Computation Graph

y3162

Namespace:CG

Class Inheritance Hierarchy

- Node
  - Leaf1
  - Leaf2
  - Concatenation
  - MMtoM
    - Add
    - Sub
  - MMto1
    - Dots
    - MSE
  - MtoM
    - ReLU
    - Sigmoid
    - Tanh
    - Softmax
  - Mto1
    - Norm2
  - Affine
  - Filter2d
    - Convolution2d
    - MaxPooling2d
    - AveragePooling2d

Class:Node

This is a base class of following classes.

Type	Property	Description
const size_t	domsize	The size of the *data* of the domain.
const size_t	height	The number of rows in the *data*.
const size_t	width	The number of columns in the *data*.
vec1<dtype>	data	The 1-dimensional vector which retains data processed by this node.
vec1<dtype>	grad	The partial derivative of the cost function with respect to *data*.
vec1<Node∗>	forward	The vector of pointers to the next layer's nodes.
vec1<Node∗>	backward	The vector of pointers to the previous layer's nodes.
int	f_count	The number of nodes coming from the next layer during backpropagation.
int	b_count	The number of nodes coming from the previous layer during forwardpropagation.

Type	Method	Description
void	pushThis(Node ∗node)	Add this as an argument's *forward*.
virtual void	calcData()	Calculate *data* by using previous layer's nodes.
void	forwardPropagation()	After all forward propagations from the previous layer are completed, propagate to the next layer.
virtual void	calcPartialDerivative()	Calculate the *grad* for the previous layer by using the *grad* at this node.
void	backwardPropagation()	After receiving all backward propagations from the next layer, propagate backward to the previous layer.
virtual void	updateParameters(dtype eta)	Update this node's parameters.
void	update(dtype eta)	After updating all parameters of a layer, update the parameters of the previous layer.

Class:Leaf1 extends Node

This class represents a leaf node of 1-dimension.

Constructor	Description
Leaf1 (size_t size)	Create a leaf node with the specified size as its data's size.

Type	Method	Description
void	getInput(vec1<dtype> input)	Provide a 1-dimensional vector as input to the *data*.

Class:Leaf2 extends Node

This class represents a leaf node of 2-dimension.

Constructor	Description
Leaf2 (size_t height, size_t width)	Create a leaf node with the specified height and width as its data's size.

Type	Method	Description
void	getInput(vec1<dtype> input)	Provide flattened 2-dimensional vector as input to the *data*.
void	getInput(vec2<dtype> input)	Provide a 2-dimensional vector as input to the *data*.

Class:Concatenation extends Node

This class represents a node that concatenates the outputs of the previous layers.

Constructor	Description
Concatenation (vec1<Node*>)	Create a Concatenation node.

Class:MMtoM extends Node

This class represents a node that processes the *data* from the previous layer's nodes and updates the *data* for this node. $\mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R}^m$.

Constructor	Description
MMtoM (Node* node1, Node* node2)	Create a MMtoM node with specified two nodes.

Class:Add extends MMtoM

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.
$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.
$\boldsymbol{d} \in \mathbb{R}^m$: The *data* in this node.

$$\boldsymbol{d} = \boldsymbol{x} + \boldsymbol{y} \Longrightarrow d_i = x_i + y_i$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d_i}{\partial x_i}\frac{\partial L}{\partial d_i} = \frac{\partial L}{\partial d_i}$$
$$\frac{\partial L}{\partial y_i} = \frac{\partial d_i}{\partial y_i}\frac{\partial L}{\partial d_i} = \frac{\partial L}{\partial d_i}$$

Class:Sub extends MMtoM

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.
$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.
$\boldsymbol{d} \in \mathbb{R}^m$: The *data* in this node.

$$\boldsymbol{d} = \boldsymbol{x} - \boldsymbol{y} \Longrightarrow d_i = x_i - y_i$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d_i}{\partial x_i}\frac{\partial L}{\partial d_i} = \frac{\partial L}{\partial d_i}$$
$$\frac{\partial L}{\partial y_i} = \frac{\partial d_i}{\partial y_i}\frac{\partial L}{\partial d_i} = -\frac{\partial L}{\partial d_i}$$

Class:MMto1 extends Node

This class represents a node that processes the *data* from the previous layer's nodes and updates the *data* for this node. $\mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R}$.

