

ceras

Generated by Doxygen 1.9.2

1 Namespace Index	1
1.1 Namespace List	1
2 Concept Index	3
2.1 Concepts	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Namespace Documentation	11
6.1 ceras Namespace Reference	11
6.1.1 Typedef Documentation	19
6.1.1.1 ada_delta	19
6.1.1.2 ada_grad	19
6.1.1.3 default_allocator	19
6.1.1.4 rms_prop	19
6.1.2 Function Documentation	20
6.1.2.1 abs()	20
6.1.2.2 abs_loss()	20
6.1.2.3 Add()	20
6.1.2.4 arg()	20
6.1.2.5 as_tensor()	21
6.1.2.6 AveragePooling2D()	21
6.1.2.7 BatchNormalization()	21
6.1.2.8 binary_accuracy()	22
6.1.2.9 binary_cross_entropy_loss()	22
6.1.2.10 binary_step()	22
6.1.2.11 bind()	22
6.1.2.12 computation_graph()	23
6.1.2.13 Concatenate()	23
6.1.2.14 conj()	23
6.1.2.15 Conv2D() [1/2]	24
6.1.2.16 Conv2D() [2/2]	25
6.1.2.17 crelu()	25
6.1.2.18 cross_entropy()	26
6.1.2.19 cross_entropy_loss()	26
6.1.2.20 Dense()	26
6.1.2.21 divide()	27

6.1.2.22 Dropout()	27
6.1.2.23 elementwise_multiply()	27
6.1.2.24 elementwise_product()	28
6.1.2.25 elu()	28
6.1.2.26 ELU()	28
6.1.2.27 exponential()	28
6.1.2.28 Flatten()	29
6.1.2.29 gaussian()	29
6.1.2.30 gelu()	29
6.1.2.31 get_default_session()	30
6.1.2.32 hadamard_product()	30
6.1.2.33 hard_sigmoid()	30
6.1.2.34 heaviside_step()	30
6.1.2.35 hinge_loss()	31
6.1.2.36 imag()	31
6.1.2.37 Input()	31
6.1.2.38 inverse()	31
6.1.2.39 leaky_relu()	31
6.1.2.40 LeakyReLU()	32
6.1.2.41 lisht()	32
6.1.2.42 lstm()	32
6.1.2.43 mae()	32
6.1.2.44 make_binary_operator()	33
6.1.2.45 make_compiled_model()	33
6.1.2.46 make_trainable()	33
6.1.2.47 make_unary_operator()	33
6.1.2.48 MaxPooling2D()	33
6.1.2.49 mean()	34
6.1.2.50 mean_absolute_error()	34
6.1.2.51 mean_reduce()	34
6.1.2.52 mean_squared_error()	34
6.1.2.53 mean_squared_logarithmic_error()	34
6.1.2.54 minus()	35
6.1.2.55 mish()	35
6.1.2.56 mse()	35
6.1.2.57 Multiply()	35
6.1.2.58 negative()	35
6.1.2.59 negative_relu()	36
6.1.2.60 norm()	36
6.1.2.61 operator"!=()	36
6.1.2.62 operator*() [1/4]	36
6.1.2.63 operator*() [2/4]	37

6.1.2.64 operator*() [3/4]	37
6.1.2.65 operator*() [4/4]	37
6.1.2.66 operator+() [1/6]	37
6.1.2.67 operator+() [2/6]	37
6.1.2.68 operator+() [3/6]	38
6.1.2.69 operator+() [4/6]	38
6.1.2.70 operator+() [5/6]	38
6.1.2.71 operator+() [6/6]	38
6.1.2.72 operator-() [1/6]	38
6.1.2.73 operator-() [2/6]	39
6.1.2.74 operator-() [3/6]	39
6.1.2.75 operator-() [4/6]	39
6.1.2.76 operator-() [5/6]	39
6.1.2.77 operator-() [6/6]	39
6.1.2.78 operator/()	40
6.1.2.79 operator<()	40
6.1.2.80 operator<=()	40
6.1.2.81 operator==([1/2]	40
6.1.2.82 operator==([2/2]	40
6.1.2.83 operator>()	41
6.1.2.84 operator>=()	41
6.1.2.85 plus()	41
6.1.2.86 polar()	41
6.1.2.87 prelu()	41
6.1.2.88 real()	42
6.1.2.89 reduce_mean()	42
6.1.2.90 reduce_sum()	42
6.1.2.91 relu()	42
6.1.2.92 ReLU()	43
6.1.2.93 relu6()	43
6.1.2.94 replace_placeholder_with_expression()	43
6.1.2.95 Reshape()	44
6.1.2.96 run()	44
6.1.2.97 selu()	44
6.1.2.98 sigmoid()	44
6.1.2.99 silu()	45
6.1.2.100 soft_sign()	45
6.1.2.101 Softmax()	45
6.1.2.102 softmax()	45
6.1.2.103 softplus()	46
6.1.2.104 softsign()	46
6.1.2.105 square()	46

6.1.2.106 squared_loss()	47
6.1.2.107 Subtract()	47
6.1.2.108 sum_reduce()	47
6.1.2.109 swish()	47
6.1.2.110 tank_shrink()	48
6.1.2.111 unit_step()	48
6.1.2.112 UpSampling2D()	48
6.1.3 Variable Documentation	48
6.1.3.1 Adadelta	48
6.1.3.2 Adagrad	49
6.1.3.3 Adam	49
6.1.3.4 BinaryCrossentropy	49
6.1.3.5 BinaryCrossEntropy	49
6.1.3.6 CategoricalCrossentropy	50
6.1.3.7 CategoricalCrossEntropy	50
6.1.3.8 Hinge	50
6.1.3.9 is_binary_operator_v	50
6.1.3.10 is_complex_v	50
6.1.3.11 is_constant_v	51
6.1.3.12 is_place_holder_v	51
6.1.3.13 is_tensor_v	51
6.1.3.14 is_unary_operator_v	51
6.1.3.15 is_value_v	51
6.1.3.16 is_variable_v	51
6.1.3.17 MAE	52
6.1.3.18 MeanAbsoluteError	52
6.1.3.19 MeanSquaredError	52
6.1.3.20 MSE	53
6.1.3.21 random_generator	53
6.1.3.22 random_seed	53
6.1.3.23 RMSprop	53
6.1.3.24 SGD	53
6.2 ceras::ceras_private Namespace Reference	54
6.3 ceras::dataset Namespace Reference	54
6.4 ceras::dataset::fashion_mnist Namespace Reference	54
6.4.1 Function Documentation	54
6.4.1.1 load_data()	54
6.5 ceras::dataset::mnist Namespace Reference	55
6.5.1 Function Documentation	55
6.5.1.1 load_data()	55

7 Concept Documentation

57

7.1 ceras::Binary_Operator Concept Reference	57
7.1.1 Concept definition	57
7.1.2 Detailed Description	57
7.2 ceras::Complex Concept Reference	57
7.2.1 Concept definition	57
7.2.2 Detailed Description	57
7.3 ceras::Constant Concept Reference	58
7.3.1 Concept definition	58
7.4 ceras::Expression Concept Reference	58
7.4.1 Concept definition	58
7.4.2 Detailed Description	58
7.5 ceras::Operator Concept Reference	58
7.5.1 Concept definition	58
7.5.2 Detailed Description	58
7.6 ceras::Place_Holder Concept Reference	58
7.6.1 Concept definition	59
7.7 ceras::Tensor Concept Reference	59
7.7.1 Concept definition	59
7.8 ceras::Unary_Operator Concept Reference	59
7.8.1 Concept definition	59
7.8.2 Detailed Description	59
7.9 ceras::Value Concept Reference	59
7.9.1 Concept definition	59
7.10 ceras::Variable Concept Reference	59
7.10.1 Concept definition	59
8 Class Documentation	61
8.1 ceras::adadelta< Loss, T > Struct Template Reference	61
8.1.1 Member Typedef Documentation	61
8.1.1.1 tensor_type	62
8.1.2 Constructor & Destructor Documentation	62
8.1.2.1 adadelta()	62
8.1.3 Member Function Documentation	62
8.1.3.1 forward()	62
8.1.4 Member Data Documentation	62
8.1.4.1 iterations_	62
8.1.4.2 learning_rate_	62
8.1.4.3 loss_	63
8.1.4.4 rho_	63
8.2 ceras::adagrad< Loss, T > Struct Template Reference	63
8.2.1 Member Typedef Documentation	63
8.2.1.1 tensor_type	64

8.2.2 Constructor & Destructor Documentation	64
8.2.2.1 adagrad()	64
8.2.3 Member Function Documentation	64
8.2.3.1 forward()	64
8.2.4 Member Data Documentation	64
8.2.4.1 decay_	64
8.2.4.2 iterations_	64
8.2.4.3 learning_rate_	65
8.2.4.4 loss_	65
8.3 ceras::adam< Loss, T > Struct Template Reference	65
8.3.1 Member Typedef Documentation	66
8.3.1.1 tensor_type	66
8.3.2 Constructor & Destructor Documentation	66
8.3.2.1 adam()	66
8.3.3 Member Function Documentation	66
8.3.3.1 forward()	66
8.3.4 Member Data Documentation	66
8.3.4.1 amsgrad_	66
8.3.4.2 beta_1_	67
8.3.4.3 beta_2_	67
8.3.4.4 iterations_	67
8.3.4.5 learning_rate_	67
8.3.4.6 loss_	67
8.4 ceras::binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output↔_Shape_Calculator > Struct Template Reference	67
8.4.1 Detailed Description	68
8.4.2 Member Typedef Documentation	68
8.4.2.1 tensor_type	68
8.4.3 Constructor & Destructor Documentation	69
8.4.3.1 binary_operator()	69
8.4.4 Member Function Documentation	69
8.4.4.1 backward()	69
8.4.4.2 forward()	69
8.4.4.3 shape()	69
8.4.5 Member Data Documentation	70
8.4.5.1 backward_action_	70
8.4.5.2 forward_action_	70
8.4.5.3 lhs_input_data_	70
8.4.5.4 lhs_op_	70
8.4.5.5 output_data_	70
8.4.5.6 output_shape_calculator_	71
8.4.5.7 rhs_input_data_	71

8.4.5.8 rhs_op_	71
8.5 ceras::compiled_model< Model, Optimizer, Loss > Struct Template Reference	71
8.5.1 Member Typedef Documentation	72
8.5.1.1 io_layer_type	72
8.5.1.2 optimizer_type	72
8.5.2 Constructor & Destructor Documentation	72
8.5.2.1 compiled_model()	72
8.5.3 Member Function Documentation	72
8.5.3.1 evaluate()	72
8.5.3.2 fit()	73
8.5.3.3 operator()()	74
8.5.3.4 predict()	74
8.5.3.5 train_on_batch()	74
8.5.3.6 trainable()	74
8.5.4 Member Data Documentation	75
8.5.4.1 compiled_optimizer_	75
8.5.4.2 ground_truth_place_holder_	75
8.5.4.3 input_place_holder_	75
8.5.4.4 loss_	75
8.5.4.5 model_	75
8.5.4.6 optimizer_	75
8.6 ceras::complex< Real_Ex, Imag_Ex > Struct Template Reference	76
8.6.1 Member Data Documentation	76
8.6.1.1 imag_	76
8.6.1.2 real_	76
8.7 ceras::constant< Tsor > Struct Template Reference	76
8.7.1 Detailed Description	77
8.7.2 Member Typedef Documentation	77
8.7.2.1 tensor_type	77
8.7.3 Constructor & Destructor Documentation	77
8.7.3.1 constant()	77
8.7.4 Member Function Documentation	77
8.7.4.1 backward()	78
8.7.4.2 forward()	78
8.7.4.3 shape()	78
8.7.5 Member Data Documentation	78
8.7.5.1 data_	78
8.8 ceras::gradient_descent< Loss, T > Struct Template Reference	78
8.8.1 Member Typedef Documentation	79
8.8.1.1 tensor_type	79
8.8.2 Constructor & Destructor Documentation	79
8.8.2.1 gradient_descent()	79

8.8.3 Member Function Documentation	79
8.8.3.1 forward()	79
8.8.4 Member Data Documentation	79
8.8.4.1 learning_rate_	80
8.8.4.2 loss_	80
8.8.4.3 momentum_	80
8.9 ceras::identity_output_shape_calculator Struct Reference	80
8.9.1 Detailed Description	80
8.9.2 Member Function Documentation	80
8.9.2.1 operator() [1/3]	81
8.9.2.2 operator() [2/3]	81
8.9.2.3 operator() [3/3]	81
8.10 ceras::is_binary_operator< T > Struct Template Reference	81
8.11 ceras::is_binary_operator< binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > > Struct Template Reference	82
8.12 ceras::is_complex< T > Struct Template Reference	82
8.13 ceras::is_complex< complex< Real_Ex, Imag_Ex > > Struct Template Reference	82
8.14 ceras::is_constant< T > Struct Template Reference	83
8.15 ceras::is_constant< constant< Tsor > > Struct Template Reference	83
8.16 ceras::is_place_holder< T > Struct Template Reference	83
8.17 ceras::is_place_holder< place_holder< Tsor > > Struct Template Reference	84
8.18 ceras::is_tensor< T > Struct Template Reference	84
8.19 ceras::is_tensor< tensor< T, A > > Struct Template Reference	84
8.20 ceras::is_unary_operator< T > Struct Template Reference	85
8.21 ceras::is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > > Struct Template Reference	85
8.22 ceras::is_value< T > Struct Template Reference	85
8.23 ceras::is_value< value< T > > Struct Template Reference	86
8.24 ceras::is_variable< T > Struct Template Reference	86
8.25 ceras::is_variable< variable< Tsor > > Struct Template Reference	86
8.26 ceras::model< Ex, Ph > Struct Template Reference	87
8.26.1 Detailed Description	87
8.26.2 Member Typedef Documentation	88
8.26.2.1 input_layer_type	88
8.26.2.2 output_layer_type	88
8.26.3 Constructor & Destructor Documentation	88
8.26.3.1 model() [1/2]	88
8.26.3.2 model() [2/2]	89
8.26.4 Member Function Documentation	89
8.26.4.1 compile()	89
8.26.4.2 input() [1/2]	89
8.26.4.3 input() [2/2]	89
8.26.4.4 load_weights()	90

8.26.4.5 operator>()	90
8.26.4.6 output() [1/2]	90
8.26.4.7 output() [2/2]	90
8.26.4.8 predict() [1/3]	91
8.26.4.9 predict() [2/3]	91
8.26.4.10 predict() [3/3]	92
8.26.4.11 save_weights()	92
8.26.4.12 summary()	92
8.26.4.13 trainable()	92
8.26.5 Member Data Documentation	92
8.26.5.1 expression_	92
8.26.5.2 input_layer_	93
8.26.5.3 output_layer_	93
8.26.5.4 place_holder_	93
8.27 ceras::place_holder< Tsor > Struct Template Reference	93
8.27.1 Member Typedef Documentation	94
8.27.1.1 tensor_type	94
8.27.2 Constructor & Destructor Documentation	94
8.27.2.1 place_holder() [1/4]	94
8.27.2.2 place_holder() [2/4]	94
8.27.2.3 place_holder() [3/4]	94
8.27.2.4 place_holder() [4/4]	94
8.27.3 Member Function Documentation	94
8.27.3.1 backward()	95
8.27.3.2 bind()	95
8.27.3.3 forward()	95
8.27.3.4 operator=() [1/2]	95
8.27.3.5 operator=() [2/2]	95
8.27.3.6 reset()	95
8.27.3.7 shape() [1/2]	96
8.27.3.8 shape() [2/2]	96
8.28 ceras::place_holder_state< Tsor > Struct Template Reference	96
8.28.1 Member Data Documentation	96
8.28.1.1 data_	96
8.28.1.2 shape_hint_	96
8.29 ceras::regularizer< Float > Struct Template Reference	97
8.29.1 Member Typedef Documentation	97
8.29.1.1 value_type	97
8.29.2 Constructor & Destructor Documentation	97
8.29.2.1 regularizer()	97
8.29.3 Member Data Documentation	97
8.29.3.1 l1_	98

8.29.3.2 l2_	98
8.29.3.3 synchronized_	98
8.30 ceras::rmsprop< Loss, T > Struct Template Reference	98
8.30.1 Member Typedef Documentation	99
8.30.1.1 tensor_type	99
8.30.2 Constructor & Destructor Documentation	99
8.30.2.1 rmsprop()	99
8.30.3 Member Function Documentation	99
8.30.3.1 forward()	99
8.30.4 Member Data Documentation	99
8.30.4.1 decay_	100
8.30.4.2 iterations_	100
8.30.4.3 learning_rate_	100
8.30.4.4 loss_	100
8.30.4.5 rho_	100
8.31 ceras::ceras_private::session< Tsor > Struct Template Reference	100
8.31.1 Member Typedef Documentation	101
8.31.1.1 place_holder_type	101
8.31.1.2 variable_state_type	101
8.31.1.3 variable_type	101
8.31.2 Constructor & Destructor Documentation	102
8.31.2.1 session() [1/3]	102
8.31.2.2 session() [2/3]	102
8.31.2.3 session() [3/3]	102
8.31.2.4 ~session()	102
8.31.3 Member Function Documentation	102
8.31.3.1 bind()	102
8.31.3.2 deserialize()	103
8.31.3.3 operator=() [1/2]	103
8.31.3.4 operator=() [2/2]	103
8.31.3.5 rebind()	103
8.31.3.6 remember()	103
8.31.3.7 restore()	103
8.31.3.8 restore_original()	104
8.31.3.9 run()	104
8.31.3.10 save()	104
8.31.3.11 save_original()	104
8.31.3.12 serialize()	104
8.31.3.13 tap()	104
8.31.4 Member Data Documentation	105
8.31.4.1 place_holders_	105
8.31.4.2 variables_	105

8.32 ceras::sgd< Loss, T > Struct Template Reference	105
8.32.1 Member Typedef Documentation	106
8.32.1.1 tensor_type	106
8.32.2 Constructor & Destructor Documentation	106
8.32.2.1 sgd()	106
8.32.3 Member Function Documentation	106
8.32.3.1 forward()	106
8.32.4 Member Data Documentation	106
8.32.4.1 decay_	107
8.32.4.2 iterations_	107
8.32.4.3 learning_rate_	107
8.32.4.4 loss_	107
8.32.4.5 momentum_	107
8.32.4.6 nesterov_	107
8.33 ceras::tensor< T, Allocator > Struct Template Reference	108
8.33.1 Member Typedef Documentation	110
8.33.1.1 allocator	110
8.33.1.2 self_type	110
8.33.1.3 shared_vector	110
8.33.1.4 value_type	110
8.33.1.5 vector_type	111
8.33.2 Constructor & Destructor Documentation	111
8.33.2.1 tensor() [1/7]	111
8.33.2.2 tensor() [2/7]	111
8.33.2.3 tensor() [3/7]	111
8.33.2.4 tensor() [4/7]	111
8.33.2.5 tensor() [5/7]	112
8.33.2.6 tensor() [6/7]	112
8.33.2.7 tensor() [7/7]	112
8.33.3 Member Function Documentation	112
8.33.3.1 as_scalar()	112
8.33.3.2 as_type()	112
8.33.3.3 back() [1/2]	113
8.33.3.4 back() [2/2]	113
8.33.3.5 begin() [1/2]	113
8.33.3.6 begin() [2/2]	113
8.33.3.7 cbegin()	113
8.33.3.8 cend()	113
8.33.3.9 copy()	114
8.33.3.10 crbegin()	114
8.33.3.11 creep_to()	114
8.33.3.12 crend()	114

8.33.3.13 data() [1/2]	114
8.33.3.14 data() [2/2]	114
8.33.3.15 deep_copy() [1/2]	115
8.33.3.16 deep_copy() [2/2]	115
8.33.3.17 empty()	115
8.33.3.18 end() [1/2]	115
8.33.3.19 end() [2/2]	115
8.33.3.20 front() [1/2]	115
8.33.3.21 front() [2/2]	116
8.33.3.22 map()	116
8.33.3.23 ndim()	116
8.33.3.24 operator*=() [1/2]	116
8.33.3.25 operator*=() [2/2]	116
8.33.3.26 operator+=() [1/2]	116
8.33.3.27 operator+=() [2/2]	117
8.33.3.28 operator-()	117
8.33.3.29 operator-=() [1/2]	117
8.33.3.30 operator-=() [2/2]	117
8.33.3.31 operator/=() [1/2]	117
8.33.3.32 operator/=() [2/2]	117
8.33.3.33 operator=() [1/2]	118
8.33.3.34 operator=() [2/2]	118
8.33.3.35 operator[]() [1/2]	118
8.33.3.36 operator[]() [2/2]	118
8.33.3.37 rbegin() [1/2]	118
8.33.3.38 rbegin() [2/2]	118
8.33.3.39 rend() [1/2]	119
8.33.3.40 rend() [2/2]	119
8.33.3.41 reset()	119
8.33.3.42 reshape()	119
8.33.3.43 resize()	119
8.33.3.44 shape()	120
8.33.3.45 shrink_to()	120
8.33.3.46 size()	120
8.33.3.47 slice()	120
8.33.4 Member Data Documentation	120
8.33.4.1 memory_offset_	120
8.33.4.2 shape_	121
8.33.4.3 vector_	121
8.34 ceras::tensor_deduction< L, R > Struct Template Reference	121
8.34.1 Member Typedef Documentation	121
8.34.1.1 op_type	121

8.34.1.2 <code>tensor_type</code>	121
8.35 <code>ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator ></code> Struct Template Reference	122
8.35.1 Detailed Description	122
8.35.2 Member Typedef Documentation	123
8.35.2.1 <code>tensor_type</code>	123
8.35.3 Constructor & Destructor Documentation	123
8.35.3.1 <code>unary_operator()</code>	123
8.35.4 Member Function Documentation	123
8.35.4.1 <code>backward()</code>	123
8.35.4.2 <code>forward()</code>	123
8.35.4.3 <code>shape()</code>	124
8.35.5 Member Data Documentation	124
8.35.5.1 <code>backward_action_</code>	124
8.35.5.2 <code>forward_action_</code>	124
8.35.5.3 <code>input_data_</code>	124
8.35.5.4 <code>op_</code>	124
8.35.5.5 <code>output_data_</code>	125
8.35.5.6 <code>output_shape_calculator_</code>	125
8.36 <code>ceras::value< T ></code> Struct Template Reference	125
8.36.1 Member Typedef Documentation	126
8.36.1.1 <code>tensor_type</code>	126
8.36.1.2 <code>value_type</code>	126
8.36.2 Constructor & Destructor Documentation	126
8.36.2.1 <code>value()</code> [1/4]	126
8.36.2.2 <code>value()</code> [2/4]	126
8.36.2.3 <code>value()</code> [3/4]	126
8.36.2.4 <code>value()</code> [4/4]	127
8.36.3 Member Function Documentation	127
8.36.3.1 <code>backward()</code>	127
8.36.3.2 <code>forward()</code>	127
8.36.3.3 <code>operator=()</code> [1/2]	127
8.36.3.4 <code>operator=()</code> [2/2]	127
8.36.3.5 <code>shape()</code>	127
8.36.4 Member Data Documentation	128
8.36.4.1 <code>data_</code>	128
8.37 <code>ceras::variable< Tsor ></code> Struct Template Reference	128
8.37.1 Member Typedef Documentation	129
8.37.1.1 <code>tensor_type</code>	129
8.37.1.2 <code>value_type</code>	129
8.37.2 Constructor & Destructor Documentation	129
8.37.2.1 <code>variable()</code> [1/4]	129

8.37.2.2 variable() [2/4]	129
8.37.2.3 variable() [3/4]	130
8.37.2.4 variable() [4/4]	130
8.37.3 Member Function Documentation	130
8.37.3.1 backward()	130
8.37.3.2 contexts() [1/2]	130
8.37.3.3 contexts() [2/2]	130
8.37.3.4 data() [1/2]	130
8.37.3.5 data() [2/2]	131
8.37.3.6 forward()	131
8.37.3.7 gradient() [1/2]	131
8.37.3.8 gradient() [2/2]	131
8.37.3.9 operator=() [1/2]	131
8.37.3.10 operator=() [2/2]	131
8.37.3.11 reset()	131
8.37.3.12 shape()	132
8.37.3.13 trainable() [1/2]	132
8.37.3.14 trainable() [2/2]	132
8.37.4 Member Data Documentation	132
8.37.4.1 regularizer_	132
8.37.4.2 state_	132
8.37.4.3 trainable_	132
8.38 ceras::variable_state< Tsor > Struct Template Reference	133
8.38.1 Member Data Documentation	133
8.38.1.1 contexts_	133
8.38.1.2 data_	133
8.38.1.3 gradient_	133
9 File Documentation	135
9.1 /home/feng/workspace/github.repo/ceras/include/activation.hpp File Reference	135
9.2 activation.hpp	137
9.3 /home/feng/workspace/github.repo/ceras/include/ceras.hpp File Reference	143
9.4 ceras.hpp	143
9.5 /home/feng/workspace/github.repo/ceras/include/complex_operator.hpp File Reference	144
9.6 complex_operator.hpp	146
9.7 /home/feng/workspace/github.repo/ceras/include/config.hpp File Reference	148
9.8 config.hpp	148
9.9 /home/feng/workspace/github.repo/ceras/include/constant.hpp File Reference	148
9.10 constant.hpp	149
9.11 /home/feng/workspace/github.repo/ceras/include/dataset.hpp File Reference	150
9.12 dataset.hpp	150
9.13 /home/feng/workspace/github.repo/ceras/include/includes.hpp File Reference	152

9.13.1 Macro Definition Documentation	153
9.13.1.1 STB_IMAGE_IMPLEMENTATION	153
9.13.1.2 STB_IMAGE_RESIZE_IMPLEMENTATION	153
9.13.1.3 STB_IMAGE_WRITE_IMPLEMENTATION	153
9.14 includes.hpp	154
9.15 /home/feng/workspace/github.repo/ceras/include/layer.hpp File Reference	154
9.16 layer.hpp	156
9.17 /home/feng/workspace/github.repo/ceras/include/loss.hpp File Reference	159
9.18 loss.hpp	160
9.19 /home/feng/workspace/github.repo/ceras/include/metric.hpp File Reference	163
9.20 metric.hpp	164
9.21 /home/feng/workspace/github.repo/ceras/include/model.hpp File Reference	164
9.22 model.hpp	165
9.23 /home/feng/workspace/github.repo/ceras/include/operation.hpp File Reference	168
9.23.1 Function Documentation	175
9.23.1.1 abs()	175
9.23.1.2 acos()	175
9.23.1.3 acosh()	175
9.23.1.4 argmax()	176
9.23.1.5 argmin()	176
9.23.1.6 asin()	176
9.23.1.7 asinh()	176
9.23.1.8 assign()	176
9.23.1.9 atan()	177
9.23.1.10 atan2()	177
9.23.1.11 atanh()	177
9.23.1.12 average_pooling_2d()	177
9.23.1.13 batch_normalization()	178
9.23.1.14 cbrt()	178
9.23.1.15 ceil()	178
9.23.1.16 clip()	178
9.23.1.17 concat() [1/2]	178
9.23.1.18 concat() [2/2]	179
9.23.1.19 concatenate() [1/2]	179
9.23.1.20 concatenate() [2/2]	179
9.23.1.21 conv2d()	179
9.23.1.22 cos()	179
9.23.1.23 cosh()	180
9.23.1.24 cropping_2d()	180
9.23.1.25 drop_out()	180
9.23.1.26 dropout()	180
9.23.1.27 equal()	181

9.23.1.28 erf()	181
9.23.1.29 erfc()	181
9.23.1.30 exp()	182
9.23.1.31 exp2()	182
9.23.1.32 expand_dims()	182
9.23.1.33 expm1()	182
9.23.1.34 fabs()	183
9.23.1.35 flatten()	183
9.23.1.36 floor()	183
9.23.1.37 general_conv2d()	183
9.23.1.38 hypot()	184
9.23.1.39 identity()	184
9.23.1.40 img2col()	184
9.23.1.41 llrint()	184
9.23.1.42 llround()	185
9.23.1.43 log()	185
9.23.1.44 log10()	185
9.23.1.45 log1p()	185
9.23.1.46 log2()	186
9.23.1.47 lrint()	186
9.23.1.48 lround()	186
9.23.1.49 max_pooling_2d()	186
9.23.1.50 maximum()	186
9.23.1.51 minimum()	187
9.23.1.52 nearbyint()	187
9.23.1.53 normalization_batch()	187
9.23.1.54 ones_like()	187
9.23.1.55 poisson()	188
9.23.1.56 random_normal_like()	188
9.23.1.57 reduce_max()	188
9.23.1.58 reduce_min()	189
9.23.1.59 reduce_sum()	189
9.23.1.60 repeat()	190
9.23.1.61 reshape()	190
9.23.1.62 rint()	190
9.23.1.63 round()	190
9.23.1.64 sign()	191
9.23.1.65 sin()	191
9.23.1.66 sinh()	191
9.23.1.67 sliding_2d()	192
9.23.1.68 sqrt()	192
9.23.1.69 tan()	192

9.23.1.70 tanh()	192
9.23.1.71 transpose()	193
9.23.1.72 trunc()	193
9.23.1.73 up_sampling_2d()	193
9.23.1.74 upsampling_2d()	193
9.23.1.75 zero_padding_2d()	193
9.23.1.76 zeros_like()	194
9.23.2 Variable Documentation	194
9.23.2.1 sqr	194
9.23.2.2 y	194
9.24 operation.hpp	194
9.25 /home/feng/workspace/github.repo/ceras/include/optimizer.hpp File Reference	242
9.26 optimizer.hpp	243
9.27 /home/feng/workspace/github.repo/ceras/include/place_holder.hpp File Reference	249
9.28 place_holder.hpp	250
9.29 /home/feng/workspace/github.repo/ceras/include/recurrent.hpp File Reference	251
9.29.1 Variable Documentation	252
9.29.1.1 units_	252
9.30 recurrent.hpp	252
9.31 /home/feng/workspace/github.repo/ceras/include/session.hpp File Reference	253
9.32 session.hpp	254
9.33 /home/feng/workspace/github.repo/ceras/include/tensor.hpp File Reference	256
9.34 tensor.hpp	257
9.35 /home/feng/workspace/github.repo/ceras/include/value.hpp File Reference	262
9.36 value.hpp	263
9.37 /home/feng/workspace/github.repo/ceras/include/variable.hpp File Reference	264
9.38 variable.hpp	265
9.39 /home/feng/workspace/github.repo/ceras/include/xmodel.hpp File Reference	267
9.40 xmodel.hpp	268
Index	271

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ceras	11
ceras::ceras_private	54
ceras::dataset	54
ceras::dataset::fashion_mnist	54
ceras::dataset::mnist	55

Chapter 2

Concept Index

2.1 Concepts

Here is a list of all concepts with brief descriptions:

ceras::Binary_Operator	
A type that represents a binary operator	57
ceras::Complex	
A type that represents a complex expression	57
ceras::Constant	58
ceras::Expression	
A type that represents a unary operator, a binary operator, a variable, a place_holder, a constant or a value	58
ceras::Operator	
A type that represents an unary or a binary operator	58
ceras::Place_Holder	58
ceras::Tensor	59
ceras::Unary_Operator	
A type that represents an unary operator	59
ceras::Value	59
ceras::Variable	59

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ceras::compiled_model< Model, Optimizer, Loss >	71
ceras::complex< Real_Ex, Imag_Ex >	76
enable_id	
ceras::adadelta< Loss, T >	61
ceras::adagrad< Loss, T >	63
ceras::adam< Loss, T >	65
ceras::binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output_↔ Shape_Calculator >	67
ceras::constant< Tsor >	76
ceras::gradient_descent< Loss, T >	78
ceras::place_holder< Tsor >	93
ceras::rmsprop< Loss, T >	98
ceras::sgd< Loss, T >	105
ceras::tensor< T, Allocator >	108
ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > . .	122
ceras::value< T >	125
ceras::variable< Tsor >	128
enable_shared	
ceras::adadelta< Loss, T >	61
ceras::adagrad< Loss, T >	63
ceras::adam< Loss, T >	65
ceras::gradient_descent< Loss, T >	78
ceras::rmsprop< Loss, T >	98
ceras::sgd< Loss, T >	105
enable_shared_state	
ceras::place_holder< Tsor >	93
std::false_type	
ceras::is_binary_operator< T >	81
ceras::is_complex< T >	82
ceras::is_constant< T >	83
ceras::is_place_holder< T >	83
ceras::is_tensor< T >	84
ceras::is_unary_operator< T >	85
ceras::is_value< T >	85
ceras::is_variable< T >	86

ceras::identity_output_shape_calculator	80
ceras::model< Ex, Ph >	87
ceras::place_holder_state< Tsor >	96
ceras::regularizer< Float >	97
ceras::ceras_private::session< Tsor >	100
ceras::tensor_deduction< L, R >	121
ceras::tensor_deduction< Lhs_Operator, Rhc_Operator >	121
std::true_type	
ceras::is_binary_operator< binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >	82
ceras::is_complex< complex< Real_Ex, Imag_Ex > >	82
ceras::is_constant< constant< Tsor > >	83
ceras::is_place_holder< place_holder< Tsor > >	84
ceras::is_tensor< tensor< T, A > >	84
ceras::is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >	85
ceras::is_value< value< T > >	86
ceras::is_variable< variable< Tsor > >	86
ceras::variable_state< Tsor >	133

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ceras::adadelta< Loss, T >	61
ceras::adagrad< Loss, T >	63
ceras::adam< Loss, T >	65
ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >	
A binary operator is composed of a.) a left-side input expression, b.) a right-side input expression, c.) a forward action and d.) a backward action	
ceras::compiled_model< Model, Optimizer, Loss >	71
ceras::complex< Real_Ex, Imag_Ex >	76
ceras::constant< Tsor >	
Creates a constant expression from a tensor-like object	
ceras::gradient_descent< Loss, T >	78
ceras::identity_output_shape_calculator	
The default identity output shape calculator for unary/binary operators. Should be overridden for some special operators	
ceras::is_binary_operator< T >	81
ceras::is_binary_operator< binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >	82
ceras::is_complex< T >	82
ceras::is_complex< complex< Real_Ex, Imag_Ex >	82
ceras::is_constant< T >	83
ceras::is_constant< constant< Tsor >	83
ceras::is_place_holder< T >	83
ceras::is_place_holder< place_holder< Tsor >	84
ceras::is_tensor< T >	84
ceras::is_tensor< tensor< T, A >	84
ceras::is_unary_operator< T >	85
ceras::is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >	85
ceras::is_value< T >	85
ceras::is_value< value< T >	86
ceras::is_variable< T >	86
ceras::is_variable< variable< Tsor >	86
ceras::model< Ex, Ph >	87
ceras::place_holder< Tsor >	93
ceras::place_holder_state< Tsor >	96

ceras::regularizer< Float >	97
ceras::rmsprop< Loss, T >	98
ceras::ceras_private::session< Tsor >	100
ceras::sgd< Loss, T >	105
ceras::tensor< T, Allocator >	108
ceras::tensor_deduction< L, R >	121
ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >	
A unary operator is composed of a.) an input expression, b.) a forward action and c.) a backward action	
action	122
ceras::value< T >	125
ceras::variable< Tsor >	128
ceras::variable_state< Tsor >	133

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

/home/feng/workspace/github.repo/ceras/include/activation.hpp	135
/home/feng/workspace/github.repo/ceras/include/ceras.hpp	143
/home/feng/workspace/github.repo/ceras/include/complex_operator.hpp	144
/home/feng/workspace/github.repo/ceras/include/config.hpp	148
/home/feng/workspace/github.repo/ceras/include/constant.hpp	148
/home/feng/workspace/github.repo/ceras/include/dataset.hpp	150
/home/feng/workspace/github.repo/ceras/include/includes.hpp	152
/home/feng/workspace/github.repo/ceras/include/layer.hpp	154
/home/feng/workspace/github.repo/ceras/include/loss.hpp	159
/home/feng/workspace/github.repo/ceras/include/metric.hpp	163
/home/feng/workspace/github.repo/ceras/include/model.hpp	164
/home/feng/workspace/github.repo/ceras/include/operation.hpp	168
/home/feng/workspace/github.repo/ceras/include/optimizer.hpp	242
/home/feng/workspace/github.repo/ceras/include/place_holder.hpp	249
/home/feng/workspace/github.repo/ceras/include/recurrent.hpp	251
/home/feng/workspace/github.repo/ceras/include/session.hpp	253
/home/feng/workspace/github.repo/ceras/include/tensor.hpp	256
/home/feng/workspace/github.repo/ceras/include/value.hpp	262
/home/feng/workspace/github.repo/ceras/include/variable.hpp	264
/home/feng/workspace/github.repo/ceras/include/xmodel.hpp	267

Chapter 6

Namespace Documentation

6.1 ceras Namespace Reference

Namespaces

- namespace [ceras_private](#)
- namespace [dataset](#)

Classes

- struct [adadelata](#)
- struct [adagrad](#)
- struct [adam](#)
- struct [binary_operator](#)

A binary operator is composed of a.) a left-side input expression, b.) a right-side input expression, c.) a forward action and d.) a backward action.

- struct [compiled_model](#)
- struct [complex](#)
- struct [constant](#)

Creates a constant expression from a tensor-like object.

- struct [gradient_descent](#)
- struct [identity_output_shape_calculator](#)

The default identity output shape calculator for unary/binary operators. Should be overridden for some special operators.

- struct [is_binary_operator](#)
- struct [is_binary_operator< binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >](#)
- struct [is_complex](#)
- struct [is_complex< complex< Real_Ex, Imag_Ex > >](#)
- struct [is_constant](#)
- struct [is_constant< constant< Tsor > >](#)
- struct [is_place_holder](#)
- struct [is_place_holder< place_holder< Tsor > >](#)
- struct [is_tensor](#)
- struct [is_tensor< tensor< T, A > >](#)
- struct [is_unary_operator](#)
- struct [is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >](#)
- struct [is_value](#)

- struct [is_value< value< T > >](#)
- struct [is_variable](#)
- struct [is_variable< variable< Tsor > >](#)
- struct [model](#)
- struct [place_holder](#)
- struct [place_holder_state](#)
- struct [regularizer](#)
- struct [rmsprop](#)
- struct [sgd](#)
- struct [tensor](#)
- struct [tensor_deduction](#)
- struct [unary_operator](#)

A unary operator is composed of a.) an input expression, b.) a forward action and c.) a backward action.

- struct [value](#)
- struct [variable](#)
- struct [variable_state](#)

Concepts

- concept [Complex](#)
A type that represents a complex expression.
- concept [Constant](#)
- concept [Unary_Operator](#)
A type that represents an unary operator.
- concept [Binary_Operator](#)
A type that represents a binary operator.
- concept [Operator](#)
A type that represents an unary or a binary operator.
- concept [Expression](#)
A type that represents a unary operator, a binary operator, a variable, a place_holder, a constant or a value.
- concept [Place_Holder](#)
- concept [Tensor](#)
- concept [Value](#)
- concept [Variable](#)

Typedefs

- template<typename Loss , typename T >
using [ada_grad](#) = [adagrad](#)< Loss, T >
- template<typename Loss , typename T >
using [rms_prop](#) = [rmsprop](#)< Loss, T >
- template<typename Loss , typename T >
using [ada_delta](#) = [adadelat](#)< Loss, T >
- template<typename T >
using [default_allocator](#) = [buffered_allocator](#)< T, 256 >

Functions

- `template<std::floating_point Float>`
`constexpr auto heaviside_step (Float f) noexcept`
Step activation function, an unary operator.
- `template<Expression Ex>`
`constexpr auto soft_sign (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto unit_step (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto binary_step (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto gaussian (Ex const &ex) noexcept`
Gaussian activation function, an unary operator.
- `template<Expression Ex>`
`constexpr auto softmax (Ex const &ex) noexcept`
Softmax activation function, an unary operator.
- `template<Expression Ex>`
`auto selu (Ex const &ex) noexcept`
*Scaled Exponential Linear Unit (SELU) activation function, an unary operator. If $x > 0$, returns $1.0507 x$; Otherwise, returns $1.67326 * 1.0507 * (\exp(x) - 1)$*
- `template<Expression Ex>`
`auto softplus (Ex const &ex) noexcept`
Softplus function, an unary operator. Returns $\log(\exp(x) + 1)$.
- `template<Expression Ex>`
`auto softsign (Ex const &ex) noexcept`
Softsign function, an unary operator. Returns $x / (\text{abs}(x) + 1)$.
- `template<Expression Ex>`
`auto sigmoid (Ex const &ex) noexcept`
Sigmoid function, an unary operator. Returns $1 / (\exp(-x) + 1)$.
- `template<Expression Ex>`
`auto relu (Ex const &ex) noexcept`
Relu function, an unary operator. Returns x if positive, 0 otherwise.
- `template<Expression Ex>`
`auto relu6 (Ex const &ex) noexcept`
Rectified Linear 6 function, an unary operator. Returns $\min(\max(\text{features}, 0), 6)$.
- `template<typename T>`
requires `std::floating_point<T>`
`auto leaky_relu (T const factor=0.2) noexcept`
Leaky Rectified Linear function, an unary operator. Returns x if positive, αx otherwise. α defaults to 0.2 .
- `template<typename T>`
requires `std::floating_point<T>`
`auto prelu (T const factor) noexcept`
- `template<Expression Ex>`
`auto negative_relu (Ex const &ex) noexcept`
- `template<typename T = float>`
requires `std::floating_point<T>`
`auto elu (T const alpha=1.0) noexcept`
*Exponential Linear function, an unary operator. Returns x if positive, $\alpha * (\exp(x) - 1)$ otherwise. α defaults to 0.2 .*
- `template<Expression Ex>`
`auto exponential (Ex const &ex) noexcept`
Exponential function, an unary operator. Returns $\exp(x)$.

- `template<Expression Ex>`
`auto hard_sigmoid (Ex const &ex) noexcept`
Hard Sigmoid function, an unary operator. Piecewise linear approximation of the sigmoid function.
- `template<Expression Ex>`
`auto gelu (Ex const &ex) noexcept`
Gaussian Error function, an unary operator. GAUSSIAN ERROR LINEAR UNITS (GELUS) <https://arxiv.org/pdf/1606.08415.pdf> $f(x) = 0.5x (1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$ $df = x (1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)]) + \sqrt{2/\pi} x \operatorname{sech}^2[\sqrt{2/\pi} x (1+0.44715x^2) (1+0.134145x^2)]$ where $\operatorname{sech}^2(x) = 1 - \tanh^2(x)$ derivative generated using service from <https://www.symbolab.com/solver/derivative-calculator>.
- `template<Expression Ex>`
`auto swish (Ex const &ex) noexcept`
Swish activation function.
- `template<Expression Ex>`
`auto silu (Ex const &ex) noexcept`
An alias name of activation swish.
- `template<Expression Ex>`
`auto crelu (Ex const &ex) noexcept`
Concatenated Rectified Linear Units, an activation function which preserves both positive and negative phase information while enforcing non-saturated non-linearity.
- `template<Expression Ex>`
`auto tank_shrink (Ex const &ex) noexcept`
Tank shrink function.
- `template<Expression Ex>`
`auto mish (Ex const &ex) noexcept`
Mish function.
- `template<Expression Ex>`
`auto lisht (Ex const &ex) noexcept`
Lisht function.
- `template<Expression Real_Ex, Expression Imag_Ex>`
`Real_Ex real (complex< Real_Ex, Imag_Ex > const &c) noexcept`
- `template<Expression Real_Ex, Expression Imag_Ex>`
`Imag_Ex imag (complex< Real_Ex, Imag_Ex > const &c) noexcept`
- `template<Complex C>`
`auto abs (C const &c) noexcept`
Returns the magnitude of the complex expression.
- `template<Complex C>`
`auto norm (C const &c) noexcept`
Returns the squared magnitude of the complex expression.
- `template<Complex C>`
`auto conj (C const &c) noexcept`
Returns the conjugate of the complex expression.
- `template<Expression Em, Expression Ep>`
`auto polar (Em const &em, Ep const &ep) noexcept`
Returns with given magnitude and phase angle.
- `template<Complex C>`
`auto arg (C const &c) noexcept`
Calculates the phase angle (in radians) of the complex expression.
- `template<Complex C>`
`auto operator+ (C const &c) noexcept`
Returns the complex expression.
- `template<Complex C>`
`auto operator- (C const &c) noexcept`
Negatives the complex expression.

- template<Complex Cl, Complex Cr>
auto [operator+](#) (Cl const &cl, Cr const &cr) noexcept
Sums up two complex expressions.
- template<Complex Cl, Complex Cr>
auto [operator-](#) (Cl const &cl, Cr const &cr) noexcept
Subtracts one complex expression from the other one.
- template<Complex Cl, Complex Cr>
auto [operator*](#) (Cl const &cl, Cr const &cr) noexcept
Multiplies two complex expressions. Optimization here: $(a+ib)(c+id) = (ac-bd) + i(ad+bc) = (ac-bd) + i((a+b)*(c+d)-ac-bd)$*
- template<Complex C, Expression E>
auto [operator+](#) (C const &c, E const &e) noexcept
Sums up a complex expression and an expression.
- template<Complex C, Expression E>
auto [operator+](#) (E const &e, C const &c) noexcept
Sums up a complex expression and an expression.
- template<Complex C, Expression E>
auto [operator-](#) (C const &c, E const &e) noexcept
Subtracts an expression from a complex expression.
- template<Complex C, Expression E>
auto [operator-](#) (E const &e, C const &c) noexcept
Subtracts a complex expression from an expression.
- template<Complex C, Expression E>
auto [operator*](#) (C const &c, E const &e) noexcept
Multiplies a complex expression with an expression.
- template<Complex C, Expression E>
auto [operator*](#) (E const &e, C const &c) noexcept
Multiplies an expression with a complex expression.
- auto [Input](#) (std::vector< unsigned long > const &input_shape={{-1UL}})
- auto [Conv2D](#) (unsigned long output_channels, std::vector< unsigned long > const &kernel_size, std::vector< unsigned long > const &input_shape, std::string const &padding="valid", std::vector< unsigned long > const &strides={1, 1}, std::vector< unsigned long > const &dilations={1, 1}, bool use_bias=true, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f) noexcept
2D convolution layer.
- auto [Conv2D](#) (unsigned long output_channels, std::vector< unsigned long > const &kernel_size, std::string const &padding="valid", std::vector< unsigned long > const &strides={1, 1}, std::vector< unsigned long > const &dilations={1, 1}, bool use_bias=true, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f) noexcept
2D convolution layer.
- auto [Dense](#) (unsigned long output_size, bool use_bias=true, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f)
Densely-connected layer.
- auto [BatchNormalization](#) (float threshold=0.95f, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f)
Applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.
- auto [Concatenate](#) (unsigned long axis=-1) noexcept
- auto [Add](#) () noexcept
- auto [Subtract](#) () noexcept
- auto [Multiply](#) () noexcept
- template<Expression Ex>
auto [ReLU](#) (Ex const &ex) noexcept
- auto [Softmax](#) () noexcept

- `template<typename T = float>`
`auto LeakyReLU (T const factor=0.2) noexcept`
- `template<typename T = float>`
`auto ELU (T const factor=0.2) noexcept`
- `auto Reshape (std::vector< unsigned long > const &new_shape, bool include_batch_flag=true) noexcept`
- `auto Flatten () noexcept`
- `auto MaxPooling2D (unsigned long stride) noexcept`
- `auto UpSampling2D (unsigned long stride) noexcept`
- `template<typename T >`
`auto Dropout (T factor) noexcept`
- `auto AveragePooling2D (unsigned long stride) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto mean_squared_logarithmic_error (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto squared_loss (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto mean_squared_error (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto mse (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto abs_loss (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto mean_absolute_error (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto mae (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto cross_entropy (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto binary_cross_entropy_loss (Lhs_Expression const &ground_truth, Rhs_Expression const &prediction) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point F = float>`
`constexpr auto cross_entropy_loss (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex, F label_smoothing_factor=0.0) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto hinge_loss (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point FP>`
`auto binary_accuracy (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex, FP threshold=0.5) noexcept`
- `template<Expression Ex>`
`void make_trainable (Ex &ex, bool t)`
- `template<Expression Ex, Place_Holder Ph, Expression Ey>`
`auto replace_placeholder_with_expression (Ex const &ex, Ph const &old_place_holder, Ey const &new_↵ expression)`
- `template<typename Model , typename Optimizer , typename Loss >`
`auto make_compiled_model (Model const &m, Loss const &l, Optimizer const &o)`
- `template<typename Forward_Action , typename Backward_Action , typename Output_Shape_Calculator = identity_output_shape_↵ calculator>`
`constexpr auto make_unary_operator (Forward_Action const &unary_forward_action, Backward_Action const &unary_backward_action, std::string const &name="Anonymous Unary Operator", Output_Shape_↵ Calculator const &output_shape_calculator=Output_Shape_Calculator{}) noexcept`

Construct an unary operator by passing the forward/backward actions and output shape calculator.

