入数据 (一个训练样本为一个行向量), y 表示标签 (one-hot 编码, 为一个行向量), s 为求和值, a 为激活值。
- x_j^i 的上标表示第 i 行数据样本, 下标表示该行数据样本的第 j 个值。
- x, s, a 若无下角标, 则表示向量化。
- 多分类标签 y 采用 one-hot 编码。

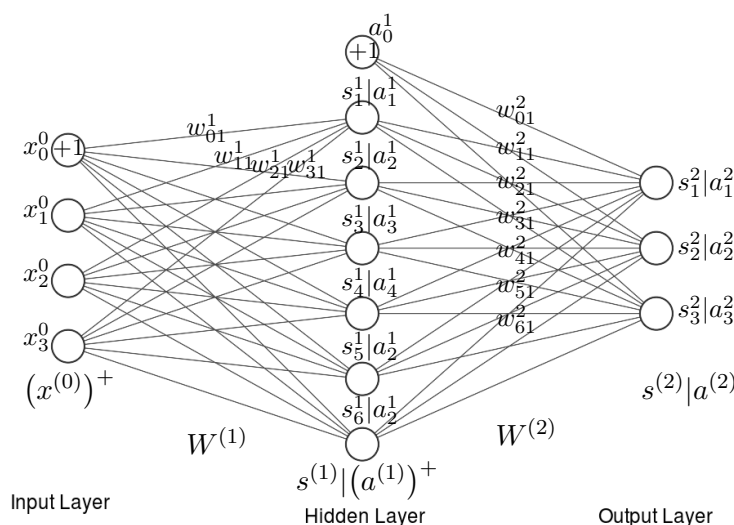
定义 1. 向量上标+运算符: 为该向量添加Bias列, 具体在第0列添加全1。

向量上标-运算符: 为该向量删去Bias列, 具体删除第0列。

例 1.
$$a = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \quad a^+ = \begin{pmatrix} 1 & a & b & c \\ 1 & d & e & f \\ 1 & g & h & i \end{pmatrix} \quad a^- = \begin{pmatrix} b & c \\ e & f \\ h & i \end{pmatrix}$$

定义 2. 同维矩阵对应位置相乘运算: $A \circ B$.

输入层三个神经元, 一个隐藏层六个神经元, 输出层三个神经元(三分类)为例。
激活函数以逻辑函数, 损失函数以均方差损失为例。



作用过程:

1. 向前传播计算误差:

$$(x^{(0)})^+ w^{(1)} = s^{(1)}, \quad \sigma(s^{(1)}) = a^{(1)}$$

$$(a^{(1)})^+ w^{(2)} = s^{(2)}, \quad \sigma(s^{(2)}) = a^{(2)}$$

$$Loss \quad L = \frac{1}{2} \sum_{k=1}^3 (a_k^{(2)} - y_k)^2$$

假设有 1 个训练样本:

- (1) x 的维度就是 1×3 。
- (2) x^+ 的维度就是 1×4 。
- (3) $w^{(1)}$ 的维度就是 4×6 。
- (4) $s^{(1)}$ 的维度就是 1×6 。
- (5) $a^{(1)}$ 的维度就是 1×6 。
- (6) $(a^{(1)})^+$ 的维度就是 1×7 。
- (7) $w^{(2)}$ 的维度就是 7×3 。
- (8) $s^{(2)}$ 的维度就是 1×3 。
- (9) $a^{(2)}$ 的维度就是 1×3 。(输出层)

2. 误差反向传播计算权值梯度:

(1) 输出层权值梯度:

$$\begin{aligned} Loss, \quad L &= \frac{1}{2} \sum_{i=1}^K (a_i^{(2)} - y_i)^2 \\ \frac{\partial L}{\partial w_{ik}^{(2)}} &= \frac{\partial}{\partial w_{ik}^{(2)}} \frac{1}{2} \sum_{i=1}^K (a_i^{(2)} - y_i)^2 \\ &= (\sigma(s_k^{(2)}) - y_k) \sigma(s_k^{(2)}) (1 - \sigma(s_k^{(2)})) \frac{\partial s_k^{(2)}}{\partial w_{ik}^{(2)}} \\ &= (\sigma(s_k^{(2)}) - y_k) \sigma(s_k^{(2)}) (1 - \sigma(s_k^{(2)})) (a_i^{(1)})^+ \\ \text{令, } \delta_k^{(2)} &= (\sigma(s_k^{(2)}) - y_k) \sigma(s_k^{(2)}) (1 - \sigma(s_k^{(2)})) \\ &= (a_k^{(2)} - y_k) a_k^{(2)} (1 - a_k^{(2)}) \\ \text{则, } \frac{\partial L}{\partial w_{ik}^{(2)}} &= \delta_k^{(2)} (a_i^{(1)})^+ \end{aligned}$$