Constructor	Description
MMto1 (Node* node1, Node* node2)	Create a MMto1 node with specified two nodes.

Class:Dots extends MMto1

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.

$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.

$d \in \mathbb{R}$: The *data* in this node.

$$d = \boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{m} x_i y_i$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d}{\partial x_i}\frac{\partial L}{\partial d} = y_i \frac{\partial L}{\partial d}$$
$$\frac{\partial L}{\partial y_i} = \frac{\partial d}{\partial y_i}\frac{\partial L}{\partial d} = x_i \frac{\partial L}{\partial d}$$

Class:MSE extends MMto1

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.

$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.

$d \in \mathbb{R}$: The *data* in this node.

$$d = \frac{1}{m} \|\boldsymbol{x} - \boldsymbol{y}\|_2^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - y_i)^2$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d}{\partial x_i}\frac{\partial L}{\partial d} = \frac{2\,(x_i - y_i)}{m}\frac{\partial L}{\partial d}$$
$$\frac{\partial L}{\partial y_i} = \frac{\partial d}{\partial y_i}\frac{\partial L}{\partial d} = -\frac{2\,(x_i - y_i)}{m}\frac{\partial L}{\partial d}$$

Class:CEE extends MMto1

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.

$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.

$d \in \mathbb{R}$: The *data* in this node.

$$d = -\boldsymbol{y} \cdot \ln \boldsymbol{x} = -\sum_{i=1}^{m} y_i \ln x_i$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d}{\partial x_i}\frac{\partial L}{\partial d} = -\frac{y_i}{x_i}\frac{\partial L}{\partial d}$$
$$\frac{\partial L}{\partial y_i} = \frac{\partial d}{\partial y_i}\frac{\partial L}{\partial d} = -\ln x_i \frac{\partial L}{\partial d}$$

Class:MtoM extends Node

This class represents a node that processes the *data* from the previous layer's node and updates the *data* for this node. $\mathbb{R}^m \longrightarrow \mathbb{R}^m$.

Constructor	Description
MtoM (Node* node1)	Create a MtoM node with a specified node.

Class:ReLU extends MtoM

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.
$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.
$\boldsymbol{d} \in \mathbb{R}^m$: The *data* in this node.

$$\boldsymbol{d} = \mathrm{ReLU}\left(\boldsymbol{x}\right) \Longrightarrow d_i = \mathrm{ReLU}\left(x_i\right) = \begin{cases} x_i & x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d_i}{\partial x_i}\frac{\partial L}{\partial d_i} = 1[x_i \geq 0]\frac{\partial L}{\partial d_i}$$

Class:Sigmoid extends MtoM

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.
$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.
$\boldsymbol{d} \in \mathbb{R}^m$: The *data* in this node.

$$\boldsymbol{d} = \mathrm{sigmoid}\left(\boldsymbol{x}\right) \Longrightarrow d_i = \mathrm{sigmoid}\left(x_i\right) = \frac{1}{1 + \exp\left(-x_i\right)}$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d_i}{\partial x_i}\frac{\partial L}{\partial d_i} = d\left(1 - d\right)\frac{\partial L}{\partial d_i}$$

Class:Tanh extends MtoM

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.
$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.
$\boldsymbol{d} \in \mathbb{R}^m$: The *data* in this node.

$$\boldsymbol{d} = \tanh\left(\boldsymbol{x}\right) \Longrightarrow d_i = \tanh\left(x_i\right) = \frac{\exp\left(x_i\right) - \exp\left(-x_i\right)}{\exp\left(x_i\right) + \exp\left(-x_i\right)}$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d_i}{\partial x_i}\frac{\partial L}{\partial d_i} = \left(1 - d_i^2\right)\frac{\partial L}{\partial d_i}$$

Class:Softmax extends MtoM

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.

$\boldsymbol{y} \in \mathbb{R}^m$: The second node from the previous layer.