- `template<typename Forward_Action , typename Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>`
`auto make_binary_operator (Forward_Action const &binary_forward_action, Backward_Action const &binary_backward_action, std::string const &name="Anonymous Binary Operator", Output_Shape_Calculator const &output_shape_calculator=Output_Shape_Calculator{}) noexcept`
- `template<Expression Ex>`
`std::string computation_graph (Ex const &ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto plus (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto operator+ (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
- `template<Expression Ex>`
`constexpr auto operator+ (Ex const &ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`auto operator* (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
- `template<Expression Ex>`
`constexpr auto negative (Ex const &ex) noexcept`
Negative operator, elementwise.
- `template<Expression Ex>`
`constexpr auto operator- (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto inverse (Ex const &ex) noexcept`
Inverse operator, elementwise.
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto elementwise_product (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
Multiply two input operators, elementwise.
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto elementwise_multiply (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto hadamard_product (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto divide (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
Divide one tensor by the other.
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto operator/ (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`
Divide one tensor by the other.
- `template<Expression Ex>`
`constexpr auto sum_reduce (Ex const &ex) noexcept`
Sum up all elements, returns a scalar.
- `template<Expression Ex>`
`constexpr auto reduce_sum (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto mean_reduce (Ex const &ex) noexcept`
Computes the mean of elements across all dimensions of an expression.
- `template<Expression Ex>`
`constexpr auto reduce_mean (Ex const &ex) noexcept`
An alias name of mean_reduce.
- `template<Expression Ex>`
`constexpr auto mean (Ex const &ex) noexcept`
An alias name of mean_reduce.
- `template<Expression Lhs_Expression, Expression Rh_Expression>`
`constexpr auto minus (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept`

- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto operator- (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Ex>`
`constexpr auto square (Ex const &ex) noexcept`
- `template<Place_Holder Ph>`
`bool operator== (Ph const &lhs, Ph const &rhs)`
- `template<Place_Holder Ph>`
`bool operator!= (Ph const &lhs, Ph const &rhs)`
- `template<Place_Holder Ph>`
`bool operator< (Ph const &lhs, Ph const &rhs)`
- `template<Place_Holder Ph>`
`bool operator> (Ph const &lhs, Ph const &rhs)`
- `template<Place_Holder Ph>`
`bool operator<= (Ph const &lhs, Ph const &rhs)`
- `template<Place_Holder Ph>`
`bool operator>= (Ph const &lhs, Ph const &rhs)`
- `auto lstm (std::unsigned long units) noexcept`
- `template<Tensor Tsor>`
`ceras_private::session< Tsor > &get_default_session ()`
Get the default global session.
- `template<Tensor Tsor>`
`auto & bind (place_holder< Tsor > &p_holder, Tsor const &value)`
Bind a tensor to a place holder.
- `template<typename Operation >`
`auto run (Operation &op)`
Run an expression.
- `template<typename T, typename A = default_allocator<T>>`
`constexpr tensor< T, A > as_tensor (T val) noexcept`
- `template<Variable Var>`
`bool operator== (Var const &lhs, Var const &rhs) noexcept`

Variables

- `template<typename T >`
`constexpr bool is_complex_v = is_complex<T>::value`
- `template<class T >`
`constexpr bool is_constant_v = is_constant<T>::value`
- `static constexpr auto MeanSquaredError`
Computes the mean of squares of errors between labels and predictions.
- `static constexpr auto MSE`
An alias name of function [MeanSquaredError](#).
- `static constexpr auto MeanAbsoluteError`
Computes the mean of absolute errors between labels and predictions.
- `static constexpr auto MAE`
An alias name of function [MeanAbsoluteError](#).
- `static constexpr auto Hinge`
- `static constexpr auto CategoricalCrossentropy`
- `static constexpr auto CategoricalCrossEntropy`
- `static constexpr auto BinaryCrossentropy`
- `static constexpr auto BinaryCrossEntropy`
- `template<class T >`
`constexpr bool is_unary_operator_v = is_unary_operator<T>::value`
- `template<class T >`
`constexpr bool is_binary_operator_v = is_binary_operator<T>::value`

- auto [Adam](#)
- auto [SGD](#)
- auto [Adagrad](#)
- auto [RMSprop](#)
- auto [Adadelta](#)
- template<class T >
constexpr bool [is_place_holder_v](#) = [is_place_holder](#)<T>::value
- static unsigned long [random_seed](#) = std::chrono::system_clock::now().time_since_epoch().count()
Random seed for the tensor library.
- static std::mt19937 [random_generator](#) {[random_seed](#)}
- template<class T >
constexpr bool [is_tensor_v](#) = [is_tensor](#)<T>::value
- template<class T >
constexpr bool [is_value_v](#) = [is_value](#)<T>::value
- template<class T >
constexpr bool [is_variable_v](#) = [is_variable](#)<T>::value

6.1.1 Typedef Documentation

6.1.1.1 [ada_delta](#)

```
template<typename Loss , typename T >
using ceras::ada\_delta = typedef adadelta< Loss, T >
```

6.1.1.2 [ada_grad](#)

```
template<typename Loss , typename T >
using ceras::ada\_grad = typedef adagrad<Loss, T>
```

6.1.1.3 [default_allocator](#)

```
template<typename T >
using ceras::default\_allocator = typedef buffered\_allocator<T, 256>
```

6.1.1.4 [rms_prop](#)

```
template<typename Loss , typename T >
using ceras::rms\_prop = typedef rmsprop< Loss, T >
```

6.1.2 Function Documentation

6.1.2.1 abs()

```
template<Complex C>
auto ceras::abs (
    C const & c ) [noexcept]
```

Returns the magnitude of the complex expression.

Parameters

c	Complex expression.
----------	---------------------

```
auto r = variable{ ... };
auto i = variable{ ... };
auto c = complex{ r, i };
auto a = abs( c );
```

6.1.2.2 abs_loss()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::abs_loss (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.3 Add()

```
auto ceras::Add ( ) [inline], [noexcept]
```

Layer that adds two layers

Example usage:

```
auto input = Input(); // (16, )
auto x1 = Dense( 8, 16 )( input );
auto x2 = Dense( 8, 16 )( input );
auto x3 = Add()( x1, x2 ); // equivalent to 'x1 + x2'
auto m = model{ input, x3 };
```

6.1.2.4 arg()

```
template<Complex C>
auto ceras::arg (
    C const & c ) [noexcept]
```

Calculates the phase angle (in radians) of the complex expression.

Parameters

c	Complex expression. Implemented as <code>atan2(imagec), real(c)).</code>
----------	--

```

auto r = variable{ ... };
auto i = variable{ ... };
auto c = complex{ r, i };
auto a = arg( c );

```

6.1.2.5 as_tensor()

```

template<typename T , typename A = default_allocator<T>>
constexpr tensor< T, A > ceras::as_tensor (
    T val ) [constexpr], [noexcept]

```

6.1.2.6 AveragePooling2D()

```

auto ceras::AveragePooling2D (
    unsigned long stride ) [inline], [noexcept]

```

Average pooling operation for spatial data.

6.1.2.7 BatchNormalization()

```

auto ceras::BatchNormalization (
    float threshold = 0.95f,
    float kernel_regularizer_l1 = 0.0f,
    float kernel_regularizer_l2 = 0.0f,
    float bias_regularizer_l1 = 0.0f,
    float bias_regularizer_l2 = 0.0f ) [inline]

```

Applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

Parameters

<i>shape</i>	Dimensionality of the input shape.
<i>threshold</i>	Momentum for the moving average.
<i>kernel_regularizer_l1</i>	L1 regularizer for the kernel. Defaults to 0.0f.
<i>kernel_regularizer_l2</i>	L2 regularizer for the kernel. Defaults to 0.0f.
<i>bias_regularizer_l1</i>	L1 regularizer for the bias vector. Defaults to 0.0f.
<i>bias_regularizer_l2</i>	L2 regularizer for the bias vector. Defaults to 0.0f.

Example code:

```

auto a = variable{ random<float>( {12, 34, 56, 78} ) };
auto b = BatchNormalization( {34, 56, 78}, 0.8f )( a ); // note the leading dimension of 'a' is interpreted
as batch size, and only the rest 3 dimensions are required here.

```

Applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

Parameters

<i>threshold</i>	Momentum for the moving average.
<i>kernel_regularizer_↵_l1</i>	L1 regularizer for the kernel. Defaults to 0.0f.
<i>kernel_regularizer_↵_l2</i>	L2 regularizer for the kernel. Defaults to 0.0f.
<i>bias_regularizer_l1</i>	L1 regularizer for the bias vector. Defaults to 0.0f.
<i>bias_regularizer_l2</i>	L2 regularizer for the bias vector. Defaults to 0.0f.

Example code:

```
auto a = variable{ random<float>( {12, 34, 56, 78} ) };
auto b = BatchNormalization( {34, 56, 78}, 0.8f )( a ); // note the leading dimension of 'a' is interpreted
               as batch size, and only the rest 3 dimensions are required here.
```

6.1.2.8 binary_accuracy()

```
template<Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point FP>
auto ceras::binary_accuracy (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex,
    FP threshold = 0.5 ) [noexcept]
```

6.1.2.9 binary_cross_entropy_loss()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::binary_cross_entropy_loss (
    Lhs_Expression const & ground_truth,
    Rhs_Expression const & prediction ) [constexpr], [noexcept]
```

6.1.2.10 binary_step()

```
template<Expression Ex>
constexpr auto ceras::binary_step (
    Ex const & ex ) [constexpr], [noexcept]
```

6.1.2.11 bind()

```
template<Tensor Tsor>
auto & ceras::bind (
    place_holder< Tsor > & p_holder,
    Tsor const & value )
```

Bind a tensor to a place holder.

Parameters

<i>p_holder</i>	The place holder.
<i>value</i>	The tensor to bind.

Returns

A default session.

6.1.2.12 computation_graph()

```
template<Expression Ex>
std::string ceras::computation_graph (
    Ex const & ex ) [inline], [noexcept]
```

Generating the computation graph, in [graph description language](#).

Parameters

<i>ex</i>	An expression.
-----------	----------------

Returns

A string describing the computation graph, in graph description language.

6.1.2.13 Concatenate()

```
auto ceras::Concatenate (
    unsigned long axis = -1 ) [inline], [noexcept]
```

Layer that concatenates two layers.

Parameters

<i>axis</i>	The concatenation axis. Default to the last channel.
-------------	--

Example usage:

```
auto l1 = variable{ tensor<float>{ {12, 11, 3} } };
auto l2 = variable{ tensor<float>{ {12, 11, 4} } };
auto l12 = Concatenate()( l1, l2 ); // should be of shape (12, 11, 7)
```

6.1.2.14 conj()

```
template<Complex C>
```

```
auto ceras::conj (
    C const & c ) [noexcept]
```

Returns the conjugate of the complex expression.

Parameters

<i>c</i>	Complex expression.
----------	---------------------

```
auto r = variable{ ... };
auto i = variable{ ... };
auto c = complex{ r, i };
auto a = conj( c );
```

6.1.2.15 Conv2D() [1/2]

```
auto ceras::Conv2D (
    unsigned long output_channels,
    std::vector< unsigned long > const & kernel_size,
    std::string const & padding = "valid",
    std::vector< unsigned long > const & strides = {1,1},
    std::vector< unsigned long > const & dilations = {1, 1},
    bool use_bias = true,
    float kernel_regularizer_l1 = 0.0f,
    float kernel_regularizer_l2 = 0.0f,
    float bias_regularizer_l1 = 0.0f,
    float bias_regularizer_l2 = 0.0f ) [inline], [noexcept]
```

2D convolution layer.

Parameters

<i>output_channels</i>	Dimensionality of the output space.
<i>kernel_size</i>	The height and width of the convolutional window.
<i>padding</i>	valid or same. valid suggests no padding. same suggests zero padding. Defaults to valid.
<i>strides</i>	The strides along the height and width direction. Defaults to (1, 1).
<i>dilations</i>	The dialation along the height and width direction. Defaults to (1, 1).
<i>use_bias</i>	Wether or not use a bias vector. Defaults to true.
<i>kernel_regularizer_l1</i>	L1 regularizer for the kernel. Defaults to 0.0f.
<i>kernel_regularizer_l2</i>	L2 regularizer for the kernel. Defaults to 0.0f.
<i>bias_regularizer_l1</i>	L1 regularizer for the bias vector. Defaults to 0.0f.
<i>bias_regularizer_l2</i>	L2 regularizer for the bias vector. Defaults to 0.0f.

Example code:

```
auto x = Input{ {28, 28, 1} };
auto y = Conv2D( 32, {3, 3}, "same" )( x );
auto z = Flatten()( y );
auto u = Dense( 10, 28*28*32 )( z );
auto m = model{ x, u };
```

6.1.2.16 Conv2D() [2/2]

```

auto ceras::Conv2D (
    unsigned long output_channels,
    std::vector< unsigned long > const & kernel_size,
    std::vector< unsigned long > const & input_shape,
    std::string const & padding = "valid",
    std::vector< unsigned long > const & strides = {1,1},
    std::vector< unsigned long > const & dilations = {1, 1},
    bool use_bias = true,
    float kernel_regularizer_l1 = 0.0f,
    float kernel_regularizer_l2 = 0.0f,
    float bias_regularizer_l1 = 0.0f,
    float bias_regularizer_l2 = 0.0f ) [inline], [noexcept]

```

2D convolution layer.

Parameters

<i>output_channels</i>	Dimensionality of the output space.
<i>kernel_size</i>	The height and width of the convolutional window.
<i>input_shape</i>	Dimensionality of the input shape.
<i>padding</i>	valid or same. valid suggests no padding. same suggests zero padding. Defaults to valid.
<i>strides</i>	The strides along the height and width direction. Defaults to (1, 1).
<i>dilations</i>	The dialation along the height and width direction. Defaults to (1, 1).
<i>use_bias</i>	Wether or not use a bias vector. Defaults to true.
<i>kernel_regularizer_l1</i>	L1 regularizer for the kernel. Defaults to 0.0f.
<i>kernel_regularizer_l2</i>	L2 regularizer for the kernel. Defaults to 0.0f.
<i>bias_regularizer_l1</i>	L1 regularizer for the bias vector. Defaults to 0.0f.
<i>bias_regularizer_l2</i>	L2 regularizer for the bias vector. Defaults to 0.0f.

Example code:

```

auto x = Input{};
auto y = Conv2D( 32, {3, 3}, {28, 28, 1}, "same" )( x );
auto z = Flatten()( y );
auto u = Dense( 10, 28*28*32 )( z );
auto m = model{ x, u };

```

6.1.2.17 crelu()

```

template<Expression Ex>
auto ceras::crelu (
    Ex const & ex ) [noexcept]

```

Concatenated Rectified Linear Units, an activation function which preserves both positive and negative phase information while enforcing non-saturated non-linearity.

Reference: Shang, Wenling, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units." ArXiv:1603.05201 [Cs], July 19, 2016.

<http://arxiv.org/abs/1603.05201>.

```

auto v = variable{ random<float>{ 3, 3 } };
auto c = crelu( v );

```

6.1.2.18 cross_entropy()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::cross_entropy (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.19 cross_entropy_loss()

```
template<Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point F = float>
constexpr auto ceras::cross_entropy_loss (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex,
    F label_smoothing_factor = 0.0 ) [constexpr], [noexcept]
```

6.1.2.20 Dense()

```
auto ceras::Dense (
    unsigned long output_size,
    bool use_bias = true,
    float kernel_regularizer_l1 = 0.0f,
    float kernel_regularizer_l2 = 0.0f,
    float bias_regularizer_l1 = 0.0f,
    float bias_regularizer_l2 = 0.0f ) [inline]
```

Densely-connected layer.

Parameters

<i>output_size</i>	Dimensionality of output shape. The output shape is (batch_size, output_size).
<i>input_size</i>	Dimensionality of input shape. The input shape is (batch_size, input_size).
<i>use_bias</i>	Using a bias vector or not. Defaults to <code>true</code> .
<i>kernel_regularizer_l1</i>	L1 regularizer for the kernel. Defaults to <code>0.0f</code> .
<i>kernel_regularizer_l2</i>	L2 regularizer for the kernel. Defaults to <code>0.0f</code> .
<i>bias_regularizer_l1</i>	L1 regularizer for the bias vector. Defaults to <code>0.0f</code> .
<i>bias_regularizer_l2</i>	L2 regularizer for the bias vector. Defaults to <code>0.0f</code> .

Example code:

```
auto x = Input{ {28*28,} };
auto y = Dense( 10, 28*28 )( x );
auto m = model{ x, y };
```

Densely-connected layer.

Parameters

<i>output_size</i>	Dimensionality of output shape. The output shape is (batch_size, output_size).
<i>use_bias</i>	Using a bias vector or not. Defaults to <code>true</code> .
<i>kernel_regularizer_↵_l1</i>	L1 regularizer for the kernel. Defaults to <code>0.0f</code> .
<i>kernel_regularizer_↵_l2</i>	L2 regularizer for the kernel. Defaults to <code>0.0f</code> .
<i>bias_regularizer_l1</i>	L1 regularizer for the bias vector. Defaults to <code>0.0f</code> .
<i>bias_regularizer_l2</i>	L2 regularizer for the bias vector. Defaults to <code>0.0f</code> .

Example code:

```
auto x = Input{ {28*28,} };
auto y = Dense( 10, )( x );
auto m = model{ x, y };
```

6.1.2.21 divide()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::divide (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

Divide one tensor by the other.

```
auto x = varialbe{ tensor<float>{ {17, 12} } };
auto y = varialbe{ tensor<float>{ {17, 12} } };
auto z = divide( x, y ); // z = x / y
```

6.1.2.22 Dropout()

```
template<typename T >
auto ceras::Dropout (
    T factor ) [inline], [noexcept]
```

Applies Dropout to the input.

6.1.2.23 elementwise_multiply()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::elementwise_multiply (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.24 elementwise_product()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::elementwise_product (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

Multiply two input operators, elementwise.

```
auto x = variable{ tensor<float>{ {2, 3, 5} } };
auto y = variable{ tensor<float>{ {2, 3, 5} } };
auto z = elementwise_product( x, y ); // z = x*y;
```

6.1.2.25 elu()

```
template<typename T = float>
requires std::floating_point<T>
auto ceras::elu (
    T const alpha = 1.0 ) [noexcept]
```

Exponential Linear function, an unary operator. Returns x if positive, $\alpha * (\exp(x) - 1)$ otherwise. α defaults to 0.2.

Parameters

ex	An input operator.
----	--------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = elu(0.1f)( y );
```

6.1.2.26 ELU()

```
template<typename T = float>
auto ceras::ELU (
    T const factor = 0.2 ) [inline], [noexcept]
```

Exponential Linear Unit.

6.1.2.27 exponential()

```
template<Expression Ex>
auto ceras::exponential (
    Ex const & ex ) [inline], [noexcept]
```

Exponential function, an unary operator. Returns $\exp(x)$.

Parameters

ex	An input operator.
----	--------------------


```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = exponential( y );
```

6.1.2.28 Flatten()

```
auto ceras::Flatten ( ) [inline], [noexcept]
```

Flattens the input. Does not affect the batch size.

6.1.2.29 gaussian()

```
template<Expression Ex>
constexpr auto ceras::gaussian (
    Ex const & ex ) [constexpr], [noexcept]
```

Gaussian activation function, an unary operator.

Parameters

<i>ex</i>	An input operator
-----------	-------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = gaussian( y );
```

6.1.2.30 gelu()

```
template<Expression Ex>
auto ceras::gelu (
    Ex const & ex ) [inline], [noexcept]
```

Gaussian Error function, an unary operator. GAUSSIAN ERROR LINEAR UNITS (GELUS) <https://arxiv.org/pdf/1606.08415.pdf> $f(x) = 0.5x (1 + \tanh[\sqrt{2\pi}(x + 0.044715x^3)])$ $df = x (1 + \tanh[\sqrt{2\pi}(x + 0.044715x^3)]) + \sqrt{2\pi} x \operatorname{sech}^2[\sqrt{2\pi}(x + 0.044715x^3)]$ where $\operatorname{sech}^2(x) = 1 - \tanh^2(x)$ derivative generated using service from <https://www.symbolab.com/solver/derivative-calculator>.

Parameters

<i>ex</i>	An input operator.
-----------	--------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = gelu( y );
```

6.1.2.31 `get_default_session()`

```
template<Tensor Tsor>
ceras_private::session< Tsor > & ceras::get_default_session ( )
```

Get the default global session.

6.1.2.32 `hadamard_product()`

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::hadamard_product (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.33 `hard_sigmoid()`

```
template<Expression Ex>
auto ceras::hard_sigmoid (
    Ex const & ex ) [inline], [noexcept]
```

Hard Sigmoid function, an unary operator. Piecewise linear approximation of the sigmoid function.

Parameters

<i>ex</i>	An input operator.
-----------	--------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = hard_sigmoid( y );
```

6.1.2.34 `heaviside_step()`

```
template<std::floating_point Float>
constexpr auto ceras::heaviside_step (
    Float f ) [constexpr], [noexcept]
```

Step activation function, an unary operator.

Parameters

<i>ex</i>	An input operator
-----------	-------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = heaviside_step( y );
```

6.1.2.35 hinge_loss()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::hinge_loss (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.36 imag()

```
template<Expression Real_Ex, Expression Imag_Ex>
Imag_Ex ceras::imag (
    complex< Real_Ex, Imag_Ex > const & c ) [noexcept]
```

@brief Returns the imaginary part of the complex expression.

Parameters

<i>c</i>	A complex expression.
----------	-----------------------

6.1.2.37 Input()

```
auto ceras::Input (
    std::vector< unsigned long > const & input_shape = {{-1UL}} ) [inline]
```

6.1.2.38 inverse()

```
template<Expression Ex>
constexpr auto ceras::inverse (
    Ex const & ex ) [constexpr], [noexcept]
```

Inverse operator, elementwise.

```
auto x = variable{ ... };
auto ix = inverse( x );
```

6.1.2.39 leaky_relu()

```
template<typename T >
requires std::floating_point<T>
auto ceras::leaky_relu (
    T const factor = 0.2 ) [noexcept]
```

Leaky Rectified Linear function, an unary operator. Returns x if positive, αx otherwise. α defaults to 0.2.

Parameters

<i>ex</i>	An input operator.
-----------	--------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = leaky_relu(0.1f)( y );
```

6.1.2.40 LeakyReLU()

```
template<typename T = float>
auto ceras::LeakyReLU (
    T const factor = 0.2 ) [inline], [noexcept]
```

leaky relu activation function.

6.1.2.41 lisht()

```
template<Expression Ex>
auto ceras::lisht (
    Ex const & ex ) [noexcept]
```

Lisht function.

```
auto v = variable{ random<float>{ 3, 3 } };
auto c = lisht( v );
```

6.1.2.42 lstm()

```
auto ceras::lstm (
    std::unsigned long units ) [inline], [noexcept]
```

6.1.2.43 mae()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::mae (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.44 make_binary_operator()

```
template<typename Forward_Action , typename Backward_Action , typename Output_Shape_Calculator
= identity_output_shape_calculator>
auto ceras::make_binary_operator (
    Forward_Action const & binary_forward_action,
    Backward_Action const & binary_backward_action,
    std::string const & name = "Anonymous Binary Operator",
    Output_Shape_Calculator const & output_shape_calculator = Output_Shape_Calculator{}
) [noexcept]
```

6.1.2.45 make_compiled_model()

```
template<typename Model , typename Optimizer , typename Loss >
auto ceras::make_compiled_model (
    Model const & m,
    Loss const & l,
    Optimizer const & o ) [inline]
```

6.1.2.46 make_trainable()

```
template<Expression Ex>
void ceras::make_trainable (
    Ex & ex,
    bool t )
```

Setting an expression's trainable flag

6.1.2.47 make_unary_operator()

```
template<typename Forward_Action , typename Backward_Action , typename Output_Shape_Calculator
= identity_output_shape_calculator>
constexpr auto ceras::make_unary_operator (
    Forward_Action const & unary_forward_action,
    Backward_Action const & unary_backward_action,
    std::string const & name = "Anonymous Unary Operator",
    Output_Shape_Calculator const & output_shape_calculator = Output_Shape_Calculator{}
) [constexpr], [noexcept]
```

Construct an unary operator by passing the forward/backward actions and output shape calculator.

6.1.2.48 MaxPooling2D()

```
auto ceras::MaxPooling2D (
    unsigned long stride ) [inline], [noexcept]
```

Max pooling operation for 2D spatial data.

6.1.2.49 mean()

```
template<Expression Ex>
constexpr auto ceras::mean (
    Ex const & ex ) [constexpr], [noexcept]
```

An alias name of `mean_reduce`.

6.1.2.50 mean_absolute_error()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::mean_absolute_error (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.51 mean_reduce()

```
template<Expression Ex>
constexpr auto ceras::mean_reduce (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes the mean of elements across all dimensions of an expression.

Parameters

<code>ex</code>	Incoming expression.
-----------------	----------------------

Example code:

```
auto va = place_holder<tensor<float>>{};
auto vb = variable{ random<float>{ 3, 4 } };
auto diff = mean_reduce( va, vb );
```

6.1.2.52 mean_squared_error()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::mean_squared_error (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.53 mean_squared_logarithmic_error()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::mean_squared_logarithmic_error (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.54 minus()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::minus (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.55 mish()

```
template<Expression Ex>
auto ceras::mish (
    Ex const & ex ) [noexcept]
```

Mish function.

```
auto v = variable{ random<float>{ 3, 3 } };
auto c = mish( v );
```

6.1.2.56 mse()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::mse (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.57 Multiply()

```
auto ceras::Multiply ( ) [inline], [noexcept]
```

Layer that elementwise multiplies two layers

Example usage:

```
auto input = Input(); // (16, )
auto x1 = Dense( 8, 16 )( input );
auto x2 = Dense( 8, 16 )( input );
auto x3 = Multiply()( x1, x2 ); // equivalent to 'elementwise_multiply(x1, x2)'
auto m = model{ input, x3 };
```

6.1.2.58 negative()

```
template<Expression Ex>
constexpr auto ceras::negative (
    Ex const & ex ) [constexpr], [noexcept]
```

Negative operator, elementwise.

```
auto x = variable{ ... };
auto ix = negative( x );
```

6.1.2.59 negative_relu()

```
template<Expression Ex>
auto ceras::negative_relu (
    Ex const & ex ) [noexcept]
```

6.1.2.60 norm()

```
template<Complex C>
auto ceras::norm (
    C const & c ) [noexcept]
```

Returns the squared magnitude of the complex expression.

Parameters

<i>c</i>	Complex expression.
-----------------	---------------------

```
auto r = variable{ ... };
auto i = variable{ ... };
auto c = complex{ r, i };
auto a = norm( c );
```

6.1.2.61 operator"!="()

```
template<Place_Holder Ph>
bool ceras::operator!= (
    Ph const & lhs,
    Ph const & rhs )
```

6.1.2.62 operator*() [1/4]

```
template<Complex C, Expression E>
auto ceras::operator* (
    C const & c,
    E const & e ) [noexcept]
```

Multiplies a complex expression with an expression.

6.1.2.63 operator*() [2/4]

```
template<Complex Cl, Complex Cr>
auto ceras::operator* (
    Cl const & cl,
    Cr const & cr ) [noexcept]
```

Multiplies two complex expressions. Optimization here: $(a+ib)*(c+id) = (ac-bd) + i(ad+bc) = (ac-bd) + i((a+b)*(c+d)-ac-bd)$

```
auto c1 = complex{ ..., ... };
auto c2 = complex{ ..., ... };
auto c12 = c1 * c2;
```

6.1.2.64 operator*() [3/4]

```
template<Complex C, Expression E>
auto ceras::operator* (
    E const & e,
    C const & c ) [noexcept]
```

Multiplies an expression with a complex expression.

6.1.2.65 operator*() [4/4]

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
auto ceras::operator* (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [noexcept]
```

6.1.2.66 operator+() [1/6]

```
template<Complex C>
auto ceras::operator+ (
    C const & c ) [noexcept]
```

Returns the complex expression.

6.1.2.67 operator+() [2/6]

```
template<Complex C, Expression E>
auto ceras::operator+ (
    C const & c,
    E const & e ) [noexcept]
```

Sums up a complex expression and an expression.

6.1.2.68 operator+() [3/6]

```
template<Complex Cl, Complex Cr>
auto ceras::operator+ (
    Cl const & cl,
    Cr const & cr ) [noexcept]
```

Sums up two complex expressions.

6.1.2.69 operator+() [4/6]

```
template<Complex C, Expression E>
auto ceras::operator+ (
    E const & e,
    C const & c ) [noexcept]
```

Sums up a complex expression and an expression.

6.1.2.70 operator+() [5/6]

```
template<Expression Ex>
constexpr auto ceras::operator+ (
    Ex const & ex ) [constexpr], [noexcept]
```

6.1.2.71 operator+() [6/6]

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::operator+ (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.72 operator-() [1/6]

```
template<Complex C>
auto ceras::operator- (
    C const & c ) [noexcept]
```

Negatives the complex expression.

6.1.2.73 operator-() [2/6]

```
template<Complex C, Expression E>
auto ceras::operator- (
    C const & c,
    E const & e ) [noexcept]
```

Subtracts an expression from a compression expression.

6.1.2.74 operator-() [3/6]

```
template<Complex Cl, Complex Cr>
auto ceras::operator- (
    Cl const & cl,
    Cr const & cr ) [noexcept]
```

Subtracts one complex expression from the other one.

6.1.2.75 operator-() [4/6]

```
template<Complex C, Expression E>
auto ceras::operator- (
    E const & e,
    C const & c ) [noexcept]
```

Subtracts a complex expression from an expression.

6.1.2.76 operator-() [5/6]

```
template<Expression Ex>
constexpr auto ceras::operator- (
    Ex const & ex ) [constexpr], [noexcept]
```

6.1.2.77 operator-() [6/6]

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::operator- (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.78 operator/()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::operator/ (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

Divide one tensor by the other.

```
auto x = varialbe{ tensor<float>{ {17, 12} } };
auto y = varialbe{ tensor<float>{ {17, 12} } };
auto z = x/y; // same as 'divide( x, y );'
```

6.1.2.79 operator<()

```
template<Place_Holder Ph>
bool ceras::operator< (
    Ph const & lhs,
    Ph const & rhs )
```

6.1.2.80 operator<=()

```
template<Place_Holder Ph>
bool ceras::operator<= (
    Ph const & lhs,
    Ph const & rhs )
```

6.1.2.81 operator==() [1/2]

```
template<Place_Holder Ph>
bool ceras::operator==(
    Ph const & lhs,
    Ph const & rhs )
```

6.1.2.82 operator==() [2/2]

```
template<Variable Var>
bool ceras::operator==(
    Var const & lhs,
    Var const & rhs ) [noexcept]
```

6.1.2.83 operator>()

```
template<Place_Holder Ph>
bool ceras::operator> (
    Ph const & lhs,
    Ph const & rhs )
```

6.1.2.84 operator>=()

```
template<Place_Holder Ph>
bool ceras::operator>= (
    Ph const & lhs,
    Ph const & rhs )
```

6.1.2.85 plus()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::plus (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.86 polar()

```
template<Expression Em, Expression Ep>
auto ceras::polar (
    Em const & em,
    Ep const & ep ) [noexcept]
```

Returns with given magnitude and phase angle.

Parameters

<i>em</i>	Magnitude.
<i>ep</i>	Phase.

```
auto r = variable{ ... };
auto i = variable{ ... };
auto a = polar( r, i );
```

6.1.2.87 prelu()

```
template<typename T >
requires std::floating_point<T>
```

```
auto ceras::prelu (
    T const factor ) [noexcept]
```

@PReLU is an alias name of Leaky_ReLU

6.1.2.88 real()

```
template<Expression Real_Ex, Expression Imag_Ex>
Real_Ex ceras::real (
    complex< Real_Ex, Imag_Ex > const & c ) [noexcept]
```

@bref Returns the real part of the complex expression.

Parameters

<i>c</i>	A complex expression.
----------	-----------------------

6.1.2.89 reduce_mean()

```
template<Expression Ex>
constexpr auto ceras::reduce_mean (
    Ex const & ex ) [constexpr], [noexcept]
```

An alias name of mean_reduce.

6.1.2.90 reduce_sum()

```
template<Expression Ex>
constexpr auto ceras::reduce_sum (
    Ex const & ex ) [constexpr], [noexcept]
```

6.1.2.91 relu()

```
template<Expression Ex>
auto ceras::relu (
    Ex const & ex ) [noexcept]
```

Relu function, an unary operator. Returns x if positive, 0 otherwise.

Parameters

<i>ex</i>	An input operator.
-----------	--------------------

```

auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = relu( y );

```

6.1.2.92 ReLU()

```

template<Expression Ex>
auto ceras::ReLU (
    Ex const & ex ) [inline], [noexcept]

```

Rectified Linear Unit activation function.

6.1.2.93 relu6()

```

template<Expression Ex>
auto ceras::relu6 (
    Ex const & ex ) [noexcept]

```

Rectified Linear 6 function, an unary operator. Returns $\min(\max(\text{features}, 0), 6)$.

Parameters

<i>ex</i>	An input operator.
-----------	--------------------

```

auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = relu6( y );

```

6.1.2.94 replace_placeholder_with_expression()

```

template<Expression Ex, Place_Holder Ph, Expression Ey>
auto ceras::replace_placeholder_with_expression (
    Ex const & ex,
    Ph const & old_place_holder,
    Ey const & new_expression )

```

Replacing a [place_holder](#) with an expression.

Parameters

<i>ex</i>	Can be a unary operator, binary operator, variable, place_holder , a constant or a value
<i>old_place_holder</i>	An place holder in <i>ex</i>
<i>new_expression</i>	An expression that will replace <i>old_place_holder</i> in <i>ex</i> .

Returns

A expression inheriting the topology of *ex*, but with *old_place_holder* replaced by *new_expression*

6.1.2.95 Reshape()

```
auto ceras::Reshape (
    std::vector< unsigned long > const & new_shape,
    bool include_batch_flag = true ) [inline], [noexcept]
```

Reshapes inputs into the given shape.

6.1.2.96 run()

```
template<typename Operation >
auto ceras::run (
    Operation & op )
```

Run an expression.

Parameters

<i>op</i>	An expression.
-----------	----------------

Returns

The result of the expression.

6.1.2.97 selu()

```
template<Expression Ex>
auto ceras::selu (
    Ex const & ex ) [inline], [noexcept]
```

Scaled Exponential Linear Unit (SELU) activation function, an unary operator. If $x > 0$, returns $1.0507 x$; Otherwise, returns $1.67326 * 1.0507 * (\exp(x) - 1)$

Parameters

<i>ex</i>	An input operator
-----------	-------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = selu( y );
```

6.1.2.98 sigmoid()

```
template<Expression Ex>
auto ceras::sigmoid (
    Ex const & ex ) [inline], [noexcept]
```

Sigmoid function, an unary operator. Returns $1 / (\exp(-x) + 1)$.

Parameters

ex	An input operator.
----	--------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = sigmoid( y );
```

6.1.2.99 silu()

```
template<Expression Ex>
auto ceras::silu (
    Ex const & ex ) [noexcept]
```

An alias name of activation swish.

6.1.2.100 soft_sign()

```
template<Expression Ex>
constexpr auto ceras::soft_sign (
    Ex const & ex ) [constexpr], [noexcept]
```

6.1.2.101 Softmax()

```
auto ceras::Softmax ( ) [inline], [noexcept]
```

Softmax activation function.

6.1.2.102 softmax()

```
template<Expression Ex>
constexpr auto ceras::softmax (
    Ex const & ex ) [constexpr], [noexcept]
```

Softmax activation function, an unary operator.

Parameters

ex	An input operator
----	-------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = softmax( y );
```

6.1.2.103 softplus()

```
template<Expression Ex>
auto ceras::softplus (
    Ex const & ex ) [inline], [noexcept]
```

Softplus function, an unary operator. Returns $\log(\exp(x) + 1)$.

Parameters

<i>ex</i>	An input operator
-----------	-------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = softplus( y );
```

6.1.2.104 softsign()

```
template<Expression Ex>
auto ceras::softsign (
    Ex const & ex ) [inline], [noexcept]
```

Softsign function, an unary operator. Returns $x / (\text{abs}(x) + 1)$.

Parameters

<i>ex</i>	An input operator.
-----------	--------------------

```
auto x = Input();
auto y = Dense( 10, 28*28 )( x );
auto output = softsign( y );
```

6.1.2.105 square()

```
template<Expression Ex>
constexpr auto ceras::square (
    Ex const & ex ) [constexpr], [noexcept]
```

Returns the square of the input

Parameters

<i>ex</i>	The input operator.
-----------	---------------------

Returns

An instance of a [unary_operator](#) that evaluate the squared value of the input operator.

Example code:

```
auto e = variable<tensor<float>>{ /*...*/ };
auto square = square(e);
```

6.1.2.106 squared_loss()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto ceras::squared_loss (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

6.1.2.107 Subtract()

```
auto ceras::Subtract ( ) [inline], [noexcept]
```

Layer that subtracts two layers

Example usage:

```
auto input = Input(); // (16, )
auto x1 = Dense( 8, 16 )( input );
auto x2 = Dense( 8, 16 )( input );
auto x3 = Subtract()( x1, x2 ); // equivalent to 'x1 - x2'
auto m = model{ input, x3 };
```

6.1.2.108 sum_reduce()

```
template<Expression Ex>
constexpr auto ceras::sum_reduce (
    Ex const & ex ) [constexpr], [noexcept]
```

Sum up all elements, returns a scalar.

```
auto x = variable{ ... };
auto y = sum_reduce( x );
```

6.1.2.109 swish()

```
template<Expression Ex>
auto ceras::swish (
    Ex const & ex ) [noexcept]
```

Swish activation function.

Reference: Ramachandran, Prajit, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions." ArXiv:1710.05941 [Cs], October 16, 2017. <http://arxiv.org/abs/1710.05941>.

Parameters

<i>ex</i>	Input expression.
-----------	-------------------

6.1.2.110 tank_shrink()

```
template<Expression Ex>
auto ceras::tank_shrink (
    Ex const & ex ) [noexcept]
```

Tank shrink function.

```
auto v = variable{ random<float>{ 3, 3 } };
auto c = tank_shrink( v );
```

6.1.2.111 unit_step()

```
template<Expression Ex>
constexpr auto ceras::unit_step (
    Ex const & ex ) [constexpr], [noexcept]
```

6.1.2.112 UpSampling2D()

```
auto ceras::UpSampling2D (
    unsigned long stride ) [inline], [noexcept]
```

Upsampling layer for 2D inputs.

6.1.3 Variable Documentation**6.1.3.1 Adadelta**

```
auto ceras::Adadelta [inline]
```

Initial value:

```
= [] ( auto ... args )
{
    return [=]<Expression Ex>( Ex& loss )
    {
        return adadelta{loss, args...};
    };
}
```

6.1.3.2 Adagrad

```
auto ceras::Adagrad [inline]
```

Initial value:

```
= [] ( auto ... args )
{
    return [=]<Expression Ex>( Ex& loss )
    {
        return adagrad{loss, args...};
    };
}
```

6.1.3.3 Adam

```
auto ceras::Adam [inline]
```

Initial value:

```
= [] ( auto ... args )
{
    return [=]<Expression Ex>( Ex& loss )
    {
        return adam{loss, args...};
    };
}
```

6.1.3.4 BinaryCrossentropy

```
constexpr auto ceras::BinaryCrossentropy [inline], [static], [constexpr]
```

Initial value:

```
= [] ()
{
    return []<Expression Ex >( Ex const& output )
    {
        return [=]<Place_Holder Ph>( Ph const& ground_truth )
        {
            return binary_cross_entropy_loss( ground_truth, output );
        };
    };
}
```

6.1.3.5 BinaryCrossEntropy

```
constexpr auto ceras::BinaryCrossEntropy [inline], [static], [constexpr]
```

Initial value:

```
= [] ()
{
    return BinaryCrossentropy();
}
```

6.1.3.6 CategoricalCrossentropy

```
constexpr auto ceras::CategoricalCrossentropy [inline], [static], [constexpr]
```

Initial value:

```
= []<std::floating_point F=float>( F label_smoothing_factor = 0.0)
{
    return [=]<Expression Ex >( Ex const& output )
    {
        return [=]<Place_Holder Ph>( Ph const& ground_truth )
        {
            return cross_entropy_loss( ground_truth, output, label_smoothing_factor );
        };
    };
}
```

6.1.3.7 CategoricalCrossEntropy

```
constexpr auto ceras::CategoricalCrossEntropy [inline], [static], [constexpr]
```

Initial value:

```
= []<std::floating_point F=float>(F label_smoothing_factor = 0.0)
{
    return CategoricalCrossentropy(label_smoothing_factor);
}
```

6.1.3.8 Hinge

```
constexpr auto ceras::Hinge [inline], [static], [constexpr]
```

Initial value:

```
= [] ()
{
    return []<Expression Ex >( Ex const& output )
    {
        return [=]<Place_Holder Ph>( Ph const& ground_truth )
        {
            return hinge_loss( ground_truth, output );
        };
    };
}
```

6.1.3.9 is_binary_operator_v

```
template<class T >
```

```
constexpr bool ceras::is_binary_operator_v = is_binary_operator<T>::value [inline], [constexpr]
```

If T is an instance of a [binary_operator](#), the constant value equals to true. Otherwise this value is false.

6.1.3.10 is_complex_v

```
template<typename T >
```

```
constexpr bool ceras::is_complex_v = is_complex<T>::value [constexpr]
```

6.1.3.11 is_constant_v

```
template<class T >
constexpr bool ceras::is_constant_v = is_constant<T>::value [inline], [constexpr]
```

6.1.3.12 is_place_holder_v

```
template<class T >
constexpr bool ceras::is_place_holder_v = is_place_holder<T>::value [inline], [constexpr]
```

6.1.3.13 is_tensor_v

```
template<class T >
constexpr bool ceras::is_tensor_v = is_tensor<T>::value [inline], [constexpr]
```

6.1.3.14 is_unary_operator_v

```
template<class T >
constexpr bool ceras::is_unary_operator_v = is_unary_operator<T>::value [inline], [constexpr]
```

If T is an instance of a [unary_operator](#), the constant value equals to `true`. `false` otherwise.

6.1.3.15 is_value_v

```
template<class T >
constexpr bool ceras::is_value_v = is_value<T>::value [inline], [constexpr]
```

6.1.3.16 is_variable_v

```
template<class T >
constexpr bool ceras::is_variable_v = is_variable<T>::value [inline], [constexpr]
```

6.1.3.17 MAE

```
constexpr auto ceras::MAE [inline], [static], [constexpr]
```

Initial value:

```
= [] ()
{
    return MeanAbsoluteError();
}
```

An alias name of function [MeanAbsoluteError](#).

6.1.3.18 MeanAbsoluteError

```
constexpr auto ceras::MeanAbsoluteError [inline], [static], [constexpr]
```

Initial value:

```
= [] ()
{
    return []<Expression Ex >( Ex const& output )
    {
        return []<Place_Holder Ph>( Ph const& ground_truth )
        {
            return mean_absolute_error( ground_truth, output );
        };
    };
}
```

Computes the mean of absolute errors between labels and predictions.

```
auto input = place_holder<tensor<float>>{};
auto v = variable<tensor<float>>{ ones<float>({12, 34}) };
auto output = input * v;
auto m = model{ input, output };
auto cm = m.compile( MeanAbsoluteError(), Adam(128/*batch size*/, 0.01f/*learning rate*/) );
```

see also [mean_absolute_error](#)

6.1.3.19 MeanSquaredError

```
constexpr auto ceras::MeanSquaredError [inline], [static], [constexpr]
```

Initial value:

```
= [] ()
{
    return []<Expression Ex >( Ex const& output )
    {
        return []<Place_Holder Ph>( Ph const& ground_truth )
        {
            return mean_squared_error( ground_truth, output );
        };
    };
}
```

Computes the mean of squares of errors between labels and predictions.

```
auto input = place_holder<tensor<float>>{};
auto v = variable<tensor<float>>{ ones<float>({12, 34}) };
auto output = input * v;
auto m = model{ input, output };
auto cm = m.compile( MeanSquaredError(), Adam(128/*batch size*/, 0.01f/*learning rate*/) );
```

see also [mean_squared_error](#)

6.1.3.20 MSE

```
constexpr auto ceras::MSE [inline], [static], [constexpr]
```

Initial value:

```
= [] ()  
{  
    return MeanSquaredError();  
}
```

An alias name of function [MeanSquaredError](#).

6.1.3.21 random_generator

```
std::mt19937 ceras::random_generator {random_seed} [static]
```

6.1.3.22 random_seed

```
unsigned long ceras::random_seed = std::chrono::system_clock::now().time_since_epoch().count()  
[static]
```

Random seed for the tensor library.

To reproduce the result involving random variates such as `rand`, `normal`, `poisson`, it is necessary to fix the random seed by
`random_seed=42;`

6.1.3.23 RMSprop

```
auto ceras::RMSprop [inline]
```

Initial value:

```
= [] ( auto ... args )  
{  
    return [=]<Expression Ex>( Ex& loss )  
    {  
        return rmsprop{loss, args...};  
    };  
}
```

6.1.3.24 SGD

```
auto ceras::SGD [inline]
```

Initial value:

```
= [] ( auto ... args )  
{  
    return [=]<Expression Ex>( Ex& loss )  
    {  
        return sgd{loss, args...};  
    };  
}
```

6.2 ceras::ceras_private Namespace Reference

Classes

- struct [session](#)

6.3 ceras::dataset Namespace Reference

Namespaces

- namespace [fashion_mnist](#)
- namespace [mnist](#)

6.4 ceras::dataset::fashion_mnist Namespace Reference

Functions

- auto [load_data](#) (std::string const &path=std::string{"/dataset/fashion_mnist"})

6.4.1 Function Documentation

6.4.1.1 load_data()

```
auto ceras::dataset::fashion_mnist::load_data (
    std::string const & path = std::string{"/dataset/fashion_mnist"} ) [inline]
```

Loads the fashion-MNIST dataset.

Parameters

<i>path</i>	Path where to cache the dataset locally. Default to "/dataset/fashion_mnist", should be updated if running the program somewhere else.
-------------	--

Returns

A tuple of 4 tensors: x_train, y_train, x_test, y_test. x_train, x_test: uint8 arrays of grayscale image data with shapes (num_samples, 28, 28). y_train, y_test: uint8 tensor of digit labels (integers in range 0-9) with shapes (num_samples, 10). Note: for digit 0, the corresponding array is `[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]]`.

Label Description 0 T-shirt/top 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Sandal 6 Shirt 7 Sneaker 8 Bag 9 Ankle boot

Example usage:

```
auto const& [x_train, y_train, x_test, y_test] =
    ceras::dataset::mnist::load_data("/home/feng/dataset/fashion_mnist");
```

The copyright for Fashion-MNIST is held by Zalando SE. Fashion-MNIST is licensed under the MIT license.

6.5 ceras::dataset::mnist Namespace Reference

Functions

- auto [load_data](#) (std::string const &path=std::string{"/dataset/mnist"})

6.5.1 Function Documentation

6.5.1.1 load_data()

```
auto ceras::dataset::mnist::load_data (
    std::string const & path = std::string{"/dataset/mnist"} ) [inline]
```

Loads the MNIST dataset.

Parameters

<i>path</i>	Path where to cache the dataset locally. Default to "/dataset/mnist", should be updated if running the program somewhere else.
-------------	--

Returns

A tuple of 4 tensors: x_train, y_train, x_test, y_test. x_train, x_test: uint8 arrays of grayscale image data with shapes (num_samples, 28, 28). y_train, y_test: uint8 tensor of digit labels (integers in range 0-9) with shapes (num_samples, 10). Note: for digit 0, the corresponding array is `[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]`.

Example usage:

```
auto const& [x_train, y_train, x_test, y_test] =
    ceras::dataset::mnist::load_data("/home/feng/dataset/mnist");
```

Yann LeCun and Corinna Cortes hold the copyright of MNIST dataset, which is available under the terms of the Creative Commons Attribution-Share Alike 3.0 license.

Chapter 7

Concept Documentation

7.1 ceras::Binary_Operator Concept Reference

A type that represents a binary operator.

```
#include <operation.hpp>
```

7.1.1 Concept definition

```
template<typename T>  
concept ceras::Binary_Operator = is_binary_operator_v<T>
```

7.1.2 Detailed Description

A type that represents a binary operator.

<>

7.2 ceras::Complex Concept Reference

A type that represents a complex expression.

```
#include <complex_operator.hpp>
```

7.2.1 Concept definition

```
template<typename T>  
concept ceras::Complex = is_complex_v<T>
```

7.2.2 Detailed Description

A type that represents a complex expression.

7.3 ceras::Constant Concept Reference

```
#include <constant.hpp>
```

7.3.1 Concept definition

```
template<typename T>
concept ceras::Constant = is_constant_v<T>
```

7.4 ceras::Expression Concept Reference

A type that represents a unary operator, a binary operator, a variable, a place_holder, a constant or a value.

```
#include <operation.hpp>
```

7.4.1 Concept definition

```
template<typename T>
concept ceras::Expression = Operator<T> || Variable<T> || Place_Holder<T> || Constant<T> || Value<T>
```

7.4.2 Detailed Description

A type that represents a unary operator, a binary operator, a variable, a place_holder, a constant or a value.

<>

7.5 ceras::Operator Concept Reference

A type that represents an unary or a binary operator.

```
#include <operation.hpp>
```

7.5.1 Concept definition

```
template<typename T>
concept ceras::Operator = Unary_Operator<T> || Binary_Operator<T>
```

7.5.2 Detailed Description

A type that represents an unary or a binary operator.

<>

7.6 ceras::Place_Holder Concept Reference

```
#include <place_holder.hpp>
```

7.6.1 Concept definition

```
template<typename T>
concept ceras::Place_Holder = is_place_holder_v<T>
```

7.7 ceras::Tensor Concept Reference

```
#include <tensor.hpp>
```

7.7.1 Concept definition

```
template<typename T>
concept ceras::Tensor = is_tensor_v<T>
```

7.8 ceras::Unary_Operator Concept Reference

A type that represents an unary operator.