(2) 隐藏层权值梯度:

$$\begin{aligned} \frac{\partial L}{\partial w_{ij}^{(1)}} &= \frac{\partial}{\partial w_{ij}^{(1)}} \frac{1}{2} \sum_{k \in K} (a_k^{(2)} - y_k)^2 \\ &= \sum_{k \in K} (a_k^{(2)} - y_k) \frac{\partial \sigma(s_k^{(2)})}{\partial w_{ij}^{(1)}} \end{aligned}$$

$$\begin{aligned}
&= \sum_{k \in K} (a_k^{(2)} - y_k) \sigma(s_k^{(2)}) (1 - \sigma(s_k^{(2)})) \frac{\partial s_k^{(2)}}{\partial w_{ij}^{(1)}} \\
&= \sum_{k \in K} (a_k^{(2)} - y_k) \sigma(s_k^{(2)}) (1 - \sigma(s_k^{(2)})) \frac{\partial s_k^{(2)}}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}} \\
&= \sum_{k \in K} (a_k^{(2)} - y_k) \sigma(s_k^{(2)}) (1 - \sigma(s_k^{(2)})) w_{jk}^{(2)} \frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}} \\
&= \frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}} \sum_{k \in K} (a_k^{(2)} - y_k) \sigma(s_k^{(2)}) (1 - \sigma(s_k^{(2)})) w_{jk}^{(2)} \\
&= a_j^{(1)} (1 - a_j^{(1)}) \frac{\partial s_j^{(1)}}{w_{ij}^{(1)}} \sum_{k \in K} \delta_k^{(2)} w_{jk}^{(2)} \\
&= a_j^{(1)} (1 - a_j^{(1)}) (x_i^{(0)})^+ \sum_{k \in K} \delta_k^{(2)} w_{jk}^{(2)} \\
\text{令, } \quad \delta_j^{(1)} &= a_j^{(1)} (1 - a_j^{(1)}) \sum_{k \in K} \delta_k^{(2)} w_{jk}^{(2)} \\
\text{则, } \quad \frac{\partial L}{\partial w_{ij}^{(1)}} &= \delta_j^{(1)} (x_i^{(0)})^+
\end{aligned}$$

3. 更新权值:

$$W = W - \eta \frac{\partial Loss}{\partial W}$$

6.1.2 作用过程推广

用户指定神经网络相关参数:

变量名称	符号	备注
训练样本维度	N_t	
训练样本数量	N_T	
隐藏层数	H	包含H个隐藏层
第i层神经元个数	$N^{(i)}$	$i = 0, 1, \dots, H, H + 1$ $i = H + 1$, 输出层, $N^i = K$ 这里定义不带 bias 神经元。
第i层的激活函数	$\sigma^{(i)}$	在常用的 sigmoid , tanh , relu , leaky relu 中选择
分类类别数	K	
损失函数	S_l	在常用的 MeanSquareError , CorssEntropyLoss 中选择

常见激活函数:

1. sigmoid

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

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

2. tanh

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 1 - \frac{2e^{-z}}{e^z + e^{-z}}$$

$$\sigma'(z) = 1 - \sigma(z)^2$$

3. relu

$$\sigma(z) = \begin{cases} 0, & \text{if } z < 0 \\ z, & \text{otherwise.} \end{cases}$$

$$\sigma'(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{otherwise.} \end{cases}$$

4. leaky relu

$$\sigma(z) = \begin{cases} z, & \text{if } z > 0 \\ az, & \text{otherwise.} \end{cases}$$

$$\sigma'(z) = \begin{cases} 1, & \text{if } z > 0 \\ a, & \text{otherwise.} \end{cases}$$

5. softmax

第 i 个类目的概率, $p_i = a_i^{(N+1)} = \sigma(s_i^{(N+1)}) = \frac{e^{s_i^{(N+1)}}}{\sum_{j=1}^K e^{s_j^{(N+1)}}}$

$$\frac{\partial p_i}{\partial s_j} = \frac{\partial \frac{e^{s_i}}{\sum_{k=1}^N e^{s_k}}}{\partial s_j}$$

$$f(x) = \frac{g(x)}{h(x)}, \quad f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h^2(x)}$$

$$g(x) = e^{s_i}, \quad h(x) = \sum_{k=1}^N e^{s_k}$$

$$i == j$$

$$i \neq j$$

$$\begin{aligned}
\frac{\partial \frac{e^{s_i}}{\sum_{k=1}^N e^{s_k}}}{\partial s_j} &= \frac{e^{s_i} \sum_{k=1}^N e^{s_k} - e^{s_j} e^{s_i}}{\left(\sum_{k=1}^N e^{s_k}\right)^2} & \frac{\partial \frac{e^{s_i}}{\sum_{k=1}^N e^{s_k}}}{\partial s_j} &= \frac{0 - e^{s_j} e^{s_i}}{\left(\sum_{k=1}^N e^{s_k}\right)^2} \\
&= \frac{e^{s_i} (\sum_{k=1}^N e^{s_k} - e^{s_j})}{\left(\sum_{k=1}^N e^{s_k}\right)^2} & &= -\frac{e^{s_i}}{\sum_{k=1}^N e^{s_k}} \frac{e^{s_j}}{\sum_{k=1}^N e^{s_k}} \\
&= p_i(1 - p_j) & &= -p_i p_j
\end{aligned}$$

总结:

$$\begin{aligned}
\delta_{ij} &= \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \\
\Rightarrow \frac{\partial p_i}{\partial s_j} &= p_i(\delta_{ij} - p_j)
\end{aligned}$$