$\boldsymbol{d} \in \mathbb{R}^m$: The *data* in this node.

$$\boldsymbol{d} = \text{Softmax}\left(\boldsymbol{x}\right) \Longrightarrow d_i = \text{Softmax}\left(\boldsymbol{x}\right)_i = \frac{\exp\left(x_i\right)}{\sum_{j=1}^{m} \exp\left(x_j\right)}$$

Let L be the cost function,

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial d_j}{\partial x_i} \frac{\partial L}{\partial d_j} = \sum_{j=1}^{m} d_j \left(\delta_{i,j} - d_i\right) \frac{\partial L}{\partial d_j}$$

Class:Mto1 extends Node

This class represents a node that processes the *data* from the previous layer's node and updates the *data* for this node. $\mathbb{R}^m \longrightarrow \mathbb{R}$.

Constructor	Description
Mto1 (Node* node1, Node* node2)	Create a Mto1 node with a specified nodes.

Class:Norm2 extends Mto1

$\boldsymbol{x} \in \mathbb{R}^m$: The first node from the previous layer.
$d \in \mathbb{R}$: The *data* in this node.

$$d = \|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{m} x_i^2}$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \frac{\partial d}{\partial x_i}\frac{\partial L}{\partial d} = \frac{x_i}{d}\frac{\partial L}{\partial d}$$

Class:Affine extends Node

This class represents a node that performs an affine transformation.$\mathbb{R}^m \longrightarrow \mathbb{R}^n$

Type	Property	Description
vec2<dtype>	weight	The affine transformation matrix for this affine transformation.
vec2<dtype>	gradWeight	The sum of the gradients of the *weight*.
const dtype	bias	The bias in this affine transformation.

Constructor	Description
Affine (Node* node1, vec2<dtype> Weight, dtype bias)	Create an affine node with specified *weight* and *bias*.
Affine (Node* node1, vec2<dtype> Weight)	Create an affine node with specified *weight*. *bias* is set to 1.

$\boldsymbol{x} \in \mathbb{R}^m$: The node from the previous layer.

$\boldsymbol{W} \in \mathbb{R}^{(m+1)\times n}$: The weights in this node.

$b \in \mathbb{R}$: The *bias* in this node.

$\boldsymbol{d} \in \mathbb{R}^n$: The *data* in this node.

$$\boldsymbol{d} = \boldsymbol{W}^T \begin{pmatrix} \boldsymbol{x} \\ b \end{pmatrix} \Longrightarrow d_i = \sum_{j=1}^{m} W_{j,i} x_j + W_{m+1,i} b$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial d_j}{\partial x_i} \frac{\partial L}{\partial d_j} = \sum_{j=1}^{n} W_{i,j} \frac{\partial L}{\partial d_j}$$

$$\frac{\partial L}{\partial W_{i,j}} = \sum_{k=1}^{n} \frac{\partial d_k}{\partial W_{i,j}} \frac{\partial L}{\partial d_k} = \begin{cases} x_i \dfrac{\partial L}{\partial d_j} & 1 \le i \le m \\ b \dfrac{\partial L}{\partial d_j} & \text{otherwise} \end{cases}$$

Class:Filter2d extends Node

This class represents a node that performs convolution.

Type	Property	Description
const size_t	kheight	The height of kernel/filter.
const size_t	kwidth	The width of kernel/filter.
const size_t	pl	The size of padding in the left.
const size_t	pt	The size of padding in the top.
const size_t	sw	The stride width.

Constructor	Description
Filter (vec1<Node∗>nodes, size_t kernelHeight, size_t kernelWidth, size_t stride, size_t topPadding, size_t leftPadding, size_t height, size_t width)	Create a Filter node with specified size of kernels/fileters and size of padding to output the specified size of *data*.