```
#include <operation.hpp>
```

7.8.1 Concept definition

```
template<typename T>
concept ceras::Unary_Operator = is_unary_operator_v<T>
```

7.8.2 Detailed Description

A type that represents an unary operator.

<>

7.9 ceras::Value Concept Reference

```
#include <value.hpp>
```

7.9.1 Concept definition

```
template<typename T>
concept ceras::Value = is_value_v<T>
```

7.10 ceras::Variable Concept Reference

```
#include <variable.hpp>
```

7.10.1 Concept definition

```
template<typename T>
concept ceras::Variable = is_variable_v<T>
```

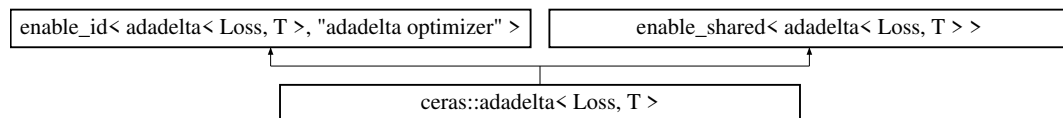

Chapter 8

Class Documentation

8.1 ceras::adadelta< Loss, T > Struct Template Reference

```
#include <optimizer.hpp>
```

Inheritance diagram for ceras::adadelta< Loss, T >:



Public Types

- typedef [tensor](#)< T > [tensor_type](#)

Public Member Functions

- [adadelta](#) (Loss &loss, std::size_t batch_size, T rho=0.9) noexcept
- void [forward](#) ()

Public Attributes

- Loss & [loss_](#)
- T [rho_](#)
- T [learning_rate_](#)
- unsigned long [iterations_](#)

8.1.1 Member Typedef Documentation

8.1.1.1 tensor_type

```
template<typename Loss , typename T >
typedef tensor< T > ceras::adadelta< Loss, T >::tensor_type
```

8.1.2 Constructor & Destructor Documentation

8.1.2.1 adadelta()

```
template<typename Loss , typename T >
ceras::adadelta< Loss, T >::adadelta (
    Loss & loss,
    std::size_t batch_size,
    T rho = 0.9 ) [inline], [noexcept]
```

8.1.3 Member Function Documentation

8.1.3.1 forward()

```
template<typename Loss , typename T >
void ceras::adadelta< Loss, T >::forward ( ) [inline]
```

8.1.4 Member Data Documentation

8.1.4.1 iterations_

```
template<typename Loss , typename T >
unsigned long ceras::adadelta< Loss, T >::iterations_
```

8.1.4.2 learning_rate_

```
template<typename Loss , typename T >
T ceras::adadelta< Loss, T >::learning_rate_
```

8.1.4.3 loss_

```
template<typename Loss , typename T >
Loss& ceras::adadelat< Loss, T >::loss_
```

8.1.4.4 rho_

```
template<typename Loss , typename T >
T ceras::adadelat< Loss, T >::rho_
```

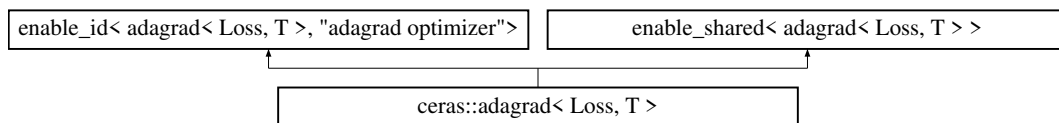
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/optimizer.hpp>

8.2 ceras::adagrad< Loss, T > Struct Template Reference

```
#include <optimizer.hpp>
```

Inheritance diagram for ceras::adagrad< Loss, T >:



Public Types

- typedef [tensor](#)< T > [tensor_type](#)

Public Member Functions

- [adagrad](#) (Loss &loss, std::size_t batch_size, T learning_rate=1.0e-1, T decay=0.0) noexcept
- void [forward](#) ()

Public Attributes

- Loss & [loss_](#)
- T [learning_rate_](#)
- T [decay_](#)
- unsigned long [iterations_](#)

8.2.1 Member Typedef Documentation

8.2.1.1 tensor_type

```
template<typename Loss , typename T >
typedef tensor< T > ceras::adagrad< Loss, T >::tensor\_type
```

8.2.2 Constructor & Destructor Documentation

8.2.2.1 adagrad()

```
template<typename Loss , typename T >
ceras::adagrad< Loss, T >::adagrad (
    Loss & loss,
    std::size_t batch_size,
    T learning_rate = 1.0e-1,
    T decay = 0.0 ) [inline], [noexcept]
```

8.2.3 Member Function Documentation

8.2.3.1 forward()

```
template<typename Loss , typename T >
void ceras::adagrad< Loss, T >::forward ( ) [inline]
```

8.2.4 Member Data Documentation

8.2.4.1 decay_

```
template<typename Loss , typename T >
T ceras::adagrad< Loss, T >::decay_
```

8.2.4.2 iterations_

```
template<typename Loss , typename T >
unsigned long ceras::adagrad< Loss, T >::iterations_
```

8.2.4.3 learning_rate_

```
template<typename Loss , typename T >
T ceras::adagrad< Loss, T >::learning_rate_
```

8.2.4.4 loss_

```
template<typename Loss , typename T >
Loss& ceras::adagrad< Loss, T >::loss_
```

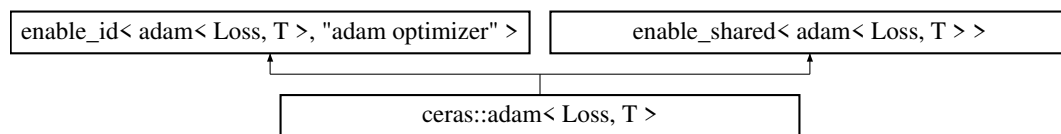
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/optimizer.hpp>

8.3 ceras::adam< Loss, T > Struct Template Reference

```
#include <optimizer.hpp>
```

Inheritance diagram for ceras::adam< Loss, T >:



Public Types

- typedef [tensor](#)< T > [tensor_type](#)

Public Member Functions

- [adam](#) (Loss &loss, std::size_t batch_size, T learning_rate=1.0e-1, T beta_1=0.9, T beta_2=0.999, bool amsgrad=false) noexcept
- void [forward](#) ()

Public Attributes

- Loss & [loss_](#)
- T [learning_rate_](#)
- T [beta_1_](#)
- T [beta_2_](#)
- bool [amsgrad_](#)
- unsigned long [iterations_](#)

8.3.1 Member Typedef Documentation

8.3.1.1 tensor_type

```
template<typename Loss , typename T >
typedef tensor< T > ceras::adam< Loss, T >::tensor\_type
```

8.3.2 Constructor & Destructor Documentation

8.3.2.1 adam()

```
template<typename Loss , typename T >
ceras::adam< Loss, T >::adam (
    Loss & loss,
    std::size_t batch_size,
    T learning_rate = 1.0e-1,
    T beta_1 = 0.9,
    T beta_2 = 0.999,
    bool amsgrad = false ) [inline], [noexcept]
```

8.3.3 Member Function Documentation

8.3.3.1 forward()

```
template<typename Loss , typename T >
void ceras::adam< Loss, T >::forward ( ) [inline]
```

8.3.4 Member Data Documentation

8.3.4.1 amsgrad_

```
template<typename Loss , typename T >
bool ceras::adam< Loss, T >::amsgrad_
```

8.3.4.2 beta_1_

```
template<typename Loss , typename T >
T ceras::adam< Loss, T >::beta_1_
```

8.3.4.3 beta_2_

```
template<typename Loss , typename T >
T ceras::adam< Loss, T >::beta_2_
```

8.3.4.4 iterations_

```
template<typename Loss , typename T >
unsigned long ceras::adam< Loss, T >::iterations_
```

8.3.4.5 learning_rate_

```
template<typename Loss , typename T >
T ceras::adam< Loss, T >::learning_rate_
```

8.3.4.6 loss_

```
template<typename Loss , typename T >
Loss& ceras::adam< Loss, T >::loss_
```

The documentation for this struct was generated from the following file:

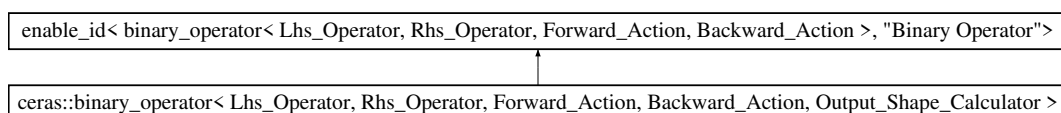
- </home/feng/workspace/github.repo/ceras/include/optimizer.hpp>

8.4 ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > Struct Template Reference

A binary operator is composed of a.) a left-side input expression, b.) a right-side input expression, c.) a forward action and d.) a backward action.

```
#include <operation.hpp>
```

Inheritance diagram for ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >:



Public Types

- typedef [tensor_deduction](#)< Lhs_Operator, Rhs_Operator >::[tensor_type](#) [tensor_type](#)

Public Member Functions

- [binary_operator](#) (Lhs_Operator const &lhs_op, Rhs_Operator const &rhs_op, Forward_Action const &forward_action, Backward_Action const &backward_action, Output_Shape_Calculator const &output_shape_calculator) noexcept
- auto [forward](#) ()
- void [backward](#) ([tensor_type](#) const &grad)
Backward action, grad back-propagated.
- std::vector< unsigned long > [shape](#) () const noexcept
Calculate the output shape.

Public Attributes

- Lhs_Operator [lhs_op_](#)
- Rhs_Operator [rhs_op_](#)
- Forward_Action [forward_action_](#)
- Backward_Action [backward_action_](#)
- Output_Shape_Calculator [output_shape_calculator_](#)
- [tensor_type](#) [lhs_input_data_](#)
- [tensor_type](#) [rhs_input_data_](#)
- [tensor_type](#) [output_data_](#)

8.4.1 Detailed Description

```
template<typename Lhs_Operator, typename Rhs_Operator, typename Forward_Action, typename Backward_Action, typename
Output_Shape_Calculator = identity_output_shape_calculator>
struct ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >
```

A binary operator is composed of a.) a left-side input expression, b.) a right-side input expression, c.) a forward action and d.) a backward action.

8.4.2 Member Typedef Documentation

8.4.2.1 [tensor_type](#)

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
typedef tensor\_deduction<Lhs_Operator,Rhs_Operator>::tensor\_type ceras::binary\_operator<
Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >::tensor\_type
```


8.4.3 Constructor & Destructor Documentation

8.4.3.1 binary_operator()

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_↵
Shape_Calculator >::binary_operator (
    Lhs_Operator const & lhs_op,
    Rhs_Operator const & rhs_op,
    Forward_Action const & forward_action,
    Backward_Action const & backward_action,
    Output_Shape_Calculator const & output_shape_calculator ) [inline], [noexcept]
```

8.4.4 Member Function Documentation

8.4.4.1 backward()

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
void ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action,
Output_Shape_Calculator >::backward (
    tensor_type const & grad ) [inline]
```

Backward action, grad back-propagated.

8.4.4.2 forward()

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
auto ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action,
Output_Shape_Calculator >::forward ( ) [inline]
```

8.4.4.3 shape()

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
std::vector< unsigned long > ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_↵
Action, Backward_Action, Output_Shape_Calculator >::shape ( ) const [inline], [noexcept]
```

Calculate the output shape.

8.4.5 Member Data Documentation

8.4.5.1 backward_action_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
Backward_Action ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_↵
_Action, Output_Shape_Calculator >::backward_action_
```

8.4.5.2 forward_action_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
Forward_Action ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_↵
_Action, Output_Shape_Calculator >::forward_action_
```

8.4.5.3 lhs_input_data_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
tensor_type ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_↵
_Action, Output_Shape_Calculator >::lhs_input_data_
```

8.4.5.4 lhs_op_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
Lhs_Operator ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_↵
_Action, Output_Shape_Calculator >::lhs_op_
```

8.4.5.5 output_data_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
tensor_type ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_↵
_Action, Output_Shape_Calculator >::output_data_
```

8.4.5.6 output_shape_calculator_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
Output_Shape_Calculator ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action,
Backward_Action, Output_Shape_Calculator >::output_shape_calculator_
```

8.4.5.7 rhs_input_data_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
tensor_type ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_↵
Action, Output_Shape_Calculator >::rhs_input_data_
```

8.4.5.8 rhs_op_

```
template<typename Lhs_Operator , typename Rhs_Operator , typename Forward_Action , typename
Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>
Rhs_Operator ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_↵
Action, Output_Shape_Calculator >::rhs_op_
```

The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/operation.hpp

8.5 ceras::compiled_model< Model, Optimizer, Loss > Struct Template Reference

```
#include <model.hpp>
```

Public Types

- typedef Model::input_layer_type [io_layer_type](#)
- typedef decltype(std::declval< Optimizer >()(std::declval< Loss & >())) [optimizer_type](#)

Public Member Functions

- [compiled_model](#) (Model const &m, [io_layer_type](#) const &input_place_holder, [io_layer_type](#) const &ground_↵_truth_place_holder, Loss const &loss, Optimizer const &optimizer)
- template<Tensor Tsr>
auto [evaluate](#) (Tsr const &inputs, Tsr const &outputs, unsigned long batch_size=32)
- template<Tensor Tsr>
auto [fit](#) (Tsr const &inputs, Tsr const &outputs, unsigned long batch_size, unsigned long epoch=1, int verbose=0, double validation_split=0.0)
- template<Tensor Tsr>
auto [train_on_batch](#) (Tsr const &input, Tsr const &output)
- template<Tensor Tsr>
auto [predict](#) (Tsr const &input_tensor)
- template<Expression Exp>
auto [operator\(\)](#) (Exp const &ex) const noexcept
- void [trainable](#) (bool t)

Public Attributes

- Model `model_`
- `io_layer_type` `input_place_holder_`
- `io_layer_type` `ground_truth_place_holder_`
- Loss `loss_`
- Optimizer `optimizer_`
- `optimizer_type` `compiled_optimizer_`

8.5.1 Member Typedef Documentation

8.5.1.1 `io_layer_type`

```
template<typename Model , typename Optimizer , typename Loss >
typedef Model::input_layer_type ceras::compiled_model< Model, Optimizer, Loss >::io_layer_type
```

8.5.1.2 `optimizer_type`

```
template<typename Model , typename Optimizer , typename Loss >
typedef decltype(std::declval<Optimizer>() (std::declval<Loss&>())) ceras::compiled_model<
Model, Optimizer, Loss >::optimizer_type
```

8.5.2 Constructor & Destructor Documentation

8.5.2.1 `compiled_model()`

```
template<typename Model , typename Optimizer , typename Loss >
ceras::compiled_model< Model, Optimizer, Loss >::compiled_model (
    Model const & m,
    io_layer_type const & input_place_holder,
    io_layer_type const & ground_truth_place_holder,
    Loss const & loss,
    Optimizer const & optimizer ) [inline]
```

8.5.3 Member Function Documentation

8.5.3.1 `evaluate()`

```
template<typename Model , typename Optimizer , typename Loss >
template<Tensor Tsor>
auto ceras::compiled_model< Model, Optimizer, Loss >::evaluate (
    Tsor const & inputs,
    Tsor const & outputs,
    unsigned long batch_size = 32 ) [inline]
```

Calculate the loss for the model in test model.

Parameters

<i>inputs</i>	Input data. A tensor of shape (samples, input_shape).
<i>outputs</i>	Output data. A tensor of shape (samples, output_shape).
<i>batch_size</i>	Number of samples per batch of computation. Default to 32.

Returns

Test loss. A scalar.

8.5.3.2 fit()

```
template<typename Model , typename Optimizer , typename Loss >
template<Tensor Tsor>
auto ceras::compiled_model< Model, Optimizer, Loss >::fit (
    Tsor const & inputs,
    Tsor const & outputs,
    unsigned long batch_size,
    unsigned long epoch = 1,
    int verbose = 0,
    double validation_split = 0.0 ) [inline]
```

Train the model on the selected dataset for a fixed numbers of epochs.

Parameters

<i>inputs</i>	Input data. A tensor of shape (samples, input_shape).
<i>outputs</i>	Input data. A tensor of shape (samples, output_shape).
<i>batch_size</i>	Number of samples per gradient update. Should agree with the batch size in the optimizer.
<i>epoch</i>	Number of epoches to train the dataset.
<i>verbose</i>	Verbosity mode. 0 for silent. 1 for one line per epoch.
<i>validation_split</i>	Fraction of the training data that will be used for validation. A floating number in range [0, 1].

Returns

A tuple of two vectors. The first vector gives the historical errors on the training data. The second vector gives the historical errors on the validation data.

Example:

```
model m{ ... };
auto cm = m.compile( ... );
tensor<float> inputs, outputs;
//...
unsigned long batch_size = 32;
unsigned long epoch = 10;
int verbose = 1;
double validation_split = 0.2;
auto errors = cm.fit( inputs, outputs, batch_size, epoch, verbose, validation_split );
```

8.5.3.3 operator()

```
template<typename Model , typename Optimizer , typename Loss >
template<Expression Exp>
auto ceras::compiled_model< Model, Optimizer, Loss >::operator() (
    Exp const & ex ) const [inline], [noexcept]
```

8.5.3.4 predict()

```
template<typename Model , typename Optimizer , typename Loss >
template<Tensor Tsor>
auto ceras::compiled_model< Model, Optimizer, Loss >::predict (
    Tsor const & input_tensor ) [inline]
```

8.5.3.5 train_on_batch()

```
template<typename Model , typename Optimizer , typename Loss >
template<Tensor Tsor>
auto ceras::compiled_model< Model, Optimizer, Loss >::train_on_batch (
    Tsor const & input,
    Tsor const & output ) [inline]
```

Running a single updated on a single batch of data.

Parameters

<i>input</i>	The input data to train the model. A tensor of shape (batch_size, input_shape).
<i>output</i>	The output data to train the model. A tensor of shape (batch_size, output_shape).

Returns

Training loss. A scalar.

Example code:

```
auto m = model{ ... };
auto cm = m.compile( ... );
for ( auto idx : range( 1024 ) )
{
    auto x = ...; // get batch input
    auto y = ...; // get batch output
    cm.train_on_batch( x, y );
}
```

8.5.3.6 trainable()

```
template<typename Model , typename Optimizer , typename Loss >
void ceras::compiled_model< Model, Optimizer, Loss >::trainable (
    bool t ) [inline]
```

8.5.4 Member Data Documentation

8.5.4.1 compiled_optimizer_

```
template<typename Model , typename Optimizer , typename Loss >  
optimizer_type ceras::compiled_model< Model, Optimizer, Loss >::compiled_optimizer_
```

8.5.4.2 ground_truth_place_holder_

```
template<typename Model , typename Optimizer , typename Loss >  
io_layer_type ceras::compiled_model< Model, Optimizer, Loss >::ground_truth_place_holder_
```

8.5.4.3 input_place_holder_

```
template<typename Model , typename Optimizer , typename Loss >  
io_layer_type ceras::compiled_model< Model, Optimizer, Loss >::input_place_holder_
```

8.5.4.4 loss_

```
template<typename Model , typename Optimizer , typename Loss >  
Loss ceras::compiled_model< Model, Optimizer, Loss >::loss_
```

8.5.4.5 model_

```
template<typename Model , typename Optimizer , typename Loss >  
Model ceras::compiled_model< Model, Optimizer, Loss >::model_
```

8.5.4.6 optimizer_

```
template<typename Model , typename Optimizer , typename Loss >  
Optimizer ceras::compiled_model< Model, Optimizer, Loss >::optimizer_
```

The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/[model.hpp](#)

8.6 `ceras::complex< Real_Ex, Imag_Ex >` Struct Template Reference

```
#include <complex_operator.hpp>
```

Public Attributes

- Real_Ex [real_](#)
- Imag_Ex [imag_](#)

8.6.1 Member Data Documentation

8.6.1.1 `imag_`

```
template<Expression Real_Ex, Expression Imag_Ex>
Imag_Ex ceras::complex< Real\_Ex, Imag\_Ex >::imag\_
```

8.6.1.2 `real_`

```
template<Expression Real_Ex, Expression Imag_Ex>
Real_Ex ceras::complex< Real\_Ex, Imag\_Ex >::real\_
```

The documentation for this struct was generated from the following file:

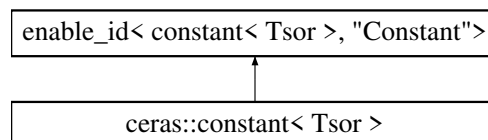
- [/home/feng/workspace/github.repo/ceras/include/complex_operator.hpp](#)

8.7 `ceras::constant< Tsor >` Struct Template Reference

Creates a constant expression from a tensor-like object.

```
#include <constant.hpp>
```

Inheritance diagram for `ceras::constant< Tsor >`:



Public Types

- typedef Tsor [tensor_type](#)

Public Member Functions

- `constant` (`tensor_type` const &data)
- void `backward` (auto) const
- `tensor_type forward` () const
- auto `shape` () const

Public Attributes

- `tensor_type data_`

8.7.1 Detailed Description

```
template<Tensor Tzor>
struct ceras::constant< Tzor >
```

Creates a constant expression from a tensor-like object.

```
auto c = constant{ zeros<float>( {3, 3, 3} ) };
```

8.7.2 Member Typedef Documentation

8.7.2.1 tensor_type

```
template<Tensor Tzor>
typedef Tzor ceras::constant< Tzor >::tensor_type
```

8.7.3 Constructor & Destructor Documentation

8.7.3.1 constant()

```
template<Tensor Tzor>
ceras::constant< Tzor >::constant (
    tensor_type const & data ) [inline]
```

8.7.4 Member Function Documentation

8.7.4.1 backward()

```
template<Tensor Tsor>
void ceras::constant< Tsor >::backward (
    auto ) const [inline]
```

8.7.4.2 forward()

```
template<Tensor Tsor>
tensor_type ceras::constant< Tsor >::forward ( ) const [inline]
```

8.7.4.3 shape()

```
template<Tensor Tsor>
auto ceras::constant< Tsor >::shape ( ) const [inline]
```

8.7.5 Member Data Documentation

8.7.5.1 data_

```
template<Tensor Tsor>
tensor_type ceras::constant< Tsor >::data_
```

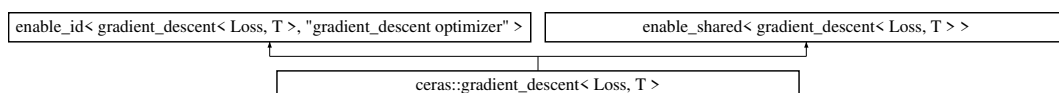
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/constant.hpp>

8.8 ceras::gradient_descent< Loss, T > Struct Template Reference

```
#include <optimizer.hpp>
```

Inheritance diagram for ceras::gradient_descent< Loss, T >:



Public Types

- typedef `tensor< T >` `tensor_type`

Public Member Functions

- [gradient_descent](#) (Loss &loss, std::size_t batch_size, T learning_rate=1.0e-3, T momentum=0.0) noexcept
- void [forward](#) ()

Public Attributes

- Loss & [loss_](#)
- T [learning_rate_](#)
- T [momentum_](#)

8.8.1 Member Typedef Documentation

8.8.1.1 tensor_type

```
template<typename Loss , typename T >
typedef tensor< T > ceras::gradient\_descent< Loss, T >::tensor\_type
```

8.8.2 Constructor & Destructor Documentation

8.8.2.1 gradient_descent()

```
template<typename Loss , typename T >
ceras::gradient\_descent< Loss, T >::gradient\_descent (
    Loss & loss,
    std::size_t batch_size,
    T learning_rate = 1.0e-3,
    T momentum = 0.0 ) [inline], [noexcept]
```

8.8.3 Member Function Documentation

8.8.3.1 forward()

```
template<typename Loss , typename T >
void ceras::gradient\_descent< Loss, T >::forward ( ) [inline]
```

8.8.4 Member Data Documentation

8.8.4.1 learning_rate_

```
template<typename Loss , typename T >
T ceras::gradient_descent< Loss, T >::learning_rate_
```

8.8.4.2 loss_

```
template<typename Loss , typename T >
Loss& ceras::gradient_descent< Loss, T >::loss_
```

8.8.4.3 momentum_

```
template<typename Loss , typename T >
T ceras::gradient_descent< Loss, T >::momentum_
```

The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/optimizer.hpp>

8.9 ceras::identity_output_shape_calculator Struct Reference

The default identity output shape calculator for unary/binary operators. Should be overridden for some special operators.

```
#include <operation.hpp>
```

Public Member Functions

- `std::vector< unsigned long > operator() (std::vector< unsigned long > const &input_shape) const noexcept`
- `std::vector< unsigned long > operator() (std::vector< unsigned long > const &lhs_input_shape, std::vector< unsigned long > const &rhs_input_shape) const noexcept`
- `std::vector< unsigned long > operator() () const noexcept`

8.9.1 Detailed Description

The default identity output shape calculator for unary/binary operators. Should be overridden for some special operators.

8.9.2 Member Function Documentation

8.9.2.1 operator>() [1/3]

```
std::vector< unsigned long > ceras::identity_output_shape_calculator::operator() ( ) const
[inline], [noexcept]
```

8.9.2.2 operator>() [2/3]

```
std::vector< unsigned long > ceras::identity_output_shape_calculator::operator() (
    std::vector< unsigned long > const & input_shape ) const [inline], [noexcept]
```

8.9.2.3 operator>() [3/3]

```
std::vector< unsigned long > ceras::identity_output_shape_calculator::operator() (
    std::vector< unsigned long > const & lhs_input_shape,
    std::vector< unsigned long > const & rhs_input_shape ) const [inline], [noexcept]
```

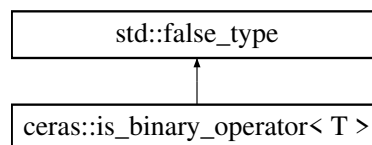
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/operation.hpp>

8.10 ceras::is_binary_operator< T > Struct Template Reference

```
#include <operation.hpp>
```

Inheritance diagram for ceras::is_binary_operator< T >:



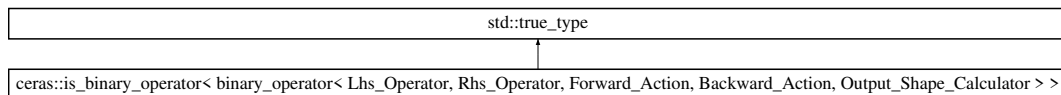
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/operation.hpp>

8.11 `ceras::is_binary_operator< binary_operator< Lhs_Operator, Rh_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >` Struct Template Reference

```
#include <operation.hpp>
```

Inheritance diagram for `ceras::is_binary_operator< binary_operator< Lhs_Operator, Rh_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >`:



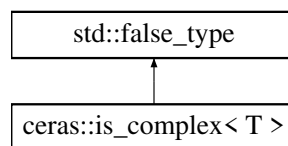
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/operation.hpp>

8.12 `ceras::is_complex< T >` Struct Template Reference

```
#include <complex_operator.hpp>
```

Inheritance diagram for `ceras::is_complex< T >`:



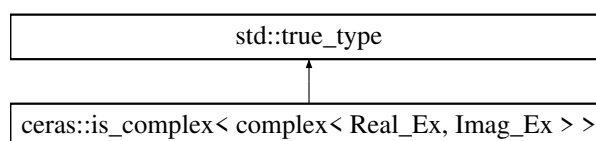
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/complex_operator.hpp

8.13 `ceras::is_complex< complex< Real_Ex, Imag_Ex > >` Struct Template Reference

```
#include <complex_operator.hpp>
```

Inheritance diagram for `ceras::is_complex< complex< Real_Ex, Imag_Ex > >`:



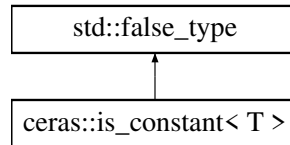
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/complex_operator.hpp

8.14 ceras::is_constant< T > Struct Template Reference

```
#include <constant.hpp>
```

Inheritance diagram for ceras::is_constant< T >:



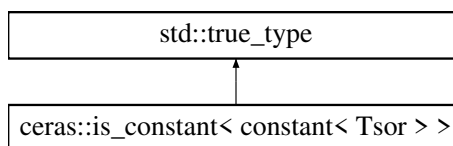
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/[constant.hpp](#)

8.15 ceras::is_constant< constant< Tsor > > Struct Template Reference

```
#include <constant.hpp>
```

Inheritance diagram for ceras::is_constant< constant< Tsor > >:



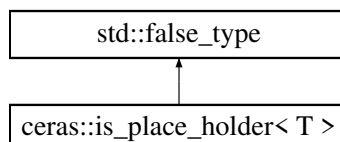
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/[constant.hpp](#)

8.16 ceras::is_place_holder< T > Struct Template Reference

```
#include <place_holder.hpp>
```

Inheritance diagram for ceras::is_place_holder< T >:



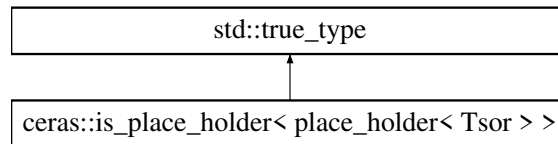
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/[place_holder.hpp](#)

8.17 `ceras::is_place_holder< place_holder< Tsor > >` Struct Template Reference

```
#include <place_holder.hpp>
```

Inheritance diagram for `ceras::is_place_holder< place_holder< Tsor > >`:



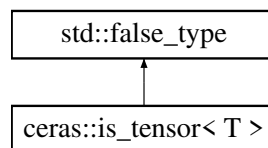
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/place_holder.hpp

8.18 `ceras::is_tensor< T >` Struct Template Reference

```
#include <tensor.hpp>
```

Inheritance diagram for `ceras::is_tensor< T >`:



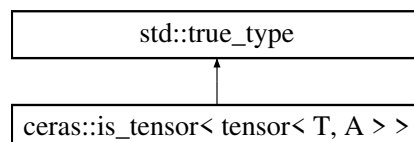
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/tensor.hpp>

8.19 `ceras::is_tensor< tensor< T, A > >` Struct Template Reference

```
#include <tensor.hpp>
```

Inheritance diagram for `ceras::is_tensor< tensor< T, A > >`:



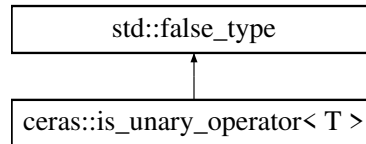
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/tensor.hpp>

8.20 ceras::is_unary_operator< T > Struct Template Reference

```
#include <operation.hpp>
```

Inheritance diagram for ceras::is_unary_operator< T >:



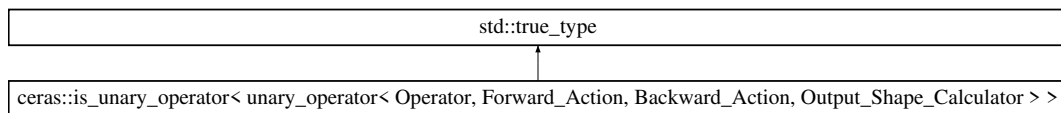
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/operation.hpp>

8.21 ceras::is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > > Struct Template Reference

```
#include <operation.hpp>
```

Inheritance diagram for ceras::is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >:



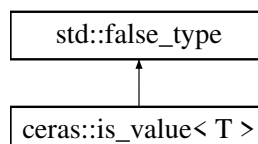
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/operation.hpp>

8.22 ceras::is_value< T > Struct Template Reference

```
#include <value.hpp>
```

Inheritance diagram for ceras::is_value< T >:



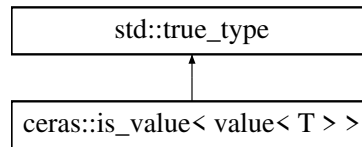
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/value.hpp>

8.23 `ceras::is_value< value< T > >` Struct Template Reference

```
#include <value.hpp>
```

Inheritance diagram for `ceras::is_value< value< T > >`:



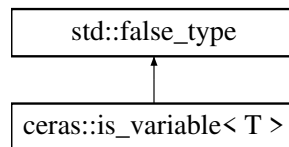
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/value.hpp>

8.24 `ceras::is_variable< T >` Struct Template Reference

```
#include <variable.hpp>
```

Inheritance diagram for `ceras::is_variable< T >`:



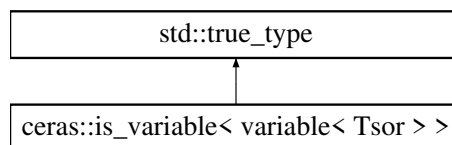
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/variable.hpp>

8.25 `ceras::is_variable< variable< Tsor > >` Struct Template Reference

```
#include <variable.hpp>
```

Inheritance diagram for `ceras::is_variable< variable< Tsor > >`:



The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/variable.hpp>

8.26 ceras::model< Ex, Ph > Struct Template Reference

```
#include <model.hpp>
```

Public Types

- typedef Ph [input_layer_type](#)
- typedef Ex [output_layer_type](#)

Public Member Functions

- [input_layer_type](#) [input](#) () const noexcept
- [output_layer_type](#) [output](#) () const noexcept
- [model](#) ([input_layer_type](#) const &[place_holder](#), [output_layer_type](#) const &expression)
- template<Tensor Tsr>
 - auto [predict](#) (Tsr const &input_tensor)
- template<Expression Exp>
 - auto [operator](#)() (Exp const &ex) const noexcept
- template<typename Loss , typename Optimizer >
 - auto [compile](#) (Loss const &l, Optimizer const &o)
- void [trainable](#) (bool t)
- void [save_weights](#) (std::string const &file)
- void [load_weights](#) (std::string const &file)
- void [summary](#) (std::string const &file_name=std::string{}) const noexcept
- constexpr [model](#) (Input_List const &input_layer, Output_List const &output_layer)
- constexpr auto [input](#) () const
 - Returns the input layer(s) of the model in a 'list', which is are [place_holders](#).*
- constexpr auto [output](#) () const
 - Returns the output layer(s) of the model in a 'list', which is are expressions.*
- template<Tensor Tsr>
 - constexpr auto [predict](#) (Tsr const &input_tensor) const
- template<List Tsr_List>
 - auto [predict](#) (Tsr_List const &input_tensor) const

Public Attributes

- [output_layer_type](#) [expression_](#)
 - output layer of the model.*
- [input_layer_type](#) [place_holder_](#)
- Input_List [input_layer_](#)
- Output_List [output_layer_](#)

8.26.1 Detailed Description

```
template<Expression Ex, Place_Holder Ph>
struct ceras::model< Ex, Ph >
```

Groups an input layer (a place holder) and an output layer (an expression template) into an object.

Template Parameters

<i>Ex</i>	The expression template for the output layer.
<i>Ph</i>	The place holder expression for the input layer

8.26.2 Member Typedef Documentation

8.26.2.1 input_layer_type

```
template<Expression Ex, Place_Holder Ph>
typedef Ph ceras::model< Ex, Ph >::input_layer_type
```

8.26.2.2 output_layer_type

```
template<Expression Ex, Place_Holder Ph>
typedef Ex ceras::model< Ex, Ph >::output_layer_type
```

8.26.3 Constructor & Destructor Documentation

8.26.3.1 model() [1/2]

```
template<Expression Ex, Place_Holder Ph>
ceras::model< Ex, Ph >::model (
    input_layer_type const & place_holder,
    output_layer_type const & expression ) [inline]
```

Parameters

<i>place_holder</i>	The input layer of the model, a place holder.
<i>expression</i>	The output layer of the model, a expression template.

Example code to generate a model:

```
auto input = Input();
auto l1 = relu( Dense( 1024, 28*28 )( input ) );
auto output = sigmoid( Dense( 10, 1024 )( l1 ) );
auto m = model{ input, output };
```

8.26.3.2 model() [2/2]

```
template<Expression Ex, Place_Holder Ph>
constexpr ceras::model< Ex, Ph >::model (
    Input_List const & input_layer,
    Output_List const & output_layer ) [inline], [constexpr]
```

8.26.4 Member Function Documentation

8.26.4.1 compile()

```
template<Expression Ex, Place_Holder Ph>
template<typename Loss , typename Optimizer >
auto ceras::model< Ex, Ph >::compile (
    Loss const & l,
    Optimizer const & o ) [inline]
```

Compile the model for training

Parameters

<i>l</i>	The loss to minimize.
<i>o</i>	The optimizer to do the optimization.

Returns

An instance of [compiled_model](#).

Example useage:

```
model m{ ... };
unsigned long batch_size = 16;
float learning_rate = 0.001f;
auto cm = m.compile( MeanSquaredError(), SGD( batch_size, learning_rate ) );
```

8.26.4.2 input() [1/2]

```
template<Expression Ex, Place_Holder Ph>
constexpr auto ceras::model< Ex, Ph >::input ( ) const [inline], [constexpr]
```

Returns the input layer(s) of the model in a 'list', which is are [place_holders](#).

8.26.4.3 input() [2/2]

```
template<Expression Ex, Place_Holder Ph>
input_layer_type ceras::model< Ex, Ph >::input ( ) const [inline], [noexcept]
```

Returns the input layer of the model, which is a [place_holder](#).

8.26.4.4 load_weights()

```
template<Expression Ex, Place_Holder Ph>
void ceras::model< Ex, Ph >::load_weights (
    std::string const & file ) [inline]
```

Loads all variables from a file

8.26.4.5 operator()()

```
template<Expression Ex, Place_Holder Ph>
template<Expression Exp>
auto ceras::model< Ex, Ph >::operator() (
    Exp const & ex ) const [inline], [noexcept]
```

Generating a new expression by using the current model.

Parameters

ex	An expression that represents the input to the model.
----	---

Returns

An expression that replacing the input node with a new epxression.

Example code

```
auto x = Input(); // input, (28*28,)
auto y = Dense( 128, 28*28 )( x );
auto m1 = model( x, y ); // this model is [(28*28,) -> (128,)]
auto u = Input(); // new input, (32,)
auto v = Dense( 28*28, 32 )( u );
auto m2 = model( u, v );
auto input = Input(); // (32, )
auto ouptut = m1( m2( input ) ); // this new expression is [(32,) -> (28*28,) -> (128,)], note x is not in
    this expression any more
auto m = model( input, output ); // create a new model
```

8.26.4.6 output() [1/2]

```
template<Expression Ex, Place_Holder Ph>
constexpr auto ceras::model< Ex, Ph >::output ( ) const [inline], [constexpr]
```

Returns the output layer(s) of the model in a 'list', which is are expressions.

8.26.4.7 output() [2/2]

```
template<Expression Ex, Place_Holder Ph>
output_layer_type ceras::model< Ex, Ph >::output ( ) const [inline], [noexcept]
```

Returns the output layer of the model.

8.26.4.8 predict() [1/3]

```
template<Expression Ex, Place_Holder Ph>
template<Tensor Tsor>
auto ceras::model< Ex, Ph >::predict (
    Tsor const & input_tensor ) [inline]
```

Making prediction by binding the nput data to the place_holder_ and evaluating expression_.

Parameters

<i>input_tensor</i>	The input samples.
---------------------	--------------------

Returns

The result this model predicts.

Example to predict

```
auto input = Input();
auto l1 = relu( Dense( 1024, 28*28 )( input ) );
auto output = sigmoid( Dense( 10, 1024 )( l1 ) );
// ... train the model after defining a loss and an optimizer
auto m = model{ input, output };
auto test_data = random( {128, 28*28} ); // batch size is 128
auto result = model.predict( test_data ); // should produce an tensor of (128, 10)
```

8.26.4.9 predict() [2/3]

```
template<Expression Ex, Place_Holder Ph>
template<Tensor Tsor>
constexpr auto ceras::model< Ex, Ph >::predict (
    Tsor const & input_tensor ) const [inline], [constexpr]
```

Making prediction by binding the nput data to the place_holder_ and evaluating expression_.

Parameters

<i>input_tensor</i>	The input samples.
---------------------	--------------------

Returns

The result this model predicts.

Example to predict

```
auto input = Input();
auto l1 = relu( Dense( 1024, 28*28 )( input ) );
auto output = sigmoid( Dense( 10, 1024 )( l1 ) );
// ... train the model after defining a loss and an optimizer
auto m = model{ input, output };
auto test_data = random( {128, 28*28} ); // batch size is 128
auto result = model.predict( test_data ); // should produce an tensor of (128, 10)
```

8.26.4.10 predict() [3/3]

```
template<Expression Ex, Place_Holder Ph>
template<List Tsor_List>
auto ceras::model< Ex, Ph >::predict (
    Tsor_List const & input_tensor ) const [inline]
```

8.26.4.11 save_weights()

```
template<Expression Ex, Place_Holder Ph>
void ceras::model< Ex, Ph >::save_weights (
    std::string const & file ) [inline]
```

Writes all variables to a file

8.26.4.12 summary()

```
template<Expression Ex, Place_Holder Ph>
void ceras::model< Ex, Ph >::summary (
    std::string const & file_name = std::string{} ) const [inline], [noexcept]
```

Print the model summary to console or to a file.

Parameters

<i>file_name</i>	The file to save the summary. If empty, the summary will be printed to console. Empty by default.
------------------	---

8.26.4.13 trainable()

```
template<Expression Ex, Place_Holder Ph>
void ceras::model< Ex, Ph >::trainable (
    bool t ) [inline]
```

8.26.5 Member Data Documentation**8.26.5.1 expression_**

```
template<Expression Ex, Place_Holder Ph>
output_layer_type ceras::model< Ex, Ph >::expression_
```

output layer of the model.

8.26.5.2 input_layer_

```
template<Expression Ex, Place_Holder Ph>
Input_List ceras::model< Ex, Ph >::input_layer_
```

8.26.5.3 output_layer_

```
template<Expression Ex, Place_Holder Ph>
Output_List ceras::model< Ex, Ph >::output_layer_
```

8.26.5.4 place_holder_

```
template<Expression Ex, Place_Holder Ph>
input_layer_type ceras::model< Ex, Ph >::place_holder_
```

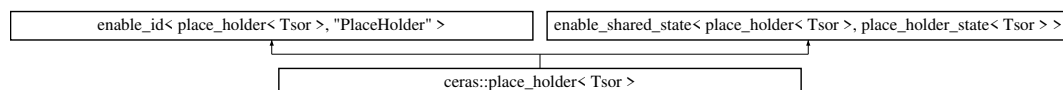
The documentation for this struct was generated from the following files:

- /home/feng/workspace/github.repo/ceras/include/model.hpp
- /home/feng/workspace/github.repo/ceras/include/xmodel.hpp

8.27 ceras::place_holder< Tsor > Struct Template Reference

```
#include <place_holder.hpp>
```

Inheritance diagram for ceras::place_holder< Tsor >:



Public Types

- typedef Tsor [tensor_type](#)

Public Member Functions

- [place_holder](#) ([place_holder](#) const &other)=default
- [place_holder](#) ([place_holder](#) &&other)=default
- [place_holder](#) & [operator=](#) ([place_holder](#) const &other)=default
- [place_holder](#) & [operator=](#) ([place_holder](#) &&other)=default
- [place_holder](#) ()
- [place_holder](#) (std::vector< unsigned long > const &shape_hint)
- void [bind](#) (Tsor data)
- Tsor const [forward](#) () const
- void [reset](#) () noexcept
- void [backward](#) (auto) const noexcept
- void [shape](#) (std::vector< unsigned long > const &shape_hint) noexcept
- std::vector< unsigned long > [shape](#) () const noexcept

8.27.1 Member Typedef Documentation

8.27.1.1 tensor_type

```
template<Tensor Tsor>
typedef Tsor ceras::place\_holder< Tsor >::tensor\_type
```

8.27.2 Constructor & Destructor Documentation

8.27.2.1 place_holder() [1/4]

```
template<Tensor Tsor>
ceras::place\_holder< Tsor >::place\_holder (
    place\_holder< Tsor > const & other ) [default]
```

8.27.2.2 place_holder() [2/4]

```
template<Tensor Tsor>
ceras::place\_holder< Tsor >::place\_holder (
    place\_holder< Tsor > && other ) [default]
```

8.27.2.3 place_holder() [3/4]

```
template<Tensor Tsor>
ceras::place\_holder< Tsor >::place\_holder ( ) [inline]
```

8.27.2.4 place_holder() [4/4]

```
template<Tensor Tsor>
ceras::place\_holder< Tsor >::place\_holder (
    std::vector< unsigned long > const & shape_hint ) [inline]
```

8.27.3 Member Function Documentation

8.27.3.1 backward()

```
template<Tensor Tsor>
void ceras::place_holder< Tsor >::backward (
    auto ) const [inline], [noexcept]
```

8.27.3.2 bind()

```
template<Tensor Tsor>
void ceras::place_holder< Tsor >::bind (
    Tsor data ) [inline]
```

8.27.3.3 forward()

```
template<Tensor Tsor>
Tsor const ceras::place_holder< Tsor >::forward ( ) const [inline]
```

8.27.3.4 operator=() [1/2]

```
template<Tensor Tsor>
place_holder & ceras::place_holder< Tsor >::operator= (
    place_holder< Tsor > && other ) [default]
```

8.27.3.5 operator=() [2/2]

```
template<Tensor Tsor>
place_holder & ceras::place_holder< Tsor >::operator= (
    place_holder< Tsor > const & other ) [default]
```

8.27.3.6 reset()

```
template<Tensor Tsor>
void ceras::place_holder< Tsor >::reset ( ) [inline], [noexcept]
```

8.27.3.7 `shape()` [1/2]

```
template<Tensor Tsor>
std::vector< unsigned long > ceras::place\_holder< Tsor >::shape ( ) const [inline], [noexcept]
```

8.27.3.8 `shape()` [2/2]

```
template<Tensor Tsor>
void ceras::place\_holder< Tsor >::shape (
    std::vector< unsigned long > const & shape_hint ) [inline], [noexcept]
```

The documentation for this struct was generated from the following file:

- [/home/feng/workspace/github.repo/ceras/include/place_holder.hpp](#)

8.28 `ceras::place_holder_state< Tsor >` Struct Template Reference

```
#include <place_holder.hpp>
```

Public Attributes

- Tsor [data_](#)
- std::vector< unsigned long > [shape_hint_](#)

8.28.1 Member Data Documentation

8.28.1.1 `data_`

```
template<Tensor Tsor>
Tsor ceras::place\_holder\_state< Tsor >::data_
```

8.28.1.2 `shape_hint_`

```
template<Tensor Tsor>
std::vector< unsigned long> ceras::place\_holder\_state< Tsor >::shape_hint_
```

The documentation for this struct was generated from the following file:

- [/home/feng/workspace/github.repo/ceras/include/place_holder.hpp](#)

8.29 ceras::regularizer< Float > Struct Template Reference

```
#include <variable.hpp>
```

Public Types

- typedef Float [value_type](#)

Public Member Functions

- constexpr [regularizer](#) ([value_type](#) l1=0.0, [value_type](#) l2=0.0, bool synchronized=false) noexcept

Public Attributes

- [value_type](#) l1_
- [value_type](#) l2_
- bool [synchronized_](#)

8.29.1 Member Typedef Documentation

8.29.1.1 value_type

```
template<typename Float >  
typedef Float ceras::regularizer< Float >::value_type
```

8.29.2 Constructor & Destructor Documentation

8.29.2.1 regularizer()

```
template<typename Float >  
constexpr ceras::regularizer< Float >::regularizer (  
    value\_type l1 = 0.0,  
    value\_type l2 = 0.0,  
    bool synchronized = false ) [inline], [constexpr], [noexcept]
```

8.29.3 Member Data Documentation

8.29.3.1 l1_

```
template<typename Float >
value_type ceras::regularizer< Float >::l1_
```

8.29.3.2 l2_

```
template<typename Float >
value_type ceras::regularizer< Float >::l2_
```

8.29.3.3 synchronized_

```
template<typename Float >
bool ceras::regularizer< Float >::synchronized_
```

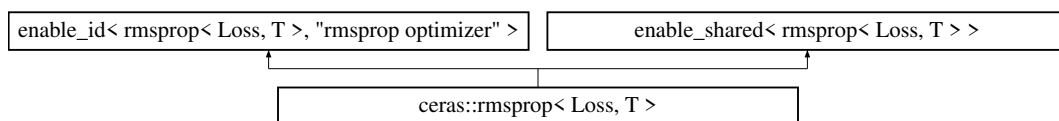
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/variable.hpp>

8.30 ceras::rmsprop< Loss, T > Struct Template Reference

```
#include <optimizer.hpp>
```

Inheritance diagram for ceras::rmsprop< Loss, T >:



Public Types

- typedef [tensor](#)< T > [tensor_type](#)

Public Member Functions

- [rmsprop](#) (Loss &loss, std::size_t batch_size, T learning_rate=1.0e-1, T rho=0.9, T decay=0.0) noexcept
- void [forward](#) ()

Public Attributes

- Loss & [loss_](#)
- T [learning_rate_](#)
- T [rho_](#)
- T [decay_](#)
- unsigned long [iterations_](#)

8.30.1 Member Typedef Documentation

8.30.1.1 tensor_type

```
template<typename Loss , typename T >
typedef tensor< T > ceras::rmsprop< Loss, T >::tensor\_type
```

8.30.2 Constructor & Destructor Documentation

8.30.2.1 rmsprop()

```
template<typename Loss , typename T >
ceras::rmsprop< Loss, T >::rmsprop (
    Loss & loss,
    std::size_t batch\_size,
    T learning\_rate = 1.0e-1,
    T rho = 0.9,
    T decay = 0.0 ) [inline], [noexcept]
```

8.30.3 Member Function Documentation

8.30.3.1 forward()

```
template<typename Loss , typename T >
void ceras::rmsprop< Loss, T >::forward ( ) [inline]
```

8.30.4 Member Data Documentation

8.30.4.1 decay_

```
template<typename Loss , typename T >
T ceras::rmsprop< Loss, T >::decay_
```

8.30.4.2 iterations_

```
template<typename Loss , typename T >
unsigned long ceras::rmsprop< Loss, T >::iterations_
```

8.30.4.3 learning_rate_

```
template<typename Loss , typename T >
T ceras::rmsprop< Loss, T >::learning_rate_
```

8.30.4.4 loss_

```
template<typename Loss , typename T >
Loss& ceras::rmsprop< Loss, T >::loss_
```

8.30.4.5 rho_

```
template<typename Loss , typename T >
T ceras::rmsprop< Loss, T >::rho_
```

The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/optimizer.hpp>

8.31 ceras::ceras_private::session< Tsor > Struct Template Reference

```
#include <session.hpp>
```

Public Types

- typedef [place_holder< Tsor >](#) [place_holder_type](#)
- typedef [variable< Tsor >](#) [variable_type](#)
- typedef [variable_state< Tsor >](#) [variable_state_type](#)

Public Member Functions

- [session](#) ()
- [session](#) ([session](#) const &)=delete
- [session](#) ([session](#) &&)=default
- [session](#) & [operator=](#) ([session](#) const &)=delete
- [session](#) & [operator=](#) ([session](#) &&)=default
- [session](#) & [rebind](#) ([place_holder_type](#) &p_holder, Tsor const &[value](#))
- [session](#) & [bind](#) ([place_holder_type](#) &p_holder, Tsor const &[value](#))
- [session](#) & [remember](#) ([variable_type](#) const &[v](#))
- template<typename Operation >
auto [run](#) (Operation &op) const
- template<typename Operation >
void [tap](#) (Operation &op) const
- void [deserialize](#) (std::string const &file_path)
- void [serialize](#) (std::string const &file_path) const
- void [save](#) (std::string const &file_path) const
- void [restore](#) (std::string const &file_path)
- void [save_original](#) (std::string const &file_path) const
- void [restore_original](#) (std::string const &file_path)
- [~session](#) ()

Public Attributes

- std::vector< [place_holder_type](#) > [place_holders_](#)
- std::map< int, [variable_type](#) > [variables_](#)

8.31.1 Member Typedef Documentation

8.31.1.1 place_holder_type

```
template<Tensor Tsor>
typedef place\_holder<Tsor> ceras::ceras\_private::session< Tsor >::place\_holder\_type
```

8.31.1.2 variable_state_type

```
template<Tensor Tsor>
typedef variable\_state<Tsor> ceras::ceras\_private::session< Tsor >::variable\_state\_type
```

8.31.1.3 variable_type

```
template<Tensor Tsor>
typedef variable<Tsor> ceras::ceras\_private::session< Tsor >::variable\_type
```

8.31.2 Constructor & Destructor Documentation

8.31.2.1 session() [1/3]

```
template<Tensor Tsor>
ceras::ceras_private::session< Tsor >::session ( ) [inline]
```

8.31.2.2 session() [2/3]

```
template<Tensor Tsor>
ceras::ceras_private::session< Tsor >::session (
    session< Tsor > const & ) [delete]
```

8.31.2.3 session() [3/3]

```
template<Tensor Tsor>
ceras::ceras_private::session< Tsor >::session (
    session< Tsor > && ) [default]
```

8.31.2.4 ~session()

```
template<Tensor Tsor>
ceras::ceras_private::session< Tsor >::~~session ( ) [inline]
```

8.31.3 Member Function Documentation

8.31.3.1 bind()

```
template<Tensor Tsor>
session & ceras::ceras_private::session< Tsor >::bind (
    place_holder_type & p_holder,
    Tsor const & value ) [inline]
```

8.31.3.2 deserialize()

```
template<Tensor Tsor>
void ceras::ceras_private::session< Tsor >::deserialize (
    std::string const & file_path ) [inline]
```

8.31.3.3 operator=() [1/2]

```
template<Tensor Tsor>
session & ceras::ceras_private::session< Tsor >::operator= (
    session< Tsor > && ) [default]
```

8.31.3.4 operator=() [2/2]

```
template<Tensor Tsor>
session & ceras::ceras_private::session< Tsor >::operator= (
    session< Tsor > const & ) [delete]
```

8.31.3.5 rebind()

```
template<Tensor Tsor>
session & ceras::ceras_private::session< Tsor >::rebind (
    place_holder_type & p_holder,
    Tsor const & value ) [inline]
```

8.31.3.6 remember()

```
template<Tensor Tsor>
session & ceras::ceras_private::session< Tsor >::remember (
    variable_type const & v ) [inline]
```

8.31.3.7 restore()

```
template<Tensor Tsor>
void ceras::ceras_private::session< Tsor >::restore (
    std::string const & file_path ) [inline]
```

8.31.3.8 restore_original()

```
template<Tensor Tsor>
void ceras::ceras_private::session< Tsor >::restore_original (
    std::string const & file_path ) [inline]
```

8.31.3.9 run()

```
template<Tensor Tsor>
template<typename Operation >
auto ceras::ceras_private::session< Tsor >::run (
    Operation & op ) const [inline]
```

8.31.3.10 save()

```
template<Tensor Tsor>
void ceras::ceras_private::session< Tsor >::save (
    std::string const & file_path ) const [inline]
```

8.31.3.11 save_original()

```
template<Tensor Tsor>
void ceras::ceras_private::session< Tsor >::save_original (
    std::string const & file_path ) const [inline]
```

8.31.3.12 serialize()

```
template<Tensor Tsor>
void ceras::ceras_private::session< Tsor >::serialize (
    std::string const & file_path ) const [inline]
```

8.31.3.13 tap()