常见损失函数:

1. Mean Square Error

$$\begin{aligned}
Loss(z, y) &= \frac{1}{2} \sum_{i=1}^K (z_i - y_i)^2 \\
\frac{\partial Loss}{\partial z} &= (z - y) \quad \text{向量相减}
\end{aligned}$$

2. Cross Entropy Loss

$$\begin{aligned}
Loss(z, y) &= -\sum_{i=1}^K y_i \log z_i = -y_k \log z_k \quad (\text{one-hot 只有第k个分量为 1}) \\
\frac{\partial Loss}{\partial z} &= -\frac{y}{z} \quad \text{向量相除}
\end{aligned}$$

一般如果多分类问题输出层使用 **CrossEntropy** 损失, 并用 **Softmax** 激活。
 因为有:

$$\frac{\partial Loss(a^{(N+1)}, y)}{\partial s_j^{(N+1)}} = a_j^{(N+1)} - y_j$$

推导:

$$\frac{\partial Loss(a^{(N+1)}, y)}{\partial s_j^{(N+1)}} = \sum_{i=1}^K \frac{\partial Loss(a^{(N+1)}, y)}{\partial a_i} \frac{\partial a_i}{\partial s_j}$$

$$\begin{aligned}
&= -\sum_{i=1}^K \frac{y_i}{a_i} \frac{\partial a_i}{\partial s_j} \\
&= \left(-\frac{y_i}{a_i} \frac{\partial a_i}{\partial s_j} \right)_{i=j} - \sum_{i=1, i \neq j}^K \frac{y_i}{a_i} \frac{\partial a_i}{\partial s_j} \\
&= -\frac{y_j}{a_j} a_j (1 - a_j) - \sum_{i=1, i \neq j}^K \frac{y_i}{a_i} \cdot -a_i a_j \\
&= -y_j + y_j a_j + \sum_{i=1, i \neq j}^K y_i a_j \\
&= -y_j + a_j \sum_{i=1}^K y_i \\
&= a_j^{(N+1)} - y_j
\end{aligned}$$

变量说明:

变量	含义	备注
X	输入训练样本	维度 $N_T \times N_t$, 相当于激活值矩阵 $A^{(0)}$.
X^+	输入训练样本加偏置列	维度 $N_T \times (N_t + 1)$, 相当于激活值矩阵 $(A^{(0)})^+$.
$S^{(i)}$	第 i 层求和值矩阵	维度 $N_T \times N^{(i)}$, $i = 1, \dots, H, H+1$.
$A^{(i)}$	第 i 层激活值矩阵	维度 $N_T \times N^{(i)}$, $i = 0, 1, \dots, H, H+1$.
$(A^{(i)})^+$	第 i 层激活值矩阵加偏置列	维度 $N_T \times (N^{(i)} + 1)$, $i = 0, 1, \dots, H$.
$W^{(i)}$	第 i 层权值矩阵	维度 $N^{(i-1)} \times N^{(i)}$, $i = 1, \dots, H, H+1$.
$W^{(H+1)}$	输出层权值矩阵	维度 $N^{(N)} \times K$, 标签有 K 个类目。
$W_b^{(i)}$	第 i 层权值偏置矩阵	维度 $(N^{(i-1)} + 1) \times N^{(i)}$, $i = 1, \dots, H, H+1$.
$W_b^{(H+1)}$	输出层权值偏置矩阵	维度 $(N^{(N)} + 1) \times K$, 标签有 K 个类目。
$S^{(H+1)}$	输出层求和值矩阵	维度 $N_T \times K$.
$A^{(H+1)}$	输出层激活值矩阵	维度 $N_T \times K$.
Y	训练数据标签	维度 $N_T \times K$ (ONE-HOT).
$\Delta^{(i)}$	第 i 层的反向传播中间变量矩阵	维度 $N_T \times N^{(i)}$, $i = 1, \dots, H, H+1$.
$\Delta^{(H+1)}$	输出层反向传播中间变量矩阵	维度 $N_T \times K$.

神经网络向前传播

$$\begin{aligned}
(A^{(0)})^+ W_b^{(1)} &= S^{(1)} \xrightarrow{\sigma+b} (A^{(1)})^+ \Rightarrow \dots \Rightarrow \dots \\
&\Rightarrow (A^{(H-1)})^+ W_b^{(H)} = S^{(H)} \xrightarrow{\sigma+b} (A^{(H)})^+ \\
&\Rightarrow (A^{(H)})^+ W_b^{(H+1)} = S^{(H+1)} \xrightarrow{\sigma} A^{(H+1)} \\
&\Rightarrow \mathcal{L}(A^{(H+1)}, Y)
\end{aligned}$$

神经网络反向传播

1. 输出层权值求导:

$$\nabla_{W_b^{(H+1)}} \mathcal{L}(A^{(H+1)}, Y) = \frac{1}{H_T} (A^{(H)+})^T \Delta^{(H+1)} \quad (\text{矩阵化})$$

$$\Delta^{(H+1)} = \frac{\partial \mathcal{L}(A^{(H+1)}, Y)}{\partial A^{(H+1)}} \circ \sigma'(S^{(H+1)})$$