$C \in \mathbb{N}$: The number of channnels.
$\boldsymbol{X}^{(c)} \in \mathbb{R}^{m_h \times m_w}$: The c-th node from the previous layer.
$\boldsymbol{K}^{(c)} \in \mathbb{R}^{k_h \times k_w}$: The c-th kernel in this node.
$\boldsymbol{D} \in \mathbb{R}^{n_h \times n_w}$: The *data* in this node.
$p_1 \in \mathbb{N} \cup \{0\}$: The top padding.
$p_2 \in \mathbb{N} \cup \{0\}$: The bottom padding.
$q_1 \in \mathbb{N} \cup \{0\}$: The left padding.
$q_2 \in \mathbb{N} \cup \{0\}$: The right padding.
$s \in \mathbb{N}$: The stride width.
It follows that,

$$n_h = \frac{m_h + (p_1 + p_2) - k_h}{s} + 1, \quad n_w = \frac{m_w + (q_1 + q_2) - k_w}{s} + 1$$

When given m_h, m_w, k_h, k_w, n_h, n_w, s, p_1 and q_1,

$$p_2 = s(n_h - 1) + k_h - m_h - p_1, \quad q_2 = s(n_w - 1) + k_w - m_w - q_1$$

When given only m_h, m_w, k_h, k_w, n_h, n_w and s,

$$p_1 = \left\lfloor \frac{s(n_h - 1) + k_h - m_h}{2} \right\rfloor, \quad q_1 = \left\lfloor \frac{s(n_w - 1) + k_w - m_w}{2} \right\rfloor$$
$$p_2 = \left\lceil \frac{s(n_h - 1) + k_h - m_h}{2} \right\rceil, \quad q_2 = \left\lceil \frac{s(n_w - 1) + k_w - m_w}{2} \right\rceil$$

When given only m_h, m_w, k_h, k_w and s,

$$n_h = \left\lceil \frac{m_h - k_h}{s} \right\rceil + 1, \qquad n_w = \left\lceil \frac{m_w - k_w}{s} \right\rceil + 1$$
$$p_1 = \left\lfloor \frac{s(n_h - 1) + k_h - m_h}{2} \right\rfloor, \quad q_1 = \left\lfloor \frac{s(n_w - 1) + k_w - m_w}{2} \right\rfloor$$
$$p_2 = \left\lceil \frac{s(n_h - 1) + k_h - m_h}{2} \right\rceil, \quad q_2 = \left\lceil \frac{s(n_w - 1) + k_w - m_w}{2} \right\rceil$$

Class:Convolution2d extends Filter2d

This class represents a node that performs convolution.

Type	Property	Description
vec2<dtype>	kernel	The filter used during convolution
vec2<dtype>	gradKernel	The sum of the gradients of the *kernel*
dtype	bias	The bias in this affine transformation.
dtype	gradBias	The sum of the gradients of the *bias*.

Constructor	Description
Convolution2d (vec1<Node*> nodes, vec3<dtype> Kernel, dtye bias, size_t stride, size_t topPadding, size_t leftPadding, size_t height, size_t width)	Create an convolution node with specified *kernel* and *bias*. Virtually, the data of domain is zero-padded to output the specified size of *data*.
Convolution2d (vec1<Node*> nodes, vec3<dtype> Kernel, dtype bias, size_t stride, size_t height, size_t width)	Create an convolution node with specified *kernel* and *bias*. In order to output the specified size of *data*, the data of domain is virtually and automatically zero-padded.
Convolution2d (vec1<Node*> nodes, vec3<dtype> Kernel, dtye bias, size_t stride)	Create an convolution node with specified *kernel* and *bias*. In order to align with the stride width, the data of domain is virtually and automatically zero-padded.
Convolution2d (vec1<Node*> nodes, vec3<dtype> Kernel, dtye bias, size_t height, size_t width)	Create an convolution node with specified *kernel* and *bias*. In order to align with the stride width, the data of domain is virtually and automatically zero-padded with stride width 1.
Convolution2d (vec1<Node*> nodes, vec3<dtype> Kernel, dtye bias)	Create an convolution node with specified *kernel* and *bias*. Stride width is set to 1.

$C \in \mathbb{N}$: The number of channnels.

$\boldsymbol{X}^{(c)} \in \mathbb{R}^{m_h \times m_w}$: The node from the previous layer.

$\boldsymbol{K}^{(c)} \in \mathbb{R}^{k_h \times k_w}$: The kernel in this node.

$b \in \mathbb{R}$: The *bias* in this node.

$\boldsymbol{D} \in \mathbb{R}^{n_h \times n_w}$: The *data* in this node.

$p \in \mathbb{N} \cup \{0\}$: The top padding.

$q \in \mathbb{N} \cup \{0\}$: The left padding.