```
template<Tensor Tsor>
template<typename Operation >
void ceras::ceras_private::session< Tsor >::tap (
    Operation & op ) const [inline]
```

8.31.4 Member Data Documentation

8.31.4.1 place_holders_

```
template<Tensor Tsor>
std::vector<place_holder_type> ceras::ceras_private::session< Tsor >::place_holders_
```

8.31.4.2 variables_

```
template<Tensor Tsor>
std::map<int, variable_type> ceras::ceras_private::session< Tsor >::variables_
```

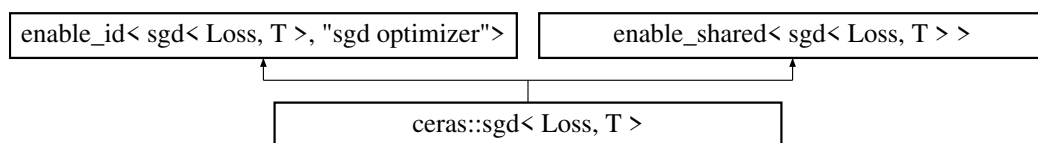
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/[session.hpp](#)

8.32 ceras::sgd< Loss, T > Struct Template Reference

```
#include <optimizer.hpp>
```

Inheritance diagram for ceras::sgd< Loss, T >:



Public Types

- typedef [tensor](#)< T > [tensor_type](#)

Public Member Functions

- [sgd](#) (Loss &loss, std::size_t batch_size, T learning_rate=1.0e-1, T momentum=0.0, T decay=0.0, bool nestorov=false) noexcept
- void [forward](#) ()

Public Attributes

- Loss & [loss_](#)
- T [learning_rate_](#)
- T [momentum_](#)
- T [decay_](#)
- bool [nesterov_](#)
- unsigned long [iterations_](#)

8.32.1 Member Typedef Documentation

8.32.1.1 `tensor_type`

```
template<typename Loss , typename T >
typedef tensor< T > ceras::sgd< Loss, T >::tensor\_type
```

8.32.2 Constructor & Destructor Documentation

8.32.2.1 `sgd()`

```
template<typename Loss , typename T >
ceras::sgd< Loss, T >::sgd (
    Loss & loss,
    std::size_t batch\_size,
    T learning\_rate = 1.0e-1,
    T momentum = 0.0,
    T decay = 0.0,
    bool nesterov = false ) [inline], [noexcept]
```

8.32.3 Member Function Documentation

8.32.3.1 `forward()`

```
template<typename Loss , typename T >
void ceras::sgd< Loss, T >::forward ( ) [inline]
```

8.32.4 Member Data Documentation

8.32.4.1 decay_

```
template<typename Loss , typename T >  
T ceras::sgd< Loss, T >::decay_
```

8.32.4.2 iterations_

```
template<typename Loss , typename T >  
unsigned long ceras::sgd< Loss, T >::iterations_
```

8.32.4.3 learning_rate_

```
template<typename Loss , typename T >  
T ceras::sgd< Loss, T >::learning_rate_
```

8.32.4.4 loss_

```
template<typename Loss , typename T >  
Loss& ceras::sgd< Loss, T >::loss_
```

8.32.4.5 momentum_

```
template<typename Loss , typename T >  
T ceras::sgd< Loss, T >::momentum_
```

8.32.4.6 nesterov_

```
template<typename Loss , typename T >  
bool ceras::sgd< Loss, T >::nesterov_
```

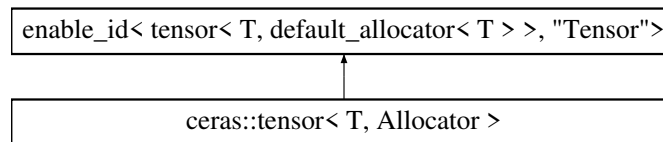
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/optimizer.hpp>

8.33 ceras::tensor< T, Allocator > Struct Template Reference

```
#include <tensor.hpp>
```

Inheritance diagram for ceras::tensor< T, Allocator >:



Public Types

- typedef T [value_type](#)
- typedef Allocator [allocator](#)
- typedef std::vector< T, Allocator > [vector_type](#)
- typedef std::shared_ptr< [vector_type](#) > [shared_vector](#)
- typedef [tensor](#) [self_type](#)

Public Member Functions

- [tensor](#) ()
- constexpr [tensor](#) (std::vector< unsigned long > const &[shape](#), std::initializer_list< T > init, const Allocator &alloc=Allocator())
Construct a vector with the specified shape, initialized value and a (default) allocator.
- constexpr [tensor](#) (std::vector< unsigned long > const &[shape](#))
Construct a vector with the specified shape. All values initialized to default. With a default constructed allocator.
- constexpr [tensor](#) (std::vector< unsigned long > const &[shape](#), T init)
*Construct a vector with the specified shape and all values initialized to *init*. With a default constructed allocator.*
- constexpr [tensor](#) ([tensor](#) const &other, unsigned long memory_offset) noexcept
- constexpr [tensor](#) ([self_type](#) const &other) noexcept
Copy-ctor.
- constexpr [tensor](#) ([self_type](#) &&other) noexcept
Move-ctor.
- constexpr [self_type](#) & operator= ([self_type](#) const &other) noexcept
Copy-assignment.
- constexpr [self_type](#) & operator= ([self_type](#) &&other) noexcept
Move-assignment.
- constexpr auto [begin](#) () noexcept
Iterator to the first element of the tensor.
- constexpr auto [begin](#) () const noexcept
Iterator to the first element of the tensor.
- constexpr auto [cbegin](#) () const noexcept
Iterator to the first element of the tensor.
- constexpr auto [end](#) () noexcept
Iterator to the element following the last element of the tensor.
- constexpr auto [end](#) () const noexcept
Iterator to the element following the last element of the tensor.
- constexpr auto [cend](#) () const noexcept

- Iterator to the element following the last element of the tensor.*
- constexpr auto [rbegin](#) () noexcept
- Reverse iterator to the first element of the tensor.*
- constexpr auto [rbegin](#) () const noexcept
- Reverse iterator to the first element of the tensor.*
- constexpr auto [crbegin](#) () const noexcept
- Reverse iterator to the first element of the tensor.*
- constexpr auto [rend](#) () noexcept
- Reverse iterator to the element following the last element of the tensor.*
- constexpr auto [rend](#) () const noexcept
- Reverse iterator to the element following the last element of the tensor.*
- constexpr auto [crend](#) () const noexcept
- Reverse iterator to the element following the last element of the tensor.*
- constexpr auto [front](#) ()
- constexpr auto [front](#) () const
- constexpr auto [back](#) ()
- constexpr auto [back](#) () const
- constexpr unsigned long [size](#) () const noexcept
- Number of elements in the tensor.*
- constexpr bool [empty](#) () const noexcept
- Check if the tensor has elements.*
- constexpr [self_type](#) & [reset](#) (T val=T{0})
- constexpr unsigned long [ndim](#) () const noexcept
- Dimension of the tensor.*
- constexpr std::vector< unsigned long > const & [shape](#) () const noexcept
- Shape of the tensor.*
- constexpr [self_type](#) & [deep_copy](#) ([self_type](#) const &other)
- A deep copy of the tensor.*
- constexpr [self_type](#) const [deep_copy](#) () const
- constexpr [self_type](#) const [copy](#) () const
- constexpr [value_type](#) & [operator\[\]](#) (unsigned long idx)
- constexpr [value_type](#) const & [operator\[\]](#) (unsigned long idx) const
- constexpr [self_type](#) & [resize](#) (std::vector< unsigned long > const &new_shape)
- Resize the tensor with a new shape.*
- constexpr [self_type](#) & [reshape](#) (std::vector< unsigned long > const &new_shape)
- Reshape tensor. -1 indicates the dimension needs recalculating.*
- constexpr [self_type](#) & [shrink_to](#) (std::vector< unsigned long > const &new_shape)
- constexpr [self_type](#) & [creep_to](#) (unsigned long new_memory_offset)
- constexpr [value_type](#) * [data](#) () noexcept
- Returns pointer to the underlying array serving as element storage.*
- constexpr const [value_type](#) * [data](#) () const noexcept
- Returns pointer to the underlying array serving as element storage.*
- template<typename Function >
- constexpr [self_type](#) & [map](#) (Function const &f)
 - Applying element-wise operation on each element in the tensor.*
- constexpr [self_type](#) & [operator+=](#) ([self_type](#) const &other)
- constexpr [self_type](#) & [operator+=](#) ([value_type](#) x)
- constexpr [self_type](#) & [operator-=](#) ([self_type](#) const &other)
- constexpr [self_type](#) & [operator-=](#) ([value_type](#) x)
- constexpr [self_type](#) & [operator*=](#) ([self_type](#) const &other)
- constexpr [self_type](#) & [operator*=](#) ([value_type](#) x)
- constexpr [self_type](#) & [operator/=](#) ([self_type](#) const &other)

- constexpr [self_type](#) & [operator/](#)= ([value_type](#) x)
- constexpr [self_type](#) const [operator-](#) () const
- constexpr [value_type](#) [as_scalar](#) () const noexcept
- template<typename U >
constexpr auto [as_type](#) () const noexcept
- [tensor slice](#) (unsigned long m, unsigned long n) const noexcept

Public Attributes

- std::vector< unsigned long > [shape_](#)
- unsigned long [memory_offset_](#)
- [shared_vector](#) [vector_](#)

8.33.1 Member Typedef Documentation

8.33.1.1 allocator

```
template<typename T , typename Allocator = default_allocator<T>>
typedef Allocator ceras::tensor< T, Allocator >::allocator
```

8.33.1.2 self_type

```
template<typename T , typename Allocator = default_allocator<T>>
typedef tensor ceras::tensor< T, Allocator >::self\_type
```

8.33.1.3 shared_vector

```
template<typename T , typename Allocator = default_allocator<T>>
typedef std::shared_ptr<vector\_type> ceras::tensor< T, Allocator >::shared\_vector
```

8.33.1.4 value_type

```
template<typename T , typename Allocator = default_allocator<T>>
typedef T ceras::tensor< T, Allocator >::value\_type
```

8.33.1.5 `vector_type`

```
template<typename T , typename Allocator = default_allocator<T>>
typedef std::vector<T, Allocator> ceras::tensor< T, Allocator >::vector_type
```

8.33.2 Constructor & Destructor Documentation

8.33.2.1 `tensor()` [1/7]

```
template<typename T , typename Allocator = default_allocator<T>>
ceras::tensor< T, Allocator >::tensor ( ) [inline]
```

@brief Construct an empty vector

8.33.2.2 `tensor()` [2/7]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr ceras::tensor< T, Allocator >::tensor (
    std::vector< unsigned long > const & shape,
    std::initializer_list< T > init,
    const Allocator & alloc = Allocator() ) [inline], [constexpr]
```

Construct a vector with the specified shape, initialized value and a (default) allocator.

8.33.2.3 `tensor()` [3/7]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr ceras::tensor< T, Allocator >::tensor (
    std::vector< unsigned long > const & shape ) [inline], [constexpr]
```

Construct a vector with the specified shape. All values initialized to default. With a default constructed allocator.

8.33.2.4 `tensor()` [4/7]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr ceras::tensor< T, Allocator >::tensor (
    std::vector< unsigned long > const & shape,
    T init ) [inline], [constexpr]
```

Construct a vector with the specified shape and all values initialized to `init`. With a default constructed allocator.

8.33.2.5 tensor() [5/7]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr ceras::tensor< T, Allocator >::tensor (
    tensor< T, Allocator > const & other,
    unsigned long memory_offset ) [inline], [constexpr], [noexcept]
```

8.33.2.6 tensor() [6/7]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr ceras::tensor< T, Allocator >::tensor (
    self_type const & other ) [inline], [constexpr], [noexcept]
```

Copy-ctor.

8.33.2.7 tensor() [7/7]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr ceras::tensor< T, Allocator >::tensor (
    self_type && other ) [inline], [constexpr], [noexcept]
```

Move-ctor.

8.33.3 Member Function Documentation**8.33.3.1 as_scalar()**

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr value_type ceras::tensor< T, Allocator >::as_scalar ( ) const [inline], [constexpr],
[noexcept]
```

8.33.3.2 as_type()

```
template<typename T , typename Allocator = default_allocator<T>>
template<typename U >
constexpr auto ceras::tensor< T, Allocator >::as_type ( ) const [inline], [constexpr], [noexcept]
```

8.33.3.3 back() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::back ( ) [inline], [constexpr]
```

@brief Reference to the last element in the tensor.

8.33.3.4 back() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::back ( ) const [inline], [constexpr]
```

@brief Reference to the last element in the tensor.

8.33.3.5 begin() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::begin ( ) const [inline], [constexpr], [noexcept]
```

Iterator to the first element of the tensor.

8.33.3.6 begin() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::begin ( ) [inline], [constexpr], [noexcept]
```

Iterator to the first element of the tensor.

8.33.3.7 cbegin()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::cbegin ( ) const [inline], [constexpr], [noexcept]
```

Iterator to the first element of the tensor.

8.33.3.8 cend()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::cend ( ) const [inline], [constexpr], [noexcept]
```

Iterator to the element following the last element of the tensor.

8.33.3.9 copy()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type const ceras::tensor< T, Allocator >::copy ( ) const [inline], [constexpr]
```

8.33.3.10 crbegin()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::crbegin ( ) const [inline], [constexpr], [noexcept]
```

Reverse iterator to the first element of the tensor.

8.33.3.11 creep_to()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::creep_to (
    unsigned long new_memory_offset ) [inline], [constexpr]
```

8.33.3.12 crend()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::crend ( ) const [inline], [constexpr], [noexcept]
```

Reverse iterator to the element following the last element of the tensor.

8.33.3.13 data() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr const value_type * ceras::tensor< T, Allocator >::data ( ) const [inline], [constexpr],
[noexcept]
```

Returns pointer to the underlying array serving as element storage.

The pointer is such that range `[data(); data() + size())` is always a valid range, even if the container is empty (`data()` is not dereferenceable in that case).

8.33.3.14 data() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr value_type * ceras::tensor< T, Allocator >::data ( ) [inline], [constexpr], [noexcept]
```

Returns pointer to the underlying array serving as element storage.

The pointer is such that range `[data(); data() + size())` is always a valid range, even if the container is empty (`data()` is not dereferenceable in that case).

8.33.3.15 deep_copy() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type const ceras::tensor< T, Allocator >::deep_copy ( ) const [inline], [constexpr]
```

8.33.3.16 deep_copy() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::deep_copy (
    self_type const & other ) [inline], [constexpr]
```

A deep copy of the tensor.

8.33.3.17 empty()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr bool ceras::tensor< T, Allocator >::empty ( ) const [inline], [constexpr], [noexcept]
```

Check if the tensor has elements.

8.33.3.18 end() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::end ( ) const [inline], [constexpr], [noexcept]
```

Iterator to the element following the last element of the tensor.

8.33.3.19 end() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::end ( ) [inline], [constexpr], [noexcept]
```

Iterator to the element following the last element of the tensor.

8.33.3.20 front() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::front ( ) [inline], [constexpr]
```

@breif Reference to the first element in the tensor.

8.33.3.21 front() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::front ( ) const [inline], [constexpr]
```

@breif Reference to the first element in the tensor.

8.33.3.22 map()

```
template<typename T , typename Allocator = default_allocator<T>>
template<typename Function >
constexpr self_type & ceras::tensor< T, Allocator >::map (
    Function const & f ) [inline], [constexpr]
```

Applying element-wise operation on each element in the tensor.

```
tensor<double> x{...};
x.map( []( double v ){ return 1.0/v+1.0; } );
```

8.33.3.23 ndim()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr unsigned long ceras::tensor< T, Allocator >::ndim ( ) const [inline], [constexpr],
[noexcept]
```

Dimension of the tensor.

8.33.3.24 operator*=() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator*= (
    self_type const & other ) [inline], [constexpr]
```

8.33.3.25 operator*=() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator*= (
    value_type x ) [inline], [constexpr]
```

8.33.3.26 operator+=() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator+= (
    self_type const & other ) [inline], [constexpr]
```


8.33.3.27 operator+=() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator+= (
    value_type x ) [inline], [constexpr]
```

8.33.3.28 operator-()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type const ceras::tensor< T, Allocator >::operator- ( ) const [inline], [constexpr]
```

8.33.3.29 operator-=() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator-= (
    self_type const & other ) [inline], [constexpr]
```

8.33.3.30 operator-=() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator-= (
    value_type x ) [inline], [constexpr]
```

8.33.3.31 operator/=() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator/= (
    self_type const & other ) [inline], [constexpr]
```

8.33.3.32 operator/=() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator/= (
    value_type x ) [inline], [constexpr]
```

8.33.3.33 operator=() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator= (
    self_type && other ) [inline], [constexpr], [noexcept]
```

Move-assignment.

8.33.3.34 operator=() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::operator= (
    self_type const & other ) [inline], [constexpr], [noexcept]
```

Copy-assignment.

8.33.3.35 operator[]() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr value_type & ceras::tensor< T, Allocator >::operator[] (
    unsigned long idx ) [inline], [constexpr]
```

8.33.3.36 operator[]() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr value_type const & ceras::tensor< T, Allocator >::operator[] (
    unsigned long idx ) const [inline], [constexpr]
```

8.33.3.37 rbegin() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::rbegin ( ) const [inline], [constexpr], [noexcept]
```

Reverse iterator to the first element of the tensor.

8.33.3.38 rbegin() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::rbegin ( ) [inline], [constexpr], [noexcept]
```

Reverse iterator to the first element of the tensor.

8.33.3.39 rend() [1/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::rend ( ) const [inline], [constexpr], [noexcept]
```

Reverse iterator to the element following the last element of the tensor.

8.33.3.40 rend() [2/2]

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr auto ceras::tensor< T, Allocator >::rend ( ) [inline], [constexpr], [noexcept]
```

Reverse iterator to the element following the last element of the tensor.

8.33.3.41 reset()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::reset (
    T val = T{0} ) [inline], [constexpr]
```

Resetting all elements in the tensor to a fixed value (default to 0), without change the shape.

Example code:

```
tensor<float> ts;
ts.reset( 0.0f );
```

8.33.3.42 reshape()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::reshape (
    std::vector< unsigned long > const & new_shape ) [inline], [constexpr]
```

Reshape tensor. -1 indicates the dimension needs recalculating.

```
tensor<float> t{ {2, 3, 4} };
auto t1 = t.reshape( {3, 8} );
auto t2 = t.reshape( {1, 4, -1UL} );
```

8.33.3.43 resize()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::resize (
    std::vector< unsigned long > const & new_shape ) [inline], [constexpr]
```

Resize the tensor with a new shape.

8.33.3.44 shape()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr std::vector< unsigned long > const & ceras::tensor< T, Allocator >::shape ( ) const
[inline], [constexpr], [noexcept]
```

Shape of the tensor.

8.33.3.45 shrink_to()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr self_type & ceras::tensor< T, Allocator >::shrink_to (
    std::vector< unsigned long > const & new_shape ) [inline], [constexpr]
```

8.33.3.46 size()

```
template<typename T , typename Allocator = default_allocator<T>>
constexpr unsigned long ceras::tensor< T, Allocator >::size ( ) const [inline], [constexpr],
[noexcept]
```

Number of elements in the tensor.

8.33.3.47 slice()

```
template<typename T , typename Allocator = default_allocator<T>>
tensor ceras::tensor< T, Allocator >::slice (
    unsigned long m,
    unsigned long n ) const [inline], [noexcept]
```

8.33.4 Member Data Documentation

8.33.4.1 memory_offset_

```
template<typename T , typename Allocator = default_allocator<T>>
unsigned long ceras::tensor< T, Allocator >::memory_offset_
```

8.33.4.2 shape_

```
template<typename T , typename Allocator = default_allocator<T>>
std::vector<unsigned long> ceras::tensor< T, Allocator >::shape_
```

8.33.4.3 vector_

```
template<typename T , typename Allocator = default_allocator<T>>
shared_vector ceras::tensor< T, Allocator >::vector_
```

The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/[tensor.hpp](#)

8.34 ceras::tensor_deduction< L, R > Struct Template Reference

```
#include <value.hpp>
```

Public Types

- using [op_type](#) = std::conditional< [is_value_v](#)< L >, R, L >::type
- using [tensor_type](#) = std::remove_cv_t< decltype(std::declval< [op_type](#) >()).forward())>

8.34.1 Member Typedef Documentation

8.34.1.1 op_type

```
template<typename L , typename R >
using ceras::tensor_deduction< L, R >::op_type = std::conditional<is\_value\_v<L>, R, L>::type
```

8.34.1.2 tensor_type

```
template<typename L , typename R >
using ceras::tensor_deduction< L, R >::tensor_type = std::remove_cv_t<decltype(std::declval<op\_type>()).forwa
```

The documentation for this struct was generated from the following file:

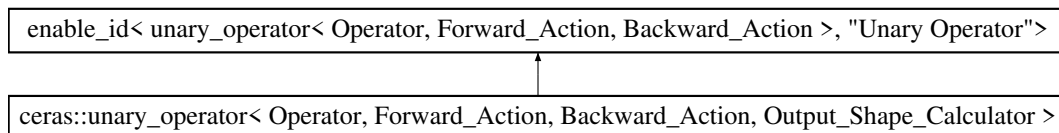
- /home/feng/workspace/github.repo/ceras/include/[value.hpp](#)

8.35 ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > Struct Template Reference

A unary operator is composed of a.) an input expression, b.) a forward action and c.) a backward action.

```
#include <operation.hpp>
```

Inheritance diagram for ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >:



Public Types

- typedef decltype(std::declval< Forward_Action >()(std::declval< decltype(op_)>()).forward())) [tensor_type](#)

Public Member Functions

- [unary_operator](#) (Operator const &op, Forward_Action const &forward_action, Backward_Action const &backward_action, Output_Shape_Calculator const &output_shape_calculator) noexcept
 - auto [forward](#) ()
 - void [backward](#) ([tensor_type](#) const &grad)
 - std::vector< unsigned long > [shape](#) () const noexcept
- Calculate the output tensor shape.*

Public Attributes

- Operator [op_](#)
- Forward_Action [forward_action_](#)
- Backward_Action [backward_action_](#)
- Output_Shape_Calculator [output_shape_calculator_](#)
- [tensor_type](#) [input_data_](#)
- [tensor_type](#) [output_data_](#)

8.35.1 Detailed Description

```
template<typename Operator, typename Forward_Action, typename Backward_Action, typename Output_Shape_Calculator =
identity_output_shape_calculator>
struct ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >
```

A unary operator is composed of a.) an input expression, b.) a forward action and c.) a backward action.

8.35.2 Member Typedef Documentation

8.35.2.1 tensor_type

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename  
Output_Shape_Calculator = identity_output_shape_calculator>  
typedef decltype( std::declval<Forward_Action>() ( std::declval<decltype(op_)>().forward() ) )  
ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >↔  
::tensor_type
```

8.35.3 Constructor & Destructor Documentation

8.35.3.1 unary_operator()

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename  
Output_Shape_Calculator = identity_output_shape_calculator>  
ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >↔  
::unary_operator (   
    Operator const & op,  
    Forward_Action const & forward_action,  
    Backward_Action const & backward_action,  
    Output_Shape_Calculator const & output_shape_calculator ) [inline], [noexcept]
```

8.35.4 Member Function Documentation

8.35.4.1 backward()

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename  
Output_Shape_Calculator = identity_output_shape_calculator>  
void ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator  
>::backward (   
    tensor_type const & grad ) [inline]
```

8.35.4.2 forward()

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename  
Output_Shape_Calculator = identity_output_shape_calculator>  
auto ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator  
>::forward ( ) [inline]
```

8.35.4.3 shape()

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename
Output_Shape_Calculator = identity_output_shape_calculator>
std::vector< unsigned long > ceras::unary_operator< Operator, Forward_Action, Backward_↵
Action, Output_Shape_Calculator >::shape ( ) const [inline], [noexcept]
```

Calculate the output tensor shape.

8.35.5 Member Data Documentation

8.35.5.1 backward_action_

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename
Output_Shape_Calculator = identity_output_shape_calculator>
Backward_Action ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_↵
Shape_Calculator >::backward_action_
```

8.35.5.2 forward_action_

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename
Output_Shape_Calculator = identity_output_shape_calculator>
Forward_Action ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_↵
Shape_Calculator >::forward_action_
```

8.35.5.3 input_data_

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename
Output_Shape_Calculator = identity_output_shape_calculator>
tensor_type ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_↵
Calculator >::input_data_
```

8.35.5.4 op_

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename
Output_Shape_Calculator = identity_output_shape_calculator>
Operator ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_↵
Calculator >::op_
```


8.35.5.5 output_data_

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename
Output_Shape_Calculator = identity_output_shape_calculator>
tensor_type ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_↵
Calculator >::output_data_
```

8.35.5.6 output_shape_calculator_

```
template<typename Operator , typename Forward_Action , typename Backward_Action , typename
Output_Shape_Calculator = identity_output_shape_calculator>
Output_Shape_Calculator ceras::unary_operator< Operator, Forward_Action, Backward_Action,
Output_Shape_Calculator >::output_shape_calculator_
```

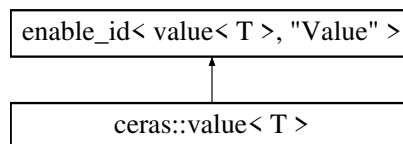
The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/operation.hpp

8.36 ceras::value< T > Struct Template Reference

```
#include <value.hpp>
```

Inheritance diagram for ceras::value< T >:



Public Types

- typedef T [value_type](#)
- typedef [tensor](#)< [value_type](#) > [tensor_type](#)

Public Member Functions

- [value](#) ()=delete
- [value](#) ([value_type](#) v) noexcept
- [value](#) ([value](#) const &) noexcept=default
- [value](#) ([value](#) &&) noexcept=default
- [value](#) & [operator](#)= ([value](#) const &) noexcept=default
- [value](#) & [operator](#)= ([value](#) &&) noexcept=default
- void [backward](#) (auto) noexcept
- template<Tensor Tsor>
Tsor const [forward](#) (Tsor const &refer) const
- std::vector< unsigned long > [shape](#) () const noexcept

Public Attributes

- [value_type data_](#)

8.36.1 Member Typedef Documentation

8.36.1.1 `tensor_type`

```
template<typename T >  
typedef tensor<value\_type> ceras::value< T >::tensor\_type
```

8.36.1.2 `value_type`

```
template<typename T >  
typedef T ceras::value< T >::value\_type
```

8.36.2 Constructor & Destructor Documentation

8.36.2.1 `value()` [1/4]

```
template<typename T >  
ceras::value< T >::value ( ) [delete]
```

8.36.2.2 `value()` [2/4]

```
template<typename T >  
ceras::value< T >::value (   
    value\_type v ) [inline], [noexcept]
```

8.36.2.3 `value()` [3/4]

```
template<typename T >  
ceras::value< T >::value (   
    value< T > const & ) [default], [noexcept]
```

8.36.2.4 value() [4/4]

```
template<typename T >
ceras::value< T >::value (
    value< T > && ) [default], [noexcept]
```

8.36.3 Member Function Documentation

8.36.3.1 backward()

```
template<typename T >
void ceras::value< T >::backward (
    auto ) [inline], [noexcept]
```

8.36.3.2 forward()

```
template<typename T >
template<Tensor Tsor>
Tsor const ceras::value< T >::forward (
    Tsor const & refer ) const [inline]
```

8.36.3.3 operator=() [1/2]

```
template<typename T >
value & ceras::value< T >::operator= (
    value< T > && ) [default], [noexcept]
```

8.36.3.4 operator=() [2/2]

```
template<typename T >
value & ceras::value< T >::operator= (
    value< T > const & ) [default], [noexcept]
```

8.36.3.5 shape()

```
template<typename T >
std::vector< unsigned long > ceras::value< T >::shape ( ) const [inline], [noexcept]
```

8.36.4 Member Data Documentation

8.36.4.1 data_

```
template<typename T >
value_type ceras::value< T >::data_
```

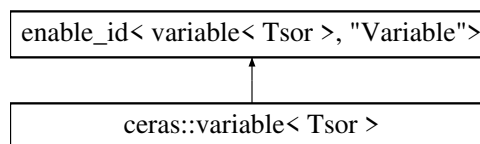
The documentation for this struct was generated from the following file:

- </home/feng/workspace/github.repo/ceras/include/value.hpp>

8.37 ceras::variable< Tsor > Struct Template Reference

```
#include <variable.hpp>
```

Inheritance diagram for ceras::variable< Tsor >:



Public Types

- typedef Tsor [tensor_type](#)
- typedef [tensor_type::value_type](#) [value_type](#)

Public Member Functions

- [variable](#) ([tensor_type](#) const &data, [value_type](#) l1=[value_type](#){0}, [value_type](#) l2=[value_type](#){0}, bool trainable=true)
- [variable](#) () noexcept
- [variable](#) ([variable](#) const &other)=default
- [variable](#) ([variable](#) &&)=default
- [variable](#) & operator= ([variable](#) &&)=default
- [variable](#) & operator= ([variable](#) const &other)=default
- [tensor_type](#) const [forward](#) () noexcept
- void [backward](#) (auto const &grad) noexcept
- std::vector< std::size_t > [shape](#) () const noexcept
- std::vector< [tensor_type](#) > & [contexts](#) ()
- std::vector< [tensor_type](#) > [contexts](#) () const
- [tensor_type](#) & [data](#) ()
- [tensor_type](#) [data](#) () const
- [tensor_type](#) & [gradient](#) ()
- [tensor_type](#) [gradient](#) () const
- void [reset](#) ()
- bool [trainable](#) () const noexcept
- void [trainable](#) (bool t)

Public Attributes

- std::shared_ptr< [variable_state< tensor_type > >](#) [state_](#)
- [regularizer< value_type >](#) [regularizer_](#)
- bool [trainable_](#)

8.37.1 Member Typedef Documentation

8.37.1.1 tensor_type

```
template<Tensor Ttor>
typedef Ttor ceras::variable< Ttor >::tensor\_type
```

8.37.1.2 value_type

```
template<Tensor Ttor>
typedef tensor_type::value_type ceras::variable< Ttor >::value\_type
```

8.37.2 Constructor & Destructor Documentation

8.37.2.1 variable() [1/4]

```
template<Tensor Ttor>
ceras::variable< Ttor >::variable (
    tensor\_type const & data,
    value\_type l1 = value\_type{0},
    value\_type l2 = value\_type{0},
    bool trainable = true ) [inline]
```

8.37.2.2 variable() [2/4]

```
template<Tensor Ttor>
ceras::variable< Ttor >::variable ( ) [inline], [noexcept]
```

8.37.2.3 variable() [3/4]

```
template<Tensor Tsor>
ceras::variable< Tsor >::variable (
    variable< Tsor > const & other ) [default]
```

8.37.2.4 variable() [4/4]

```
template<Tensor Tsor>
ceras::variable< Tsor >::variable (
    variable< Tsor > && ) [default]
```

8.37.3 Member Function Documentation**8.37.3.1 backward()**

```
template<Tensor Tsor>
void ceras::variable< Tsor >::backward (
    auto const & grad ) [inline], [noexcept]
```

8.37.3.2 contexts() [1/2]

```
template<Tensor Tsor>
std::vector< tensor_type > & ceras::variable< Tsor >::contexts ( ) [inline]
```

8.37.3.3 contexts() [2/2]

```
template<Tensor Tsor>
std::vector< tensor_type > ceras::variable< Tsor >::contexts ( ) const [inline]
```

8.37.3.4 data() [1/2]

```
template<Tensor Tsor>
tensor_type & ceras::variable< Tsor >::data ( ) [inline]
```

8.37.3.5 data() [2/2]

```
template<Tensor Tzor>
tensor_type ceras::variable< Tzor >::data ( ) const [inline]
```

8.37.3.6 forward()

```
template<Tensor Tzor>
tensor_type const ceras::variable< Tzor >::forward ( ) [inline], [noexcept]
```

8.37.3.7 gradient() [1/2]

```
template<Tensor Tzor>
tensor_type & ceras::variable< Tzor >::gradient ( ) [inline]
```

8.37.3.8 gradient() [2/2]

```
template<Tensor Tzor>
tensor_type ceras::variable< Tzor >::gradient ( ) const [inline]
```

8.37.3.9 operator=() [1/2]

```
template<Tensor Tzor>
variable & ceras::variable< Tzor >::operator= (
    variable< Tzor > && ) [default]
```

8.37.3.10 operator=() [2/2]

```
template<Tensor Tzor>
variable & ceras::variable< Tzor >::operator= (
    variable< Tzor > const & other ) [default]
```

8.37.3.11 reset()

```
template<Tensor Tzor>
void ceras::variable< Tzor >::reset ( ) [inline]
```

8.37.3.12 shape()

```
template<Tensor Tsor>
std::vector< std::size_t > ceras::variable< Tsor >::shape ( ) const [inline], [noexcept]
```

8.37.3.13 trainable() [1/2]

```
template<Tensor Tsor>
bool ceras::variable< Tsor >::trainable ( ) const [inline], [noexcept]
```

8.37.3.14 trainable() [2/2]

```
template<Tensor Tsor>
void ceras::variable< Tsor >::trainable (
    bool t ) [inline]
```

8.37.4 Member Data Documentation**8.37.4.1 regularizer_**

```
template<Tensor Tsor>
regularizer<value_type> ceras::variable< Tsor >::regularizer_
```

8.37.4.2 state_

```
template<Tensor Tsor>
std::shared_ptr<variable_state<tensor_type> > ceras::variable< Tsor >::state_
```

8.37.4.3 trainable_

```
template<Tensor Tsor>
bool ceras::variable< Tsor >::trainable_
```

The documentation for this struct was generated from the following file:

- /home/feng/workspace/github.repo/ceras/include/[variable.hpp](#)

8.38 ceras::variable_state< Tzor > Struct Template Reference

```
#include <variable.hpp>
```

Public Attributes

- Tzor [data_](#)
- Tzor [gradient_](#)
- std::vector< Tzor > [contexts_](#)

8.38.1 Member Data Documentation

8.38.1.1 contexts_

```
template<Tensor Tzor>  
std::vector<Tzor> ceras::variable\_state< Tzor >::contexts\_
```

8.38.1.2 data_

```
template<Tensor Tzor>  
Tzor ceras::variable\_state< Tzor >::data\_
```

8.38.1.3 gradient_

```
template<Tensor Tzor>  
Tzor ceras::variable\_state< Tzor >::gradient\_
```

The documentation for this struct was generated from the following file:

- [/home/feng/workspace/github.repo/ceras/include/variable.hpp](#)

Chapter 9

File Documentation

9.1 /home/feng/workspace/github.repo/ceras/include/activation.hpp File Reference

```
#include "../operation.hpp"
#include "../tensor.hpp"
#include "../utils/range.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/for_each.hpp"
#include "../utils/context_cast.hpp"
```

Namespaces

- namespace [ceras](#)

Functions

- `template<std::floating_point Float>`
`constexpr auto ceras::heaviside_step (Float f) noexcept`
Step activation function, an unary operator.
- `template<Expression Ex>`
`constexpr auto ceras::soft_sign (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto ceras::unit_step (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto ceras::binary_step (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto ceras::gaussian (Ex const &ex) noexcept`
Gaussian activation function, an unary operator.
- `template<Expression Ex>`
`constexpr auto ceras::softmax (Ex const &ex) noexcept`
Softmax activation function, an unary operator.
- `template<Expression Ex>`
`auto ceras::selu (Ex const &ex) noexcept`

- Scaled Exponential Linear Unit (SELU) activation function, an unary operator. If $x > 0$, returns $1.0507 x$; Otherwise, returns $1.67326 * 1.0507 * (\exp(x) - 1)$*
- `template<Expression Ex>`
`auto ceras::softplus (Ex const &ex) noexcept`
Softplus function, an unary operator. Returns $\log(\exp(x) + 1)$.
 - `template<Expression Ex>`
`auto ceras::softsign (Ex const &ex) noexcept`
Softsign function, an unary operator. Returns $x / (\text{abs}(x) + 1)$.
 - `template<Expression Ex>`
`auto ceras::sigmoid (Ex const &ex) noexcept`
Sigmoid function, an unary operator. Returns $1 / (\exp(-x) + 1)$.
 - `template<Expression Ex>`
`auto ceras::relu (Ex const &ex) noexcept`
Relu function, an unary operator. Returns x if positive, 0 otherwise.
 - `template<Expression Ex>`
`auto ceras::relu6 (Ex const &ex) noexcept`
Rectified Linear 6 function, an unary operator. Returns $\min(\max(\text{features}, 0), 6)$.
 - `template<typename T>`
`requires std::floating_point<T>`
`auto ceras::leaky_relu (T const factor=0.2) noexcept`
Leaky Rectified Linear function, an unary operator. Returns x if positive, αx otherwise. α defaults to 0.2 .
 - `template<typename T>`
`requires std::floating_point<T>`
`auto ceras::prelu (T const factor) noexcept`
 - `template<Expression Ex>`
`auto ceras::negative_relu (Ex const &ex) noexcept`
 - `template<typename T = float>`
`requires std::floating_point<T>`
`auto ceras::elu (T const alpha=1.0) noexcept`
*Exponential Linear function, an unary operator. Returns x if positive, $\alpha * (\exp(x) - 1)$ otherwise. α defaults to 0.2 .*
 - `template<Expression Ex>`
`auto ceras::exponential (Ex const &ex) noexcept`
Exponential function, an unary operator. Returns $\exp(x)$.
 - `template<Expression Ex>`
`auto ceras::hard_sigmoid (Ex const &ex) noexcept`
Hard Sigmoid function, an unary operator. Piecewise linear approximation of the sigmoid function.
 - `template<Expression Ex>`
`auto ceras::gelu (Ex const &ex) noexcept`
Gaussian Error function, an unary operator. GAUSSIAN ERROR LINEAR UNITS (GELUS) <https://arxiv.org/pdf/1606.08415.pdf> $f(x) = 0.5x (1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$ $\frac{df}{dx} = x (1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3))) + \sqrt{2/\pi} x \text{sech}^2(\sqrt{2/\pi}(x + 0.044715x^3)) (1 + 0.134145x^2)$ where $\text{sech}^2(x) = 1 - \tanh^2(x)$ derivative generated using service from <https://www.symbolab.com/solver/derivative-calculator>.
 - `template<Expression Ex>`
`auto ceras::swish (Ex const &ex) noexcept`
Swish activation function.
 - `template<Expression Ex>`
`auto ceras::silu (Ex const &ex) noexcept`
An alias name of activation swish.
 - `template<Expression Ex>`
`auto ceras::crelu (Ex const &ex) noexcept`
Concatenated Rectified Linear Units, an activation function which preserves both positive and negative phase information while enforcing non-saturated non-linearity.

- `template<Expression Ex>`
`auto ceras::tank_shrink (Ex const &ex) noexcept`
Tank shrink function.
- `template<Expression Ex>`
`auto ceras::mish (Ex const &ex) noexcept`
Mish function.
- `template<Expression Ex>`
`auto ceras::lisht (Ex const &ex) noexcept`
Lisht function.

9.2 activation.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef DJDWJBHNDAYTNOXLFOBDSGAQAAYPWMXJGEBYIRKEAKAQUUVGDUGGDKSDXUKSPCYNTWTDNII
2 #define DJDWJBHNDAYTNOXLFOBDSGAQAAYPWMXJGEBYIRKEAKAQUUVGDUGGDKSDXUKSPCYNTWTDNII
3
4 #include "../operation.hpp"
5 #include "../tensor.hpp"
6 #include "../utils/range.hpp"
7 #include "../utils/better_assert.hpp"
8 #include "../utils/for_each.hpp"
9 #include "../utils/context_cast.hpp"
10
11 namespace ceras
12 {
13
14     template< std::floating_point Float >
15     auto constexpr heaviside_step( Float f ) noexcept // f should not be zero
16     {
17         return [=]<Expression Ex>( Ex const& ex ) noexcept
18         {
19             return sigmoid( value( f+f ) * ex );
20         };
21     }
22
23     // alias of heaviside_step(20)
24     template <Expression Ex>
25     auto constexpr soft_sign( Ex const& ex ) noexcept // soft-sign
26     {
27         return heaviside_step( 20.0 ) ( ex );
28     }
29
30     // alias of heaviside_step(20)
31     template <Expression Ex>
32     auto constexpr unit_step( Ex const& ex ) noexcept
33     {
34         return soft_sign( ex );
35     }
36
37     // alias of heaviside_step(20)
38     template <Expression Ex>
39     auto constexpr binary_step( Ex const& ex ) noexcept
40     {
41         return soft_sign( ex );
42     }
43
44     template <Expression Ex>
45     auto constexpr gaussian( Ex const& ex ) noexcept
46     {
47         return exp( negative( square(ex) ) );
48     }
49
50     template <Expression Ex>
51     auto constexpr softmax( Ex const& ex ) noexcept
52     {
53         return make_unary_operator( [<Tensor Tsor>( Tsor const& input ) noexcept
54         {
55             better_assert( !input.empty(), "softmax forward: input tensor is
56             empty!" );
57             Tsor x = deep_copy( input );
58         }
59     );
60     }
61
62 }

```

```

96         std::size_t const last_dim = *(x.shape().rbegin());
97         std::size_t const rest_dim = x.size() / last_dim;
98         for ( auto idx : range( rest_dim ) )
99         {
100             auto [begin, end] = std::make_tuple( x.begin()+idx*last_dim,
101 x.begin()+(idx+1)*last_dim );
102             typename Tsor::value_type const mx = *std::max_element(
103 begin, end );
104             for_each( begin, end, [mx]( auto & v ){ v = std::exp( v-mx
105 ); } );
106             typename Tsor::value_type const sum = std::accumulate(
107 begin, end, typename Tsor::value_type{0} );
108             for_each( begin, end, [sum]( auto & v ){ v /= sum; } );
109             return x;
110         },
111         [<Tensor Tsor>( Tsor const&, Tsor const& output, Tsor const& grad )
112 noexcept
113 {
114     better_assert( !has_nan( grad ), "backprop: upcoming gradient
115 for activation softmax contains NaN" );
116     Tsor ans = grad;
117     for_each( ans.begin(), ans.end(), output.begin(), []( auto& a,
118 auto o ) { a *= o * ( typename Tsor::value_type{1} - o ); } );
119     return ans;
120 },
121     "Softmax"
122 )( ex );
123 }
124
125 template <Expression Ex>
126 auto inline selu( Ex const& ex ) noexcept
127 {
128     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
129     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
130
131     return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
132 {
133     typedef typename Tsor::value_type value_type;
134     value_type const lambda = 1.0507;
135     value_type const alpha = 1.67326;
136     Tsor& ans = context_cast<Tsor>( forward_cache );
137     ans.resize( input.shape() );
138     std::copy( input.begin(), input.end(), ans.begin() );
139     // if x >= 0: \lambda x
140     // if x < 0: \lambda \alpha (exp(x) - 1)
141     ans.map( [lambda, alpha](auto& x){ x = (x >= value_type{0}) ?
142 (lambda * x) : (lambda * alpha * (std::exp(x) - value_type{1})); } );
143     return ans;
144 },
145     [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor
146 const& grad ) noexcept
147 {
148     typedef typename Tsor::value_type value_type;
149     value_type const lambda = 1.0507;
150     value_type const alpha = 1.67326;
151     Tsor& ans = context_cast<Tsor>( backward_cache );
152     ans.resize( input.shape() ); // 1 / ( 1 + exp(-x) )
153     // if x >= 0: \lambda
154     // if x < 0: \lambda \alpha exp( x )
155     for_each( ans.begin(), ans.end(), input.begin(), grad.begin(),
156 [lambda, alpha]( auto& a, auto i, auto g ){ a = (i >= value_type{0}) ? (g * lambda) : (g * lambda *
157 alpha * std::exp(i)); } );
158     return ans;
159 },
160     "SeLU"
161 )( ex );
162 }
163
164 template <Expression Ex>
165 auto inline softplus( Ex const& ex ) noexcept
166 {
167     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
168     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
169
170     return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
171 {
172     Tsor& ans = context_cast<Tsor>( forward_cache );
173     ans.resize( input.shape() );
174     std::copy( input.begin(), input.end(), ans.begin() );
175     ans.map( [](auto& x){ x = std::log(1.0+std::exp(x)); } ); // ln(
176 1+e^x )
177     return ans;
178 },
179     [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor
180 const& grad ) noexcept
181 {

```

```

192         Tsor& ans = context_cast<Tsor>( backward_cache );
193         ans.resize( input.shape() ); // 1 / ( 1 + exp(-x) )
194         for_each( ans.begin(), ans.end(), input.begin(), grad.begin(),
[] ( auto& a, auto i, auto g ) { a = g / ( typename Tsor::value_type{1} - std::exp(-i) ); } );
195         return ans;
196     },
197     "SoftPlus"
198 ) ( ex );
199 }
200
201
202
203
204
205
206
207
208
209
210
211
212
213 template <Expression Ex>
214 auto inline softsign( Ex const& ex ) noexcept
215 {
216     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
217     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
218
219     return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
220     {
221         Tsor& ans = context_cast<Tsor>( forward_cache );
222         ans.resize( input.shape() );
223         std::copy( input.begin(), input.end(), ans.begin() );
224         ans.map( [] (auto& x) { x /= typename Tsor::value_type{1} +
std::abs(x); } ); // x / ( 1+|x| )
225         return ans;
226     },
227     [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor
const& grad ) noexcept
228     {
229         Tsor& ans = context_cast<Tsor>( backward_cache );
230         ans.resize( input.shape() ); // 1 / ( 1 + |x| )^2
231         for_each( ans.begin(), ans.end(), input.begin(), grad.begin(),
[] ( auto& a, auto i, auto g ) { auto tmp = typename Tsor::value_type{1} + std::abs(i); a = g /
(tmp*tmp); } );
232         return ans;
233     },
234     "SoftSign"
235 ) ( ex );
236 }
237
238
239
240
241
242
243
244
245 template <Expression Ex>
246 auto inline sigmoid( Ex const& ex ) noexcept
247 {
248     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
249     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
250     return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
251     {
252         Tsor& ans = context_cast<Tsor>( forward_cache );
253         ans.resize( input.shape() );
254         std::copy( input.begin(), input.end(), ans.begin() );
255         //auto ans = input.deep_copy();
256         ans.map( [] (auto& x) { x = 1.0 / (1.0+std::exp(-x)); } );
257         return ans;
258     },
259     [backward_cache]<Tensor Tsor>( Tsor const&, Tsor const& output, Tsor
const& grad ) noexcept
260     {
261         Tsor& ans = context_cast<Tsor>( backward_cache );
262         ans.resize( output.shape() );
263         //Tsor ans{ output.shape() };
264         for_each( ans.begin(), ans.end(), output.begin(), grad.begin(),
[] ( auto & a, auto o, auto g ) { a = g * o * ( typename Tsor::value_type{1} - o ); } );
265         return ans;
266     },
267     "Sigmoid"
268 ) ( ex );
269 }
270
271
272
273
274
275
276
277 namespace
278 {
279     struct relu_context
280     {
281         auto make_forward() const noexcept
282         {
283             return [] ( std::shared_ptr<std::any> forward_cache ) noexcept
284             {
285                 return [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
286                 {
287                     typedef typename Tsor::value_type value_type;
288                     Tsor& ans = context_cast<Tsor>( forward_cache );
289                     ans.resize( input.shape() );
290                     for_each( ans.begin(), ans.end(), input.begin(), [] (auto& o, auto x) { o =
std::max(x, value_type{0}); } );
291                 }
292             }
293         }
294     };

```

```

293         };
294     };
295 }
296
297     auto make_backward() const noexcept
298     {
299         return []<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad ) noexcept
300         {
301             typedef typename Tsor::value_type value_type;
302             Tsor ans = grad; // shallow copy
303             for_each( ans.begin(), ans.end(), input.begin(), []( auto& v, auto x ){ if ( x <=
value_type{0} ) v = value_type{0}; } );
304             return ans;
305         };
306     }
307     }; // relu_context
308
309 //anonymous namespace
310
311 template <Expression Ex>
312 auto relu( Ex const& ex ) noexcept
313 {
314     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
315     return make_unary_operator( relu_context{}.make_forward()( forward_cache ),
relu_context{}.make_backward(), "Relu" )( ex );
316 }
317
318
319
320 namespace
321 {
322     struct relu6_context
323     {
324         auto make_forward() const noexcept
325         {
326             return [] ( std::shared_ptr<std::any> forward_cache ) noexcept
327             {
328                 return [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
329                 {
330                     typedef typename Tsor::value_type value_type;
331                     Tsor& ans = context_cast<Tsor>( forward_cache );
332                     ans.resize( input.shape() );
333                     for_each( ans.begin(), ans.end(), input.begin(), []( auto& o, auto x ){ o =
std::min( value_type{6}, std::max(x, value_type{0}) ); } );
334                     return ans;
335                 };
336             };
337         }
338     };
339
340     auto make_backward() const noexcept
341     {
342         return []<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad ) noexcept
343         {
344             typedef typename Tsor::value_type value_type;
345             Tsor ans = grad; // shallow copy
346             //const typename Tsor::value_type zero{0};
347             for_each( ans.begin(), ans.end(), input.begin(), []( auto& v, auto x ){ if ( (x <=
value_type{0}) || (x >= value_type{6}) ) v = value_type{0}; } );
348             return ans;
349         };
350     }
351     }; // relu6_context
352
353 //anonymous namespace
354
355 template <Expression Ex>
356 auto relu6( Ex const& ex ) noexcept
357 {
358     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
359     return make_unary_operator( relu6_context{}.make_forward()( forward_cache ),
relu6_context{}.make_backward(), "Relu6" )( ex );
360 }
361
362
363
364 template< typename T > requires std::floating_point<T>
365 auto leaky_relu( T const factor=0.2 ) noexcept
366 {
367     better_assert( factor > T{0}, "Expecting leak_relu with a factor greater than 0, but got factor
= ", factor );
368     better_assert( factor < T{1}, "Expecting leak_relu with a factor less than 1, but got factor =
", factor );
369     return [factor]<Expression Ex>( Ex const& ex ) noexcept
370     {
371         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
372         return make_unary_operator( [factor, forward_cache]<Tensor Tsor>( Tsor const& input )
noexcept
373         {
374             Tsor& ans = context_cast<Tsor>( forward_cache );

```