当输出层使用 Softmax+CrossEntropy 时:

$$\Delta^{(H+1)} = A^{(H+1)} - Y$$

2. 隐藏层权值求导:

$$\nabla_{W_b^{(H)}} \mathcal{L}(A^{(H+1)}, Y) = \frac{1}{H_T} (A^{(H-1)+})^T \Delta^{(H)}$$

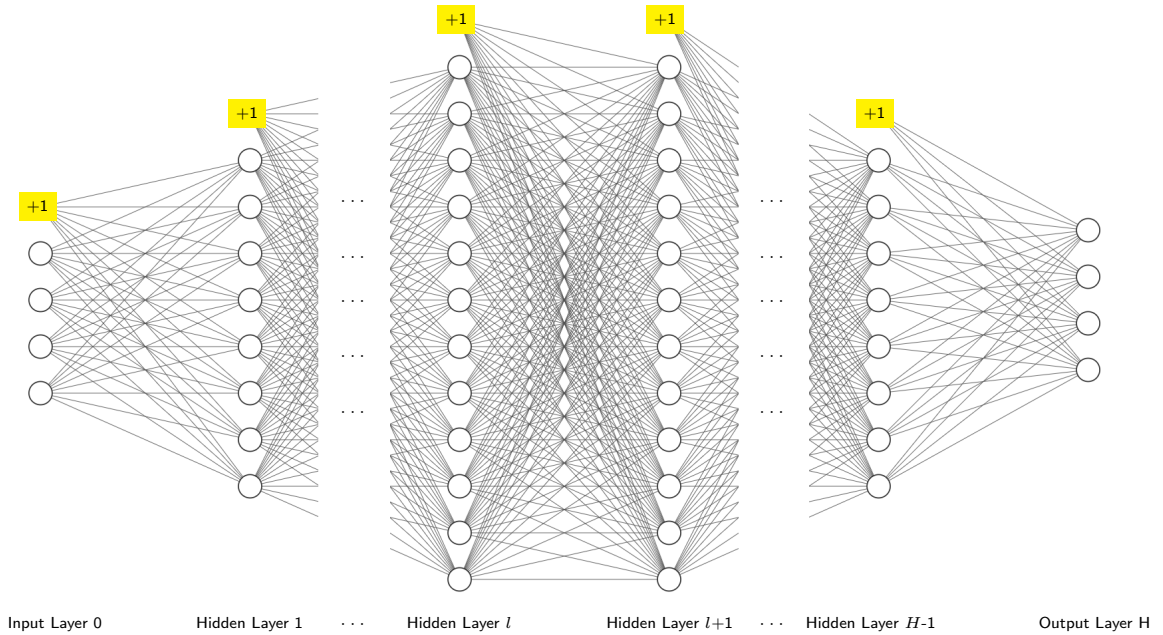
$$\text{令: } \Delta^{(H)} = (\Delta^{(H+1)} (W^{(H+1)})^T) \circ \sigma'(S^{(H)})$$

扩展: 对于第 i 个隐藏层的权值矩阵的梯度为 $(i = 1, 2, \dots, H)$:

$$\nabla_{W_b^{(i)}} \mathcal{L}(A^{(H+1)}, Y) = \frac{1}{H_T} (A^{(i-1)+})^T \Delta^{(i)}$$

$$\text{令: } \Delta^{(i)} = \Delta^{(i+1)} (W^{(i+1)})^T \circ \sigma'(S^{(i)})$$

全连接神经网络结构图



神经网络张量流:

前向传播	维度	反向传播	维度
\mathbf{X}	$N_T \times (N_t + 1)$	LOSS	Scalar
$S^{(1)}$	$N_T \times N^{(1)}$	$\Delta^{(H+1)}$	$N_T \times K$
$(A^{(1)})^+$	$N_T \times (N^{(1)} + 1)$	$\nabla_{W^{(H+1)}} \mathcal{L}$	$(N^{(H)} + 1) \times K$
$S^{(2)}$	$N_T \times N^{(2)}$	$\Delta^{(H)}$	$N_T \times (N^{(H)} + 1)$
$(A^{(2)})^+$	$N_T \times (N^{(2)} + 1)$	$\nabla_{W^{(N)}} \mathcal{L}$	$(N^{(H-1)} + 1) \times N^{(H)}$
\vdots	\vdots	\vdots	\vdots
$S^{(H)}$	$N_T \times N^{(H)}$	$\Delta^{(i)}$	$N_T \times (N^{(i)} + 1)$
$(A^{(H)})^+$	$N_T \times (N^{(H)} + 1)$	$\nabla_{W^{(i)}} \mathcal{L}$	$(N^{(i-1)} + 1) \times N^{(i)}$
$S^{(H+1)}$	$N_T \times K$	$\Delta^{(1)}$	$N_T \times (N^{(1)} + 1)$
$A^{(H+1)}$	$N_T \times K$	$\nabla_{W^{(1)}} \mathcal{L}$	$(N_t + 1) \times N^{(1)}$
comput Loss		update Weight	