$s \in \mathbb{N}$: The stride width.

$$\boldsymbol{D} = \sum_{c=1}^{C} \left(\boldsymbol{X}^{(c)} *_s \boldsymbol{K^{(c)}} \right) + b\mathbf{1}_{n_h \times n_w} \Longrightarrow D_{\alpha,\beta} = \sum_{c=1}^{C} \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} K_{i,j}^{(c)} X_{(\alpha-1)s+i-p,(\beta-1)s+j-q}^{(c)} + b$$

Let $L \in \mathbb{R}$ be the cost function,

$$\begin{aligned}
\frac{\partial L}{\partial X_{\alpha,\beta}^{(c)}} &= \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} 1[\alpha - i + p \equiv 0 (\mathrm{mod}\, s)] 1[\beta - j + q \equiv 0 (\mathrm{mod}\, s)] \frac{\partial D_{\frac{\alpha-i+p}{s}+1, \frac{\beta-j+q}{s}}}{\partial X_{\alpha,\beta}^{(c)}} \frac{\partial L}{\partial D_{\frac{\alpha-i+p}{s}+1, \frac{\beta-j+q}{s}+1}} \\
&= \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} 1[\alpha - i + p \equiv 0 (\mathrm{mod}\, s)] 1[\beta - j + q \equiv 0 (\mathrm{mod}\, s)] K_{i,j}^{(c)} \frac{\partial L}{\partial D_{\frac{\alpha-i+p}{s}+1, \frac{\beta-j+q}{s}+1}}
\end{aligned}$$

$$\frac{\partial L}{\partial K_{i,j}^{(c)}} = \sum_{\alpha=1}^{n_h} \sum_{\beta=1}^{n_w} \frac{\partial D_{\alpha,\beta}}{\partial K_{i,j}^{(c)}} \frac{\partial L}{\partial D_{\alpha,\beta}} = \sum_{\alpha=1}^{n_h} \sum_{\beta=1}^{n_w} X_{(\alpha-1)s+i-p,(\beta-1)s+j-q}^{(c)} \frac{\partial L}{\partial D_{\alpha,\beta}}$$

$$\frac{\partial L}{\partial b} = \sum_{\alpha=1}^{n_h} \sum_{\beta=1}^{n_w} \frac{\partial D_{\alpha,\beta}}{\partial b} \frac{\partial L}{\partial D_{\alpha,\beta}} = \sum_{\alpha=1}^{n_h} \sum_{\beta=1}^{n_w} \frac{\partial L}{\partial D_{\alpha,\beta}}$$

Class:MaxPooling2d extends Filter2d

This class represents a node that performs max-pooling.

Type	Property	Description
vec2<unsigned int>	maxCount	The number of elements in the data within domain that have the maximum value in a given filter area.

Constructor	Description
MaxPooling2d (Node *node1, size_t kernelHeight size_t kernelWidth, size_t stride, size_t topPadding size_t leftPadding, size_t height, size_t width)	Create a max-pooling node with specified size. Virtually, the data of domain is zero-padded to output the specified size of *data*.
MaxPooling2d (Node *node1, size_t kernelHeight size_t kernelWidth,, size_t stride, size_t height, size_t width)	Create a max-pooling node with specified size. In order to output the specified size of *data*, the data of domain is virtually and automatically zero-padded.
MaxPooling2d (Node *node1, size_t kernelHeight size_t kernelWidth, size_t stride)	Create a max-pooling node with specified size. In order to align with the stride width, the data of domain is virtually and automatically zero-padded.

$\boldsymbol{X} \in \mathbb{R}^{m_h \times m_w}$: The c-th node from the previous layer.

k_h : The height of the filter.

k_w : The width of the filter.

$\boldsymbol{D} \in \mathbb{R}^{n_h \times n_w}$: The c-th *data* in this node.

$\boldsymbol{N} \in \mathbb{N}^{n_h \times n_w}$: The number of elements in $\boldsymbol{X}$ that have the maximum value for each element in $\boldsymbol{D}$.

$p \in \mathbb{N} \cup \{0\}$: The top padding.

$q \in \mathbb{N} \cup \{0\}$: The left padding.