```

405         ans.resize( input.shape() );
406         for_each( ans.begin(), ans.end(), input.begin(), [factor](
auto& v_out, auto v_in ){ v_out = std::max( T{v_in}, T{factor*v_in} ); } );
407         return ans;
408     },
409     [factor]<Tensor T>( T const& input, T const& grad, T const&
const& grad ) noexcept
410     {
411         typedef typename T::value_type value_type;
412         T ans = grad; // OK for shallow copy
413         for_each( ans.begin(), ans.end(), input.begin(), [factor](
value_type& v_back, value_type const v_in ){ v_back = (v_in > value_type{0}) ? v_in :
factor*v_back; } );
414         return ans;
415     },
416     "LeakyRelu"
417     )( ex );
418 };
419 }
420
421 template< typename T > requires std::floating_point<T>
422 auto prelu( T const factor ) noexcept
423 {
424     return leaky_relu( factor );
425 }
426
427 template <Expression Ex>
428 auto negative_relu( Ex const& ex ) noexcept
429 {
430     return negative( relu( ex ) );
431 }
432
433 template< typename T=float > requires std::floating_point<T>
434 auto elu( T const alpha=1.0 ) noexcept
435 {
436     return [alpha]<Expression Ex>( Ex const& ex ) noexcept
437     {
438         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
439         return make_unary_operator( [alpha, forward_cache]<Tensor T>( T const& input )
noexcept
440         {
441             typedef typename T::value_type value_type;
442             T& ans = context_cast<T>( forward_cache );
443             ans.resize( input.shape() );
444             for_each( ans.begin(), ans.end(), input.begin(), [alpha](
auto& v_out, auto v_in ){ v_out = (v_in > value_type{0}) ? v_in : (alpha * (std::exp(v_in) -
value_type{1})); } );
445             return ans;
446         },
447         [alpha]<Tensor T>( T const& input, T const& grad, T const&
const& grad ) noexcept
448         {
449             typedef typename T::value_type value_type;
450             T ans = grad; // OK for shallow copy
451             for_each( ans.begin(), ans.end(), input.begin(), [alpha](
value_type& v_back, value_type const v_in ){ v_back = (v_in >= value_type{0}) ? v_in :
alpha*std::exp(v_back); } );
452             return ans;
453         },
454         "ELU"
455         )( ex );
456     };
457 }
458
459 template <Expression Ex>
460 auto inline exponential( Ex const& ex ) noexcept
461 {
462     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
463     return make_unary_operator( [forward_cache]<Tensor T>( T const& input ) noexcept
464     {
465         T& ans = context_cast<T>( forward_cache );
466         ans.resize( input.shape() );
467         std::copy( input.begin(), input.end(), ans.begin() );
468         ans.map( [](auto& x){ x = std::exp(x); } ); // exp(x)
469         better_assert( !has_nan( ans ), "exponential operator forward
output contains nan." );
470         better_assert( !has_inf( ans ), "exponential operator forward
output contains inf." );
471         return ans;
472     },
473     [<Tensor T>( T const& input, T const& output, T const& grad )
noexcept
474     {
475         T ans = grad;
476         for_each( ans.begin(), ans.end(), output.begin(), []( auto& a,

```

```

504     auto o ){ a *= o; } );
505                                     return ans;
506                                     },
507                                     "Exponential"
508     }
509
510     template <Expression Ex>
511     auto inline hard_sigmoid( Ex const& ex ) noexcept
512     {
513         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
514
515         return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
516         {
517             typedef typename Tsor::value_type value_type;
518             Tsor& ans = context_cast<Tsor>( forward_cache );
519             ans.resize( input.shape() );
520             std::copy( input.begin(), input.end(), ans.begin() );
521             ans.map([auto& x] { x = ( x > value_type{1} ) ?
522 value_type{1} : ( x < value_type{-1} ) ? value_type{0} : (x+value_type{1})/value_type{2}; });
523             return ans;
524         },
525         [<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad )
526         noexcept
527         {
528             typedef typename Tsor::value_type value_type;
529             Tsor ans = grad;
530             for_each( ans.begin(), ans.end(), input.begin(), []( auto& a,
531 auto x ) { a = ((x > value_type{1}) || (x < value_type{-1})) ? value_type{0} : (a / value_type{2});
532 } );
533             return ans;
534         },
535         "HardSigmoid"
536     ) ( ex );
537 }
538
539     template <Expression Ex>
540     auto inline gelu( Ex const& ex ) noexcept
541     {
542         auto _gelu = [<typename T>( T x )
543         {
544             auto const ans = 0.5 * x * ( 1.0 + std::tanh( 0.79788456080286535588 * x ( 1.0 +
545 0.044715*x*x ) ) );
546             return static_cast<T>( ans );
547         };
548         auto sech_2 = [<auto x )
549         {
550             return 1.0 - std::pow( std::tanh(x), 2 );
551         };
552         auto _dgelu = [sech_2]<typename T>( T x )
553         {
554             auto const sq_2_pi_x = 0.79788456080286535588 * x;
555             auto const _xx = x * x;
556             auto const ans = 0.5 * ( 1.0 + std::tanh( sq_2_pi_x * ( 1.0 + 0.044715 * _xx ) ) ) +
557 sq_2_pi_x * sech_2( sq_2_pi_x * (1.0 + 0.044715 * _xx ) * ( 1.0 + 0.134145 * _xx ) );
558             return static_cast<T>( ans );
559         };
560
561         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
562
563         return make_unary_operator( [forward_cache, _gelu]<Tensor Tsor>( Tsor const& input ) noexcept
564         {
565             //typedef typename Tsor::value_type value_type;
566             Tsor& ans = context_cast<Tsor>( forward_cache );
567             ans.resize( input.shape() );
568             std::copy( input.begin(), input.end(), ans.begin() );
569             ans.map([_gelu](auto& x) { x = _gelu(x); });
570             return ans;
571         },
572         [_dgelu]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const&
573 grad ) noexcept
574         {
575             //typedef typename Tsor::value_type value_type;
576             Tsor ans = grad;
577             for_each( ans.begin(), ans.end(), [&_dgelu]( auto& x ) { x =
578 _dgelu(x); } );
579             return ans;
580         },
581         "GeLU"
582     ) ( ex );
583 }
584
585     template< Expression Ex >
586     auto swish( Ex const& ex ) noexcept
587     {
588         return hadamard_product( ex, sigmoid( ex ) );
589     }

```

```

616     }
617
621     template< Expression Ex >
622     auto silu( Ex const& ex ) noexcept
623     {
624         return swish( ex );
625     }
626
637     template< Expression Ex >
638     auto crelu( Ex const& ex ) noexcept
639     {
640         return concatenate(-1)( relu(ex), relu(-ex) );
641     }
642
651     template< Expression Ex >
652     auto tank_shrink( Ex const& ex ) noexcept
653     {
654         return ex - tanh( ex );
655     }
656
657
666     template< Expression Ex >
667     auto mish( Ex const& ex ) noexcept
668     {
669         return ex*tanh(softplus(ex));
670     }
671
672
681     template< Expression Ex >
682     auto lisht( Ex const& ex ) noexcept
683     {
684         return ex*tanh(ex);
685     }
686
687 } // namespace ceras
688
689 #endif // DJDWJBHNDAYTNOXLFQBDGSAQAAYPWXJGEBYIRKEAKAQUUWVGDUUGDKSDXUKSPCYNTWTDNII
690

```

9.3 /home/feng/workspace/github.repo/ceras/include/ceras.hpp File Reference

```

#include "../config.hpp"
#include "../includes.hpp"
#include "../activation.hpp"
#include "../ceras.hpp"
#include "../loss.hpp"
#include "../operation.hpp"
#include "../complex_operator.hpp"
#include "../optimizer.hpp"
#include "../place_holder.hpp"
#include "../session.hpp"
#include "../tensor.hpp"
#include "../variable.hpp"
#include "../constant.hpp"
#include "../layer.hpp"
#include "../model.hpp"
#include "../dataset.hpp"

```

9.4 ceras.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef AOXWQXCBBVJUJJSJDTMLRNOHUNBOARCNCRGTAHMIKSULFJOIUGCEEEXXVRUVWTOSKPYWDJOKQ
2 #define AOXWQXCBBVJUJJSJDTMLRNOHUNBOARCNCRGTAHMIKSULFJOIUGCEEEXXVRUVWTOSKPYWDJOKQ
3

```

```

4 #include "../config.hpp"
5 #include "../includes.hpp"
6 #include "../activation.hpp"
7 #include "../ceras.hpp"
8 #include "../config.hpp"
9 #include "../loss.hpp"
10 #include "../operation.hpp"
11 #include "../complex_operator.hpp"
12 #include "../optimizer.hpp"
13 #include "../place_holder.hpp"
14 #include "../session.hpp"
15 #include "../tensor.hpp"
16 #include "../variable.hpp"
17 #include "../constant.hpp"
18 #include "../layer.hpp"
19 #include "../model.hpp"
20 #include "../dataset.hpp"
21
22 #endif //AOXWQXCBBVJUJSDTMLRNOHUNBOARCNCRGBTAHMIKSULFJOIUGCEEGXXVRUVWTOSKPYWDJOKQ
23

```

9.5 /home/feng/workspace/github.repo/ceras/include/complex_↔ operator.hpp File Reference

```
#include "../operation.hpp"
```

Classes

- struct [ceras::complex< Real_Ex, Imag_Ex >](#)
- struct [ceras::is_complex< T >](#)
- struct [ceras::is_complex< complex< Real_Ex, Imag_Ex > >](#)

Namespaces

- namespace [ceras](#)

Concepts

- concept [ceras::Complex](#)
A type that represents a complex expression.

Functions

- template<Expression Real_Ex, Expression Imag_Ex>
Real_Ex [ceras::real](#) (complex< Real_Ex, Imag_Ex > const &c) noexcept
- template<Expression Real_Ex, Expression Imag_Ex>
Imag_Ex [ceras::imag](#) (complex< Real_Ex, Imag_Ex > const &c) noexcept
- template<Complex C>
auto [ceras::abs](#) (C const &c) noexcept
Returns the magnitude of the complex expression.
- template<Complex C>
auto [ceras::norm](#) (C const &c) noexcept
Returns the squared magnitude of the complex expression.

- template<Complex C>
auto `ceras::conj` (C const &c) noexcept
Returns the conjugate of the complex expression.
- template<Expression Em, Expression Ep>
auto `ceras::polar` (Em const &em, Ep const &ep) noexcept
Returns with given magnitude and phase angle.
- template<Complex C>
auto `ceras::arg` (C const &c) noexcept
Calculates the phase angle (in radians) of the complex expression.
- template<Complex C>
auto `ceras::operator+` (C const &c) noexcept
Returns the complex expression.
- template<Complex C>
auto `ceras::operator-` (C const &c) noexcept
Negatives the complex expression.
- template<Complex Cl, Complex Cr>
auto `ceras::operator+` (Cl const &cl, Cr const &cr) noexcept
Sums up two complex expressions.
- template<Complex Cl, Complex Cr>
auto `ceras::operator-` (Cl const &cl, Cr const &cr) noexcept
Subtracts one complex expression from the other one.
- template<Complex Cl, Complex Cr>
auto `ceras::operator*` (Cl const &cl, Cr const &cr) noexcept
Multiplies two complex expressions. Optimization here: $(a+ib)(c+id) = (ac-bd) + i(ad+bc) = (ac-bd) + i((a+b)*(c+d) - ac-bd)$*
- template<Complex C, Expression E>
auto `ceras::operator+` (C const &c, E const &e) noexcept
Sums up a complex expression and an expression.
- template<Complex C, Expression E>
auto `ceras::operator+` (E const &e, C const &c) noexcept
Sums up a complex expression and an expression.
- template<Complex C, Expression E>
auto `ceras::operator-` (C const &c, E const &e) noexcept
Subtracts an expression from a complex expression.
- template<Complex C, Expression E>
auto `ceras::operator-` (E const &e, C const &c) noexcept
Subtracts a complex expression from an expression.
- template<Complex C, Expression E>
auto `ceras::operator*` (C const &c, E const &e) noexcept
Multiplies a complex expression with an expression.
- template<Complex C, Expression E>
auto `ceras::operator*` (E const &e, C const &c) noexcept
Multiplies an expression with a complex expression.

Variables

- template<typename T >
constexpr bool `ceras::is_complex_v` = `is_complex<T>::value`

9.6 complex_operator.hpp

[Go to the documentation of this file.](#)

```

1  #ifndef SJTXPMDBEYHNPQSGKFIEKTKOYWOFMOEGAHNOHJVHJIAWTBHCCFUKCHLJMJAFFRHRXEOTYEDC
2  #define SJTXPMDBEYHNPQSGKFIEKTKOYWOFMOEGAHNOHJVHJIAWTBHCCFUKCHLJMJAFFRHRXEOTYEDC
3
4  #include "../operation.hpp"
5
6  namespace ceras
7  {
8
9      template< Expression Real_Ex, Expression Imag_Ex >
10     struct complex
11     {
12         Real_Ex real_;
13         Imag_Ex imag_;
14     }; //struct complex
15
16
17     template< typename T >
18     struct is_complex : std::false_type {};
19
20     template< Expression Real_Ex, Expression Imag_Ex >
21     struct is_complex<complex<Real_Ex, Imag_Ex> : std::true_type {};
22
23     template< typename T >
24     constexpr bool is_complex_v = is_complex<T>::value;
25
26     template< typename T >
27     concept Complex = is_complex_v<T>;
28
29
30     template< Expression Real_Ex, Expression Imag_Ex >
31     Real_Ex real( complex<Real_Ex, Imag_Ex> const& c ) noexcept
32     {
33         return c.real_;
34     }
35
36     template< Expression Real_Ex, Expression Imag_Ex >
37     Imag_Ex imag( complex<Real_Ex, Imag_Ex> const& c ) noexcept
38     {
39         return c.imag_;
40     }
41
42     template< Complex C >
43     auto abs( C const& c ) noexcept
44     {
45         return hypot( real(c), imag(c) );
46     }
47
48     template< Complex C >
49     auto norm( C const& c ) noexcept
50     {
51         auto const& r = real( c );
52         auto const& i = imag( c );
53         return hadamard_product( r, r ) + hadamard_product( i, i );
54     }
55
56     template< Complex C >
57     auto conj( C const& c ) noexcept
58     {
59         return complex{ real(c), -imag(c) };
60     }
61
62     template< Expression Em, Expression Ep >
63     auto polar( Em const& em, Ep const& ep ) noexcept
64     {
65         return complex{ hadamard_product( em, cos(ep) ), hadamard_product( em, sin(ep) ) };
66     }
67
68     template< Complex C >
69     auto arg( C const& c ) noexcept
70     {
71         return atan2( imag(c), real(c) );
72     }
73
74     template< Complex C >
75     auto operator + ( C const& c ) noexcept

```

```

155     {
156         return c;
157     }
158
159     template< Complex C >
160     auto operator - ( C const& c ) noexcept
161     {
162         return complex{ negative(real(c)), negative(imag(c)) };
163     }
164
165     template< Complex Cl, Complex Cr >
166     auto operator + ( Cl const& cl, Cr const& cr ) noexcept
167     {
168         return complex{ real(cl)+real(cr), imag(cl)+imag(cr) };
169     }
170
171     template< Complex Cl, Complex Cr >
172     auto operator - ( Cl const& cl, Cr const& cr ) noexcept
173     {
174         return complex{ real(cl)-real(cr), imag(cl)-imag(cr) };
175     }
176
177     template< Complex Cl, Complex Cr >
178     auto operator * ( Cl const& cl, Cr const& cr ) noexcept
179     {
180         auto const& a = real(cl);
181         auto const& b = imag(cl);
182         auto const& c = real(cr);
183         auto const& d = imag(cr);
184         auto const& ac = a * c;           // 1st multiplication
185         auto const& bd = b * d;           // 2nd multiplication
186         auto const& a_b = a + b;
187         auto const& c_d = c + d;
188         auto const& abcd = a_b * c_d;     // 3rd multiplication
189
190         return complex{ ac-bd, abcd-ac-bd };
191     }
192
193     template< Complex C, Expression E >
194     auto operator + ( C const& c, E const& e ) noexcept
195     {
196         return complex{ real(c)+e, imag(c) };
197     }
198
199     template< Complex C, Expression E >
200     auto operator + ( E const& e, C const& c ) noexcept
201     {
202         return c + e;
203     }
204
205     template< Complex C, Expression E >
206     auto operator - ( C const& c, E const& e ) noexcept
207     {
208         return complex{ real(c)-e, imag(c) };
209     }
210
211     template< Complex C, Expression E >
212     auto operator - ( E const& e, C const& c ) noexcept
213     {
214         return c + e;
215     }
216
217     template< Complex C, Expression E >
218     auto operator * ( C const& c, E const& e ) noexcept
219     {
220         return complex{ real(c)*e, imag(c)*e };
221     }
222
223     template< Complex C, Expression E >
224     auto operator * ( E const& e, C const& c ) noexcept
225     {
226         return c * e;
227     }
228
229 } // namespace ceras
230
231 #endif//SJTXPMBEYHNPQSGKFIEKTKOYWOFMOEGAHNOHJVHJIAWTBHCCFUKCHLJMJAFFRHRXEOTYEDC
232

```

9.7 /home/feng/workspace/github.repo/ceras/include/config.hpp File Reference

9.8 config.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef FBXYAXRPGSNHIXESHOGNYHPVEWWVSRSRJLQPIRIFENBGNMGFLJNMWXDNQLHKOAGBNYGBJRLBD
2 #define FBXYAXRPGSNHIXESHOGNYHPVEWWVSRSRJLQPIRIFENBGNMGFLJNMWXDNQLHKOAGBNYGBJRLBD
3
4 namespace ceras
5 {
6     inline constexpr unsigned long version = 20211219UL;
7     inline constexpr unsigned long __version__ = version;
8
24 #ifdef _MSC_VER
25     inline constexpr unsigned long is_windows_platform = 1;
26 #else
27     inline constexpr unsigned long is_windows_platform = 0;
28 #endif
29
30 #ifdef NDEBUG
31     inline constexpr unsigned long debug_mode = 0;
32 #else
33     inline constexpr unsigned long debug_mode = 1;
34 #endif
35
36 #ifdef CBLAS
37     inline constexpr unsigned long cblas_mode = 1;
38 #else
39     inline constexpr unsigned long cblas_mode = 0;
40 #endif
41
42 #ifndef NOPARALLEL
43     inline constexpr unsigned long parallel_mode = 1;
44 #else
45     inline constexpr unsigned long parallel_mode = 0;
46 #endif
47
48 #ifdef CUDA
49     inline constexpr unsigned long cuda_mode = 1;
50 #else
51     inline constexpr unsigned long cuda_mode = 0;
52 #endif
53
54     inline int visible_device = 0; // using GPU 0 by default
55     inline unsigned long cuda_gemm_threshold = 0UL; // will be updated if in CUDA mode, always assume
        float multiplications as double is rarely used
56
57     inline constexpr double eps = 1.0e-8;
58     inline constexpr double epsilon = eps; // alias of `eps`
59
77     inline int learning_phase = 1;
78 }
79
80 #endif//FBXYAXRPGSNHIXESHOGNYHPVEWWVSRSRJLQPIRIFENBGNMGFLJNMWXDNQLHKOAGBNYGBJRLBD
81

```

9.9 /home/feng/workspace/github.repo/ceras/include/constant.hpp File Reference

```

#include "../includes.hpp"
#include "../tensor.hpp"
#include "../utils/id.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/enable_shared.hpp"

```


Classes

- struct `ceras::constant< Tsor >`
Creates a constant expression from a tensor-like object.
- struct `ceras::is_constant< T >`
- struct `ceras::is_constant< constant< Tsor > >`

Namespaces

- namespace `ceras`

Concepts

- concept `ceras::Constant`

Variables

- template<class T >
constexpr bool `ceras::is_constant_v` = `is_constant<T>::value`

9.10 constant.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef CONSTANT_HPP_INCLUDED_DLKJASLKJFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
2 #define CONSTANT_HPP_INCLUDED_DLKJASLKJFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
3
4 #include "../includes.hpp"
5 #include "../tensor.hpp"
6 #include "../utils/id.hpp"
7 #include "../utils/better_assert.hpp"
8 #include "../utils/enable_shared.hpp"
9
10 namespace ceras
11 {
12
13     template< Tensor Tsor >
14     struct constant : enable_id<constant<Tsor>, "Constant">
15     {
16         typedef Tsor tensor_type;
17         tensor_type data_;
18
19         constant( tensor_type const& data ) : enable_id<constant<tensor_type>, "Constant">{ {},
20         data_{data} {}
21
22         void backward( auto )const {}
23
24         tensor_type forward()const
25     {
26         return data_;
27     }
28
29     auto shape()const
30     {
31         return data_.shape();
32     }
33 };
34
35 template< typename T >
36 struct is_constant : std::false_type {};
37
38 template< Tensor Tsor >
39 struct is_constant< constant< Tsor > > : std::true_type {};
40
41 template< class T >
42 inline constexpr bool is_constant_v = is_constant<T>::value;
43
44 template< typename T >
45 concept Constant = is_constant_v<T>;
46
47 } // namespace ceras
48
49 #endif //CONSTANT_HPP_INCLUDED_DLKJASLKJFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
50

```

9.11 /home/feng/workspace/github.repo/ceras/include/dataset.hpp File Reference

```
#include "../tensor.hpp"
#include "../includes.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/for_each.hpp"
```

Namespaces

- namespace [ceras](#)
- namespace [ceras::dataset](#)
- namespace [ceras::dataset::mnist](#)
- namespace [ceras::dataset::fashion_mnist](#)

Functions

- auto [ceras::dataset::mnist::load_data](#) (std::string const &path=std::string{"/dataset/mnist"})
- auto [ceras::dataset::fashion_mnist::load_data](#) (std::string const &path=std::string{"/dataset/fashion_mnist"})

9.12 dataset.hpp

[Go to the documentation of this file.](#)

```
1 #ifndef RKQSLRMXHSPFGGPQCNEPEBAKCXHNXQPMXETNTTXBWEWBIQHVCFRKRFSFMLXXXRYFUKHEXYIGL
2 #define RKQSLRMXHSPFGGPQCNEPEBAKCXHNXQPMXETNTTXBWEWBIQHVCFRKRFSFMLXXXRYFUKHEXYIGL
3
4 #include "../tensor.hpp"
5 #include "../includes.hpp"
6 #include "../utils/better_assert.hpp"
7 #include "../utils/for_each.hpp"
8
9 namespace ceras::dataset
10 {
11
12     namespace mnist
13     {
14         inline auto load_data( std::string const& path = std::string{"/dataset/mnist"} )
15         {
16             std::string const training_image_path = path + std::string{"/train-images-idx3-ubyte"};
17             std::string const training_label_path = path + std::string{"/train-labels-idx1-ubyte"};
18             std::string const test_image_path = path + std::string{"/t10k-images-idx3-ubyte"};
19             std::string const test_label_path = path + std::string{"/t10k-labels-idx1-ubyte"};
20
21             auto const& load_binary = []( std::string const& filename )
22             {
23                 std::ifstream ifs( filename, std::ios::binary );
24                 better_assert( ifs.good(), "Failed to load data from ", filename );
25                 std::vector<char> buff( ( std::istreambuf_iterator<char>( ifs ) ), (
26                     std::istreambuf_iterator<char>( ) ) );
27                 std::vector<std::uint8_t> ans( buff.size() );
28                 std::copy( buff.begin(), buff.end(), reinterpret_cast<char*>( ans.data() ) );
29                 return ans;
30             };
31
32             auto const& extract_image = []( std::vector<std::uint8_t> const& image_data )
33             {
34                 unsigned long const offset = 16;
35                 unsigned long const samples = (image_data.size()-offset) / (28*28);
36                 tensor<std::uint8_t> ans{ {samples, 28, 28} };
37                 std::copy( image_data.begin()+offset, image_data.end(), ans.data() );
38                 return ans;
39             };
40
41             auto const& extract_label = []( std::vector<std::uint8_t> const& label_data )
```

```

54     {
55         unsigned long const offset = 8;
56         unsigned long const samples = label_data.size() - offset;
57         auto ans = zeros<std::uint8_t>({samples, 10});
58         auto ans_2d = matrix_view{ ans.data(), samples, 10 };
59         for ( auto idx : range( samples ) )
60             ans_2d[idx][label_data[idx+offset]] = 1;
61         return ans;
62     };
63
64     return std::make_tuple( extract_image(load_binary(training_image_path)),
65                             extract_label(load_binary(training_label_path)),
66                             extract_image(load_binary(test_image_path)),
67                             extract_label(load_binary(test_label_path)) );
68 }
69 }
70
71 namespace fashion_mnist
72 {
73     inline auto load_data( std::string const& path = std::string("./dataset/fashion_mnist") )
74     {
75         std::string const training_image_path = path + std::string("/train-images-idx3-ubyte");
76         std::string const training_label_path = path + std::string("/train-labels-idx1-ubyte");
77         std::string const test_image_path = path + std::string("/t10k-images-idx3-ubyte");
78         std::string const test_label_path = path + std::string("/t10k-labels-idx1-ubyte");
79
80         auto const& load_binary = []( std::string const& filename )
81         {
82             std::ifstream ifs( filename, std::ios::binary );
83             better_assert( ifs.good(), "Failed to load data from ", filename );
84             std::vector<char> buff( ( std::istreambuf_iterator<char>( ifs ) ), (
85 std::istreambuf_iterator<char>() ) );
86             std::vector<std::uint8_t> ans( buff.size() );
87             std::copy( buff.begin(), buff.end(), reinterpret_cast<char*>( ans.data() ) );
88             return ans;
89         };
90
91         auto const& extract_image = []( std::vector<std::uint8_t> const& image_data )
92         {
93             unsigned long const offset = 16;
94             unsigned long const samples = (image_data.size()-offset) / (28*28);
95             tensor<std::uint8_t> ans{ {samples, 28, 28} };
96             std::copy( image_data.begin()+offset, image_data.end(), ans.data() );
97             return ans;
98         };
99
100        auto const& extract_label = []( std::vector<std::uint8_t> const& label_data )
101        {
102            unsigned long const offset = 8;
103            unsigned long const samples = label_data.size() - offset;
104            auto ans = zeros<std::uint8_t>({samples, 10});
105            auto ans_2d = matrix_view{ ans.data(), samples, 10 };
106            for ( auto idx : range( samples ) )
107                ans_2d[idx][label_data[idx+offset]] = 1;
108            return ans;
109        };
110
111        return std::make_tuple( extract_image(load_binary(training_image_path)),
112                                extract_label(load_binary(training_label_path)),
113                                extract_image(load_binary(test_image_path)),
114                                extract_label(load_binary(test_label_path)) );
115    }
116 }
117
118 //load_data
119
120 //fashion_mnist
121
122
123
124 #if 0
125 namespace cifar10
126 {
127     inline auto load_data( std::string const& path = std::string{} )
128     {
129     }
130 }
131
132 namespace cifar100
133 {
134     inline auto load_data( std::string const& path = std::string{} )
135     {
136     }
137 }
138
139 namespace imdb
140 {
141     inline auto load_data( std::string const& path = std::string{} )
142     {
143     }
144 }

```

```

165
166     namespace Reuters
167     {
168         inline auto load_data( std::string const& path = std::string{} )
169         {
170         }
171     }
172
173     namespace boston_housing
174     {
175         inline auto load_data( std::string const& path = std::string{} )
176         {
177         }
178     }
179 #endif
180
181
182 } // namespace ceras
183
184 #endif //RKQSLRMXHSFPGPQCNEPEBAKCXHNXQPMXETNTTXBWEWBIQHVCFRKRFSFMLXXXRYFUKHEXYIGL
185

```

9.13 /home/feng/workspace/github.repo/ceras/include/includes.hpp File Reference

```

#include " ./config.hpp"
#include <algorithm>
#include <any>
#include <array>
#include <cassert>
#include <chrono>
#include <climits>
#include <cmath>
#include <compare>
#include <concepts>
#include <cstddef>
#include <cstdint>
#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <exception>
#include <filesystem>
#include <fstream>
#include <functional>
#include <initializer_list>
#include <iomanip>
#include <ios>
#include <iostream>
#include <istream>
#include <iterator>
#include <limits>
#include <map>
#include <memory>
#include <numeric>
#include <optional>
#include <ostream>
#include <random>
#include <ranges>
#include <regex>
#include <set>
#include <sstream>
#include <string>

```

```
#include <thread>
#include <tuple>
#include <type_traits>
#include <unordered_map>
#include <unordered_set>
#include <utility>
#include <vector>
#include "../utils/3rd_party/stb_image.h"
#include "../utils/3rd_party/stb_image_write.h"
#include "../utils/3rd_party/stb_image_resize.h"
#include "../utils/3rd_party/glob.hpp"
```

Macros

- `#define STB_IMAGE_IMPLEMENTATION`
- `#define STB_IMAGE_WRITE_IMPLEMENTATION`
- `#define STB_IMAGE_RESIZE_IMPLEMENTATION`

9.13.1 Macro Definition Documentation

9.13.1.1 STB_IMAGE_IMPLEMENTATION

```
#define STB_IMAGE_IMPLEMENTATION
```

9.13.1.2 STB_IMAGE_RESIZE_IMPLEMENTATION

```
#define STB_IMAGE_RESIZE_IMPLEMENTATION
```

9.13.1.3 STB_IMAGE_WRITE_IMPLEMENTATION

```
#define STB_IMAGE_WRITE_IMPLEMENTATION
```

9.14 includes.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef JIDOKQFMGRFETGICWFQXOAUHHPAYLQWXAGIBPFTAXKFLOWQMWEAWUOKJJBUNTLLMIHYSFJQOYA
2 #define JIDOKQFMGRFETGICWFQXOAUHHPAYLQWXAGIBPFTAXKFLOWQMWEAWUOKJJBUNTLLMIHYSFJQOYA
3
4 #include "../config.hpp"
5
6 #include <algorithm>
7 #include <any>
8 #include <array>
9 #include <cassert>
10 #include <chrono>
11 #include <climits>
12 #include <cmath>
13 #include <compare>
14 #include <concepts>
15 #include <cstdint>
16 #include <cstdlib>
17 #include <csdlib>
18 #include <cstdid>
19 #include <ctime>
20 #include <exception>
21 #include <filesystem>
22 #include <fstream>
23 #include <functional>
24 #include <initializer_list>
25 #include <iomanip>
26 #include <ios>
27 #include <iostream>
28 #include <istream>
29 #include <iterator>
30 #include <limits>
31 #include <map>
32 #include <memory>
33 #include <numeric>
34 #include <optional>
35 #include <ostream>
36 #include <random>
37 #include <ranges>
38 #include <regex>
39 #include <set>
40 #include <sstream>
41 #include <string>
42 #include <thread>
43 #include <tuple>
44 #include <type_traits>
45 #include <unordered_map>
46 #include <unordered_set>
47 #include <utility>
48 #include <vector>
49
50 //
51 // begin of 3rd party libraries
52 //
53
54 #define STB_IMAGE_IMPLEMENTATION
55 #include "../utils/3rd_party/stb_image.h"
56 #define STB_IMAGE_WRITE_IMPLEMENTATION
57 #include "../utils/3rd_party/stb_image_write.h"
58 #define STB_IMAGE_RESIZE_IMPLEMENTATION
59 #include "../utils/3rd_party/stb_image_resize.h"
60
61 #include "../utils/3rd_party/glob.hpp"
62
63 //
64 // end of 3rd party libraries
65 //
66
67 #endif//JIDOKQFMGRFETGICWFQXOAUHHPAYLQWXAGIBPFTAXKFLOWQMWEAWUOKJJBUNTLLMIHYSFJQOYA
68

```

9.15 /home/feng/workspace/github.repo/ceras/include/layer.hpp File Reference

```

#include "../operation.hpp"
#include "../activation.hpp"

```

```
#include "../loss.hpp"
#include "../optimizer.hpp"
#include "../utils/better_assert.hpp"
```

Namespaces

- namespace [ceras](#)

Functions

- auto [ceras::Input](#) (std::vector< unsigned long > const &input_shape={{-1UL}})
- auto [ceras::Conv2D](#) (unsigned long output_channels, std::vector< unsigned long > const &kernel_size, std::vector< unsigned long > const &input_shape, std::string const &padding="valid", std::vector< unsigned long > const &strides={1, 1}, std::vector< unsigned long > const &dilations={1, 1}, bool use_bias=true, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f) noexcept
2D convolution layer.
- auto [ceras::Conv2D](#) (unsigned long output_channels, std::vector< unsigned long > const &kernel_size, std::string const &padding="valid", std::vector< unsigned long > const &strides={1, 1}, std::vector< unsigned long > const &dilations={1, 1}, bool use_bias=true, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f) noexcept
2D convolution layer.
- auto [ceras::Dense](#) (unsigned long output_size, bool use_bias=true, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f)
Densely-connected layer.
- auto [ceras::BatchNormalization](#) (float threshold=0.95f, float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f)
Applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.
- auto [ceras::Concatenate](#) (unsigned long axis=-1) noexcept
- auto [ceras::Add](#) () noexcept
- auto [ceras::Subtract](#) () noexcept
- auto [ceras::Multiply](#) () noexcept
- template<Expression Ex>
auto [ceras::ReLU](#) (Ex const &ex) noexcept
- auto [ceras::Softmax](#) () noexcept
- template<typename T = float>
auto [ceras::LeakyReLU](#) (T const factor=0.2) noexcept
- template<typename T = float>
auto [ceras::ELU](#) (T const factor=0.2) noexcept
- auto [ceras::Reshape](#) (std::vector< unsigned long > const &new_shape, bool include_batch_flag=true) noexcept
- auto [ceras::Flatten](#) () noexcept
- auto [ceras::MaxPooling2D](#) (unsigned long stride) noexcept
- auto [ceras::UpSampling2D](#) (unsigned long stride) noexcept
- template<typename T >
auto [ceras::Dropout](#) (T factor) noexcept
- auto [ceras::AveragePooling2D](#) (unsigned long stride) noexcept

9.16 layer.hpp

[Go to the documentation of this file.](#)

```

1  #ifndef NLESIGQPSASUTOXPLGXCUHFGGUGYSWLQQFATNISJOSPUFHRORXBNXLSWTYRNSIWJKYFXIQXVN
2  #define NLESIGQPSASUTOXPLGXCUHFGGUGYSWLQQFATNISJOSPUFHRORXBNXLSWTYRNSIWJKYFXIQXVN
3
4  #include "../operation.hpp"
5  #include "../activation.hpp"
6  #include "../loss.hpp"
7  #include "../optimizer.hpp"
8  #include "../utils/better_assert.hpp"
9
10 // try to mimic classes defined in tensorflow.keras
11
12 namespace ceras
13 {
14
15     inline auto Input( std::vector<unsigned long> const& input_shape = {{1UL}} )
16     {
17         return place_holder<tensor<float>{ input_shape };
18     }
19
20     [[deprecated("input_shape is not required in the new Conv2D(), this interface will be removed.")]]
21     inline auto Conv2D( unsigned long output_channels, std::vector<unsigned long> const& kernel_size,
22         std::vector<unsigned long> const& input_shape, std::string const&
23         padding="valid",
24         std::vector<unsigned long> const& strides={1,1}, std::vector<unsigned long>
25         const& dilations={1, 1}, bool use_bias=true,
26         float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float
27         bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f
28     ) noexcept
29     {
30         better_assert( output_channels > 0, "Expecting output_channels larger than 0." );
31         better_assert( kernel_size.size() > 0, "Expecting kernel_size at least has 1 elements." );
32         better_assert( input_shape.size() ==3, "Expecting input_shape has 3 elements." );
33         better_assert( strides.size() > 0, "Expecting strides at least has 1 elements." );
34         return [=]<Expression Ex>( Ex const& ex ) noexcept
35         {
36             unsigned long const kernel_size_x = kernel_size[0];
37             unsigned long const kernel_size_y = kernel_size.size() == 2 ? kernel_size[1] :
38             kernel_size[0];
39             //unsigned long const kernel_size_y = kernel_size[1];
40             unsigned long const input_channels = input_shape[2];
41             unsigned long const input_x = input_shape[0];
42             unsigned long const input_y = input_shape[1];
43             unsigned long const stride_x = strides[0];
44             unsigned long const stride_y = strides.size() == 2 ? strides[1] : strides[0];
45             unsigned long const dilation_row = dilations[0];
46             unsigned long const dilation_col = dilations.size() == 2 ? dilations[1] : dilations[0];
47             //unsigned long const stride_y = strides[1];
48             auto w = variable<tensor<float>{ glorot_uniform<float>({output_channels, kernel_size_x,
49             kernel_size_y, input_channels}), kernel_regularizer_l1, kernel_regularizer_l2 };
50             auto b = variable<tensor<float>{ zeros<float>({1, 1, output_channels}), bias_regularizer_l1,
51             bias_regularizer_l2, use_bias };
52             return conv2d( input_x, input_y, stride_x, stride_y, dilation_row, dilation_col, padding ) (
53             ex, w ) + b;
54         };
55     }
56
57     inline auto Conv2D( unsigned long output_channels, std::vector<unsigned long> const& kernel_size,
58         std::string const& padding="valid",
59         std::vector<unsigned long> const& strides={1,1}, std::vector<unsigned long>
60         const& dilations={1, 1}, bool use_bias=true,
61         float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float
62         bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f
63     ) noexcept
64     {
65         better_assert( output_channels > 0, "Expecting output_channels larger than 0." );
66         better_assert( kernel_size.size() > 0, "Expecting kernel_size at least has 1 elements." );
67         better_assert( strides.size() > 0, "Expecting strides at least has 1 elements." );
68         return [=]<Expression Ex>( Ex const& ex ) noexcept
69         {
70             unsigned long const kernel_size_x = kernel_size[0];
71             unsigned long const kernel_size_y = kernel_size.size() == 2 ? kernel_size[1] :
72             kernel_size[0];
73             //unsigned long const input_channels = input_shape[2];
74             unsigned long const input_channels = *(ex.shape().rbegin());
75             unsigned long const stride_x = strides[0];
76             unsigned long const stride_y = strides.size() == 2 ? strides[1] : strides[0];
77             unsigned long const dilation_row = dilations[0];
78             unsigned long const dilation_col = dilations.size() == 2 ? dilations[1] : dilations[0];
79             auto w = variable<tensor<float>{ glorot_uniform<float>({output_channels, kernel_size_x,
80             kernel_size_y, input_channels}), kernel_regularizer_l1, kernel_regularizer_l2 };

```



```

118         auto b = variable<tensor<float>>{ zeros<float>({1, 1, output_channels}), bias_regularizer_l1,
bias_regularizer_l2, use_bias };
119         return general_conv2d( stride_x, stride_y, dilation_row, dilation_col, padding )( ex, w ) +
b;
120     };
121 }
122
123 #if 0
124 [[deprecated("input_size is not required in the new Dense()")]]
125 inline auto Dense( unsigned long output_size, unsigned long input_size, bool use_bias=true, float
kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float
bias_regularizer_l2=0.0f )
126 {
127     return [=]<Expression Ex>( Ex const& ex )
128     {
129         auto w = variable<tensor<float>>{ glorot_uniform<float>({input_size, output_size}),
kernel_regularizer_l1, kernel_regularizer_l2 };
130         auto b = variable<tensor<float>>{ zeros<float>({1, output_size}), bias_regularizer_l1,
bias_regularizer_l2, use_bias }; // if use_bias, then b is trainable; otherwise, non-trainable.
131         return ex * w + b;
132     };
133 }
134 #endif
135 inline auto Dense( unsigned long output_size, bool use_bias=true, float kernel_regularizer_l1=0.0f,
float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f )
136 {
137     return [=]<Expression Ex>( Ex const& ex )
138     {
139         unsigned long const input_size = *(ex.shape().rbegin());
140         auto w = variable<tensor<float>>{ glorot_uniform<float>({input_size, output_size}),
kernel_regularizer_l1, kernel_regularizer_l2 };
141         auto b = variable<tensor<float>>{ zeros<float>({1, output_size}), bias_regularizer_l1,
bias_regularizer_l2, use_bias }; // if use_bias, then b is trainable; otherwise, non-trainable.
142         return ex * w + b;
143     };
144 }
145
146 #if 0
147 inline auto BatchNormalization( std::vector<unsigned long> const& shape, float threshold = 0.95f,
float kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f,
float bias_regularizer_l2=0.0f )
148 {
149     return [=]<Expression Ex>( Ex const& ex )
150     {
151         unsigned long const last_dim = *(shape.rbegin());
152         auto gamma = variable{ ones<float>({last_dim, } ), kernel_regularizer_l1,
kernel_regularizer_l2 };
153         auto beta = variable{ zeros<float>({last_dim, } ), bias_regularizer_l1, bias_regularizer_l2
};
154         return batch_normalization( threshold )( ex, gamma, beta );
155     };
156 }
157 #endif
158 inline auto BatchNormalization( float threshold, std::vector<unsigned long> const& shape, float
kernel_regularizer_l1=0.0f, float kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float
bias_regularizer_l2=0.0f )
159 {
160     return BatchNormalization( shape, threshold, kernel_regularizer_l1, kernel_regularizer_l2,
bias_regularizer_l1, bias_regularizer_l2 );
161 }
162 #endif
163 inline auto BatchNormalization( float threshold = 0.95f, float kernel_regularizer_l1=0.0f, float
kernel_regularizer_l2=0.0f, float bias_regularizer_l1=0.0f, float bias_regularizer_l2=0.0f )
164 {
165     return [=]<Expression Ex>( Ex const& ex )
166     {
167         unsigned long const last_dim = *(ex.shape().rbegin());
168         auto gamma = variable{ ones<float>({last_dim, } ), kernel_regularizer_l1,
kernel_regularizer_l2 };
169         auto beta = variable{ zeros<float>({last_dim, } ), bias_regularizer_l1, bias_regularizer_l2
};
170         return batch_normalization( threshold )( ex, gamma, beta );
171     };
172 }
173
174 #if 0
175 // TODO: fix this layer
176 inline auto LayerNormalization( std::vector<unsigned long> const& shape )
177 {
178     return [=]<Expression Ex>( Ex const& ex )
179     {
180         unsigned long const last_dim = *(shape.rbegin());
181         auto gamma = variable<tensor<float>>{ ones<float>({last_dim, } ) };
182         auto beta = variable<tensor<float>>{ zeros<float>({last_dim, } ) };
183         return layer_normalization()( ex, gamma, beta );
184     };
185 }
186 #endif

```

```

252 #endif
253
265 inline auto Concatenate(unsigned long axis = -1) noexcept
266 {
267     return [=]<Expression Lhs_Expression, Expression Rhs_Expression>( Lhs_Expression const& lhs_ex,
Rhs_Expression const& rhs_ex ) noexcept
268     {
269         return concatenate( axis )( lhs_ex, rhs_ex );
270     };
271 }
272
285 inline auto Add() noexcept
286 {
287     return [=]<Expression Lhs_Expression, Expression Rhs_Expression>( Lhs_Expression const& lhs_ex,
Rhs_Expression const& rhs_ex ) noexcept
288     {
289         return lhs_ex + rhs_ex;
290     };
291 }
292
293
306 inline auto Subtract() noexcept
307 {
308     return [=]<Expression Lhs_Expression, Expression Rhs_Expression>( Lhs_Expression const& lhs_ex,
Rhs_Expression const& rhs_ex ) noexcept
309     {
310         return lhs_ex - rhs_ex;
311     };
312 }
313
326 inline auto Multiply() noexcept
327 {
328     return [=]<Expression Lhs_Expression, Expression Rhs_Expression>( Lhs_Expression const& lhs_ex,
Rhs_Expression const& rhs_ex ) noexcept
329     {
330         return hadamard_product( lhs_ex, rhs_ex );
331     };
332 }
333
337 template< Expression Ex >
338 inline auto ReLU( Ex const& ex ) noexcept
339 {
340     return relu( ex );
341 }
342
346 inline auto Softmax() noexcept
347 {
348     return [=< Expression Ex >( Ex const& ex ) noexcept
349     {
350         return softmax( ex );
351     };
352 }
353
354
358 template< typename T = float >
359 inline auto LeakyReLU( T const factor=0.2 ) noexcept
360 {
361     return leaky_relu( factor );
362 }
363
367 template< typename T = float >
368 inline auto ELU( T const factor=0.2 ) noexcept
369 {
370     return elu( factor );
371 }
372
373
377 inline auto Reshape( std::vector<unsigned long> const& new_shape, bool include_batch_flag=true )
noexcept
378 {
379     return reshape( new_shape, include_batch_flag );
380 }
381
385 inline auto Flatten() noexcept
386 {
387     return [=<Expression Ex>( Ex const& ex ) noexcept
388     {
389         return flatten( ex );
390     };
391 }
392
396 inline auto MaxPooling2D( unsigned long stride ) noexcept
397 {
398     return max_pooling_2d( stride );
399 }
400
404 inline auto UpSampling2D( unsigned long stride ) noexcept

```

```

405     {
406         return up_sampling_2d( stride );
407     }
408
409     template< typename T >
410     inline auto Dropout( T factor ) noexcept
411     {
412         return drop_out( factor );
413     }
414
415     inline auto AveragePooling2D( unsigned long stride ) noexcept
416     {
417         return average_pooling_2d( stride );
418     }
419
420     //
421     // TODO: wrap more operations from 'operation.hpp'
422     //
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437 } // namespace f
438
439 #endif // NLESIGQPSASUTOXPLGXCUHFGGUGYSWLQQFATNISJOSPUFHRORXBNXLSWTYRNSIWJKYFXIQXVN
440

```

9.17 /home/feng/workspace/github.repo/ceras/include/loss.hpp File Reference

```

#include "../operation.hpp"
#include "../tensor.hpp"
#include "../utils/debug.hpp"

```

Namespaces

- namespace [ceras](#)

Functions

- template<Expression Lhs_Expression, Expression Rh_Expression>
constexpr auto [ceras::mean_squared_logarithmic_error](#) (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept
- template<Expression Lhs_Expression, Expression Rh_Expression>
constexpr auto [ceras::squared_loss](#) (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept
- template<Expression Lhs_Expression, Expression Rh_Expression>
constexpr auto [ceras::mean_squared_error](#) (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept
- template<Expression Lhs_Expression, Expression Rh_Expression>
constexpr auto [ceras::mse](#) (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept
- template<Expression Lhs_Expression, Expression Rh_Expression>
constexpr auto [ceras::abs_loss](#) (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept
- template<Expression Lhs_Expression, Expression Rh_Expression>
constexpr auto [ceras::mean_absolute_error](#) (Lhs_Expression const &lhs_ex, Rh_Expression const &rhs_ex) noexcept

- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::mae (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::cross_entropy (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::binary_cross_entropy_loss (Lhs_Expression const &ground_truth, Rhs_Expression const &prediction) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point F = float>`
`constexpr auto ceras::cross_entropy_loss (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex, F label_smoothing_factor=0.0) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::hinge_loss (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`

Variables

- static constexpr auto [ceras::MeanSquaredError](#)
Computes the mean of squares of errors between labels and predictions.
- static constexpr auto [ceras::MSE](#)
An alias name of function [MeanSquaredError](#).
- static constexpr auto [ceras::MeanAbsoluteError](#)
Computes the mean of absolute errors between labels and predictions.
- static constexpr auto [ceras::MAE](#)
An alias name of function [MeanAbsoluteError](#).
- static constexpr auto [ceras::Hinge](#)
- static constexpr auto [ceras::CategoricalCrossentropy](#)
- static constexpr auto [ceras::CategoricalCrossEntropy](#)
- static constexpr auto [ceras::BinaryCrossentropy](#)
- static constexpr auto [ceras::BinaryCrossEntropy](#)

9.18 loss.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef APWVIJWMXHAVXUGYGVNDSEFKTMBKLBMLSHWUPRPLGFCHUBDRAHGSTDSEDNKOGTIBNQVNLXCD
2 #define APWVIJWMXHAVXUGYGVNDSEFKTMBKLBMLSHWUPRPLGFCHUBDRAHGSTDSEDNKOGTIBNQVNLXCD
3
4 #include "../operation.hpp"
5 #include "../tensor.hpp"
6 #include "../utils/debug.hpp"
7
8 namespace ceras
9 {
10
11     template < Expression Lhs_Expression, Expression Rhs_Expression >
12     auto constexpr mean_squared_logarithmic_error( Lhs_Expression const& lhs_ex, Rhs_Expression const&
13     rhs_ex ) noexcept
14     {
15         return sum_reduce( square( minus(log( value{1.0} + clip(eps)(lhs_ex) ), log( value{1.0} +
16         clip(eps)(rhs_ex) ))) );
17     }
18
19     template < Expression Lhs_Expression, Expression Rhs_Expression >
20     auto constexpr squared_loss( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
21     {
22         return sum_reduce( square( minus(lhs_ex, rhs_ex)) );
23     }
24
25     template < Expression Lhs_Expression, Expression Rhs_Expression >
26     auto constexpr mean_squared_error( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex )
27     noexcept
28     {
29         return mean_reduce( square( minus(lhs_ex, rhs_ex) ) );
30     }
31
32 }
```

```

27     }
28
29     template < Expression Lhs_Expression, Expression Rhs_Expression >
30     auto constexpr mse( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
31     {
32         return mean_squared_error( lhs_ex, rhs_ex );
33     }
34
35     template < Expression Lhs_Expression, Expression Rhs_Expression >
36     auto constexpr abs_loss( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
37     {
38         return sum_reduce( abs( minus(lhs_ex, rhs_ex)) );
39     }
40
41     template < Expression Lhs_Expression, Expression Rhs_Expression >
42     auto constexpr mean_absolute_error( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex )
43     noexcept
44     {
45         return mean_reduce( abs( minus(lhs_ex, rhs_ex)) );
46     };
47
48     template < Expression Lhs_Expression, Expression Rhs_Expression >
49     auto constexpr mae( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
50     {
51         return mean_absolute_error( lhs_ex, rhs_ex );
52     };
53
54     template < Expression Lhs_Expression, Expression Rhs_Expression >
55     auto constexpr cross_entropy( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
56     {
57         return negative( sum_reduce( hadamard_product( lhs_ex, log(rhs_ex) ) ) );
58     }
59
60     namespace
61     {
62         struct cross_entropy_loss_context
63         {
64             template< std::floating_point T >
65             auto make_forward( T label_smoothing_factor ) const noexcept
66             {
67                 return [label_smoothing_factor]<Tensor Tsor>( Tsor const& ground_truth_input, Tsor const&
68                 prediction_input ) noexcept
69                 {
70                     Tsor sm = softmax( prediction_input );
71                     typedef typename Tsor::value_type value_type;
72                     typename Tsor::value_type ans{0};
73                     unsigned long const n = *(ground_truth_input.shape().rbegin());
74                     value_type const _c0 = label_smoothing_factor / (n-1);
75                     value_type const _c1 = value_type{1} - label_smoothing_factor;
76
77                     for ( auto idx : range( ground_truth_input.size() ) )
78                     {
79                         value_type const v = ground_truth_input[idx] > eps ? _c1 : _c0;
80                         ans -= v * std::log( std::max( static_cast<typename Tsor::value_type>(eps),
81                         sm[idx] ) );
82                     }
83                     //ans -= ground_truth_input[idx] * std::log( std::max( static_cast<typename
84                     Tsor::value_type>(eps), sm[idx] ) );
85                 }
86                 auto result = as_tensor<typename Tsor::value_type, typename
87                 Tsor::allocator>(ans/(*(ground_truth_input.shape().begin())));
88                 return result;
89             };
90         };
91
92         template< std::floating_point T >
93         auto make_backward( T label_smoothing_factor) const noexcept
94         {
95             return [=]<Tensor Tsor>( Tsor const& ground_truth_input, Tsor const& prediction_input,
96             [[maybe_unused]]Tsor const& output_data, [[maybe_unused]]Tsor const& grad ) noexcept
97             {
98                 // In our implementation, the grad is always 1, unless this layer is nested
99                 // contributing to a combined weighted loss
100                 typedef typename Tsor::value_type value_type;
101                 value_type const factor = grad[0]; // the shape of grad is {1,}
102
103                 unsigned long const n = *(ground_truth_input.shape().rbegin());
104                 value_type const _c0 = label_smoothing_factor / (n-1);
105                 value_type const _c1 = value_type{1} - label_smoothing_factor;
106
107                 Tsor ground_truth_gradient = ground_truth_input;
108
109                 //Tsor sm = softmax( prediction_input ) - ground_truth_input;
110                 //return std::make_tuple( ground_truth_gradient*factor, sm*factor );
111
112                 Tsor sm = softmax( prediction_input );
113                 for ( auto idx : range( ground_truth_input.size() ) )
114                 {