优化算法

批量梯度下降 (Batch Gradient Descent, BGD) 批量梯度下降法是最原始的形式，它是指在每一次迭代时使用所有样本来进行梯度的更新。

1. 对目标函数求导:

$$\frac{\Delta J(\theta_0, \theta_1)}{\Delta \theta_j} = \frac{1}{m} \sum_{i=1}^m \left(h_{\theta} \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

其中 $i = 1, 2, \dots, m$ 表示样本数, $j = 0, 1$ 表示特征数, 这里我们使用了偏置项 $x_0^{(i)} = 1$ 。

2. 每次迭代对参数进行更新:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m \left(h_{\theta} \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

伪码:

```
repeat{
     $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$ 
    (for  $j = 0, 1$ )
}
```

1. 优点:

- 一次迭代是对所有样本进行计算，此时利用矩阵进行操作，实现了并行。
- 由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。当目标函数为凸函数时，BGD一定能够得到全局最优。

2. 缺点:

- 当样本数目 m 很大时，每迭代一步都需要对所有样本计算，训练过程很慢。

随机梯度下降 (Stochastic Gradient Descent, SGD) 随机梯度下降法不同于批量梯度下降，随机梯度下降是每次迭代使用一个样本来对参数进行更新。使得训练速度加快。

1. 对一个样本的目标函数求偏导:

$$\frac{\Delta J^{(i)}(\theta_0, \theta_1)}{\theta_j} = (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

2. 参数更新:

$$\theta_j := \theta_j - \alpha (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

伪码:

```
repeat{
    for  $i = 1, \dots, m$ {
         $\theta_j := \theta_j - \alpha (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$ 
        (for  $j = 0, 1$ )
    }
}
```

1. 优点:

- 由于不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样每一轮参数的更新速度大大加快。

2. 缺点:

- 准确度下降。由于即使在目标函数为强凸函数的情况下，SGD仍旧无法做到线性收敛。
- 可能会收敛到局部最优，由于单个样本并不能代表全体样本的趋势。
- 不易于并行实现。

小批量梯度下降 (Mini-Batch Gradient Descent, MBGD) 是对 BGD 以及 SGD 的一个折中办法。其思想是：每次迭代使用 `batch.size` 个样本来对参数进行更新。

这里我们假设 `batch.size=10`，样本数 `m=1000`。

伪码：

```
repeat{
  for i = 1, 11, 21, 31, ..., 991{
     $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$ 
    (for  $j = 0, 1$ )
  }
}
```

1. 优点：

- 通过矩阵运算，每次在一个 `batch` 上优化神经网络参数并不会比单个数据慢太多。
- 每次使用一个 `batch` 可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。（比如上例中的 30W，设置 `batch.size=100` 时，需要迭代 3000 次，远小于 SGD 的 30W 次）
- 可实现并行化。

2. 缺点：

- `batch.size` 的不当选择可能会带来一些问题。

6.2 MSVL 语言框架

数据类型：单精度浮点数。

6.2.1 辅助函数

1. absolute
2. min
3. equal
4. F2S
5. sigmoid
6. tanh
7. relu
8. leakyRelu

6.2.2 矩阵操作函数

1. MatCreate
2. MatDelete
3. MatSetVal
4. MatShape
5. MatDump
6. MatAdd
7. MatSub
8. MatMul
9. MatProduct
10. MatNumMul
11. MatNumAdd
12. MatTrans
13. MatZeros
14. MatOnes
15. MatEye
16. MatRowSum
17. MatRowMax
18. MatExp
19. MatSquare
20. MatVectorSub
21. MatVectorDiv

- 22. MatCopy
- 23. MatPlusRow
- 24. MatPlusCol

6.2.3 矩阵激活函数

- 1. MatNoneActi
- 2. MatSoftmax
- 3. MatSigmoid
- 4. MatTanh
- 5. MatRelu
- 6. MatLeakyRelu

6.2.4 矩阵激活函数导数函数

- 1. MatDerivationNoneActi 0
- 2. MatDerivationSigmoid 1
- 3. MatDerivationTanh 2
- 4. MatDerivationRelu 3
- 5. MatDerivationLeakyRelu 4
- 6. MatDerivationSoftmax 5

6.2.5 损失函数

- 1. OneHot
- 2. MSE
- 3. CrossEntropy

6.2.6 损失函数的导函数

- 1. MSEDerivative
- 2. CrossEntropyDerivative

6.2.7 神经网络的初始化

神经网络初始化介绍

- 1. AllZero initialization
- 2. Random initialization
- 3. Xavier initialization
- 4. He initialization

相关函数

1. `gaussrand_NORMAL` 标准高斯随机生成
2. `gaussrand` 高斯随机生成
3. `MatInitZero` 全零初始化
4. `MatInitRandomNormalization` 随机初始化
5. `MatInitXavier` Xavier初始化
6. `MatInitHe` 凯明初始化