$s \in \mathbb{N}$: The stride width.

$$\boldsymbol{D} = \max_{s,k_h \times k_w}(\boldsymbol{X}) \Longrightarrow D_{\alpha,\beta} = \max_{\substack{1 \le i \le k_h \\ 1 \le j \le k_w}} \left(X_{(\alpha-1)s+i-p,(\beta-1)s+j-q} \right)$$

Let $L \in \mathbb{R}$ be the cost function,

$$\frac{\partial L}{\partial X_{\alpha,\beta}} = \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} 1[\alpha - i + p \equiv 0 (\mathrm{mod}\, s)] 1[\beta - j + q \equiv 0 (\mathrm{mod}\, s)] \frac{\partial D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}}{\partial X_{\alpha,\beta}} \frac{\partial L}{\partial D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}}$$

$\alpha - i + p \equiv 0 (\mathrm{mod}\, s)$ かつ $\beta - j + q \equiv 0 (\mathrm{mod}\, s)$ のとき、

$$\begin{aligned}
\frac{\partial D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}}{\partial X_{\alpha,\beta}} &= \frac{1\left[X_{\alpha,\beta} = D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}\right]}{N_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}} \\
&= \frac{1\left[X_{\alpha,\beta} = D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}\right]}{\sum_{i'=1}^{k_h} \sum_{j'=1}^{k_w} 1\left[D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1} = X_{(\alpha-i+p)+i'-p,(\beta-j+q)+j'-q}\right]} \\
&= \frac{1\left[X_{\alpha,\beta} = D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}\right]}{\sum_{i'=1}^{k_h} \sum_{j'=1}^{k_w} 1[X_{\alpha,\beta} = X_{\alpha-i+i',\beta-j+j'}]}
\end{aligned}$$

Class:AveragePooling2d extends Filter2d

This class represents a node that performs average-pooling.

Constructor	Description
AveragePooling2d (Node *node1, size_t kernelHeight size_t kernelWidth, size_t stride, size_t topPadding size_t leftPadding, size_t height, size_t width)	Create a average-pooling node with specified size. Virtually, the data of domain is zero-padded to output the specified size of *data*.
AveragePooling2d (Node *node1, size_t kernelHeight size_t kernelWidth,, size_t stride, size_t height, size_t width)	Create a average-pooling node with specified size. In order to output the specified size of *data*, the data of domain is virtually and automatically zero-padded.
AveragePooling2d (Node *node1, size_t kernelHeight size_t kernelWidth, size_t stride)	Create a average-pooling node with specified size. In order to align with the stride width, the data of domain is virtually and automatically zero-padded.

$\boldsymbol{X} \in \mathbb{R}^{m_h \times m_w}$: The node from the previous layer.

k_h : The height of the filter.

k_w : The width of the filter.

$\boldsymbol{D} \in \mathbb{R}^{n_h \times n_w}$: The *data* in this node.

$p \in \mathbb{N} \cup \{0\}$: The top padding.

$q \in \mathbb{N} \cup \{0\}$: The left padding.

$s \in \mathbb{N}$: The stride width.

$$\boldsymbol{D} = \operatorname{avg}_{s,k_h \times k_w}(\boldsymbol{X}) \Longrightarrow D_{\alpha,\beta} = \frac{1}{k_h k_w} \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} X_{(\alpha-1)s+i-p,(\beta-1)s+j-q}$$

Let $L \in \mathbb{R}$ be the cost function,

$$\begin{aligned}
\frac{\partial L}{\partial X_{\alpha,\beta}} &= \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} 1[\alpha - i + p \equiv 0 (\operatorname{mod} s)] 1[\beta - j + q \equiv 0 (\operatorname{mod} s)] \frac{\partial D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}}{\partial X_{\alpha,\beta}} \frac{\partial L}{\partial D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}} \\
&= \frac{1}{k_h k_w} \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} 1[\alpha - i + p \equiv 0 (\operatorname{mod} s)] 1[\beta - j + q \equiv 0 (\operatorname{mod} s)] \frac{\partial L}{\partial D_{\frac{\alpha-i+p}{s}+1,\frac{\beta-j+q}{s}+1}}
\end{aligned}$$