```

```

107         value_type const v = ground_truth_gradient[idx] > eps ? _c1 : _c0;
108         sm[idx] = factor * (sm[idx] - v );
109     }
110
111     return std::make_tuple( ground_truth_gradient*factor, sm );
112 };
113 }
114
115 }; //struct cross_entropy_loss_context
116
117 } //anonymous namespace
118
119 template < Expression Lhs_Expression, Expression Rhs_Expression >
120 auto constexpr binary_cross_entropy_loss( Lhs_Expression const& ground_truth, Rhs_Expression const&
prediction ) noexcept
121 {
122     auto ones = ones_like( ground_truth );
123     auto error = negative( hadamard_product( ground_truth, log(prediction) ) + hadamard_product(
(ones - ground_truth), log(ones - prediction) ) );
124     return mean_reduce( error );
125 }
126
127
128 // beware: do not apply softmax activation before this layer, as this loss is softmax+xentropy
already
129 template < Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point F=float >
130 auto constexpr cross_entropy_loss( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex, F
label_smoothing_factor=0.0 ) noexcept
131 {
132     return make_binary_operator( cross_entropy_loss_context{}.make_forward( label_smoothing_factor
), cross_entropy_loss_context{}.make_backward( label_smoothing_factor ), "CrossEntropyLoss" )( lhs_ex,
rhs_ex );
133 }
134
135 template < Expression Lhs_Expression, Expression Rhs_Expression >
136 auto constexpr hinge_loss( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
137 {
138     return mean_reduce( maximum( value{0.0f}, value{1.0f} - hadamard_product( lhs_ex, rhs_ex ) ) );
139 }
140
141 // loss interfaces
142 // A loss is an expression. This expression takes two parameters.
143 // The first parameter is a place_holder, that will be binded to an tensor.
144 // The second parameter is an expression, that will be evaluated to compare with the tensor binded
to the first parameter
145
146 inline static constexpr auto MeanSquaredError = [] ()
147 {
148     return [] <Expression Ex >( Ex const& output )
149     {
150         return [] <Place_Holder Ph>( Ph const& ground_truth )
151         {
152             return mean_squared_error( ground_truth, output );
153         };
154     };
155 };
156
157 inline static constexpr auto MSE = [] ()
158 {
159     return MeanSquaredError();
160 };
161
162 inline static constexpr auto MeanAbsoluteError = [] ()
163 {
164     return [] <Expression Ex >( Ex const& output )
165     {
166         return [] <Place_Holder Ph>( Ph const& ground_truth )
167         {
168             return mean_absolute_error( ground_truth, output );
169         };
170     };
171 };
172
173 inline static constexpr auto MAE = [] ()
174 {
175     return MeanAbsoluteError();
176 };
177
178 inline static constexpr auto Hinge = [] ()
179 {
180     return [] <Expression Ex >( Ex const& output )
181     {
182         return [] <Place_Holder Ph>( Ph const& ground_truth )
183         {

```

```

219         return hinge_loss( ground_truth, output );
220     };
221 };
222 };
223
224 // note: do not apply softmax activation to the last layer of the model, this loss has packaged it
225 inline static constexpr auto CategoricalCrossentropy = []<std::floating_point F=float>( F
label_smoothing_factor = 0.0)
226 {
227     return [=]<Expression Ex >( Ex const& output )
228     {
229         return [=]<Place_Holder Ph>( Ph const& ground_truth )
230         {
231             return cross_entropy_loss( ground_truth, output, label_smoothing_factor );
232         };
233     };
234 };
235
236 inline static constexpr auto CategoricalCrossEntropy = []<std::floating_point F=float>(F
label_smoothing_factor = 0.0)
237 {
238     return CategoricalCrossentropy(label_smoothing_factor);
239 };
240
241 inline static constexpr auto BinaryCrossentropy = []()
242 {
243     return [=]<Expression Ex >( Ex const& output )
244     {
245         return [=]<Place_Holder Ph>( Ph const& ground_truth )
246         {
247             return binary_cross_entropy_loss( ground_truth, output );
248         };
249     };
250 };
251
252 inline static constexpr auto BinaryCrossEntropy = []()
253 {
254     return BinaryCrossentropy();
255 };
256
257
258 } // namespace ceras
259
260 #endif // APWVIJWMXHAVXUGYGVNDSEFKTMBKLBMLSHWUPRPGLFCHUBDRAHGSTDSEDNKOGTIBNQVNLXCD
261

```

9.19 /home/feng/workspace/github.repo/ceras/include/metric.hpp File Reference

```

#include "../operation.hpp"
#include "../activation.hpp"
#include "../loss.hpp"

```

Namespaces

- namespace [ceras](#)

Functions

- `template<Expression Lhs_Expression, Expression Rhc_Expression, std::floating_point FP>`
`auto ceras::binary_accuracy (Lhs_Expression const &lhs_ex, Rhc_Expression const &rhc_ex, FP`
`threshold=0.5) noexcept`

9.20 metric.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef EIDQGGJMELOKMYJNSKIUQTTOGNSMKBVITQSMTSVJDNWXPJBJKQBJDAIAIYMTPTIBLWPBCYLI
2 #define EIDQGGJMELOKMYJNSKIUQTTOGNSMKBVITQSMTSVJDNWXPJBJKQBJDAIAIYMTPTIBLWPBCYLI
3
4 #include "../operation.hpp"
5 #include "../activation.hpp"
6 #include "../loss.hpp"
7
8 namespace ceras
9 {
10
11     template< Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point FP >
12     auto binary_accuracy( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex, FP threshold=0.5 )
13     noexcept
14     {
15         return mean( equal( lhs_ex, rhs_ex ) );
16     }
17
18 }
19
20
21 } // namespace ceras
22
23 #endif // EIDQGGJMELOKMYJNSKIUQTTOGNSMKBVITQSMTSVJDNWXPJBJKQBJDAIAIYMTPTIBLWPBCYLI
24
25

```

9.21 /home/feng/workspace/github.repo/ceras/include/model.hpp File Reference

```

#include "../includes.hpp"
#include "../operation.hpp"
#include "../place_holder.hpp"
#include "../tensor.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/context_cast.hpp"
#include "../utils/tqdm.hpp"

```

Classes

- struct [ceras::compiled_model](#)< [Model](#), [Optimizer](#), [Loss](#) >
- struct [ceras::model](#)< [Ex](#), [Ph](#) >

Namespaces

- namespace [ceras](#)

Functions

- template<Expression Ex>
void [ceras::make_trainable](#) (Ex &ex, bool t)
- template<Expression Ex, Place_Holder Ph, Expression Ey>
auto [ceras::replace_placeholder_with_expression](#) (Ex const &ex, Ph const &old_place_holder, Ey const &new_expression)
- template<typename Model, typename Optimizer, typename Loss >
auto [ceras::make_compiled_model](#) (Model const &m, Loss const &l, Optimizer const &o)

9.22 model.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef BPLYFMIFNNWSGMLLEKBJMAJDBRSPHHRAYMOHTWSTCMNMFSLLYNQTTCCAQXKXSLMSLKESHASL
2 #define BPLYFMIFNNWSGMLLEKBJMAJDBRSPHHRAYMOHTWSTCMNMFSLLYNQTTCCAQXKXSLMSLKESHASL
3
4 #include "../includes.hpp"
5 #include "../operation.hpp"
6 #include "../place_holder.hpp"
7 #include "../tensor.hpp"
8 #include "../utils/better_assert.hpp"
9 #include "../utils/context_cast.hpp"
10 #include "../utils/tqdm.hpp"
11
12 namespace ceras
13 {
14     template< Expression Ex >
15     void make_trainable( Ex& ex , bool t )
16     {
17         if constexpr (is_variable_v<Ex>)
18         {
19             ex.trainable( t );
20         }
21         else if constexpr (is_binary_operator_v<Ex>)
22         {
23             make_trainable( ex.lhs_op_, t );
24             make_trainable( ex.rhs_op_, t );
25         }
26         else if constexpr (is_unary_operator_v<Ex>)
27         {
28             make_trainable( ex.op_, t );
29         }
30     }
31
32     template< Expression Ex, Place_Holder Ph, Expression Ey >
33     auto replace_placeholder_with_expression( Ex const& ex, Ph const& old_place_holder, Ey const&
34     new_expression )
35     {
36         if constexpr (is_value_v<Ex> || is_constant_v<Ex> || is_variable_v<Ex>)
37         {
38             return ex;
39         }
40         else if constexpr (is_place_holder_v<Ex>)
41         {
42             return new_expression; // assuming only one place holder in the model
43             //return (ex == old_place_holder) ? new_expression : ex;
44         }
45         else if constexpr (is_unary_operator_v<Ex>)
46         {
47             return make_unary_operator( ex.forward_action_, ex.backward_action_, ex.name_,
48             ex.output_shape_calculator_ )( replace_placeholder_with_expression( ex.op_, old_place_holder,
49             new_expression ) );
50         }
51         else if constexpr (is_binary_operator_v<Ex>)
52         {
53             return make_binary_operator( ex.forward_action_, ex.backward_action_, ex.name_,
54             ex.output_shape_calculator_ )( replace_placeholder_with_expression( ex.lhs_op_,
55             old_place_holder, new_expression ),
56             replace_placeholder_with_expression( ex.rhs_op_,
57             old_place_holder, new_expression ) );
58         }
59         else
60         {
61             better_assert( false, "replace::Should never reach here!" );
62         }
63     }
64
65     template< typename Model, typename Optimizer, typename Loss >
66     struct compiled_model
67     {
68         typedef typename Model::input_layer_type io_layer_type;
69         typedef decltype( std::declval<Optimizer>() (std::declval<Loss&>()) ) optimizer_type; // defined
70         because the compiled optimizer takes a reference to an expression as its parameter
71
72         Model model_;
73         io_layer_type input_place_holder_;
74         io_layer_type ground_truth_place_holder_;
75         Loss loss_; // MeanSquaredError()( model_.output() )( find a input );
76         Optimizer optimizer_; // Adam( ... )
77         optimizer_type compiled_optimizer_;
78
79         compiled_model( Model const& m, io_layer_type const& input_place_holder, io_layer_type const&
80         ground_truth_place_holder, Loss const& loss, Optimizer const& optimizer ) :

```

```

85         model_{m}, input_place_holder_{input_place_holder},
ground_truth_place_holder_{ground_truth_place_holder}, loss_{loss}, optimizer_{optimizer},
compiled_optimizer_{ optimizer_{loss_} } { }

86
94     template< Tensor Tsor >
95     auto evaluate( Tsor const& inputs, Tsor const& outputs, unsigned long batch_size=32 )
96     {
97         // extract size of samples
98         unsigned long const samples = *(inputs.shape().begin());
99         unsigned long const loops = samples / batch_size;
100
101         // prepare tensor for inputs
102         std::vector<unsigned long> batch_input_shape = inputs.shape();
103         batch_input_shape[0] = batch_size;
104         Tsor input_samples{ batch_input_shape };
105         unsigned long const input_size_per_batch = input_samples.size();
106
107         // prepare tensor for outputs
108         std::vector<unsigned long> batch_output_shape = outputs.shape();
109         batch_output_shape[0] = batch_size;
110         Tsor output_samples{ batch_output_shape };
111         unsigned long const output_size_per_batch = output_samples.size();
112
113         // bind tensors to place holders
114         //session<Tsor> s;
115         auto& s = get_default_session<Tsor>();
116         s.bind( input_place_holder_, input_samples );
117         s.bind( ground_truth_place_holder_, output_samples );
118
119         typedef typename Tsor::value_type value_type;
120         value_type validation_error = 0;
121
122         learning_phase = 0; // for different behaviours in normalization and drop-out layers
123
124         for ( auto l : tq::trange( loops ) )
125         {
126             // feed data
127             std::copy_n( inputs.data() + l * input_size_per_batch, input_size_per_batch,
input_samples.data() );
128             std::copy_n( outputs.data() + l * output_size_per_batch, output_size_per_batch,
output_samples.data() );
129             // forward pass
130             auto error = s.run( loss_ ).as_scalar();
131             // in case of training split, do backpropagation
132             validation_error += error;
133         }
134
135         learning_phase = 1; // for different behaviours in normalization and drop-out layers
136
137         return validation_error / loops;
138     }
139
163     template< Tensor Tsor >
164     auto fit( Tsor const& inputs, Tsor const& outputs, unsigned long batch_size, unsigned long
epoch=1, int verbose=0, double validation_split=0.0 )
165     {
166         // extract size of samples
167         unsigned long const samples = *(inputs.shape().begin());
168         unsigned long const loops_per_epoch = samples / batch_size;
169         unsigned long const training_loops = ( 1.0 - validation_split ) * loops_per_epoch;
170         unsigned long const validation_loops = loops_per_epoch - training_loops;
171
172         // prepare tensor for inputs
173         std::vector<unsigned long> batch_input_shape = inputs.shape();
174         batch_input_shape[0] = batch_size;
175         Tsor input_samples{ batch_input_shape };
176         unsigned long const input_size_per_batch = input_samples.size();
177
178         // prepare tensor for outputs
179         std::vector<unsigned long> batch_output_shape = outputs.shape();
180         batch_output_shape[0] = batch_size;
181         Tsor output_samples{ batch_output_shape };
182         unsigned long const output_size_per_batch = output_samples.size();
183
184         // bind tensors to place holders
185         //session<Tsor> s;
186         auto& s = get_default_session<Tsor>(); // .get();
187         s.bind( input_place_holder_, input_samples );
188         s.bind( ground_truth_place_holder_, output_samples );
189
190         // collect training errors
191         typedef typename Tsor::value_type value_type;
192         std::vector<value_type> training_errors;
193         std::vector<value_type> validation_errors;
194
195         learning_phase = 1; // for different behaviours in normalization and drop-out layers
196

```

```

197         for ( auto e : range( epoch ) )
198         {
199             value_type training_error = 0;
200             value_type validation_error = 0;
201             for ( auto l : tq::trange( loops_per_epoch ) )
202             {
203                 // feed data
204                 std::copy_n( inputs.data() + l * input_size_per_batch, input_size_per_batch,
input_samples.data() );
205                 std::copy_n( outputs.data() + l * output_size_per_batch, output_size_per_batch,
output_samples.data() );
206                 // forward pass
207                 auto error = s.run( loss_ ).as_scalar();
208                 // in case of training split, do backpropagation
209                 if ( l <= training_loops )
210                 {
211                     training_error += error;
212                     s.run( compiled_optimizer_ );
213                 }
214                 else // in case of validation split, just collect errors
215                 {
216                     validation_error += error;
217                 }
218             }
219             training_errors.push_back( training_error / training_loops );
220             validation_errors.push_back( validation_error / validation_loops );
221             if ( verbose )
222                 std::cout << "\nTraining error: " << training_error / training_loops << " and
validation error: " << validation_error / validation_loops << " at epoch: " << e+1 << "/" << epoch;
223             std::cout << std::endl;
224         }
225         return std::make_tuple( training_errors, validation_errors );
226     }
227
228     template< Tensor Tsor >
229     auto train_on_batch( Tsor const& input, Tsor const& output )
230     {
231         learning_phase = 1; // for different behaviours in normalization and drop-out layers
232         auto& s = get_default_session<Tsor>(); // .get();
233         s.bind( input_place_holder_, input );
234         s.bind( ground_truth_place_holder_, output );
235         //debug_log( "Training on batch forward pass." );
236         auto error = s.run( loss_ );
237         //debug_log( "Training on batch backward pass." );
238         s.run( compiled_optimizer_ );
239         return error.as_scalar();
240     }
241
242     template< Tensor Tsor>
243     auto predict( Tsor const& input_tensor )
244     {
245         auto m = model_;
246         return m.predict( input_tensor );
247     }
248
249     template< Expression Exp >
250     auto operator()( Exp const& ex ) const noexcept
251     {
252         return model_( ex );
253     }
254
255     void trainable( bool t )
256     {
257         model_.trainable( t );
258     }
259 };
260
261 template< typename Model, typename Optimizer, typename Loss >
262 inline auto make_compiled_model( Model const& m, Loss const& l, Optimizer const& o )
263 {
264     auto input_place_holder = m.input();
265     auto ground_truth_place_holder = typename Model::input_layer_type{};
266     auto loss = l( m.output() )( ground_truth_place_holder );
267     auto optimizer = o( loss );
268     return compiled_model{ m, input_place_holder, ground_truth_place_holder, loss, o };
269 }
270
271 template< Expression Ex, Place_Holder Ph >
272 struct model
273 {
274     typedef Ph         input_layer_type;
275     typedef Ex         output_layer_type;
276
277     output_layer_type expression_;
278     input_layer_type place_holder_; //< input layer of the model.
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304

```

```

308     input_layer_type input() const noexcept { return place_holder_; }
309
313     output_layer_type output() const noexcept { return expression_; }
314
327     model( input_layer_type const& place_holder, output_layer_type const& expression ) :
expression_(expression), place_holder_{place_holder} {}
328
345     template< Tensor Tsor>
346     auto predict( Tsor const& input_tensor )
347     {
348         learning_phase = 0; // for different behaviours in normalization and drop-out layers
349
350         //session<Tsor> s;
351         auto& s = get_default_session<Tsor>(); // .get();
352         s.bind( place_holder_, input_tensor );
353
354         auto ans = s.run( expression_ );
355
356         learning_phase = 1; // restore learning phase
357
358         return ans;
359     }
360
381     template< Expression Exp >
382     auto operator()( Exp const& ex ) const noexcept
383     {
384         return replace_placeholder_with_expression( expression_, place_holder_, ex );
385     }
386
401     template< typename Loss, typename Optimizer >
402     auto compile( Loss const& l, Optimizer const& o )
403     {
404         return make_compiled_model( *this, l, o );
405     }
406
407     void trainable( bool t )
408     {
409         make_trainable( expression_, t );
410     }
411
412
416     void save_weights( std::string const& file )
417     {
418         auto& s = get_default_session<tensor<float>>();
419         s.serialize( file );
420     }
421
425     void load_weights( std::string const& file )
426     {
427         auto& s = get_default_session<tensor<float>>();
428         s.deserialize( file );
429     }
430
431
436     void summary( std::string const& file_name=std::string{} ) const noexcept
437     {
438         auto g = computation_graph( expression_ );
439
440         if ( file_name.empty() )
441         {
442             std::cout << g << std::endl;
443             return;
444         }
445
446         std::ofstream ofs( file_name );
447         ofs << g;
448     }
449
450 };
451
452
453 } // namespace ceras
454
455 #endif // BPLYFMIFNNWSGMLLEKBJMAJDBRSPHHRAYMOHTWSTCMNMFSLLYNQTTCCAQXXSLMSLKESHRSALCalculate the loss for
the model in test model

```

9.23 /home/feng/workspace/github.repo/ceras/include/operation.hpp File Reference

```

#include "../includes.hpp"
#include "../place_holder.hpp"

```

```
#include "../variable.hpp"
#include "../constant.hpp"
#include "../value.hpp"
#include "../utils/range.hpp"
#include "../utils/debug.hpp"
#include "../config.hpp"
#include "../utils/context_cast.hpp"
#include "../utils/for_each.hpp"
#include "../utils/id.hpp"
#include "../utils/enable_shared.hpp"
#include "../utils/fmt.hpp"
```

Classes

- struct [ceras::identity_output_shape_calculator](#)
The default identity output shape calculator for unary/binary operators. Should be overridden for some special operators.
- struct [ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >](#)
A unary operator is composed of a.) an input expression, b.) a forward action and c.) a backward action.
- struct [ceras::binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >](#)
A binary operator is composed of a.) a left-side input expression, b.) a right-side input expression, c.) a forward action and d.) a backward action.
- struct [ceras::is_unary_operator< T >](#)
- struct [ceras::is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >](#)
- struct [ceras::is_binary_operator< T >](#)
- struct [ceras::is_binary_operator< binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >](#)

Namespaces

- namespace [ceras](#)

Concepts

- concept [ceras::Unary_Operator](#)
A type that represents an unary operator.
- concept [ceras::Binary_Operator](#)
A type that represents a binary operator.
- concept [ceras::Operator](#)
A type that represents an unary or a binary operator.
- concept [ceras::Expression](#)
A type that represents a unary operator, a binary operator, a variable, a place_holder, a constant or a value.

Functions

- `template<typename Forward_Action , typename Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>`
`constexpr auto ceras::make_unary_operator (Forward_Action const &unary_forward_action, Backward_`
`Action const &unary_backward_action, std::string const &name="Anonymous Unary Operator", Output_`
`Shape_Calculator const &output_shape_calculator=Output_Shape_Calculator{}) noexcept`
Construct an unary operator by passing the forward/backward actions and output shape calculator.
- `template<typename Forward_Action , typename Backward_Action , typename Output_Shape_Calculator = identity_output_shape_calculator>`
`auto ceras::make_binary_operator (Forward_Action const &binary_forward_action, Backward_Action const`
`&binary_backward_action, std::string const &name="Anonymous Binary Operator", Output_Shape_`
`Calculator const &output_shape_calculator=Output_Shape_Calculator{}) noexcept`
- `template<Expression Ex>`
`std::string ceras::computation_graph (Ex const &ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::plus (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::operator+ (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Ex>`
`constexpr auto ceras::operator+ (Ex const &ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`auto ceras::operator* (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Ex>`
`constexpr auto ceras::negative (Ex const &ex) noexcept`
Negative operator, elementwise.
- `template<Expression Ex>`
`constexpr auto ceras::operator- (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto ceras::inverse (Ex const &ex) noexcept`
Inverse operator, elementwise.
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::elementwise_product (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex)`
`noexcept`
Multiply two input operators, elementwise.
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::elementwise_multiply (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex)`
`noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::hadamard_product (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex)`
`noexcept`
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::divide (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
Divide one tensor by the other.
- `template<Expression Lhs_Expression, Expression Rhs_Expression>`
`constexpr auto ceras::operator/ (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
Divide one tensor by the other.
- `template<Expression Ex>`
`constexpr auto ceras::sum_reduce (Ex const &ex) noexcept`
Sum up all elements, returns a scalar.
- `template<Expression Ex>`
`constexpr auto ceras::reduce_sum (Ex const &ex) noexcept`
- `template<Expression Ex>`
`constexpr auto ceras::mean_reduce (Ex const &ex) noexcept`
Computes the mean of elements across all dimensions of an expression.

- template<Expression Ex>
constexpr auto [ceras::reduce_mean](#) (Ex const &ex) noexcept
An alias name of mean_reduce.
- template<Expression Ex>
constexpr auto [ceras::mean](#) (Ex const &ex) noexcept
An alias name of mean_reduce.
- template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto [ceras::minus](#) (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept
- template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto [ceras::operator-](#) (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept
- template<Expression Ex>
constexpr auto [ceras::square](#) (Ex const &ex) noexcept
- template<Expression Ex, Expression Ey>
*endcode **constexpr auto [hypot](#) (Ex const &ex, Ey const &ey) noexcept
- template<typename Float >
requires std::floating_point<Float>
constexpr auto [clip](#) (Float lower, Float upper=std::numeric_limits<Float>::max()) noexcept
- auto [reshape](#) (std::vector< unsigned long > const &new_shape, bool include_batch_flag=true) noexcept
- template<Expression Ex>
constexpr auto [flatten](#) (Ex const &ex) noexcept
Flatten input tensor.
- constexpr auto [expand_dims](#) (int axis=-1) noexcept
Expand input tensor with a length 1 axis inserted at index axis.
- auto [argmax](#) (unsigned long axis=0) noexcept
Returns the index with the largest value across axes of an input tensor.
- auto [argmin](#) (unsigned long axis=0) noexcept
Returns the index with the smallest value across axes of an input tensor.
- template<Expression Ex>
constexpr auto [identity](#) (Ex const &ex) noexcept
Identity operation.
- template<Expression Ex>
auto [transpose](#) (Ex const &ex) noexcept
Transpose a matrix.
- auto [img2col](#) (unsigned long const row_kernel, unsigned long col_kernel=-1, unsigned long const row_↵
padding=0, unsigned long col_padding=0, unsigned long const row_stride=1, unsigned long const col_↵
stride=1, unsigned long const row_dilation=1, unsigned long const col_dilation=1) noexcept
- auto [conv2d](#) (unsigned long row_input, unsigned long col_input, unsigned long const row_stride=1, unsigned
long const col_stride=1, unsigned long const row_dilation=1, unsigned long const col_dilation=1, std::string
const &padding="valid") noexcept
- auto [general_conv2d](#) (unsigned long const row_stride=1, unsigned long const col_stride=1, unsigned long
const row_dilation=1, unsigned long const col_dilation=1, std::string const &padding="valid") noexcept
Conv2D not constrained by the input shape.
- template<typename T >
requires std::floating_point<T>
auto [drop_out](#) (T const factor) noexcept
- template<typename T >
requires std::floating_point<T>
auto [dropout](#) (T const factor) noexcept
dropout is an alias name of drop_out.
- auto [max_pooling_2d](#) (unsigned long stride) noexcept
- auto [average_pooling_2d](#) (unsigned long stride) noexcept
- auto [up_sampling_2d](#) (unsigned long stride) noexcept
- auto [upsampling_2d](#) (unsigned long stride) noexcept

- `template<typename T = double>`
`requires std::floating_point<T>`
`auto normalization_batch (T const momentum=0.98) noexcept`
- `template<typename T >`
`requires std::floating_point<T>`
`auto batch_normalization (T const momentum=0.98) noexcept`
- `template<Expression Lhs_Expression, Expression Rhx_Expression>`
`constexpr auto concatenate (Lhs_Expression const &lhx_ex, Rhx_Expression const &rhx_ex) noexcept`
- `auto concatenate (unsigned long axe=-1)`
- `template<Expression Lhs_Expression, Expression Rhx_Expression>`
`constexpr auto concat (Lhs_Expression const &lhx_ex, Rhx_Expression const &rhx_ex) noexcept`
- `auto concat (unsigned long axe=-1)`
- `template<Expression Lhs_Expression, Expression Rhx_Expression>`
`constexpr auto maximum (Lhs_Expression const &lhx_ex, Rhx_Expression const &rhx_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhx_Expression>`
`constexpr auto minimum (Lhs_Expression const &lhx_ex, Rhx_Expression const &rhx_ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhx_Expression>`
`constexpr auto atan2 (Lhs_Expression const &lhx_ex, Rhx_Expression const &rhx_ex) noexcept`
Computes the arc tangent of y/x using the signs of arguments to determine the correct quadrant.
- `template<typename T = float>`
`requires std::floating_point<T>`
`auto random_normal_like (T mean=0.0, T stddev=1.0) noexcept`
- `template<Expression Ex>`
`auto ones_like (Ex const &ex) noexcept`
- `template<Expression Ex>`
`auto zeros_like (Ex const &ex) noexcept`
- `template<Expression Lhs_Expression, Expression Rhx_Expression, std::floating_point FP>`
`constexpr auto equal (Lhs_Expression const &lhx_ex, Rhx_Expression const &rhx_ex, FP threshold=0.5) noexcept`
- `template<Expression Ex>`
`constexpr auto sign (Ex const &ex) noexcept`
- `auto zero_padding_2d (std::vector< unsigned long > const &padding) noexcept`
Zero-padding layer for 2D input. The input should have 4-dimensions: (batch_size, row, col, channel). The output has 4-dimensions: (batch_size, new_row, new_col, channel).
- `auto cropping_2d (std::vector< unsigned long > const &padding) noexcept`
Cropping layer for 2D input. The input should have 4-dimensions: (batch_size, row, col, channel). The output has 4-dimensions: (batch_size, new_row, new_col, channel).
- `auto sliding_2d (unsigned long pixels) noexcept`
- `auto repeat (unsigned long repeats, unsigned long axis=-1) noexcept`
Repeats elements along an axis.
- `auto reduce_min (unsigned long axis=-1) noexcept`
Reduce minimal elements along an axis.
- `auto reduce_max (unsigned long axis=-1) noexcept`
Reduce maximum elements along an axis.
- `auto reduce_sum (unsigned long axis) noexcept`
Reduce sum elements along an axis.
- `template<Expression Ex>`
`constexpr auto abs (Ex const &ex) noexcept`
Computes Abs of the given expression.
- `template<Expression Ex>`
`constexpr auto acos (Ex const &ex) noexcept`
Computes Acos of the given expression.
- `template<Expression Ex>`
`constexpr auto acosh (Ex const &ex) noexcept`

Computes Acosh of the given expression.

- `template<Expression Ex>`
`constexpr auto asin (Ex const &ex) noexcept`

Computes Asin of the given expression.

- `template<Expression Ex>`
`constexpr auto asinh (Ex const &ex) noexcept`

Computes Asinh of the given expression.

- `template<Expression Ex>`
`constexpr auto atan (Ex const &ex) noexcept`

Computes Atan of the given expression.

- `template<Expression Ex>`
`constexpr auto atanh (Ex const &ex) noexcept`

Computes Atanh of the given expression.

- `template<Expression Ex>`
`constexpr auto cbrt (Ex const &ex) noexcept`

Computes Cbrt of the given expression.

- `template<Expression Ex>`
`constexpr auto ceil (Ex const &ex) noexcept`

Computes Ceil of the given expression.

- `template<Expression Ex>`
`constexpr auto cos (Ex const &ex) noexcept`

Computes Cos of the given expression.

- `template<Expression Ex>`
`constexpr auto cosh (Ex const &ex) noexcept`

Computes Cosh of the given expression.

- `template<Expression Ex>`
`constexpr auto erf (Ex const &ex) noexcept`

Computes Erf of the given expression.

- `template<Expression Ex>`
`constexpr auto erfc (Ex const &ex) noexcept`

Computes Erfc of the given expression.

- `template<Expression Ex>`
`constexpr auto exp (Ex const &ex) noexcept`

Computes Exp of the given expression.

- `template<Expression Ex>`
`constexpr auto exp2 (Ex const &ex) noexcept`

Computes Exp2 of the given expression.

- `template<Expression Ex>`
`constexpr auto expm1 (Ex const &ex) noexcept`

Computes Expm1 of the given expression.

- `template<Expression Ex>`
`constexpr auto fabs (Ex const &ex) noexcept`

Computes Fabs of the given expression.

- `template<Expression Ex>`
`constexpr auto floor (Ex const &ex) noexcept`

Computes Floor of the given expression.

- `template<Expression Ex>`
`constexpr auto llrint (Ex const &ex) noexcept`

Computes Llrint of the given expression.

- `template<Expression Ex>`
`constexpr auto llround (Ex const &ex) noexcept`

Computes Llround of the given expression.

- `template<Expression Ex>`
`constexpr auto log (Ex const &ex) noexcept`
Computes Log of the given expression.
- `template<Expression Ex>`
`constexpr auto log10 (Ex const &ex) noexcept`
Computes Log10 of the given expression.
- `template<Expression Ex>`
`constexpr auto log1p (Ex const &ex) noexcept`
Computes Log1p of the given expression.
- `template<Expression Ex>`
`constexpr auto log2 (Ex const &ex) noexcept`
Computes Log2 of the given expression.
- `template<Expression Ex>`
`constexpr auto lrint (Ex const &ex) noexcept`
Computes Lrint of the given expression.
- `template<Expression Ex>`
`constexpr auto lround (Ex const &ex) noexcept`
Computes Lround of the given expression.
- `template<Expression Ex>`
`constexpr auto nearbyint (Ex const &ex) noexcept`
Computes Nearbyint of the given expression.
- `template<Expression Ex>`
`constexpr auto rint (Ex const &ex) noexcept`
Computes Rint of the given expression.
- `template<Expression Ex>`
`constexpr auto round (Ex const &ex) noexcept`
Computes Round of the given expression.
- `template<Expression Ex>`
`constexpr auto sin (Ex const &ex) noexcept`
Computes Sin of the given expression.
- `template<Expression Ex>`
`constexpr auto sinh (Ex const &ex) noexcept`
Computes Sinh of the given expression.
- `template<Expression Ex>`
`constexpr auto sqrt (Ex const &ex) noexcept`
Computes Sqrt of the given expression.
- `template<Expression Ex>`
`constexpr auto tan (Ex const &ex) noexcept`
Computes Tan of the given expression.
- `template<Expression Ex>`
`constexpr auto tanh (Ex const &ex) noexcept`
Computes Tanh of the given expression.
- `template<Expression Ex>`
`constexpr auto trunc (Ex const &ex) noexcept`
Computes Trunc of the given expression.
- `template<Expression Lhs_Expression, Variable Rhs_Expression>`
`constexpr auto assign (Lhs_Expression const &lhs_ex, Rhs_Expression const &rhs_ex) noexcept`
- `template<Expression Ex>`
`auto poisson (Ex const &ex) noexcept`

Variables

- `template<class T >`
`constexpr bool ceras::is_unary_operator_v = is_unary_operator<T>::value`
- `template<class T >`
`constexpr bool ceras::is_binary_operator_v = is_binary_operator<T>::value`
- `*auto y = variable<tensor<float>>>{ }`
- `*auto sqr = hypot(x, y)`

9.23.1 Function Documentation

9.23.1.1 `abs()`

```
template<Expression Ex>
constexpr auto abs (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Abs of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = abs( a );
```

9.23.1.2 `acos()`

```
template<Expression Ex>
constexpr auto acos (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Acos of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = acos( a );
```

9.23.1.3 `acosh()`

```
template<Expression Ex>
constexpr auto acosh (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Acosh of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = acosh( a );
```

9.23.1.4 argmax()

```
auto argmax (
    unsigned long axis = 0 ) [inline], [noexcept]
```

Returns the index with the largest value across axes of an input tensor.

```
auto a = variable{ ... };
auto ma = argmax( 1 )( a );
```

9.23.1.5 argmin()

```
auto argmin (
    unsigned long axis = 0 ) [inline], [noexcept]
```

Returns the index with the smallest value across axes of an input tensor.

```
auto a = variable{ ... };
auto ma = argmin( 1 )( a );
```

9.23.1.6 asin()

```
template<Expression Ex>
constexpr auto asin (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Asin of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = asin( a );
```

9.23.1.7 asinh()

```
template<Expression Ex>
constexpr auto asinh (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Asinh of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = asinh( a );
```

9.23.1.8 assign()

```
template<Expression Lhs_Expression, Variable Rhs_Expression>
constexpr auto assign (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

@breif Updating the second expression's value by assining the first one to it. The second expression should be a 'variable'.

Parameters

<i>lhs_ex</i>	A mutable value.
<i>rhs_ex</i>	An expression to be assigned to lhs_ex. TODO: Fixme, this implementation is wrong

```

auto x = constant{ ... } * constant{ ... };
auto v = variable{ ... };
assgin( x, v );

```

9.23.1.9 atan()

```

template<Expression Ex>
constexpr auto atan (
    Ex const & ex ) [constexpr], [noexcept]

```

Computes Atan of the given expression.

Example code:

```

auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = atan( a );

```

9.23.1.10 atan2()

```

template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto atan2 (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]

```

Computes the arc tangent of y/x using the signs of arguments to determine the correct quadrant.

9.23.1.11 atanh()

```

template<Expression Ex>
constexpr auto atanh (
    Ex const & ex ) [constexpr], [noexcept]

```

Computes Atanh of the given expression.

Example code:

```

auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = atanh( a );

```

9.23.1.12 average_pooling_2d()

```

auto average_pooling_2d (
    unsigned long stride ) [inline], [noexcept]

```

9.23.1.13 batch_normalization()

```
template<typename T >
requires std::floating_point<T>
auto batch_normalization (
    T const momentum = 0.98 ) [inline], [noexcept]
```

9.23.1.14 cbrt()

```
template<Expression Ex>
constexpr auto cbrt (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Cbert of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = cbrt( a );
```

9.23.1.15 ceil()

```
template<Expression Ex>
constexpr auto ceil (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Ceil of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = ceil( a );
```

9.23.1.16 clip()

```
template<typename Float >
requires std::floating_point<Float>
constexpr auto clip (
    Float lower,
    Float upper = std::numeric_limits<Float>::max() ) [constexpr], [noexcept]
```

9.23.1.17 concat() [1/2]

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto concat (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

9.23.1.18 concat() [2/2]

```
auto concat (
    unsigned long axe = -1 ) [inline]
```

9.23.1.19 concatenate() [1/2]

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto concatenate (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

9.23.1.20 concatenate() [2/2]

```
auto concatenate (
    unsigned long axe = -1 ) [inline]
```

9.23.1.21 conv2d()

```
auto conv2d (
    unsigned long row_input,
    unsigned long col_input,
    unsigned long const row_stride = 1,
    unsigned long const col_stride = 1,
    unsigned long const row_dilation = 1,
    unsigned long const col_dilation = 1,
    std::string const & padding = "valid" ) [inline], [noexcept]
```

9.23.1.22 cos()

```
template<Expression Ex>
constexpr auto cos (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Cos of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = cos( a );
```

9.23.1.23 cosh()

```
template<Expression Ex>
constexpr auto cosh (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Cosh of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = cosh( a );
```

9.23.1.24 cropping_2d()

```
auto cropping_2d (
    std::vector< unsigned long > const & padding ) [inline], [noexcept]
```

Cropping layer for 2D input. The input should have 4-dimensions: (batch_size, row, col, channel). The output has 4-dimensions: (batch_size, new_row, new_col, channel).

Parameters

<i>padding</i>	If a single integer, then apply symmetric cropping to height and width. If two integers, then first is for height and the second is for width. If four integers, then is interpreted as<tt>(top_crop, bottom_crop, left_crop, right_crop).
----------------	--

Example code:

```
auto a = variable{ random<float>( {32, 32, 3} ) };
auto b = cropping_2d( {8,} )( a ); // shape for b is (32-8-8, 32-8-8, 3)
auto c = cropping_2d( {8, 4} )( a ); // shape for c is (32-8-8, 32-4-4, 3)
auto d = cropping_2d( {8, 4, 2, 1} )( a ); // shape for d is (32-8-4, 32-2-1, 3)
```

9.23.1.25 drop_out()

```
template<typename T >
requires std::floating_point<T>
auto drop_out (
    T const factor ) [inline], [noexcept]
```

9.23.1.26 dropout()

```
template<typename T >
requires std::floating_point<T>
auto dropout (
    T const factor ) [inline], [noexcept]
```

dropout is an alias name of drop_out.

9.23.1.27 equal()

```
template<Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point FP>
constexpr auto equal (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex,
    FP threshold = 0.5 ) [constexpr, [noexcept]
```

Returns the truth value of (lhs == rhs) element-wise. [+1 for true, 0 for false]

Parameters

<i>lhs_ex</i>	The first operator.
<i>rhs_ex</i>	The second operator.

Returns

An instance of a binary operator that evaluate the element-wise equality of two input operators.

Example code:

```
auto l = variable<tensor<float>>( /*...*/ );
auto r = place_holder<tensor<float>>{};
auto eq = equal(l, r);
```

9.23.1.28 erf()

```
template<Expression Ex>
constexpr auto erf (
    Ex const & ex ) [constexpr, [noexcept]
```

Computes Erf of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = erf( a );
```

9.23.1.29 erfc()

```
template<Expression Ex>
constexpr auto erfc (
    Ex const & ex ) [constexpr, [noexcept]
```

Computes Erfc of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = erfc( a );
```

9.23.1.30 exp()

```
template<Expression Ex>
constexpr auto exp (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Exp of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = exp( a );
```

9.23.1.31 exp2()

```
template<Expression Ex>
constexpr auto exp2 (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Exp2 of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = exp2( a );
```

9.23.1.32 expand_dims()

```
constexpr auto expand_dims (
    int axis = -1 ) [inline], [constexpr], [noexcept]
```

Expand input tensor with a length 1 axis inserted at index axis.

```
auto x = variable<float>{ {2, 3, 4} }
auto x0 = expand_dims(0)( x ); // new shape is ( 1, 2, 3, 4 )
auto x1 = expand_dims(1)( x ); // new shape is ( 2, 1, 3, 4 )
auto x2 = expand_dims(2)( x ); // new shape is ( 2, 3, 1, 4 )
auto x3 = expand_dims(-1)( x ); // new shape is ( 2, 3, 4, 1 )
```

9.23.1.33 expm1()

```
template<Expression Ex>
constexpr auto expm1 (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Expm1 of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = expm1( a );
```

9.23.1.34 fabs()

```
template<Expression Ex>
constexpr auto fabs (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Fabs of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = fabs( a );
```

9.23.1.35 flatten()

```
template<Expression Ex>
constexpr auto flatten (
    Ex const & ex ) [constexpr], [noexcept]
```

Flatten input tensor.

```
auto x = .....; // an operator returns tensor of shape ( 12, 34, 1 2 )
auto f = flatten( x ); // returns tensor of shape (12*34*1*2, )
```

9.23.1.36 floor()

```
template<Expression Ex>
constexpr auto floor (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Floor of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = floor( a );
```

9.23.1.37 general_conv2d()

```
auto general_conv2d (
    unsigned long const row_stride = 1,
    unsigned long const col_stride = 1,
    unsigned long const row_dilation = 1,
    unsigned long const col_dilation = 1,
    std::string const & padding = "valid" ) [inline], [noexcept]
```

Conv2D not constrained by the input shape.

9.23.1.38 hypot()

```
template<Expression Ex, Expression Ey>
*endcode **constexpr auto hypot (
    Ex const & ex,
    Ey const & ey ) [constexpr], [noexcept]
```

9.23.1.39 identity()

```
template<Expression Ex>
constexpr auto identity (
    Ex const & ex ) [constexpr], [noexcept]
```

Identity operation.

9.23.1.40 img2col()

```
auto img2col (
    unsigned long const row_kernel,
    unsigned long col_kernel = -1,
    unsigned long const row_padding = 0,
    unsigned long col_padding = 0,
    unsigned long const row_stride = 1,
    unsigned long const col_stride = 1,
    unsigned long const row_dilation = 1,
    unsigned long const col_dilation = 1 ) [inline], [noexcept]
```

9.23.1.41 llrint()

```
template<Expression Ex>
constexpr auto llrint (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Llrint of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = llrint( a );
```

9.23.1.42 llround()

```
template<Expression Ex>
constexpr auto llround (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Llround of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = llround( a );
```

9.23.1.43 log()

```
template<Expression Ex>
constexpr auto log (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Log of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = log( a );
```

9.23.1.44 log10()

```
template<Expression Ex>
constexpr auto log10 (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Log10 of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = log10( a );
```

9.23.1.45 log1p()

```
template<Expression Ex>
constexpr auto log1p (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Log1p of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = log1p( a );
```

9.23.1.46 log2()

```
template<Expression Ex>
constexpr auto log2 (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Log2 of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = log2( a );
```

9.23.1.47 lrint()

```
template<Expression Ex>
constexpr auto lrint (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Lrint of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = lrint( a );
```

9.23.1.48 lround()

```
template<Expression Ex>
constexpr auto lround (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Lround of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = lround( a );
```

9.23.1.49 max_pooling_2d()

```
auto max_pooling_2d (
    unsigned long stride ) [inline], [noexcept]
```

9.23.1.50 maximum()

```
template<Expression Lhs_Expression, Expression Rhx_Expression>
constexpr auto maximum (
    Lhs_Expression const & lhs_ex,
    Rhx_Expression const & rhs_ex ) [constexpr], [noexcept]
```

9.23.1.51 minimum()

```
template<Expression Lhs_Expression, Expression Rhs_Expression>
constexpr auto minimum (
    Lhs_Expression const & lhs_ex,
    Rhs_Expression const & rhs_ex ) [constexpr], [noexcept]
```

9.23.1.52 nearbyint()

```
template<Expression Ex>
constexpr auto nearbyint (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Nearbyint of the given expression.

Example code:

```
auto a = variable{ random<float>({2, 3, 5}) };
auto b = nearbyint( a );
```

9.23.1.53 normalization_batch()

```
template<typename T = double>
requires std::floating_point<T>
auto normalization_batch (
    T const momentum = 0.98 ) [inline], [noexcept]
```

9.23.1.54 ones_like()

```
template<Expression Ex>
auto ones_like (
    Ex const & ex ) [noexcept]
```

`ones_like` produces a tensor of the same shape as the input expression, but with every element to be 1.

Returns

An unary operator that takes an unary operator, and producing an output tensor Example Code:

```
auto va = variable{ ones<float>({3, 3, 3}) };
auto v_rand = ones_like( va ); // this expression will produces a tensor of shape (3, 3, 3), with every
element to be 1.
```

9.23.1.55 poisson()

```
template<Expression Ex>
auto poisson (
    Ex const & ex ) [noexcept]
```

`poisson` produces random tensor from a normal distribution

Returns

An unary operator that takes an unary operator, and producing output tensor subjects to a Poisson distribution. The shape of the output tensor has the same shape corresponding to the input unary operator.

Example Code

```
auto va = variable{ ones<float>({3, 3, 3}) };
auto v_rand = poisson( va ); // this expression will produces a tensor of shape (3, 3, 3) subjects to a
    Poisson distribution
```

9.23.1.56 random_normal_like()

```
template<typename T = float>
requires std::floating_point<T>
auto random_normal_like (
    T mean = 0.0,
    T stddev = 1.0 ) [inline], [noexcept]
```

`random_normal_like` produces random tensor from a normal distribution

Parameters

<i>mean</i>	Mean of the normal distribution, a scalar.
<i>stddev</i>	Standard deviation of the normal distribution, a scalar.

Returns

An unary operator that takes an unary operator, and producing output tensor from a normal distribution. The shape of the output tensor has the same shape corresponding to the input unary operator.

Example Code

```
auto va = variable{ ones<float>({3, 3, 3}) };
auto v_rand = random_normal_like( 1.0, 4.0 )( va ); // this expression will produces a tensor of shape (3,
    3, 3) from a normal distribution with parameters (1.0, 4.0)
```

9.23.1.57 reduce_max()

```
auto reduce_max (
    unsigned long axis = -1 ) [inline], [noexcept]
```

Reduce maximum elements along an axis.

Parameters

<i>axis</i>	The axis along which to reduce maximum values. Defaults to the last axis.
-------------	---

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = reduce_max( 0 )( a ); // <- output shape is ( 3, 5 )
auto b = reduce_max( 1 )( a ); // <- output shape is ( 2, 5 )
auto b = reduce_max( 2 )( a ); // <- output shape is ( 2, 3 )
auto b = reduce_max( ) ( a ); // <- output shape is ( 2, 3 )
```

9.23.1.58 reduce_min()

```
auto reduce_min (
    unsigned long axis = -1 ) [inline], [noexcept]
```

Reduce minimal elements along an axis.

Parameters

<i>axis</i>	The axis along which to reduce minimal values. Defaults to the last axis.
-------------	---

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = reduce_min( 0 )( a ); // <- output shape is ( 3, 5 )
auto b = reduce_min( 1 )( a ); // <- output shape is ( 2, 5 )
auto b = reduce_min( 2 )( a ); // <- output shape is ( 2, 3 )
auto b = reduce_min( ) ( a ); // <- output shape is ( 2, 3 )
```

9.23.1.59 reduce_sum()

```
auto reduce_sum (
    unsigned long axis ) [inline], [noexcept]
```

Reduce sum elements along an axis.

Parameters

<i>axis</i>	The axis along which to reduce sum.
-------------	-------------------------------------

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = reduce_sum( 0 )( a ); // <- output shape is ( 3, 5 )
auto b = reduce_sum( 1 )( a ); // <- output shape is ( 2, 5 )
auto b = reduce_sum( 2 )( a ); // <- output shape is ( 2, 3 )
auto b = reduce_sum( -1 )( a ); // <- output shape is ( 2, 3 )
```

9.23.1.60 repeat()

```
auto repeat (
    unsigned long repeats,
    unsigned long axis = -1 ) [inline], [noexcept]
```

Repeats elements along an axis.

Parameters

<i>repeats</i>	The number of repetitions for each element.
<i>axis</i>	The axis along which to repeat values. Defaults to the last axis.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b0 = repeat( 2, 0 )( a ); // <- output shape is ( 4, 3, 5 )
auto b1 = repeat( 2, 1 )( a ); // <- output shape is ( 2, 6, 5 )
auto b2 = repeat( 2, 2 )( a ); // <- output shape is ( 2, 3, 10 )
auto bx = repeat( 2 )( a ); // <- output shape is ( 2, 3, 10 )
```

9.23.1.61 reshape()

```
auto reshape (
    std::vector< unsigned long > const & new_shape,
    bool include_batch_flag = true ) [inline], [noexcept]
```

9.23.1.62 rint()

```
template<Expression Ex>
constexpr auto rint (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Rint of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = rint( a );
```

9.23.1.63 round()

```
template<Expression Ex>
constexpr auto round (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Round of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = round( a );
```

9.23.1.64 sign()

```
template<Expression Ex>
constexpr auto sign (
    Ex const & ex ) [constexpr], [noexcept]
```

Returns the sign. [1 for positive, 0 for 0 and -1 for negative]

Parameters

ex	The input operator.
-----------	---------------------

Returns

An instance of a unary_operator that evaluate the sign of the input operator.

Example code:

```
auto e = variable<tensor<float>>{ /*...*/ };
auto si = sign(e);
```

9.23.1.65 sin()

```
template<Expression Ex>
constexpr auto sin (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Sin of the given expression.

Example code:

```
auto a = variable{ random<float>({2, 3, 5}) };
auto b = sin( a );
```

9.23.1.66 sinh()

```
template<Expression Ex>
constexpr auto sinh (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Sinh of the given expression.