6.2.8 神经网络用户自定义参数

用户自定义参数	参数表示	数据类型	备注
输入训练样本的数量	<code>N_sample</code>	<code>int</code>	NULL
单一训练样本的维度	<code>D_sample</code>	<code>int</code>	NULL
训练样本的真值	<code>Xval</code>	<code>float *</code>	数组长度为 <code>N_sample * D_sample</code>
训练样本的标签	<code>Yval</code>	<code>float *</code>	数组长度为 <code>N_sample</code>
权值初始化方式	<code>Style_initWeight</code>	<code>int</code>	0 ->全0初始化 1 ->随机初始化 2 ->Xavier初始化 3 ->何凯明初始化
神经网络隐藏层层数	<code>N_hidden</code>	<code>int</code>	注意：这里是隐藏层层数。
各层神经元个数	<code>N_layerNeuron</code>	<code>int *</code>	数组长度为 0(输入层),1,...,N_hidden,N_hidden+1(输出层)。
各层激活函数	<code>Nstr_ActiFsHidden</code>	<code>int *</code>	数组长度为 0(输入层),1,...,N_hidden,N_hidden+1(输出层)。 取值的映射关系： 0 ->no activation (input layer) 1 ->sigmoid 2 ->tanh 3 ->relu 4 ->leaky relu 5 ->softmax (output layer)
分类类别数	<code>N_out</code>	<code>int</code>	NULL
输出层损失函数	<code>Nstr_LossF</code>	<code>int</code>	取值映射关系： 0 ->MSE 1 ->CE

6.2.9 神经网络的运算所需空间变量及创建所需空间函数

神经网络的运算所需空间变量	参数表示	数据类型	备注
神经网络激活值矩阵	P_ActiMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
神经网络激活值矩阵加偏置列	P_ActiMatPlus	Mat *	列表索引i=0(输入层),1,...,N_hidden 加偏置列偏置列全部置1.输出层不需要Plus
神经网络求和矩阵	P_SumMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
神经网络权值矩阵	P_WeightMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
神经网络权值偏置矩阵	P_WeightBiasMat	Mat*	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
训练数据标签	Mat.oneHot	Mat	row=N_sample col=N_out
反向传播中间变量矩阵	P_DeltaMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
权值偏置矩阵导数变量矩阵	P_NablaWbMat	Mat*	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
激活函数对求和值导数矩阵	P_ActiFunDerivation	Mat*	列表索引i = 0(输入层), 1, ..., N hidden, N hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义

创建所需空间函数:

1. SpaceCreateActi 创建激活值矩阵所需空间
2. SpaceCreateActiPlus 创建激活值Plus矩阵所需空间
3. SpaceCreateSum 创建求和值矩阵所需空间
4. SpaceCreateWeight 创建权值矩阵所需空间
5. SpaceCreateWeightBias 创建权值偏置矩阵所需空间
6. SpaceCreateDelta 创建反向传播中间变量矩阵所需空间

6.2.10 初始化神经网络

函数:

1. NNinit

说明:

1. 输入数据喂入激活值 (plus) 矩阵
2. 标签数据喂入矩阵并且进行独热编码
3. 选择常用的初始化方式进行权值初始化

6.2.11 神经网络前向传播

神经网络前向传播所需参数变量	参数表示	数据类型	备注
神经网络激活值矩阵	P_ActiMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
神经网络激活值矩阵加偏置列	P_ActiMatPlus	Mat *	列表索引i=0(输入层),1,...,N_hidden 加偏置列偏置列全部置1.输出层不需要Plus
神经网络求和矩阵	P_SumMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层) i=0 时输入层求和矩阵行列置零无实际含义
神经网络权值偏置矩阵	P_WeightBiasMat	Mat *	列表索引i = 0(输入层), 1, ..., N_hidden, N_hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
训练数据标签	Mat_oneHot	Mat	row=N_sample col=N_out
神经网络隐藏层层数	N_hidden	int	注意：这里是隐藏层层数。
各层激活函数	NStr_ActiFsHidden	int *	数组长度为 0(输入层),1,...,N_hidden,N_hidden+1(输出层). 取值的映射关系： 0 ->no activation (input layer) 1 ->sigmoid 2 ->tanh 3 ->relu 4 ->leaky relu 5 ->softmax (output layer)
输出层损失函数	Nstr_LossF	int	取值映射关系： 0 ->MSE 1 ->CE

1. MatActivate 激活函数选择函数
2. LossFunction 损失函数选择函数
3. NNforward 神经网络前向传播函数-返回损失

6.2.12 神经网络反向传播

神经网络反向传播所需参数变量	参数表示	数据类型	备注
神经网络隐藏层层数	N_hidden	int	注意：这里是隐藏层层数
输入样本的数量	N_sample	int	样本数量（训练样本）
各层神经元个数	N_layerNeuron	int *	数组长度为 0(输入层),1,...,N_hidden,N_hidden+1(输出层)
神经网络求和矩阵	P_SumMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
i=0 时输入层求和矩阵行列置零无实际含义	P_ActiMat	Mat *	列表索引i=0(输入层),1,...,N_hidden, N_hidden+1(输出层)
神经网络激活值矩阵加偏置列	P_ActiMatPlus	Mat *	列表索引i=0(输入层),1,...,N_hidden 加偏置列偏置列全部置1.输出层不需要Plus
反向传播中间变量矩阵	P_DeltaMat	Mat *	列表索引i = 0(输入层), 1, ..., N_hidden, N_hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义
训练数据标签	Mat_oneHot	Mat	row = N_sample col = N_out
训练数据标签	Mat_oneHot	Mat	row=N_sample col=N_out
各层激活函数	NStr_ActiFsHidden	int *	数组长度为 0(输入层),1,...,N_hidden,N_hidden+1(输出层). 取值的映射关系： 0 ->no activation (input layer) 1 ->sigmoid 2 ->tanh 3 ->relu 4 ->leaky relu 5 ->softmax (output layer)
输出层损失函数	Nstr_LossF	int	取值映射关系： 0 ->MSE 1 ->CE
激活函数对求和值导数矩阵	P_ActiFunDerivation	Mat *	列表索引i = 0(输入层), 1, ..., N_hidden, N_hidden + 1(输出层) i = 0 时输入层求和矩阵行列置零无实际含义

1. ActiFunDerivation

2. LossFunDerivation
3. NNOutputLayerBackward
4. NNBackward

6.2.13 优化算法

1. BGD (批量梯度下降)
2. SGD (随机梯度下降)
3. MBGD (小批量梯度下降)
4. Adam

demo 成型，损失减小，可收敛。

6.3 MSVL 语言框架重构

6.3.1 对象结构体

1. 矩阵 结构体

```
typedef struct {  
    int row, col;    // rowNum and columnNum [int]  
    float** element; // element, two dimensions  
}Mat;
```

2. BP单层 结构体

```
typedef struct{  
    Mat ActiMat;           // active value Matrix without bias column  
    Mat ActiMatPlus;       // active value Matrix with bias column  
    Mat SumMat;            // sum value Matrix  
    Mat WeightMat;         // weight value Matrix without bias row  
    Mat WeightBiasMat;     // weight value Matrix with bias row  
    Mat DeltaMat;         // backtrack temporary variable Matrix  
    Mat NablaWbMat;        // gradient Matrix for weight with bias  
    Mat ActiFunDerivationMat; // active Function Dervation Matrix  
  
    int NeuronNum;         // number of neuron [int]  
    int AcitFuncNum;       // active function [int]  
}FCLayer;
```

3. BP神经网络 结构体

```
typedef struct{  
    int CurrentSampleNum;    // number of current samples [int]  
    int SampleDimensionNum;  // dimensions(features) of sample [int]  
    int HiddenLayerNum;     // number of hidden layer [int]  
    int WeightInitWayNum;    // weight initialization mode [int]  
  
    FCLayer *Layer;         // layers of FCNN  
    Mat OnehotMat;          // onehot code Matrix  
  
    int ClassificationNum;   // Number of categories classified [int]  
    int LossFuncNum;        // loss function [int]  
}FCNN;
```


4. 用户自定义参数 结构体

```
typedef struct{
    int CompleteSampleNum;    // number of all samples [int]
    int TrainSampleNum;      // number of training samples [int]
    int TestSampleNum;       // number of test samples [int]
    int ValidationNum;       // number of validation samples [int]
    int SampleDimensionNum;  // dimensions(features) of sample [int]
    int HiddenLayerNum;     // number of hidden layer [int]
    int WeightInitWayNum;    // weight initialization mode [int]
    float *XValArray;        // samples features value [float*]
    float *YValArray;        // samples labels value [float*]
    int *NeuronNumArray;     // numbers of every layers Neuron [int*]
    int *ActiFuncNumArray;   // activate functions of every layers [int*]
    int ClassificationNum;   // Number of categories classified [int]
    int LossFuncNum;        // loss function [int]
}Custom;
```

5. 数据集 结构体

```
typedef struct{
    Mat CompleteFeatureDataSet; // complete featrue Mat for FCNN
    Mat CompleteLabelDataSet;  // complete label Mat without onehot
    Mat CompleteTrainFeature;  // complete featrue Mat for FCNN training
    Mat CompleteTrainLabel;    // complete label Mat without onehot
    Mat *BatchTrainFeature;    // batch featrue Mat for FCNN training [three
    // dimensions]
    Mat *BatchTrainLabel;      // batch label Mat without onehot [three
    // dimensions]
    Mat *BatchTrainLabelOneHot; // batch label Mat with onehot [three
    // dimensions]
    Mat TestFeature;           // featrue Mat for FCNN test
    Mat TestLabel;             // label Mat without onehot
    Mat TestLabelOneHot;       // label Mat with onehot
    //Mat ValidationFeature; // featrue Mat for FCNN Validation
    //Mat ValidationLabel;    // label Mat withoutone

    int CompleteSampleNum;    // number of all samples [int]
    int TrainSampleNum;      // number of training samples [int]
    int TestSampleNum;       // number of test samples [int]
    //int ValidationNum;     // number of validation samples [int]
```

```

int SampleDimensionNum; // dimensions(features) of sample [int]
int ClassificationNum;  // Number of categories classified [int]
int BatchSize;          // batch size for dataset
int BatchNum;           // number of batch
int remainder;          // the last batch size
}DataSet;