Example code:

```
auto a = variable{ random<float>({2, 3, 5}) };
auto b = sinh( a );
```

9.23.1.67 sliding_2d()

```
auto sliding_2d (
    unsigned long pixels )    [inline], [noexcept]
```

9.23.1.68 sqrt()

```
template<Expression Ex>
constexpr auto sqrt (
    Ex const & ex )    [constexpr], [noexcept]
```

Computes Sqrt of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = sqrt( a );
```

9.23.1.69 tan()

```
template<Expression Ex>
constexpr auto tan (
    Ex const & ex )    [constexpr], [noexcept]
```

Computes Tan of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = tan( a );
```

9.23.1.70 tanh()

```
template<Expression Ex>
constexpr auto tanh (
    Ex const & ex )    [constexpr], [noexcept]
```

Computes Tanh of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = tanh( a );
```

9.23.1.71 transpose()

```
template<Expression Ex>
auto transpose (
    Ex const & ex ) [noexcept]
```

Transpose a matrix.

9.23.1.72 trunc()

```
template<Expression Ex>
constexpr auto trunc (
    Ex const & ex ) [constexpr], [noexcept]
```

Computes Trunc of the given expression.

Example code:

```
auto a = variable{ random<float>( {2, 3, 5} ) };
auto b = trunc( a );
```

9.23.1.73 up_sampling_2d()

```
auto up_sampling_2d (
    unsigned long stride ) [inline], [noexcept]
```

9.23.1.74 upsampling_2d()

```
auto upsampling_2d (
    unsigned long stride ) [inline], [noexcept]
```

9.23.1.75 zero_padding_2d()

```
auto zero_padding_2d (
    std::vector< unsigned long > const & padding ) [inline], [noexcept]
```

Zero-padding layer for 2D input. The input should have 4-dimensions: (batch_size, row, col, channel). The output has 4-dimensions: (batch_size, new_row, new_col, channel).

Parameters

<i>padding</i>	If a single integer, then apply symmetric padding to height and width. If two integers, then first is for height and the second is for width. If four integers, then is interpreted as<tt>(top_pad, bottom_pad, left_pad, right_pad).
----------------	---

Example code:

```
auto a = variable{ random<float>({16, 16, 3}) };
auto b = zero_padding_2d( {8,} )( a ); // shape for b is (8+16+8, 8+16+8, 3)
auto c = zero_padding_2d( {8, 4} )( a ); // shape for c is (8+16+8, 4+16+4, 3)
auto d = zero_padding_2d( {8, 4, 2, 1} )( a ); // shape for d is (8+16+4, 2+16+1, 3)
```

9.23.1.76 zeros_like()

```
template<Expression Ex>
auto zeros_like (
    Ex const & ex ) [noexcept]
```

`zeros_like` produces a tensor of the same shape as the input expression, but with every element to be 0.

Returns

An unary operator that takes an unary operator, and producing an output tensor Example Code:

```
auto va = variable{ ones<float>({3, 3, 3}) };
auto v_rand = zeros_like( va ); // this expression will produces a tensor of shape (3, 3, 3), with
every element to be 0.
```

9.23.2 Variable Documentation

9.23.2.1 sqr

```
* auto sqr = hypot( x, y )
```

9.23.2.2 y

```
* auto y = variable<tensor<float>>{ }
```

9.24 operation.hpp

[Go to the documentation of this file.](#)

```
1 #ifndef IPKVWSJOCMGVVRASCBLPYHFBCHRIVEXYBOMMDAKFAUDFYVYOOOISLRXJNUJKPJJEVMLDPRDSNM
2 #define IPKVWSJOCMGVVRASCBLPYHFBCHRIVEXYBOMMDAKFAUDFYVYOOOISLRXJNUJKPJJEVMLDPRDSNM
3
4 #include "../includes.hpp"
5 #include "../place_holder.hpp"
6 #include "../variable.hpp"
7 #include "../constant.hpp"
8 #include "../value.hpp"
9 #include "../utils/range.hpp"
10 #include "../utils/debug.hpp"
11 #include "../config.hpp"
12 #include "../utils/context_cast.hpp"
13 #include "../utils/for_each.hpp"
14 #include "../utils/id.hpp"
15 #include "../utils/enable_shared.hpp"
16 #include "../utils/fmt.hpp"
17
```

```

18 namespace ceras
19 {
20
24     struct identity_output_shape_calculator
25     {
26         std::vector<unsigned long> operator()( std::vector<unsigned long> const& input_shape ) const
noexcept
27         {
28             return input_shape;
29         }
30
31         std::vector<unsigned long> operator()( std::vector<unsigned long> const& lhs_input_shape,
std::vector<unsigned long> const& rhs_input_shape ) const noexcept
32         {
33             return lhs_input_shape.size() > rhs_input_shape.size() ? lhs_input_shape : rhs_input_shape;
34         }
35
36         std::vector<unsigned long> operator()() const noexcept
37         {
38             return std::vector<unsigned long>{ {-1UL}, };
39         }
40     }; // struct identity_output_shape_calculator
41
42
46     template< typename Operator, typename Forward_Action, typename Backward_Action, typename
Output_Shape_Calculator = identity_output_shape_calculator >
47     struct unary_operator : enable_id<unary_operator<Operator, Forward_Action, Backward_Action>, "Unary
Operator">
48     {
49         Operator op_;
50         Forward_Action forward_action_;
51         Backward_Action backward_action_;
52         Output_Shape_Calculator output_shape_calculator_;
53
54         typedef decltype( std::declval<Forward_Action>() ( std::declval<decltype(op_)>().forward() ) )
tensor_type;
55
56         tensor_type input_data_;
57         tensor_type output_data_;
58
59         unary_operator( Operator const& op, Forward_Action const& forward_action, Backward_Action const&
backward_action, Output_Shape_Calculator const& output_shape_calculator ) noexcept :
60             op_{op}, forward_action_{ forward_action }, backward_action_{ backward_action },
output_shape_calculator_{ output_shape_calculator } { }
61
62         auto forward()
63         {
64             input_data_ = op_.forward();
65             output_data_ = forward_action_( input_data_ );
66             return output_data_;
67         }
68
69         void backward( tensor_type const& grad )
70         {
71             auto const& current_gradient = backward_action_( input_data_, output_data_, grad );
72             op_.backward( current_gradient );
73         }
74
75         std::vector<unsigned long> shape() const noexcept
76         {
77             return output_shape_calculator_( op_.shape() );
78         }
79     };
80
81
88     template< typename Forward_Action, typename Backward_Action, typename Output_Shape_Calculator=
identity_output_shape_calculator >
89     auto constexpr make_unary_operator( Forward_Action const& unary_forward_action,
Backward_Action const& unary_backward_action,
90                                         std::string const& name = "Anonymous Unary Operator",
Output_Shape_Calculator const& output_shape_calculator =
91                                         Output_Shape_Calculator{} ) noexcept
92     {
93         {
94             return [&]( auto const& op ) noexcept
95             {
96                 auto ans = unary_operator{ op, unary_forward_action, unary_backward_action,
output_shape_calculator };
97                 ans.name_ = name;
98                 return ans;
99             };
100         }
101
102
106     template< typename Lhs_Operator, typename Rhs_Operator, typename Forward_Action, typename
Backward_Action, typename Output_Shape_Calculator= identity_output_shape_calculator >
107     struct binary_operator :enable_id<binary_operator<Lhs_Operator, Rhs_Operator, Forward_Action,
Backward_Action>, "Binary Operator">

```

```

108     {
109         Lhs_Operator lhs_op_;
110         Rhs_Operator rhs_op_;
111         Forward_Action forward_action_;
112         Backward_Action backward_action_; // backward action for binary operator produces a tuple of two
tensors
113         Output_Shape_Calculator output_shape_calculator_;
114
115         typedef typename tensor_deduction<Lhs_Operator, Rhs_Operator>::tensor_type tensor_type; //
defined in value.hpp
116
117         tensor_type lhs_input_data_;
118         tensor_type rhs_input_data_;
119         tensor_type output_data_;
120
121         binary_operator( Lhs_Operator const& lhs_op, Rhs_Operator const& rhs_op, Forward_Action const&
forward_action, Backward_Action const& backward_action, Output_Shape_Calculator const&
output_shape_calculator ) noexcept :
122             lhs_op_{lhs_op}, rhs_op_{rhs_op}, forward_action_{ forward_action }, backward_action_{
backward_action }, output_shape_calculator_{ output_shape_calculator } { }
123
124         auto forward()
125         {
126             static_assert( !(is_value_v<Lhs_Operator> && is_value_v<Rhs_Operator>), "Not valid for two
values" );
127
128             if constexpr ( is_value_v<Lhs_Operator> )
129             {
130                 rhs_input_data_ = rhs_op_.forward();
131                 lhs_input_data_ = lhs_op_.forward( rhs_input_data_ );
132             }
133             else if constexpr ( is_value_v<Rhs_Operator> )
134             {
135                 lhs_input_data_ = lhs_op_.forward();
136                 rhs_input_data_ = rhs_op_.forward( lhs_input_data_ );
137             }
138             else
139             {
140                 lhs_input_data_ = lhs_op_.forward();
141                 rhs_input_data_ = rhs_op_.forward();
142             }
143             output_data_ = forward_action_( lhs_input_data_, rhs_input_data_ );
144             return output_data_;
145         }
146
147         void backward( tensor_type const& grad )
148         {
149             auto const& [current_gradient_lhs, current_gradient_rhs] = backward_action_(
lhs_input_data_, rhs_input_data_, output_data_, grad );
150             lhs_op_.backward( current_gradient_lhs );
151             rhs_op_.backward( current_gradient_rhs );
152         }
153
154         std::vector<unsigned long> shape() const noexcept
155         {
156             if constexpr ( is_value_v<Lhs_Operator> )
157                 return rhs_op_.shape();
158             else if constexpr ( is_value_v<Rhs_Operator> )
159                 return lhs_op_.shape();
160             else
161                 return output_shape_calculator_( lhs_op_.shape(), rhs_op_.shape() );
162         }
163     };
164
165     template< typename Forward_Action, typename Backward_Action, typename Output_Shape_Calculator=
identity_output_shape_calculator >
166     auto make_binary_operator( Forward_Action const& binary_forward_action,
167                               Backward_Action const& binary_backward_action,
168                               std::string const& name = "Anonymous Binary Operator",
169                               Output_Shape_Calculator const& output_shape_calculator =
Output_Shape_Calculator{} ) noexcept
170     {
171         return [&]( auto const& lhs_op, auto const& rhs_op ) noexcept
172         {
173             auto ans = binary_operator{ lhs_op, rhs_op, binary_forward_action, binary_backward_action,
output_shape_calculator };
174             ans.name_ = name;
175             return ans;
176         };
177     }
178
179     template< typename T >
180     struct is_unary_operator : std::false_type{};
181
182     template< typename Operator, typename Forward_Action, typename Backward_Action, typename

```



```

Output_Shape_Calculator >
191     struct is_unary_operator< unary_operator<Operator, Forward_Action, Backward_Action,
Output_Shape_Calculator> > : std::true_type {};
192
196     template< class T >
197     inline constexpr bool is_unary_operator_v = is_unary_operator<T>::value;
198
203     template< typename T >
204     concept Unary_Operator = is_unary_operator_v<T>;
205
206
207     template< typename T >
208     struct is_binary_operator : std::false_type{};
209
210     template< typename Lhs_Operator, typename Rhs_Operator, typename Forward_Action, typename
Backward_Action, typename Output_Shape_Calculator >
211     struct is_binary_operator< binary_operator<Lhs_Operator, Rhs_Operator, Forward_Action,
Backward_Action, Output_Shape_Calculator> > : std::true_type {};
212
216     template< class T >
217     inline constexpr bool is_binary_operator_v = is_binary_operator<T>::value;
218
223     template< typename T >
224     concept Binary_Operator = is_binary_operator_v<T>;
225
230     template< typename T >
231     concept Operator = Unary_Operator<T> || Binary_Operator<T>;
232
237     template< typename T >
238     concept Expression = Operator<T> || Variable<T> || Place_Holder<T> || Constant<T> || Value<T>;
239
240
246     template< Expression Ex >
247     inline std::string computation_graph( Ex const& ex ) noexcept
248     {
249         auto generate_node_and_label = [<Expression Expr>( Expr const& expr ) noexcept
250         {
251             std::string const id = std::to_string( expr.id() );
252             std::string const name = expr.name();
253             std::string node = std::string{"n"} + id;
254
255             std::vector<long long> shape;
256             {
257                 std::vector<unsigned long> _shape = expr.shape();
258                 shape.resize( _shape.size() );
259                 std::copy( _shape.begin(), _shape.end(), shape.begin() );
260                 if ( _shape.size() > 0 && _shape[0] == -1UL )
261                     shape[0] = -1;
262             }
263
264             std::string label = fmt::format( "{} <shape:{}> [id:{}]", name, shape, id);
265             return std::make_tuple( node, label );
266         };
267
268         auto generate_dot = [&generate_node_and_label]<Expression Expr>( Expr const& expr, auto const&
_generate_dot ) noexcept
269         {
270             auto const& [node, label] = generate_node_and_label( expr );
271             std::string const& expr_dot = node + std::string{" [label=\""} + label + std::string{"\""}
;\n";
272
273             if constexpr( is_unary_operator_v<Expr> )
274             {
275                 auto const& [n_node, n_label] = generate_node_and_label( expr.op_ );
276                 std::string const& arrow_relation = n_node + std::string{" -> "} + node + std::string{"\""}
;\n";
277
278                 std::string const& op_dot = _generate_dot( expr.op_, _generate_dot );
279                 return expr_dot + arrow_relation + op_dot;
280             }
281             else if constexpr( is_binary_operator_v<Expr> )
282             {
283                 // for LHS operator
284                 auto const& [n_lhs_node, n_lhs_label] = generate_node_and_label( expr.lhs_op_ );
285                 std::string const& arrow_lhs_relation = n_lhs_node + std::string{" -> "} + node +
std::string{"\""};\n";
286                 std::string const& op_lhs_dot = _generate_dot( expr.lhs_op_, _generate_dot );
287
288                 // for RHS operator
289                 auto const& [n_rhs_node, n_rhs_label] = generate_node_and_label( expr.rhs_op_ );
290                 std::string const& arrow_rhs_relation = n_rhs_node + std::string{" -> "} + node +
std::string{"\""};\n";
291                 std::string const& op_rhs_dot = _generate_dot( expr.rhs_op_, _generate_dot );
292
293                 return expr_dot + arrow_lhs_relation + arrow_rhs_relation + op_lhs_dot + op_rhs_dot;
294             }
295             else if constexpr ( is_variable_v<Expr> )
296             {

```

```

296         std::vector<unsigned long> const& shape = expr.shape();
297         bool const training_state = expr.trainable();
298
299         // shape
300         std::stringstream ss;
301         std::copy( shape.begin(), shape.end(), std::ostream_iterator<unsigned long>( ss, " " )
302 );
303         std::string const& str_shape = ss.str() + (training_state ? std::string("), trainable")
304 : std::string("), non-trainable"));
305         // trainable state
306         std::string const& new_label = label + std::string{"([" + str_shape + std::string{"]"};
307
308         if (!training_state)
309             return node + std::string{" [shape=box,label=\"\"} + new_label + std::string{"\""]
310 ;\n"};
311
312         return node + std::string{" [peripheries=3,style=filled,color=\".7 .3
313 1.0\",shape=box,label=\"\"} + new_label + std::string{"\""] ;\n"};
314     }
315     else
316     {
317         return expr_dot;
318     }
319 };
320
321 std::string const& head = "\n\ndigraph g {\n";
322 std::string const& tail = "}\n\n";
323 return head + generate_dot( ex, generate_dot ) + tail;
324 }
325
326 namespace
327 {
328     struct plus_context
329     {
330         auto make_forward() const noexcept
331         {
332             return [<Tensor Tsor>( Tsor const& lhs_tensor, Tsor const& rhs_tensor ) noexcept
333             {
334                 better_assert( !has_nan( lhs_tensor ), "forward propagation for operator plus:
335 lhs_tensor contains Nan!" );
336                 better_assert( !has_nan( rhs_tensor ), "forward propagation for operator plus:
337 rhs_tensor contains Nan!" );
338                 return add( lhs_tensor, rhs_tensor );
339             }];
340         }
341
342         auto const make_backward() const noexcept
343         {
344             return [<Tensor Tsor>( Tsor const& lhs_input, Tsor const& rhs_input, Tsor const&, Tsor
345 const& grad ) noexcept
346             {
347                 better_assert( !has_nan( grad ), "backprop: upcoming gradient for operator +
348 contains NaN!" );
349
350                 auto const& grad_fun = [&grad]( auto const& input )
351                 {
352                     Tsor ans = grad.deep_copy();
353                     while( input.ndim() < ans.ndim() )
354                         ans = sum( ans, 0 );
355                     auto const& shape = input.shape();
356                     for ( auto axis : range( input.ndim() ) )
357                         if ( shape[axis] == 1 )
358                             ans = sum( ans, axis, true );
359                     return ans;
360                 };
361                 return std::make_tuple( grad_fun( lhs_input), grad_fun( rhs_input ) );
362             }];
363         }
364     }; // plus_context
365 } //anonymous namespace
366
367 template< Expression Lhs_Expression, Expression Rhs_Expression >
368 auto constexpr plus( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
369 {
370     auto const& shape_calculator = [] ( std::vector<unsigned long> const& l, std::vector<unsigned
371 long> const& r ) noexcept
372     {
373         return broadcast_shape( l, r );
374     };
375
376     return make_binary_operator( plus_context{}.make_forward(), plus_context{}.make_backward(),
377 "Plus", shape_calculator )( lhs_ex, rhs_ex );
378 }
379
380 template< Expression Lhs_Expression, Expression Rhs_Expression >

```

```

373     auto constexpr operator + ( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
374     {
375         return plus( lhs_ex, rhs_ex );
376     }
377
378     template< Expression Ex >
379     auto constexpr operator + ( Ex const& ex ) noexcept
380     {
381         return ex;
382     }
383
384     namespace
385     {
386         struct multiplication_context
387         {
388             auto make_forward() const noexcept
389             {
390                 return [] ( std::shared_ptr<std::any> forward_cache ) noexcept
391                 {
392                     return [forward_cache]<Tensor Tsor>( Tsor const& lhs_tensor, Tsor const& rhs_tensor
393 ) noexcept
394                     {
395                         Tsor& ans = context_cast<Tsor>( forward_cache );
396                         multiply( lhs_tensor, rhs_tensor, ans );
397                         return ans;
398                     };
399                 };
400             auto make_backward() const noexcept
401             {
402                 return [] ( std::shared_ptr<std::any> backward_cache_lhs, std::shared_ptr<std::any>
backward_cache_rhs ) noexcept
403                 {
404                     return [backward_cache_lhs, backward_cache_rhs]<Tensor Tsor>( Tsor const& lhs_input,
Tsor const& rhs_input, Tsor const&, Tsor const& grad ) noexcept
405                     {
406                         // left branch <-- grad * rhs^T
407                         auto const& g_shape = grad.shape();
408                         auto const[m, n] = std::make_tuple( g_shape[0], g_shape[1] ); // 4, 1
409                         auto const k = *(lhs_input.shape().rbegin()); // 13
410
411                         Tsor& lhs_grad = context_cast<Tsor>( backward_cache_lhs );
412                         lhs_grad.resize( lhs_input.shape() );
413
414                         gemm( grad.data(), false, rhs_input.data(), true, m, n, k, lhs_grad.data() );
415
416                         // right branch <-- lhs^T * grad
417                         Tsor& rhs_grad = context_cast<Tsor>( backward_cache_rhs );
418                         rhs_grad.resize( rhs_input.shape() );
419                         gemm( lhs_input.data(), true, grad.data(), false, k, m, n, rhs_grad.data() );
420
421                         return std::make_tuple( lhs_grad, rhs_grad );
422                     };
423                 };
424             };
425         }; //multiplication_context
426     }; //anonymous namespace
427
428     template< Expression Lhs_Expression, Expression Rhs_Expression >
429     auto operator * ( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
430     {
431
432
433         // case of Value * Operator and Operator * Value
434         if constexpr( is_value_v<Lhs_Expression> || is_value_v<Rhs_Expression> )
435         {
436             return elementwise_product( lhs_ex, rhs_ex );
437         }
438         else
439         {
440             auto const& shape_calculator = [] ( std::vector<unsigned long> const& l, std::vector<unsigned
long> const& r ) noexcept
441             {
442                 better_assert( l.size() == 2, fmt::format( "expecting l size of 2, but got {}", l.size()
) );
443                 better_assert( r.size() == 2, fmt::format( "expecting r size of 2, but got {}", r.size()
) );
444                 better_assert( l[1] == r[0], fmt::format( "expecting l[1] == r[0], but l[1]={},
r[0]={}", l[1], r[0] ) ); // TODO: what if unknown dimension???
445                 return std::vector<unsigned long>{ {l[0], r[1]} };
446             };
447             std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
448             std::shared_ptr<std::any> backward_cache_lhs = std::make_shared<std::any>();
449             std::shared_ptr<std::any> backward_cache_rhs = std::make_shared<std::any>();
450             return make_binary_operator( multiplication_context{}.make_forward() (forward_cache),
multiplication_context{}.make_backward() (backward_cache_lhs, backward_cache_rhs), "Multiply",
shape_calculator ) ( lhs_ex, rhs_ex );

```

```

451     }
452 }
453
454
455 template <Expression Ex>
456 auto constexpr negative( Ex const& ex ) noexcept
457 {
458     return make_unary_operator( [<Tensor Tsr>( Tsr const& tensor ) noexcept
459     {
460         better_assert( !has_nan( tensor ), "forward propagation for
operator log: tensor contains NaN!" );
461         return -tensor;
462     },
463     [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
464     noexcept
465     {
466         better_assert( !has_nan( grad ), "input gradient for operator
negative contains NaN!" );
467         return -grad;
468     },
469     "Negative"
470     )( ex );
471 };
472
473 template <Expression Ex>
474 auto constexpr operator - ( Ex const& ex ) noexcept
475 {
476     return negative( ex );
477 }
478
479
480 template <Expression Ex>
481 auto constexpr inverse( Ex const& ex ) noexcept
482 {
483     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
484     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
485     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& tensor ) noexcept
486     {
487         Tsr& ans = context_cast<Tsr>( forward_cache );
488         ans.resize( tensor.shape() );
489         //for_each( tensor.begin(), tensor.end(), ans.begin(), [(auto
490         (1.0/std::max(eps, x)) : (1.0/std::min(-eps, x)); ));
491         for_each( tensor.begin(), tensor.end(), ans.begin(), [(auto
492         const x, auto& y) { y = (x > 0.0) ?
493         (1.0/std::max(eps, x)) : (1.0/std::min(-eps, x)); });
494         return ans;
495     },
496     [backward_cache]<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr
497     const& grad ) noexcept
498     {
499         Tsr& ans = context_cast<Tsr>( backward_cache );
500         ans.resize( input.shape() );
501         //for_each( ans.begin(), ans.end(), grad.begin(), input.begin(),
502         [( auto& x, auto y, auto z ){ x = - y / std::max(z*z, eps); } );
503         for_each( ans.begin(), ans.end(), grad.begin(), input.begin(),
504         [( auto& x, auto y, auto z ){ x = - y / (z*z); } );
505         ans.resize( grad.shape() );
506         return ans;
507     },
508     "Inverse"
509     )( ex );
510 };
511
512
513 template< Expression Lhs_Expression, Expression Rhs_Expression >
514 auto constexpr elementwise_product( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex )
515 noexcept
516 {
517     return make_binary_operator( [<Tensor Tsr>( Tsr const& lhs_tensor, Tsr const& rhs_tensor )
518     noexcept
519     {
520         return elementwise_product( lhs_tensor, rhs_tensor );
521     },
522     [<Tensor Tsr>( Tsr const& lhs_input, Tsr const& rhs_input, Tsr
523     const&, Tsr const grad ) noexcept
524     {
525         auto const& grad_fun = [&grad]( auto const& input, auto const&
526         other_input )
527         {
528             Tsr ans = elementwise_product( grad, other_input );
529             while( input.ndim() < ans.ndim() )
530                 ans = sum( ans, 0 );
531             auto const& shape = input.shape();
532             for ( auto axis : range( input.ndim() ) )
533                 if ( shape[axis] == 1 )
534                     ans = sum( ans, axis, true );
535         }
536     }

```

```

547         return ans;
548     };
549     return std::make_tuple( grad_fun( lhs_input, rhs_input ),
grad_fun( rhs_input, lhs_input ) );
550 },
551     "HadamardProduct"
552 )( lhs_ex, rhs_ex );
553 };
554
555 template< Expression Lhs_Expression, Expression Rhs_Expression >
556 auto constexpr elementwise_multiply( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex )
noexcept
557 {
558     return elementwise_product( lhs_ex, rhs_ex );
559 }
560
561 template< Expression Lhs_Expression, Expression Rhs_Expression >
562 auto constexpr hadamard_product( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex )
noexcept
563 {
564     return elementwise_product( lhs_ex, rhs_ex );
565 }
566
567
568 template< Expression Lhs_Expression, Expression Rhs_Expression >
569 auto constexpr divide( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
570 {
571     return elementwise_product( lhs_ex, inverse( rhs_ex ) );
572 }
573
574 template< Expression Lhs_Expression, Expression Rhs_Expression >
575 auto constexpr operator / ( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
576 {
577     return divide( lhs_ex, rhs_ex );
578 }
579
580
581
582 template <Expression Ex>
583 auto constexpr sum_reduce( Ex const& ex ) noexcept
584 {
585     return make_unary_operator( [<Tensor Tsr>( Tsr const& tsor ) noexcept
586     {
587         better_assert( !has_nan( tsor ), "forward propagation for
operator sum_reduce: tensor contains NaN!" );
588         return reduce_sum( tsor );
589     },
590     [<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr const& grad )
591     noexcept
592     {
593         better_assert( !has_nan( grad ), "input gradient for operator
sum_reduce contains NaN!" );
594         better_assert( grad.size() == 1, "sum_reduce should only output
one value" );
595         Tsr ans = ones_like( input );
596         ans *= grad[0];
597         return ans;
598     },
599     "Sum",
600     [<std::vector<unsigned long> const& ) noexcept { return
std::vector<unsigned long>( {1,} ); }
601     )( ex );
602 }
603
604 template <Expression Ex>
605 auto constexpr reduce_sum( Ex const& ex ) noexcept
606 {
607     return sum_reduce( ex );
608 }
609
610 template <Expression Ex>
611 auto constexpr mean_reduce( Ex const& ex ) noexcept
612 {
613     return make_unary_operator( [<Tensor Tsr>( Tsr const& tsor ) noexcept
614     {
615         better_assert( !has_nan( tsor ), "forward propagation for
operator mean: tensor contains NaN!" );
616         return reduce_mean( tsor );
617     },
618     [<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr const& grad )
619     noexcept
620     {
621         better_assert( !has_nan( grad ), "input gradient for operator
mean_reduce contains NaN!" );
622         better_assert( grad.size() == 1, "mean_reduce should only output
one value" );

```

```

657         Tsor ans = ones_like( input );
658         ans *= grad[0];
659         unsigned long const batch_size = (input.shape().size() == 1) ?
1 : (*input.shape().begin());
660         ans /= static_cast<typename Tsor::value_type>(batch_size);
661         return ans;
662     },
663     "Mean",
664     []( std::vector<unsigned long> const& ) noexcept { return
std::vector<unsigned long>{ {1,} }; }
665     )( ex );
666 }
667
668 template <Expression Ex>
669 auto constexpr reduce_mean( Ex const& ex ) noexcept
670 {
671     return mean_reduce( ex );
672 }
673
674 template <Expression Ex>
675 auto constexpr mean( Ex const& ex ) noexcept
676 {
677     return mean_reduce( ex );
678 }
679
680 template< Expression Lhs_Expression, Expression Rhs_Expression >
681 auto constexpr minus( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
682 {
683     if constexpr (is_value_v<Rhs_Expression>)
684     {
685         return negative( plus( negative(lhs_ex), rhs_ex ) );
686     }
687     else
688     {
689         return plus( lhs_ex, negative(rhs_ex) );
690     }
691 }
692
693 template< Expression Lhs_Expression, Expression Rhs_Expression >
694 auto constexpr operator - ( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
695 {
696     return minus( lhs_ex, rhs_ex );
697 }
698
699 template <Expression Ex>
700 auto constexpr square( Ex const& ex ) noexcept
701 {
702     return make_unary_operator( [<Tensor Tsor>( Tsor const& tsor ) noexcept
703     {
704         better_assert( !has_nan( tsor ), "forward propagation for
operator square: tensor contains NaN!" );
705         Tsor ans = tsor.deep_copy();
706         std::for_each( ans.data(), ans.data() + ans.size(), []( auto & v
707         ){ v *= v; } );
708         return ans;
709     },
710     [<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad )
711     noexcept
712     {
713         better_assert( !has_nan( grad ), "input gradient for operator
square contains NaN!" );
714         Tsor ans = input.deep_copy();
715         ans *= grad;
716         ans *= typename Tsor::value_type{2};
717         return ans;
718     },
719     "Square"
720     )( ex );
721 }
722
723 template <Expression Ex, Expression Ey>
724 auto constexpr hypot( Ex const& ex, Ey const& ey ) noexcept
725 {
726     return sqrt( square(ex) + square(ey) );
727 }
728
729 template <typename Float> requires std::floating_point<Float>
730 auto constexpr clip( Float lower, Float upper=std::numeric_limits<Float>::max() ) noexcept

```

```

770     {
771         return [lower, upper]<Expression Ex>( Ex const& ex ) noexcept
772     {
773         return make_unary_operator( [lower, upper]<Tensor Tsr>( Tsr const& tsor ) noexcept
774         {
775             better_assert( !has_nan( tsor ), "forward propagation for
operator clip: tensor contains Nan!" );
776             Tsr ans = tsor.deep_copy();
777             clip( ans, lower, upper );
778             return ans;
779         },
780         [lower, upper]<Tensor Tsr>( Tsr const& input, Tsr const&,
Tsr const& grad ) noexcept
781         {
782             better_assert( !has_nan( grad ), "input gradient for
operator clip contains NaN!" );
783             const typename Tsr::value_type zero{0};
784             Tsr ans = grad;
785             for ( auto idx : range( input.size() ) )
786                 ans[idx] = (input[idx] < lower) ? zero :
787                             (input[idx] > upper) ? zero :
788                             ans[idx];
789             return ans;
790         },
791         "Clip"
792     )( ex );
793     };
794 }
795
796 // include_batch_flag:
797 //
798 // true: considering the batch size at the first dim
799 //      - for an input of (1, 3, 4), expecting an incoming expression of shape like [BS, 12, 1 1]
800 //      - expected output of shape [BS, 1, 3, 4]
801 // false: do not consider the batch size
802 //      - for an input of (1, 3, 4), expecting an incoming expression of shape like [12, 1]
803 //      - expected output of shape [1, 3, 4]
804 auto inline reshape( std::vector<unsigned long> const& new_shape, bool include_batch_flag=true )
noexcept
805 {
806     return [new_shape, include_batch_flag]<Expression Ex>( Ex const& ex ) noexcept
807     {
808         return make_unary_operator
809         (
810             [new_shape, include_batch_flag]<Tensor Tsr>( Tsr const& tsor ) noexcept
811             {
812                 unsigned long const new_size = std::accumulate( new_shape.begin(), new_shape.end(),
1UL, []( auto x, auto y ){ return x+y; } );
813                 unsigned long const total_size = tsor.size();
814                 unsigned long const batch_size = total_size / new_size;
815
816                 better_assert( batch_size * new_size == total_size, "size mismatch for reshape
operator, expect ", batch_size*new_size, " but total input size is ", total_size, ", where batch_size
is ", batch_size );
817
818                 if ( !include_batch_flag )
819                 {
820                     better_assert( batch_size == 1, "expecting batch size of 1 while not including
batch, but got ", batch_size );
821                     Tsr ans{tsor};
822                     ans.reshape( new_shape );
823                     return ans;
824                 }
825
826                 std::vector<unsigned long> batched_new_shape;
827                 {
828                     batched_new_shape.resize( 1 + new_shape.size() );
829                     batched_new_shape[0] = batch_size;
830                     std::copy( new_shape.begin(), new_shape.end(), batched_new_shape.begin()+1 );
831                 }
832
833                 Tsr ans{ tsor };
834                 ans.reshape( batched_new_shape );
835                 return ans;
836             },
837             [<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr const& grad ) noexcept
838             {
839                 Tsr ans{ grad };
840                 ans.reshape( input.shape() );
841                 return ans;
842             },
843             "Reshape",
844             [new_shape, include_batch_flag]( std::vector<unsigned long> const& shape ) noexcept
845             {
846
847                 //debug_log( fmt::format("Calculating Reshape layer size include_batch_flag = {}",
include_batch_flag) );

```

```

848         //debug_log( fmt::format("Calculating Reshape layer size with shape = {}", shape) );
849         //debug_log( fmt::format("Calculating Reshape layer size with new_shape = {}",
new_shape) );
850
851         if ( include_batch_flag == false )
852             return new_shape;
853
854         unsigned long const new_size = std::accumulate( new_shape.begin(), new_shape.end(),
1UL, []( auto x, auto y ){ return x*y; } );
855         unsigned long const total_size = std::accumulate( shape.begin(), shape.end(), 1UL,
[]( unsigned long x, unsigned long y ){ return x*y; } );
856         unsigned long const batch_size = total_size / new_size;
857         std::vector<unsigned long> batched_new_shape;
858         {
859             batched_new_shape.resize( 1 + new_shape.size() );
860             batched_new_shape[0] = batch_size;
861             std::copy( new_shape.begin(), new_shape.end(), batched_new_shape.begin()+1 );
862         }
863         return batched_new_shape;
864
865         /*
866         std::vector<unsigned long> ans;
867         ans.resize( new_shape.size()+1 );
868         ans[0] = shape[0];
869         std::copy( new_shape.begin(), new_shape.end(), ans.begin()+1 );
870         return ans;
871         */
872     }
873     )( ex );
874 };
875 }
876
877 template <Expression Ex>
878 auto constexpr flatten( Ex const& ex ) noexcept
879 {
880     return make_unary_operator
881     (
882         [<Tensor Tsr>( Tsr const& tsr ) noexcept
883         {
884             better_assert( tsr.ndim() > 1, "Expecting dimension of incoming tensor to be greater
than 1, but got ", tsr.ndim() );
885             unsigned long const batch_size = *(tsr.shape().begin());
886             unsigned long const rem = tsr.size() / batch_size;
887             Tsr ans = tsr;
888             return ans.reshape( {batch_size, rem} );
889         },
890         [<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr const& grad ) noexcept
891         {
892             Tsr ans = grad;
893             return ans.reshape( input.shape() );
894         },
895         "Flatten",
896         []( std::vector<unsigned long> const& shape ) noexcept
897         {
898             unsigned long const total = std::accumulate( shape.begin()+1, shape.end(), 1, [](
unsigned long x, unsigned long y ){ return x*y; } );
899             return std::vector<unsigned long>{ {shape[0], total,} }; // the 1st dim is batch size
900         }
901     )( ex );
902 }
903
904 constexpr auto inline expand_dims( int axis=-1 ) noexcept
905 {
906     return [=]<Expression Ex>( Ex const& ex ) noexcept
907     {
908         return make_unary_operator
909         (
910             [=]<Tensor Tsr>( Tsr const& tsr ) noexcept
911             {
912                 Tsr ans = tsr;
913                 std::vector<unsigned long> shape = ans.shape();
914                 int const _axis = (axis == -1) ? shape.size() : axis;
915                 shape.insert( shape.begin()+_axis, 1UL );
916                 ans.reshape( shape );
917                 return ans;
918             },
919             [=]<Tensor Tsr>( Tsr const& input, Tsr const& /*output*/, Tsr const& grad ) noexcept
920             {
921                 Tsr ans = grad;
922                 ans.reshape( input.shape() );
923                 return ans;
924             },
925             "ExpandDims",
926             [axis]( std::vector<unsigned long> const& shape ) noexcept
927             {
928                 std::vector<unsigned long> ans = shape;
929                 if ( axis == -1 ) axis = ans.size();
930             }
931         )
932     }
933 }

```



```

947         ans.insert( ans.begin()+axis, 1 );
948     }
949     )(ex);
950 };
951 }
952
953
954
955
956
957
958
959
960
961 auto inline argmax( unsigned long axis=0 ) noexcept
962 {
963     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
964     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
965
966     return [=]<Expression Ex>( Ex const& ex ) noexcept
967     {
968         return make_unary_operator
969         (
970             [=]<Tensor Tsor>( Tsor const& input ) noexcept
971             {
972                 std::vector<unsigned long> const& shape = input.shape();
973                 better_assert( axis < shape.size(), fmt::format("axis {} is greater than the
dimension of the input tensor shape {}", axis, shape) );
974
975                 // calculate the output tensor shape
976                 std::vector<unsigned long> output_shape = shape;
977                 std::copy( output_shape.begin()+axis+1, output_shape.end(),
output_shape.begin()+axis );
978                 output_shape.resize( output_shape.size() - 1 );
979                 Tsor& ans = context_cast<Tsor>( forward_cache );
980                 ans.resize( output_shape );
981
982                 // viewing the input tensor as a 3D tensor, and viewing the output tensor as a 2D
tensor
983                 unsigned long const bs = std::accumulate( shape.begin(), shape.begin()+axis, 1UL,
[] ( unsigned long x, unsigned long y ) { return x*y; } );
984                 unsigned long const row = shape[axis];
985                 unsigned long const col = std::accumulate( shape.begin()+axis+1, shape.end(), 1UL,
[] ( unsigned long x, unsigned long y ) { return x*y; } );
986                 auto cube_input = view_3d( input.data(), bs, row, col );
987                 auto matrix_output = view_2d( ans.data(), bs, col );
988
989                 for ( auto _bs : range( bs ) )
990                     for ( auto _col : range( col ) )
991                     {
992                         unsigned long mx_idx = 0;
993                         auto mx = cube_input[_bs][0][_col];
994                         for ( auto _row : range( row ) )
995                         {
996                             if ( cube_input[_bs][_row][_col] > mx )
997                             {
998                                 mx = cube_input[_bs][_row][_col];
999                                 mx_idx = _row;
1000                             }
1001                         }
1002                         matrix_output[_bs][_col] = mx_idx;
1003                     }
1004                 return ans;
1005             },
1006             [=]<Tensor Tsor>( Tsor const& input, Tsor const& /*output*/, Tsor const& /*grad*/ )
noexcept
1007             {
1008                 Tsor& back_ans = context_cast<Tsor>( backward_cache );
1009                 back_ans.resize( input.shape() );
1010                 for_each( back_ans.begin(), back_ans.end(), [] ( auto& v ) { v = 0.0; } ); // always
return zero
1011                 return back_ans;
1012             },
1013             "Argmax",
1014             [axis]( std::vector<unsigned long> const& shape ) noexcept
1015             {
1016                 std::vector<unsigned long> ans = shape;
1017                 std::copy( ans.begin()+axis+1, ans.end(), ans.begin()+axis );
1018                 ans.resize( ans.size() - 1 );
1019                 return ans;
1020             }
1021         )(ex);
1022     };
1023 }
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033 auto inline argmin( unsigned long axis=0 ) noexcept
1034 {
1035     return [=]<Expression Ex>( Ex const& ex ) noexcept
1036     {
1037         return argmax(axis)( -ex );
1038     };
1039 }
1040

```

```

1041
1042
1046     template <Expression Ex>
1047     auto constexpr identity( Ex const& ex ) noexcept
1048     {
1049         return ex;
1050     }
1051
1055     template< Expression Ex >
1056     auto transpose( Ex const& ex ) noexcept
1057     {
1058         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1059         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
1060         return make_unary_operator
1061         (
1062             [forward_cache]<Tensor Tsor>( Tsor const& tsor ) noexcept
1063             {
1064                 tsor.ndim() );
1065
1066                 typedef typename Tsor::value_type value_type;
1067
1068                 std::vector<unsigned long> const shape = tsor.shape();
1069                 auto const [row, col] = std::make_tuple( shape[0], shape[1] );
1070                 view_2d<value_type> v_in{ tsor.data(), row, col };
1071
1072                 Tsor& ans = context_cast<Tsor>( forward_cache );
1073                 ans.resize( {col, row} );
1074                 view_2d<value_type> v_out{ ans.data(), col, row };
1075
1076                 for ( auto r : range( row ) )
1077                     for ( auto c : range( col ) )
1078                         v_out[c][r] = v_in[r][c];
1079
1080                 return ans;
1081             },
1082             [backward_cache]<Tensor Tsor>( Tsor const&, Tsor const&, Tsor const& grad ) noexcept
1083             {
1084                 typedef typename Tsor::value_type value_type;
1085
1086                 std::vector<unsigned long> const shape = grad.shape();
1087                 auto const [row, col] = std::make_tuple( shape[0], shape[1] );
1088                 view_2d<value_type> v_in{ grad.data(), row, col };
1089
1090                 Tsor& back_ans = context_cast<Tsor>( backward_cache );
1091                 back_ans.resize( {col, row} );
1092
1093                 view_2d<value_type> v_out{ back_ans.data(), col, row };
1094
1095                 for ( auto r : range( row ) )
1096                     for ( auto c : range( col ) )
1097                         v_out[c][r] = v_in[r][c];
1098
1099                 return back_ans;
1100             },
1101             "Transpose",
1102             [] ( std::vector<unsigned long> const shape ) noexcept
1103             {
1104                 better_assert( shape.size() == 2, fmt::format( "expecting shape size of 2, but got {}",
1105 shape.size() ) );
1106                 return std::vector<unsigned long>{ {shape[1], shape[0]} };
1107             }
1108         )( ex );
1109
1110     auto inline img2col( unsigned long const row_kernel, unsigned long col_kernel=-1,
1111                         unsigned long const row_padding=0, unsigned long col_padding=0,
1112                         unsigned long const row_stride=1, unsigned long const col_stride=1,
1113                         unsigned long const row_dilation=1, unsigned long const col_dilation=1 )
1114     noexcept
1115     {
1116         if ( col_kernel == (unsigned long)-1 ) col_kernel = row_kernel;
1117
1118         std::shared_ptr<std::vector<std::uint32_t> s_index_record =
1119         std::make_shared<std::vector<std::uint32_t>(); // col_img[idx] = img[index_record[idx]] -- (-1) for
1120         zero padding
1121
1122         auto img2col_forward = [s_index_record]<Tensor Tsor>
1123         (
1124             Tsor const& input_img, Tsor& output_col_mat,
1125             unsigned long kernel_row, unsigned long kernel_col,
1126             unsigned long padding_row, unsigned long padding_col,
1127             unsigned long stride_row, unsigned long stride_col,
1128             unsigned long dilation_row, unsigned long dilation_col
1129         ) noexcept
1130         {
1131             typedef typename Tsor::value_type value_type;

```

```

1129         std::vector<std::uint32_t>& index_record = *s_index_record; //32 bit should be enough for
memory address offset
1130
1131         std::vector<unsigned long> input_shape = input_img.shape();
1132         better_assert( input_shape.size() == 4, "Expecting a 4D tensor." );
1133         auto const [BS, R, C, CH] = std::make_tuple( input_shape[0], input_shape[1],
input_shape[2], input_shape[3] );
1134
1135         unsigned long const output_row = ( R + 2 * padding_row - ( dilation_row * (kernel_row - 1)
+ 1 ) ) / stride_row + 1;
1136         unsigned long const output_col = ( C + 2 * padding_col - ( dilation_col * (kernel_col - 1)
+ 1 ) ) / stride_col + 1;
1137         unsigned long const output_column_matrix_row = kernel_row * kernel_col * CH;
1138         unsigned long const output_column_matrix_col = BS * output_row * output_col;
1139
1140         output_col_mat.resize( {output_column_matrix_row, output_column_matrix_col} );
1141
1142         if ( index_record.size() != output_column_matrix_row * output_column_matrix_col ) //
first-run?
1143         {
1144             index_record.resize( output_column_matrix_row * output_column_matrix_col );
1145
1146             for ( auto bs : range( BS ) )
1147             {
1148                 std::int64_t const col_offset = bs * output_row * output_col * kernel_row *
kernel_col * CH;
1149                 std::int64_t const im_offset = bs * R * C * CH;
1150                 for ( auto c : range( CH * kernel_row * kernel_col ) )
1151                 {
1152                     std::int64_t const w_offset = c % kernel_col;
1153                     std::int64_t const h_offset = ( c / kernel_col ) % kernel_row;
1154                     std::int64_t const c_im = c / ( kernel_col * kernel_row );
1155
1156                     for ( auto h : range( output_row ) )
1157                     {
1158                         std::int64_t const im_row_idx = h * stride_row - padding_row + h_offset *
dilation_row;
1159                         for ( auto w : range( output_col ) )
1160                         {
1161                             std::int64_t const im_col_idx = w * stride_col - padding_col + w_offset
* dilation_col;
1162                             std::int64_t const im_idx = im_offset + ( im_row_idx * C + im_col_idx ) *
CH + c_im;
1163                             std::int64_t const col_idx = col_offset + ( c * output_row + h ) *
output_col + w;
1164                             index_record[col_idx] = static_cast<std::uint32_t>((im_row_idx<0 ||
im_row_idx>=static_cast<std::int64_t>(R) || im_col_idx<0 || im_col_idx>=static_cast<std::int64_t>(C))
? 0xffffffff : im_idx);
1165                         }
1166                     }
1167                 }
1168             }
1169             // re-arrange [bs, new_R, new_C] --> [new_R, new_c*bs]
1170             {
1171                 std::vector<std::uint32_t> re_arranged_index;
1172                 re_arranged_index.resize( index_record.size() );
1173
1174                 view_3d<std::uint32_t> re_arranged_mat{ re_arranged_index.data(),
output_column_matrix_row, BS, output_row*output_col };
1175                 view_3d<std::uint32_t> index_record_mat{ index_record.data(), BS,
output_column_matrix_row, output_row*output_col };
1176
1177                 for ( auto bs : range( BS ) )
1178                     for ( auto r : range( output_column_matrix_row ) )
1179                         for ( auto c : range( output_row*output_col ) )
1180                             re_arranged_mat[r][bs][c] = index_record_mat[bs][r][c];
1181                 // overwrite index record
1182                 std::copy( re_arranged_index.begin(), re_arranged_index.end(), index_record.begin()
);
1183             }
1184         }
1185
1186         // fill-in
1187         for ( auto idx : range( output_col_mat.size() ) )
1188         {
1189             auto const index = index_record[idx];
1190             output_col_mat[idx] = (index == 0xffffffff) ? value_type{0} : input_img[index];
1191         }
1192     };
1193
1194     auto img2col_backward = [s_index_record]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor
const& grad, Tsor& ans ) noexcept
1195     {
1196         typedef typename Tsor::value_type value_type;
1197         ans.resize( input.shape() );
1198         std::fill( ans.begin(), ans.end(), value_type{0} );
1199     }

```

```

1200         std::vector<std::uint32_t>& index_record = *s_index_record; //32 bit should be enough for
memory address offset
1201         for ( auto idx : range( grad.size() ) )
1202         {
1203             auto const index = index_record[idx];
1204             if ( index != 0xffffffff )
1205                 ans[index] += grad[idx];
1206         }
1207     };
1208
1209     std::shared_ptr<std::any> output_cache = std::make_shared<std::any>();
1210     std::shared_ptr<std::any> back_grad_cache = std::make_shared<std::any>();
1211
1212     return [row_kernel, col_kernel, row_padding, col_padding, row_stride, col_stride, row_dilation,
col_dilation, img2col_forward, img2col_backward, output_cache, back_grad_cache]<Expression Ex>( Ex
const& ex ) noexcept
1213     {
1214         return make_unary_operator
1215         (
1216             [=]<Tensor Tsor>( Tsor const& tsor ) noexcept
1217             {
1218                 Tsor& output = context_cast<Tsor>( output_cache );
1219                 img2col_forward( tsor, output, row_kernel, col_kernel, row_padding, col_padding,
row_stride, col_stride, row_dilation, col_dilation );
1220                 return Tsor{output};
1221             },
1222             [=]<Tensor Tsor>( Tsor const& input, Tsor const& output, Tsor const& grad ) noexcept
1223             {
1224                 Tsor& back_grad = context_cast<Tsor>( back_grad_cache );
1225                 img2col_backward( input, output, grad, back_grad );
1226                 return Tsor{back_grad};
1227             },
1228             "Img2Col",
1229             [=]( std::vector<unsigned long> const& shape ) noexcept
1230             {
1231                 better_assert( shape.size() == 4, fmt::format("Expecting a 4D tensor, but got {}.",
shape.size()) );
1232                 auto const [BS, R, C, CH] = std::make_tuple( shape[0], shape[1], shape[2], shape[3]
);
1233
1234                 unsigned long const output_row = ( R + 2 * row_padding - ( row_dilation *
(row_kernel - 1) + 1 ) ) / row_stride + 1;
1235                 unsigned long const output_col = ( C + 2 * col_padding - ( col_dilation *
(col_kernel - 1) + 1 ) ) / col_stride + 1;
1236                 unsigned long const output_column_matrix_row = row_kernel * col_kernel * CH;
1237                 unsigned long const output_column_matrix_col = BS * output_row * output_col;
1238                 return std::vector<unsigned long>{ {output_column_matrix_row,
output_column_matrix_col} };
1239             }
1240         )( ex );
1241     };
1242 }
1243
1244 auto inline conv2d
1245 (
1246     unsigned long row_input, unsigned long col_input,
1247     unsigned long const row_stride=1, unsigned long const col_stride=1,
1248     unsigned long const row_dilation=1, unsigned long const col_dilation=1,
1249     std::string const& padding="valid"
1250 ) noexcept
1251 {
1252     // lhs_ex is for one 4D tensor of [BS, R, C, CH]
1253     // rhs_ex is for NC 4D filter of [1, r, c, CH], thus the shape is [NC, r, c, CH]
1254     // the output tensor is of shape [BS, ..., .., NC]
1255     //
1256     // Note: the rhs expression is fixed as a variable, as we need to extract the kernel shape
from it
1257     //
1258     //return [row_input, col_input, row_stride, col_stride, row_dilation, col_dilation, padding
]<Expression Ex, Variable Va>( Ex const& lhs_ex, Va const& rhs_ex ) noexcept
1259     return [row_input, col_input, row_stride, col_stride, row_dilation, col_dilation, padding
]<Expression Ex, Expression Ey>( Ex const& lhs_ex, Ey const& rhs_ex ) noexcept
1260     {
1261         std::vector<unsigned long> const& shape = rhs_ex.shape();
1262         better_assert( shape.size() == 4 );
1263         auto const[new_channel, row_kernel, col_kernel, channel] = std::make_tuple( shape[0],
shape[1], shape[2], shape[3] );
1264         //TODO: optimization in case of small kernels of (1, 1), (3, 3)
1265         unsigned long row_padding = 0;
1266         unsigned long col_padding = 0;
1267         if ( padding == "same" )
1268         {
1269             unsigned long const row_padding_total = (row_kernel + (row_kernel - 1) * (row_dilation
- 1) - row_stride);
1270             better_assert( !(row_padding_total & 0x1), "Expecting total row padding to be even, but
got ", row_padding_total, " With row input ", row_input, " and row_stride ", row_stride );
1271             unsigned long const col_padding_total = (col_kernel + (col_kernel - 1) * (col_dilation

```