```

batch 机制的实现:建立 **batch** 结构体, 分割数据集, 复制到 **FCNN** 中。

其中 $\text{TrainSampleNum} = \text{BatchSize} * \text{BatchNum} + \text{remainder}$, 当 $\text{remainder} = 0$ 时, 代表能被整除。

6.3.2 开发流程

1. 用户自定义参数及总数据录入;
2. 数据集构建;
3. 神经网络初始化;
4. 前向传播构建;
5. 反向传播构建;
6. 优化算法构建;
7. 神经网络训练构建;
8. 神经网络测试构建。

神经网络用户自定义参数及总数据录入 目的是将相关参数及总的数据赋值给用户自定义结构体中 (其中注意对数组数据的赋值, 传引用即可), 再将各个参数分发给各个结构体。

数据集构建 根据用户输入的数据相关参数将训练测试数据集构建完成。构建路线:

1. 根据训练集数据量以及 **Batch.Size** 大小计算并导入相关参数。
2. 根据整体数据集的各个参数, 开辟所需的全部空间。
3. 将整体数据集数组根据训练测试集数目划分成两个数组, 包括训练特征数组和训练标签数组, 测试特征数组和测试标签数组。
4. 再将训练特征数组和训练标签数组划分成若干个块大小的块训练特征数组和块训练标签数组。
5. 最后将数据喂入到相应的矩阵结构体当中。

神经网络初始化

1. 开辟神经网络所需空间并导入每层相关参数注意创建顺序不可变化。

(1) 创建神经网络层空间

<1> FCLayer

(2) 导入每层参数

<1> NeuronNum

<2> AcitFuncNum

(3) 创建神经网络运算空间

<1> ActiMat

<2> ActiMatPlus

<3> SumMat

<4> ActiFunDerivationMat

<5> DeltaMat

<6> OnehotMat (FCNN)

(4) 创建神经网络参数空间(参数空间的大小不随输入神经网络样本大小而改变)

<1> WeightMat

<2> WeightBiasMat

<3> NablaWbMat

2. 进行权值初始化

(a) WeightMat

(b) WeightBiasMat

神经网络前向传播求损失 神经网络向前传播将小批量输入样本导入神经网络输入层激活值矩阵，并将对应的标签数据进行独热编码，然后进行矩阵运算损失处理得到标量损失值。其中要注意小批量输入样本的数量不一定和已经开辟的神经网络有关矩阵空间的数量相一致，要做检查处理。

1. 小批量余数处理

2. 输入特征数据拷贝到激活值矩阵

3. 输入标签独热编码数据拷贝到神经网络独热编码矩阵中

4. 前向传播矩阵相乘，激活，扩展激活值矩阵

5. 预测值矩阵与真实值矩阵求损失，返回

神经网络反向传播求梯度 神经网络先经过一次正向传播得到损失之后，再求其损失对所有权值偏置的导数得到梯度，用到了反向传播算法。注意，每一次反向传播是建立

在一次正向传播求得损失的基础之上的。

1. 输出层反向传播，求得输出层的中间变量矩阵以及权值导数
2. 隐藏层反向传播，求得各隐藏层的中间变量矩阵以及权值导数

Adam算法构建 Adam算法使用了动量变量 v_t 和RMSPProp算法中小批量随机梯度按元素平方的指数加权移动平均变量 s_t ，并在时间步0将它们中每个元素初始化为0。

默认参数变量: $\beta_1 = 0.9, \beta_2 = 0.999, \eta = 10^{-3}, \epsilon = 10^{-8}$ 。

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) g_t \quad (1)$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) g_t \odot g_t \quad (2)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_1^t} \quad (3)$$

$$\hat{s}_t = \frac{s_t}{1 - \beta_2^t} \quad (4)$$

$$g'_t = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon} \quad (5)$$

$$x_t = x_{t-1} - g'_t \quad (6)$$

1. 创建Adam参数结构体。
2. 初始化结构体参数。
 - (a) 初始化参变量为默认值。
 - (b) 创建变量所需空间。
 - (c) 时间步设置为0。
3. 根据六个公式更新梯度。

6.4 FCNN应用

6.4.1 Minst 手写字体识别

1. 数据集构建。

minst 手写字体识别数据集包括一共 70000 张灰度图片，每张图片 28×28 个像素，图片数据集包括 0~9 十类。全体数据集是经过出最大值归一化后的数据集。

2. 神经网络搭建。

```
CompleteSampleNum: 70000
TrainSampleNum:    60000
TestSampleNum:     10000
SampleDimensionNum: 784
HiddenLayerNum:    2
WeightInitWayNum:  3
ClassificationNum: 10
LossFuncNum:       1
BatchSize:         100
XValArray:(length = 54880000)
YValArray:(length = 70000)
NeuronNumArray:
784  512  256  10
ActiFuncNumArray:(include output layer Acti-Function)
0  3  3  5
OptimizationMethod: Mini-Batch Stochastic gradient descent
```

1. BSGD 收敛性不佳，指的是时间以及准确度上都达不到要求；
2. 开发Adam优化算法，提高准确度提高了五个百分点，以及收敛时间都缩短了1.5倍。
3. 对稀疏矩阵的运算做优化，收敛时间有缩短了一倍。