```

- 1) - col_stride);
1272         better_assert( !(col_padding_total & 0x1), "Expecting total col padding to be even, but
got ", col_padding_total );
1273         row_padding = ((row_kernel&1)+row_padding_total) >> 1;
1274         col_padding = ((col_kernel&1)+col_padding_total) >> 1;
1275     }
1276
1277     unsigned long const row_output = ( row_input + 2 * row_padding - ( row_dilation *
(row_kernel - 1) + 1 ) ) / row_stride + 1;
1278     unsigned long const col_output = ( col_input + 2 * row_padding - ( col_dilation *
(col_kernel - 1) + 1 ) ) / col_stride + 1;
1279
1280     auto lhs_ex_as_col = img2col(row_kernel, col_kernel, row_padding, col_padding, row_stride,
col_stride, row_dilation, col_dilation)( lhs_ex ); // [BS, R, C, CH] ==> [r*c*CH, BS*new_row*new_col]
1281
1282     auto rhs_ex_flatten = reshape({row_kernel*col_kernel*channel,})( rhs_ex ); // [NC, r, c,
CH] ==> [NC, r*c*CH]
1283
1284     auto flatten_output = rhs_ex_flatten * lhs_ex_as_col; // [NC, BS * new_row * new_col]
1285
1286     auto tr_output = transpose( flatten_output ); // [BS*new_row*new_col, NC]
1287
1288     auto ans = reshape({row_output, col_output, new_channel})( tr_output );
1289
1290     return ans;
1291 };
1292 }
1293
1294
1295 auto inline general_conv2d
1296 (
1297     unsigned long const row_stride=1, unsigned long const col_stride=1,
1298     unsigned long const row_dilation=1, unsigned long const col_dilation=1,
1299     std::string const& padding="valid"
1300 ) noexcept
1301 {
1302     // lhs_ex is for one 4D tensor of [BS, R, C, CH]
1303     // rhs_ex is for NC 4D filter of [1, r, c, CH], thus the shape is [NC, r, c, CH]
1304     // the output tensor is of shape [BS, .., .., NC]
1305     //
1306     // Note: the rhs expression is fixed as a variable, as we need to extract the kernel shape
1307     from it
1308     //
1309     return [ row_stride, col_stride, row_dilation, col_dilation, padding ]<Expression Ex,
Expression Ey>( Ex const& lhs_ex, Ey const& rhs_ex ) noexcept
1310     {
1311         auto const& lhs_shape = lhs_ex.shape();
1312         better_assert( lhs_shape.size() == 4, fmt::format( "expecting lhs_shape size of 4, but got
{}", lhs_shape.size() ) );
1313         auto [_bs, row_input, col_input, _ch] = std::make_tuple( lhs_shape[0], lhs_shape[1],
lhs_shape[2], lhs_shape[3] );
1314
1315         std::vector<unsigned long> const& shape = rhs_ex.shape();
1316         better_assert( shape.size() == 4 );
1317         auto const[new_channel, row_kernel, col_kernel, channel] = std::make_tuple( shape[0],
shape[1], shape[2], shape[3] );
1318         unsigned long row_padding = 0;
1319         unsigned long col_padding = 0;
1320         if ( padding == "same" )
1321         {
1322             unsigned long const row_padding_total = (row_kernel + (row_kernel - 1) * (row_dilation
- 1) - row_stride);
1323             unsigned long const col_padding_total = (col_kernel + (col_kernel - 1) * (col_dilation
- 1) - col_stride);
1324             row_padding = ((row_kernel&1)+row_padding_total) >> 1;
1325             col_padding = ((col_kernel&1)+col_padding_total) >> 1;
1326         }
1327
1328         unsigned long const row_output = ( row_input + 2 * row_padding - ( row_dilation *
(row_kernel - 1) + 1 ) ) / row_stride + 1;
1329         unsigned long const col_output = ( col_input + 2 * row_padding - ( col_dilation *
(col_kernel - 1) + 1 ) ) / col_stride + 1;
1330
1331         auto lhs_ex_as_col = img2col(row_kernel, col_kernel, row_padding, col_padding, row_stride,
col_stride, row_dilation, col_dilation)( lhs_ex ); // [BS, R, C, CH] ==> [r*c*CH, BS*new_row*new_col]
1332
1333         auto rhs_ex_flatten = reshape({row_kernel*col_kernel*channel,})( rhs_ex ); // [NC, r, c,
CH] ==> [NC, r*c*CH]
1334
1335         auto flatten_output = rhs_ex_flatten * lhs_ex_as_col; // [NC, BS * new_row * new_col]
1336
1337         auto tr_output = transpose( flatten_output ); // [BS*new_row*new_col, NC]
1338
1339         auto ans = reshape({row_output, col_output, new_channel})( tr_output );
1340
1341         return ans;
1342     };
1343 }
1344

```

```

1345     }
1346
1347
1348
1349
1350
1351
1352
1353     template< typename T > requires std::floating_point<T>
1354     inline auto drop_out( T const factor ) noexcept
1355     {
1356         better_assert( factor < T{1}, "Expecting drop out rate less than 1, but got factor = ", factor
1357     );
1358         better_assert( factor > T{0}, "Expecting drop out rate greater than 0, but got factor = ",
1359         factor );
1360
1361         std::shared_ptr<std::any> mask = std::make_shared<std::any>();
1362         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1363         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
1364
1365         return [factor, mask, forward_cache, backward_cache]<Expression Ex>( Ex const& ex ) noexcept
1366         {
1367             return make_unary_operator
1368             (
1369                 [factor, mask, forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
1370                 {
1371                     typedef typename Tsor::value_type value_type;
1372
1373                     if ( learning_phase == 0 ) // defined in 'config.hpp'
1374                         return input;
1375
1376                     std::any& mask_ = *mask;
1377                     // first run, initialize mask
1378                     if ( !mask_.has_value() )
1379                     {
1380                         Tsor const random_tensor = random<value_type>( input.shape() );
1381                         Tsor mask__( input.shape() );
1382                         for ( auto idx : range( input.size() ) )
1383                             if ( random_tensor[ idx ] > factor )
1384                                 mask__[ idx ] = 1;
1385                         mask_ = mask__; // initialize
1386                     }
1387
1388                     Tsor& mask__ = std::any_cast<Tsor&>( mask_ );
1389
1390                     Tsor& ans = context_cast<Tsor>( forward_cache );
1391                     ans.deep_copy( input );
1392
1393                     for ( auto idx : range( input.size() ) )
1394                         ans[idx] *= mask__[idx] / (value_type{1} - factor);
1395                     return ans;
1396                 },
1397                 [mask, backward_cache]<Tensor Tsor>( Tsor const&, Tsor const&, Tsor const& grad )
1398                 noexcept
1399                 {
1400                     if ( learning_phase == 0 ) // defined in 'config.hpp'
1401                         return grad;
1402
1403                     Tsor& mask__ = std::any_cast<Tsor&>( *mask );
1404
1405                     Tsor& ans = context_cast<Tsor>( backward_cache );
1406                     ans.deep_copy( grad );
1407
1408                     for ( auto idx : range( grad.size() ) )
1409                         ans[idx] *= mask__[idx];
1410                     return ans;
1411                 },
1412                 "Dropout"
1413             )( ex );
1414         };
1415     }
1416
1417     template< typename T > requires std::floating_point<T>
1418     inline auto dropout( T const factor ) noexcept
1419     {
1420         return drop_out( factor );
1421     }
1422
1423
1424     namespace
1425     {
1426
1427         struct max_pooling_2d_context
1428         {
1429
1430             auto make_forward() const noexcept
1431             {

```

```

1432         return []( unsigned long stride, std::shared_ptr<std::any> mask,
1433         std::shared_ptr<std::any> forward_cache ) noexcept
1434         {
1435             return [=]<Tensor Tsor>( Tsor const& input ) noexcept
1436             {
1437                 typedef typename Tsor::value_type value_type;
1438                 better_assert( input.ndim() == 4, "Expecting a 4D tensor, but got ",
1439                 input.ndim() );
1440
1441                 Tsor& mask__ = context_cast<Tsor>( mask );
1442                 mask__.resize( input.shape() );
1443
1444                 std::vector<unsigned long> shape = input.shape();
1445                 auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
1446                 shape[2], shape[3]);
1447
1448                 Tsor input_ = input;
1449                 view_4d<value_type> ts{ input_.data(), batch_size, row, col, channel };
1450                 view_4d<value_type> tm{ mask__.data(), batch_size, row, col, channel };
1451
1452                 Tsor& ans = context_cast<Tsor>( forward_cache );
1453                 ans.resize( {batch_size, row/stride, col/stride, channel} );
1454                 view_4d<value_type> t1{ ans.data(), batch_size, row/stride, col/stride, channel
1455                 };
1456
1457                 for ( auto bs : range(batch_size) )
1458                     for ( auto r : range(row/stride) ) // row for t1
1459                         for ( auto c : range(col/stride) ) // col for t1
1460                             for ( auto ch : range(channel) )
1461                                 {
1462                                     unsigned long current_row_max = r * stride;
1463                                     unsigned long current_col_max = c * stride;
1464                                     for ( auto _r : range( (r*stride), ((r*stride)+stride) ) ) //
1465                                     row for ts
1466                                         for ( auto _c : range( (c*stride), ((c*stride)+stride) ) )
1467                                         // col for ts
1468                                             {
1469                                                 if ( ts[bs][_r][_c][ch] >
1470                                                 ts[bs][current_row_max][current_col_max][ch] )
1471                                                 {
1472                                                     current_row_max = _r;
1473                                                     current_col_max = _c;
1474                                                 }
1475                                                 tm[bs][current_row_max][current_col_max][ch] = 1.0; //mark as
1476                                                 max
1477                                                 t1[bs][r][c][ch] =
1478                                                 ts[bs][current_row_max][current_col_max][ch]; // update value
1479                                             }
1480                                     return ans;
1481                                 };
1482                     };
1483             };
1484         };
1485
1486         auto make_backward() const noexcept
1487         {
1488             return []( unsigned long stride, std::shared_ptr<std::any> mask,
1489             std::shared_ptr<std::any> backward_cache ) noexcept
1490             {
1491                 return [=]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad )
1492                 noexcept
1493                 {
1494                     typedef typename Tsor::value_type value_type;
1495                     std::vector<unsigned long> const& shape = input.shape();
1496                     auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
1497                     shape[2], shape[3]);
1498
1499                     Tsor& mask__ = std::any_cast<Tsor&>( *mask );
1500                     view_4d<value_type> tm{ mask__.data(), batch_size, row, col, channel };
1501
1502                     Tsor& ans = context_cast<Tsor>( backward_cache );
1503                     ans.resize( input.shape() );
1504
1505                     view_4d<value_type> ta{ ans.data(), batch_size, row, col, channel };
1506
1507                     Tsor grad_ = grad;
1508                     view_4d<value_type> tg{ grad_.data(), batch_size, row/stride, col/stride,
1509                     channel };
1510
1511                     for ( auto bs : range( batch_size ) )
1512                         for ( auto r : range( row ) )
1513                             for ( auto c : range( col ) )
1514                                 for ( auto ch : range( channel ) )
1515                                     if ( std::abs(tm[bs][r][c][ch] - 1.0) < 1.0e-5 )
1516                                         ta[bs][r][c][ch] = tg[bs][r/stride][c/stride][ch];
1517
1518                     return ans;
1519                 };
1520             };
1521         };

```

```

1506         };
1507     };
1508 }
1509
1510 }; // max_pooling_2d_context
1511
1512 } // anonymous namespace
1513
1514
1515 // comment: maybe using function 'reduce' to reduce the cod complexity? at a price of
performance?
1516 inline auto max_pooling_2d( unsigned long stride ) noexcept
1517 {
1518     better_assert( stride > 1, "Expecting max_pooling_2d stride greater than 1, but got ", stride
);
1519
1520     std::shared_ptr<std::any> mask = std::make_shared<std::any>();
1521     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1522     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
1523
1524     return [stride, mask, forward_cache, backward_cache]<Expression Ex>( Ex const& ex ) noexcept
1525     {
1526         return make_unary_operator
1527         (
1528             max_pooling_2d_context{}.make_forward()( stride, mask, forward_cache ),
1529             max_pooling_2d_context{}.make_backward()( stride, mask, backward_cache ),
1530             "MaxPooling2D",
1531             [=]( std::vector<unsigned long> const& shape ) noexcept
1532             {
1533                 better_assert( shape.size()==4, fmt::format( "expecting shape size of 4, but got
{}", shape.size() ) );
1534                 return std::vector<unsigned long>{ {shape[0], shape[1]/stride, shape[2]/stride,
shape[3]} };
1535             }
1536         )( ex );
1537     };
1538 }
1539
1540 inline auto average_pooling_2d( unsigned long stride ) noexcept
1541 {
1542     better_assert( stride > 1, "Expecting average_pooling_2d stride greater than 1, but got ",
stride );
1543
1544     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1545     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
1546
1547     return [stride, forward_cache, backward_cache]<Expression Ex>( Ex const& ex ) noexcept
1548     {
1549         return make_unary_operator
1550         (
1551             [stride, forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept // [BS, R, C, CH]
--> [BS, R/s, C/s, CH]
1552         {
1553             typedef typename Tsor::value_type value_type;
1554             better_assert( input.ndim() == 4, "Expecting a 4D tensor, but got ", input.ndim()
);
1555
1556             std::vector<unsigned long> shape = input.shape();
1557             auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
shape[2], shape[3]);
1558             Tsor input_ = input;
1559             view_4d<value_type> ts{ input_.data(), batch_size, row, col, channel };
1560
1561             Tsor& ans = context_cast<Tsor>( forward_cache );
1562             ans.resize( {batch_size, row/stride, col/stride, channel} );
1563             std::fill( ans.begin(), ans.end(), value_type{0} );
1564
1565             view_4d<value_type> t1{ ans.data(), batch_size, row/stride, col/stride, channel };
1566
1567             value_type const factor = value_type{1} / static_cast<value_type>(stride*stride);
1568             for ( auto bs : range(batch_size) )
1569                 for ( auto r : range(row/stride) ) // row for t1
1570                     for ( auto c : range(col/stride) ) // col for t1
1571                         for ( auto ch : range(channel) )
1572                             for ( auto _r : range( (r*stride), ((r*stride)+stride) ) ) // row
for ts
1573                                 for ( auto _c : range( (c*stride), ((c*stride)+stride) ) ) //
col for ts
1574                                     t1[bs][r][c][ch] += ts[bs][_r][_c][ch] * factor;
1575             return ans;
1576         },
1577         [stride, backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad
) noexcept
1578     {
1579         typedef typename Tsor::value_type value_type;
1580         std::vector<unsigned long> const& shape = input.shape();
1581         auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],

```



```

    shape[2], shape[3]);
1582
1583     Tsor& ans = context_cast<Tsor>( backward_cache );
1584     ans.resize( input.shape() );
1585
1586     view_4d<value_type> ta{ ans.data(), batch_size, row, col, channel };
1587
1588     Tsor grad_ = grad;
1589     view_4d<value_type> tg{ grad_.data(), batch_size, row/stride, col/stride, channel
};
1590
1591     value_type const factor = value_type{1} / static_cast<value_type>(stride*stride);
1592     for ( auto bs : range( batch_size ) )
1593         for ( auto r : range( row ) )
1594             for ( auto c : range( col ) )
1595                 for ( auto ch : range( channel ) )
1596                     ta[bs][r][c][ch] = factor * tg[bs][r/stride][c/stride][ch];
1597     return ans;
1598 },
1599 "AveragePooling2D",
1600 [=]( std::vector<unsigned long> const& shape ) noexcept
1601 {
1602     better_assert( shape.size()==4, fmt::format( "expecting shape size of 4, but got
{}", shape.size() ) );
1603     return std::vector<unsigned long>{ {shape[0], shape[1]/stride, shape[2]/stride,
shape[3]} };
1604 }
1605 )( ex );
1606 };
1607 }
1608
1609 namespace
1610 {
1611     struct up_sampling_2d_context
1612     {
1613         auto make_forward() const noexcept
1614         {
1615             return [=]( unsigned long stride, std::shared_ptr<std::any> forward_cache ) noexcept
1616             {
1617                 return [=]<Tensor Tsor>( Tsor const& input ) noexcept
1618                 {
1619                     typedef typename Tsor::value_type value_type;
1620                     better_assert( input.ndim() == 4, "Expecting a 4D tensor, but got ",
input.ndim() );
1621
1622                     std::vector<unsigned long> shape = input.shape();
1623                     auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
shape[2], shape[3]);
1624
1625                     Tsor input_ = input;
1626                     view_4d<value_type> ts{ input_.data(), batch_size, row, col, channel };
1627
1628                     Tsor& ans = context_cast<Tsor>( forward_cache );
1629                     ans.resize( {batch_size, row*stride, col*stride, channel} );
1630                     std::fill( ans.begin(), ans.end(), value_type{0} );
1631
1632                     view_4d<value_type> t1{ ans.data(), batch_size, row*stride, col*stride, channel
};
1633
1634                     for ( auto bs : range(batch_size) )
1635                         for ( auto r : range(row) ) // row for ts
1636                             for ( auto c : range(col) ) // col for ts
1637                                 for ( auto ch : range(channel) )
1638                                     for ( auto _r : range( (r*stride), ((r*stride)+stride) ) ) //
row for t1
1639                                         for ( auto _c : range( (c*stride), ((c*stride)+stride) ) )
// col for t1
1640                                             t1[bs][_r][_c][ch] = ts[bs][r][c][ch];
1641
1642                     return ans;
1643                 };
1644             };
1645         }
1646     };
1647
1648     auto make_backward() const noexcept
1649     {
1650         return [=]( unsigned long stride, std::shared_ptr<std::any> backward_cache ) noexcept
1651         {
1652             return [=]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad )
noexcept
1653             {
1654                 typedef typename Tsor::value_type value_type;
1655                 std::vector<unsigned long> const& shape = input.shape();
1656                 auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
shape[2], shape[3]);
1657
1658                 Tsor& ans = context_cast<Tsor>( backward_cache );
1659                 ans.resize( input.shape() );
1660                 std::fill( ans.begin(), ans.end(), value_type{0} );

```

```

1658
1659         view_4d<value_type> ta{ ans.data(), batch_size, row, col, channel };
1660
1661         Tsor grad_ = grad;
1662         view_4d<value_type> tg{ grad_.data(), batch_size, row*stride, col*stride,
channel };
1663
1664         for ( auto bs : range( batch_size ) )
1665             for ( auto r : range( row ) )
1666                 for ( auto c : range( col ) )
1667                     for ( auto ch : range( channel ) )
1668                         for ( auto _r : range( (r*stride), ((r*stride)+stride) ) ) //
row for tg
1669                             for ( auto _c : range( (c*stride), ((c*stride)+stride) ) )
// col for tg
1670                                 ta[bs][r][c][ch] += tg[bs][_r][_c][ch];
1671
1672         return ans;
1673     };
1674 }
1675 }; // up_sampling_2d_context
1676
1677 } // anonymous namespace
1678
1679 inline auto up_sampling_2d( unsigned long stride ) noexcept
1680 {
1681     better_assert( stride > 1, "Expecting up_sampling_pooling_2d stride greater than 1, but got ",
stride );
1682
1683     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1684     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
1685
1686     return [stride, forward_cache, backward_cache]<Expression Ex>( Ex const& ex ) noexcept
1687     {
1688         return make_unary_operator
1689         (
1690             up_sampling_2d_context{}.make_forward()( stride, forward_cache ),
1691             up_sampling_2d_context{}.make_backward()( stride, backward_cache ),
1692             "UpSampling2D",
1693             [=]( std::vector<unsigned long> const& shape ) noexcept
1694             {
1695                 better_assert( shape.size()==4, fmt::format( "expecting shape size of 4, but got
{}", shape.size() ) );
1696                 return std::vector<unsigned long>{ {shape[0], shape[1]*stride, shape[2]*stride,
shape[3]} };
1697             }
1698         )( ex );
1699     };
1700 }
1701
1702 // an alias name
1703 inline auto upsampling_2d( unsigned long stride ) noexcept
1704 {
1705     return up_sampling_2d( stride );
1706 }
1707
1708
1709
1710 template< typename T=double > requires std::floating_point<T>
1711 inline auto normalization_batch( T const momentum=0.98 ) noexcept
1712 {
1713     std::shared_ptr<std::any> global_average_cache = std::make_shared<std::any>();
1714     std::shared_ptr<std::any> global_variance_cache = std::make_shared<std::any>();
1715     std::shared_ptr<std::any> average_cache = std::make_shared<std::any>();
1716     std::shared_ptr<std::any> variance_cache = std::make_shared<std::any>();
1717     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1718     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
1719
1720     return [=]<Expression Ex>( Ex const& ex ) noexcept
1721     {
1722         return make_unary_operator
1723         (
1724             [=]<Tensor Tsor>( Tsor const& input ) noexcept
1725             {
1726                 better_assert( input.ndim() > 1, "normalization_batch requires input dimension at
least 2, got ", input.ndim() );
1727
1728                 typedef typename Tsor::value_type value_type;
1729                 //typedef typename Tsor::allocator allocator;
1730
1731                 std::vector<unsigned long> const& shape = input.shape();
1732                 unsigned long const channels = *(shape.rbegin());
1733                 unsigned long const rest_dims = input.size() / channels;
1734
1735                 view_2d<value_type> input_{ input.data(), rest_dims, channels };
1736
1737                 // case of prediction phase, in this phase, the batch size could be 1, and it is

```

```

not possible to calculate the variance
1738         if ( learning_phase == 0 ) // defined in 'config.hpp'
1739         {
1740             // fix for the special case when prediction is executed before the training,
1741             typically in a GAN
1742             Tsor& global_average_test = context_cast<Tsor>( global_average_cache );
1743             if ( global_average_test.empty() )
1744                 return input;
1745             // normal case. i.e., the global_average_cache and global_variance_cache are
1746             not empty
1747             Tsor& global_average = context_extract<Tsor>( global_average_cache );
1748             Tsor& global_variance = context_extract<Tsor>( global_variance_cache );
1749             Tsor& ans = context_cast<Tsor>( forward_cache, zeros_like( input ) );
1750             ans.resize( input.shape() ); // well, the batch sizes for training and for
1751             prediction are not necessarily same
1752             view_2d<value_type> ans_( ans.data(), rest_dims, channels );
1753             {
1754                 for ( auto r : range( rest_dims ) )
1755                     for ( auto c : range( channels ) )
1756                         ans_[r][c] = ( input_[r][c] - global_average[c] ) / std::sqrt(
1757                             global_variance[c] + eps );
1758             }
1759             return ans;
1760         }
1761         //calculate average along the last channel
1762         Tsor& average = context_cast<Tsor>( average_cache );
1763         {
1764             average.resize( {channels,} );
1765             std::fill( average.begin(), average.end(), value_type{0} );
1766             for ( auto idx : range( rest_dims ) )
1767                 for ( auto jdx : range( channels ) )
1768                     average[jdx] += input_[idx][jdx];
1769             average /= static_cast<value_type>(rest_dims);
1770         }
1771         //calculate Variance along the last channel
1772         Tsor& variance = context_cast<Tsor>( variance_cache );
1773         {
1774             variance.resize( {channels,} );
1775             std::fill( variance.begin(), variance.end(), value_type{0} );
1776             for ( auto idx : range( rest_dims ) )
1777                 for ( auto jdx : range( channels ) )
1778                     variance[jdx] += std::pow( input_[idx][jdx] - average[jdx], 2 );
1779             variance /= static_cast<value_type>( rest_dims );
1780         }
1781         Tsor& ans = context_cast<Tsor>( forward_cache );
1782         ans.resize( input.shape() ); // the batch sizes for training and for prediction are
1783         not necessarily same
1784         view_2d<value_type> ans_( ans.data(), rest_dims, channels );
1785         {
1786             for ( auto idx : range( rest_dims ) )
1787                 for ( auto jdx : range( channels ) )
1788                     ans_[idx][jdx] = ( input_[idx][jdx] - average[jdx] ) / std::sqrt(
1789                         variance[jdx] + eps );
1790         }
1791         // update global average and global variance
1792         {
1793             Tsor& global_average = context_cast<Tsor>( global_average_cache, zeros_like(
1794                 average ) );
1795             // Note: No obvious different is observed between initializing global_variance
1796             to zeros and to ones with MNIST example:
1797             //      initializing global_variance to zeros, after 10 epochs mnist gives an
1798             error of 0.026
1799             //      initializing global_variance to ones, after 10 epochs mnist gives an
1800             error of 0.028
1801             Tsor& global_variance = context_cast<Tsor>( global_variance_cache, zeros_like(
1802                 variance ) );
1803             //Tsor& global_variance = context_cast<Tsor>( global_variance_cache, ones_like(
1804                 variance ) );
1805             for ( auto idx : range( global_average.size() ) )
1806             {
1807                 global_average[idx] = global_average[idx] * momentum + average[idx] * ( 1.0
1808                 - momentum );
1809                 global_variance[idx] = global_variance[idx] * momentum + variance[idx] * (
1810                 1.0 - momentum );
1811             }
1812         }

```

```

1810
1811         return ans;
1812     },
1813
1814     [=]<Tensor T>( T& input, T& variance, T& grad ) noexcept
1815     {
1816         typedef typename T::value_type value_type;
1817         T& variance = context_extract<T>( variance_cache );
1818
1819         std::vector<unsigned long> const& shape = input.shape();
1820         unsigned long const channels = *(shape.rbegin());
1821         unsigned long const rest_dims = input.size() / channels;
1822
1823         T& ans = context_cast<T>( backward_cache, zeros_like( input ) );
1824         view_2d<value_type> ans{ans.data(), rest_dims, channels };
1825         view_2d<value_type> grad{grad.data(), rest_dims, channels };
1826         for ( auto r : range( rest_dims ) )
1827             for ( auto c : range( channels ) )
1828                 ans[r][c] = grad[r][c] / std::sqrt( variance[c] + eps );
1829         return ans;
1830     },
1831     "Normalization"
1832 )( ex );
1833 };
1834 }
1835
1836
1837
1838 template< typename T > requires std::floating_point<T>
1839 inline auto batch_normalization( T const momentum=0.98 ) noexcept
1840 {
1841     return [=]<Expression Ex, Variable Va>( Ex const& ex, Va const& gamma, Va const& beta )
1842     noexcept
1843     {
1844         return elementwise_product( normalization_batch(momentum)(ex), gamma ) + beta; // multiply
1845         and sum along the batch: normalization is of shape [BS, R, C, CH], gamma/beta are of shape [R, C, CH]
1846     };
1847 }
1848
1849 //
1850 // example:
1851 //
1852 //     variable<tensor<float>> a {... };
1853 //     variable<tensor<float>> b {... };
1854 //     auto cab = concatenate( a, b )();
1855 //
1856 template< Expression Lhs_Expression, Expression Rhs_Expression >
1857 auto constexpr concatenate( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
1858 {
1859     return [&]( unsigned long axe = -1 ) noexcept
1860     {
1861         return make_binary_operator
1862         (
1863             [axe]<Tensor T>( T& lhs_tensor, T& rhs_tensor ) noexcept
1864             {
1865                 return concatenate( lhs_tensor, rhs_tensor, axe );
1866             },
1867             [axe]<Tensor T>( T& lhs_input, T& rhs_input, T& grad ) noexcept
1868             const grad ) noexcept
1869             {
1870                 typedef typename T::value_type value_type;
1871
1872                 T l_ans{ lhs_input.shape() };
1873                 T r_ans{ rhs_input.shape() };
1874                 better_assert( l_ans.size() + r_ans.size() == grad.size(), "size mismatch: lhs
1875 size is ", l_ans.size(), " rhs size is ", r_ans.size(), " and grad size is ", grad.size(),
1876 " with lhs dim is ", l_ans.ndim(), " and rhs dim is ",
1877 r_ans.ndim() );
1878
1879                 // 2D view of grad
1880                 unsigned long const ax = (axe == (unsigned long)(-1)) ? grad.ndim()-1 : axe;
1881                 unsigned long const g_col = std::accumulate( grad.shape().begin()+ax,
1882 grad.shape().end(), 1UL, []( unsigned long x, unsigned long y ){ return x*y; } );
1883                 unsigned long const g_row = grad.size() / g_col;
1884                 view_2d<value_type> v_g{ grad.data(), g_row, g_col };
1885
1886                 // 2D view of l_ans
1887                 unsigned long const lhs_row = g_row;
1888                 unsigned long const lhs_col = lhs_input.size() / lhs_row;
1889                 view_2d<value_type> v_l{ l_ans.data(), lhs_row, lhs_col };
1890
1891                 // 2D view of r_ans
1892                 unsigned long const rhs_row = g_row;
1893                 unsigned long const rhs_col = rhs_input.size() / rhs_row;
1894                 view_2d<value_type> v_r{ r_ans.data(), rhs_row, rhs_col };

```

```

1891         better_assert( g_col == lhs_col + rhs_col, "last dimension not agree" );
1892
1893         for ( unsigned long idx = 0; idx != g_row; ++idx )
1894         {
1895             std::copy( v_g[idx], v_g[idx]+lhs_col, v_l[idx] );           // fill idx-th row
1896         of 'v_l'
1897             std::copy( v_g[idx]+lhs_col, v_g[idx]+g_col, v_r[idx] );     // fill idx-th row
1898         of 'v_r'
1899         }
1900
1901         return std::make_tuple( l_ans, r_ans );
1902     },
1903     "Concatenate",
1904     [axe]( std::vector<unsigned long> const& l, std::vector<unsigned long> const& r )
1905 noexcept
1906     {
1907         better_assert( l.size() == r.size(), fmt::format( "expecting of same size, but
1908 lhs.size is {} and rhs.size is {}.", l.size(), r.size() ) );
1909         // more assertion ?
1910         std::vector<unsigned long> ans = l;
1911         if ( axe > ans.size() ) axe = ans.size() - 1;
1912         ans[axe] += r[axe];
1913         return ans;
1914     }
1915 ) ( lhs_ex, rhs_ex );
1916 };
1917
1918 // just to keep this interface agrees with Keras
1919 inline auto concatenate( unsigned long axe = -1 )
1920 {
1921     return [=]< Expression Lhs_Expression, Expression Rhs_Expression >( Lhs_Expression const&
1922 lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
1923     {
1924         return concatenate( lhs_ex, rhs_ex ) ( axe );
1925     };
1926 }
1927
1928 // alias of 'concatenate'
1929 template< Expression Lhs_Expression, Expression Rhs_Expression >
1930 auto constexpr concat( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
1931 {
1932     return concatenate( lhs_ex, rhs_ex ) ();
1933 }
1934
1935 // alias of 'concatenate'
1936 inline auto concat( unsigned long axe = -1 )
1937 {
1938     return concatenate( axe );
1939 }
1940
1941 template< Expression Lhs_Expression, Expression Rhs_Expression >
1942 auto constexpr maximum( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
1943 {
1944     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1945     std::shared_ptr<std::any> mask_cache = std::make_shared<std::any>();
1946     std::shared_ptr<std::any> backward_cache_lhs = std::make_shared<std::any>();
1947     std::shared_ptr<std::any> backward_cache_rhs = std::make_shared<std::any>();
1948     return make_binary_operator
1949     (
1950         [=]<Tensor Tsor>( Tsor const& lhs_tensor, Tsor const& rhs_tensor ) noexcept
1951         {
1952             better_assert( lhs_tensor.shape() == rhs_tensor.shape(), "tensor shape mismatch." );
1953
1954             Tsor& ans = context_cast<Tsor>( forward_cache );
1955             ans.resize( lhs_tensor.shape() );
1956             Tsor& mask = context_cast<Tsor>( mask_cache ); // 1 if lhs element is larger, 0 if rhs
1957 element is larger
1958             mask.resize( lhs_tensor.shape() );
1959
1960             for_each( lhs_tensor.begin(), lhs_tensor.end(), rhs_tensor.begin(), ans.begin(),
1961 mask.begin(), []( auto const l, auto const r, auto& a, auto& m ) { m = l > r ? 1.0 : 0.0; a = l > r
1962 ? l : r; } );
1963
1964             return ans;
1965         },
1966         [=]<Tensor Tsor>( Tsor const& lhs_input, Tsor const& rhs_input, Tsor const&, Tsor const&
1967 grad ) noexcept
1968         {
1969             Tsor& mask = context_cast<Tsor>( mask_cache ); // 1 if lhs element is larger, 0 if rhs
1970 element is larger
1971
1972             Tsor& l_ans = context_cast<Tsor>( backward_cache_lhs );
1973             l_ans.resize( lhs_input.shape() );
1974             Tsor& r_ans = context_cast<Tsor>( backward_cache_rhs );

```

```

1968         r_ans.resize( rhs_input.shape() );
1969
1970         for_each( grad.begin(), grad.end(), mask.begin(), l_ans.begin(), r_ans.begin(), [](
auto const g, auto const m, auto& l, auto& r ) { if ( m > 0.5 ) { l = g; r = 0.0; } else { l = 0.0; r
= g; } } );
1971
1972         return std::make_tuple( l_ans, r_ans );
1973     },
1974     "Maximum"
1975 )( lhs_ex, rhs_ex );
1976 }
1977
1978 template< Expression Lhs_Expression, Expression Rhs_Expression >
1979 auto constexpr minimum( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
1980 {
1981     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
1982     std::shared_ptr<std::any> mask_cache = std::make_shared<std::any>();
1983     std::shared_ptr<std::any> backward_cache_lhs = std::make_shared<std::any>();
1984     std::shared_ptr<std::any> backward_cache_rhs = std::make_shared<std::any>();
1985     return make_binary_operator
1986     (
1987         [=]<Tensor Tsor>( Tsor const& lhs_tensor, Tsor const& rhs_tensor ) noexcept
1988         {
1989             better_assert( lhs_tensor.shape() == rhs_tensor.shape(), "tensor shape mismatch." );
1990
1991             Tsor& ans = context_cast<Tsor>( forward_cache );
1992             ans.resize( lhs_tensor.shape() );
1993             Tsor& mask = context_cast<Tsor>( mask_cache ); // 1 if lhs element is larger, 0 if rhs
element is larger
1994             mask.resize( lhs_tensor.shape() );
1995
1996             for_each( lhs_tensor.begin(), lhs_tensor.end(), rhs_tensor.begin(), ans.begin(),
mask.begin(), []( auto const l, auto const r, auto& a, auto& m ) { m = l > r ? 0.0: 1.0 ; a = l > r
? r: l; } );
1997
1998             return ans;
1999         },
2000         [=]<Tensor Tsor>( Tsor const& lhs_input, Tsor const& rhs_input, Tsor const&, Tsor const&
grad ) noexcept
2001         {
2002             Tsor& mask = context_cast<Tsor>( mask_cache ); // 1 if lhs element is larger, 0 if rhs
element is larger
2003
2004             Tsor& l_ans = context_cast<Tsor>( backward_cache_lhs );
2005             l_ans.resize( lhs_input.shape() );
2006             Tsor& r_ans = context_cast<Tsor>( backward_cache_rhs );
2007             r_ans.resize( rhs_input.shape() );
2008
2009             for_each( grad.begin(), grad.end(), mask.begin(), l_ans.begin(), r_ans.begin(), [](
auto const g, auto const m, auto& l, auto& r ) { if ( m < 0.5 ) { l = g; r = 0.0; } else { l = 0.0; r
= g; } } );
2010
2011             return std::make_tuple( l_ans, r_ans );
2012         },
2013         "Minumum"
2014     )( lhs_ex, rhs_ex );
2015 }
2016
2017 template< Expression Lhs_Expression, Expression Rhs_Expression >
2018 auto constexpr atan2( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
2019 {
2020     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2021     std::shared_ptr<std::any> backward_cache_lhs = std::make_shared<std::any>();
2022     std::shared_ptr<std::any> backward_cache_rhs = std::make_shared<std::any>();
2023     return make_binary_operator
2024     (
2025         [=]<Tensor Tsor>( Tsor const& lhs_tensor, Tsor const& rhs_tensor ) noexcept
2026         {
2027             better_assert( lhs_tensor.shape() == rhs_tensor.shape(), "tensor shape mismatch." );
2028             Tsor& ans = context_cast<Tsor>( forward_cache );
2029             ans.resize( lhs_tensor.shape() );
2030             for_each( lhs_tensor.begin(), lhs_tensor.end(), rhs_tensor.begin(), ans.begin(), [](
auto const l, auto const r, auto& a ) { a = std::atan2(l, r); } );
2031
2032             return ans;
2033         },
2034         [=]<Tensor Tsor>( Tsor const& lhs_input, Tsor const& rhs_input, Tsor const&, Tsor const&
grad ) noexcept
2035         {
2036             Tsor& l_ans = context_cast<Tsor>( backward_cache_lhs );
2037             l_ans.resize( lhs_input.shape() );
2038             Tsor& r_ans = context_cast<Tsor>( backward_cache_rhs );
2039             r_ans.resize( rhs_input.shape() );
2040             for_each( grad.begin(), grad.end(), l_ans.begin(), r_ans.begin(), lhs_input.begin(),
rhs_input.begin(), []( auto const g, auto& l, auto& r, auto const x, auto const y ) { auto const c =
x*x+y*y; l = -g*y/c; r = g*x/c; } );
2041
2042             return std::make_tuple( l_ans, r_ans );
2043         },
2044     )

```

```

2045         "Arctan2"
2046     )( lhs_ex, rhs_ex );
2047 }
2048
2049
2062 template< typename T=float > requires std::floating_point<T>
2063 inline auto random_normal_like( T mean = 0.0, T stddev = 1.0 ) noexcept
2064 {
2065     return [=]<Expression Ex>(Ex const& ex ) noexcept
2066     {
2067         return make_unary_operator
2068         (
2069             [=]<Tensor Tsr>( Tsr const& tsor ) noexcept
2070             {
2071                 return randn_like( tsor, mean, stddev );
2072             },
2073             [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad ) noexcept
2074             {
2075                 return zeros_like( grad );
2076             },
2077             "RandomNormalLike"
2078         )(ex);
2079     };
2080 }
2081
2091 template< Expression Ex>
2092 auto ones_like( Ex const& ex ) noexcept
2093 {
2094     return make_unary_operator
2095     (
2096         [<Tensor Tsr>( Tsr const& tsor ) noexcept { return ones_like( tsor ); },
2097         [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad ) noexcept { return
2098         zeros_like( grad ); },
2099         "OnesLike"
2100     )(ex);
2101 }
2102
2111 template< Expression Ex>
2112 auto zeros_like( Ex const& ex ) noexcept
2113 {
2114     return make_unary_operator
2115     (
2116         [<Tensor Tsr>( Tsr const& tsor ) noexcept { return zeros_like( tsor ); },
2117         [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad ) noexcept { return
2118         zeros_like( grad ); },
2119         "ZerosLike"
2120     )(ex);
2121 }
2122
2136 template< Expression Lhs_Expression, Expression Rhs_Expression, std::floating_point FP >
2137 auto constexpr equal( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex, FP threshold=0.5
2138 ) noexcept
2139 {
2140     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2141     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
2142     return make_binary_operator
2143     (
2144         [=]<Tensor Tsr>( Tsr const& lhs_tensor, Tsr const& rhs_tensor ) noexcept
2145         {
2146             typedef typename Tsr::value_type value_type;
2147             better_assert( lhs_tensor.shape() == rhs_tensor.shape(), "equal: tensor shape
2148             mismatch." );
2149             Tsr& ans = context_cast<Tsr>( forward_cache );
2150             ans.resize( lhs_tensor.shape() );
2151             for_each( lhs_tensor.begin(), lhs_tensor.end(), rhs_tensor.begin(), ans.begin(),
2152             [threshold]( auto l, auto r, auto& v ){ v = (std::abs(l-r) > threshold) ? value_type{0} :
2153             value_type{1}; } );
2154             return ans;
2155         },
2156         [=]<Tensor Tsr>( Tsr const& lhs_input, Tsr const& rhs_input, Tsr const&, Tsr const&
2157         grad ) noexcept
2158         {
2159             typedef typename Tsr::value_type value_type;
2160             Tsr& ans = context_cast<Tsr>( backward_cache );
2161             std::fill( ans.begin(), ans.end(), value_type{0} );
2162             return std::make_tuple( ans, ans );
2163         },
2164         "Equal"
2165     )( lhs_ex, rhs_ex );
2166 }
2167
2176 template <Expression Ex>
2177 auto constexpr sign( Ex const& ex ) noexcept
2178 {
2179     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2180     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();

```

```

2181         return make_unary_operator
2182     (
2183         [=]<Tensor Tsr>( Tsr const& input ) noexcept
2184     {
2185         typedef typename Tsr::value_type value_type;
2186         Tsr& ans = context_cast<Tsr>( forward_cache );
2187         ans.resize( input.shape() );
2188         for_each( input.begin(), input.end(), ans.begin(), []( auto x, auto& v ){ v =
2189 (value_type{0} < x) - (x < value_type{0}); } );
2190         return ans;
2191     },
2192     [=]<Tensor Tsr>( Tsr const&input, Tsr const&, Tsr const& grad ) noexcept
2193 {
2194     typedef typename Tsr::value_type value_type;
2195     Tsr& ans = context_cast<Tsr>( backward_cache );
2196     ans.resize( input.shape() );
2197     std::fill( ans.begin(), ans.end(), value_type{0} ); //TF gives zeros, we follow TF here
2198     return ans;
2199 },
2200 "Sign"
2201 )( ex );
2202 };
2203
2204 namespace
2205 {
2206     struct zero_padding_2d_context
2207     {
2208         auto make_forward() const noexcept
2209         {
2210             return []( unsigned long top, unsigned long bottom, unsigned long left, unsigned long
2211 right, std::shared_ptr<std::any> forward_cache ) noexcept
2212             {
2213                 return [=]<Tensor Tsr>( Tsr const& input ) noexcept
2214                 {
2215                     typedef typename Tsr::value_type value_type;
2216                     better_assert( input.ndim() == 4, "Expecting a 4D tensor, but got ",
2217 input.ndim() );
2218
2219                     // 4D view of input tensor
2220                     std::vector<unsigned long> shape = input.shape();
2221                     auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
2222 shape[2], shape[3]);
2223                     Tsr input_ = input;
2224                     view_4d<value_type> ts{ input_.data(), batch_size, row, col, channel };
2225
2226                     // 4D view of output tensor
2227                     Tsr& ans = context_cast<Tsr>( forward_cache );
2228                     ans.resize( {batch_size, top+row+bottom, left+col+right, channel} );
2229                     view_4d<value_type> ta{ ans.data(), batch_size, top+row+bottom, left+col+right,
2230 channel };
2231
2232                     for ( auto bs : range( batch_size ) )
2233                     for ( auto r : range( row ) )
2234                     for ( auto c : range( col ) )
2235                     for ( auto ch : range( channel ) )
2236                         ta[bs][top+r][left+c][ch] = ts[bs][r][c][ch];
2237
2238                     return ans;
2239                 };
2240             };
2241         }
2242     };
2243
2244     auto make_backward() const noexcept
2245     {
2246         return []( unsigned long top, unsigned long bottom, unsigned long left, unsigned long
2247 right, std::shared_ptr<std::any> backward_cache ) noexcept
2248         {
2249             return [=]<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr const& grad )
2250 noexcept
2251             {
2252                 typedef typename Tsr::value_type value_type;
2253                 std::vector<unsigned long> const& shape = input.shape();
2254                 auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
2255 shape[2], shape[3]);
2256
2257                 Tsr& ans = context_cast<Tsr>( backward_cache );
2258                 ans.resize( input.shape() );
2259                 std::fill( ans.begin(), ans.end(), value_type{0} );
2260
2261                 view_4d<value_type> ta{ ans.data(), batch_size, row, col, channel };
2262
2263                 Tsr grad_ = grad;
2264                 view_4d<value_type> tg{ grad_.data(), batch_size, top+row+bottom,
2265 left+col+right, channel };
2266

```



```

2259         for ( auto bs : range( batch_size ) )
2260             for ( auto r : range( row ) )
2261                 for ( auto c : range( col ) )
2262                     for ( auto ch : range( channel ) )
2263                         ta[bs][r][c][ch] = tg[bs][r+top][c+left][ch];
2264         return ans;
2265     };
2266 };
2267 }
2268 }; // zero_padding_2d_context
2269 } // anonymous namespace
2270
2284 inline auto zero_padding_2d( std::vector<unsigned long> const& padding ) noexcept
2285 {
2286     // extracting paddings
2287     unsigned long top, bottom, left, right;
2288     if ( padding.size() == 1 )
2289         std::tie( top, bottom, left, right ) = std::make_tuple( padding[0], padding[0], padding[0],
padding[0] );
2290     else if ( padding.size() == 2 )
2291         std::tie( top, bottom, left, right ) = std::make_tuple( padding[0], padding[0], padding[1],
padding[1] );
2292     else if ( padding.size() == 4 )
2293         std::tie( top, bottom, left, right ) = std::make_tuple( padding[0], padding[1], padding[2],
padding[3] );
2294     else
2295         better_assert( false, "Expecting padding has size of 1, 2 or 4, but got: ", padding.size()
);
2296
2297     // checking extracted paddings
2298     better_assert( top >= 1, "Expecting zero_padding_2d top padding no less than 1, but got ", top
);
2299     better_assert( bottom >= 1, "Expecting zero_padding_2d bottom padding no less than 1, but got
", bottom );
2300     better_assert( left >= 1, "Expecting zero_padding_2d left padding no less than 1, but got ",
left );
2301     better_assert( right >= 1, "Expecting zero_padding_2d right padding no less than 1, but got ",
right );
2302
2303     // to avoid re-allocating memory for tensors
2304     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2305     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
2306
2307     return [top, bottom, left, right, forward_cache, backward_cache]<Expression Ex>( Ex const& ex )
noexcept
2308     {
2309         return make_unary_operator
2310         (
2311             zero_padding_2d_context{}.make_forward()( top, bottom, left, right, forward_cache ),
2312             zero_padding_2d_context{}.make_backward()( top, bottom, left, right, backward_cache ),
2313             "ZeroPadding2D",
2314             [=]( std::vector<unsigned long> const& shape ) noexcept { return std::vector<unsigned
long>( {shape[0], shape[1]+top+bottom, shape[2]+left+right, shape[3]} ); }
2315         )( ex );
2316     };
2317 }
2318
2319
2320
2321 namespace
2322 {
2323     struct cropping_2d_context
2324     {
2325         auto make_forward() const noexcept
2326         {
2327             return []( unsigned long top, unsigned long bottom, unsigned long left, unsigned long
right, std::shared_ptr<std::any> forward_cache ) noexcept
2328             {
2329                 return [=]<Tensor Tsor>( Tsor const& input ) noexcept
2330                 {
2331                     typedef typename Tsor::value_type value_type;
2332                     better_assert( input.ndim() == 4, "Expecting a 4D tensor, but got ",
input.ndim() );
2333
2334                     // check shape, not too large
2335
2336                     // 4D view of input tensor
2337                     std::vector<unsigned long> shape = input.shape();
2338                     auto const [batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
shape[2], shape[3]);
2339
2340                     Tsor input_ = input;
2341                     view_4d<value_type> ts{ input_.data(), batch_size, row, col, channel };
2342
2343                     better_assert( row-top-bottom > 0, fmt::format("Cropping2D: expecting a smaller
cropping dimension in row: row:{}, top:{}, bottom:{}, row, top, bottom ) );
2344                     better_assert( col-left-right > 0, fmt::format("Cropping2D: expecting a smaller
cropping dimension in col: col:{}, left:{}, right:{}, col, left, right ) );
2345

```

```

2344         // 4D view of output tensor
2345         Tsor& ans = context_cast<Tsor>( forward_cache );
2346         ans.resize( {batch_size, row-top-bottom, col-left-right, channel} );
2347         view_4d<value_type> ta{ ans.data(), batch_size, row-top-bottom, col-left-right,
channel };
2348
2349         for ( auto bs : range( batch_size ) )
2350             for ( auto r : range( row-top-bottom ) )
2351                 for ( auto c : range( col-left-right ) )
2352                     for ( auto ch : range( channel ) )
2353                         ta[bs][r][c][ch] = ts[bs][top+r][left+c][ch];
2354
2355         return ans;
2356     };
2357 };
2358 }
2359
2360     auto make_backward() const noexcept
2361     {
2362         return []( unsigned long top, unsigned long bottom, unsigned long left, unsigned long
right, std::shared_ptr<std::any> backward_cache ) noexcept
2363         {
2364             return [=]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad )
noexcept
2365             {
2366                 typedef typename Tsor::value_type value_type;
2367                 std::vector<unsigned long> const& shape = grad.shape();
2368                 auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
shape[2], shape[3]);
2369
2370                 Tsor& ans = context_cast<Tsor>( backward_cache );
2371                 ans.resize( input.shape() );
2372                 std::fill( ans.begin(), ans.end(), value_type{0} );
2373
2374                 view_4d<value_type> ta{ ans.data(), batch_size, row+top+bottom, col+left+right,
channel };
2375
2376                 Tsor grad_ = grad;
2377                 view_4d<value_type> tg{ grad_.data(), batch_size, row, col, channel };
2378
2379                 for ( auto bs : range( batch_size ) )
2380                     for ( auto r : range( row ) )
2381                         for ( auto c : range( col ) )
2382                             for ( auto ch : range( channel ) )
2383                                 ta[bs][r+top][c+left][ch] = tg[bs][r][c][ch];
2384
2385                 return ans;
2386             };
2387         };
2388     }; // cropping_2d_context
2389 } //anonymous namespace
2390
2391     inline auto cropping_2d( std::vector<unsigned long> const& padding ) noexcept
2392     {
2393         // extracting paddings
2394         unsigned long top, bottom, left, right;
2395         if ( padding.size() == 1 )
2396             std::tie( top, bottom, left, right ) = std::make_tuple( padding[0], padding[0], padding[0],
padding[0] );
2397         else if ( padding.size() == 2 )
2398             std::tie( top, bottom, left, right ) = std::make_tuple( padding[0], padding[0], padding[1],
padding[1] );
2399         else if ( padding.size() == 4 )
2400             std::tie( top, bottom, left, right ) = std::make_tuple( padding[0], padding[1], padding[2],
padding[3] );
2401         else
2402             better_assert( false, "Expecting padding has size of 1, 2 or 4, but got: ", padding.size()
);
2403
2404         // checking extracted paddings
2405         better_assert( top >= 1, "Expecting cropping_2d top padding no less than 1, but got ", top );
2406         better_assert( bottom >= 1, "Expecting cropping_2d bottom padding no less than 1, but got ",
bottom );
2407         better_assert( left >= 1, "Expecting cropping_2d left padding no less than 1, but got ", left
);
2408         better_assert( right >= 1, "Expecting cropping_2d right padding no less than 1, but got ",
right );
2409
2410         // to avoid re-allocating memory for tensors
2411         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2412         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
2413
2414         return [top, bottom, left, right, forward_cache, backward_cache]<Expression Ex>( Ex const& ex )
noexcept
2415         {
2416             return make_unary_operator
2417             (
2418

```

```

2431         cropping_2d_context{}.make_forward()( top, bottom, left, right, forward_cache ),
2432         cropping_2d_context{}.make_backward()( top, bottom, left, right, backward_cache ),
2433         "ZeroPadding2D",
2434         [=]( std::vector<unsigned long> const& shape ) noexcept
2435         {
2436             return std::vector<unsigned long>{ {shape[0], shape[1]-top-bottom,
shape[2]-left-right, shape[3]} };
2437         }
2438     )( ex );
2439 };
2440 }
2441
2442 namespace
2443 {
2444     inline auto detailed_sliding_2d( unsigned long const pixels, std::shared_ptr<std::any>
shift_cache,
2445                                     std::shared_ptr<std::any> forward_cache,
std::shared_ptr<std::any> backward_cache ) noexcept
2446     {
2447         return [=]<Expression Ex>( Ex const& ex ) noexcept // <- the output has been zero-padded by
n pixels
2448         {
2449             return make_unary_operator
2450             (
2451                 [=]<Tensor Tsor>( Tsor const& tsor ) noexcept
2452                 {
2453                     if (learning_phase != 1)
2454                         return tsor;
2455
2456                     typedef typename Tsor::value_type value_type;
2457                     std::vector<unsigned long> const& shape = tsor.shape();
2458                     auto const[batch_size, row, col, channel] = std::make_tuple(shape[0], shape[1],
shape[2], shape[3]);
2459                     view_4d vi{tsor.data(), batch_size, row, col, channel};
2460
2461                     tensor<long> shifts = context_cast<tensor<long>>( shift_cache );
2462                     shifts.resize( {channel, 2} );
2463                     { //generating random shifts
2464                         std::uniform_int_distribution<long> distribution( -pixels, pixels );
2465                         for ( auto& v : shifts )
2466                             v = distribution(random_generator);
2467                     }
2468                     view_2d _shifts{shifts.data(), channel, 2};
2469
2470                     Tsor& ans = context_cast<Tsor>( forward_cache );
2471                     ans.resize( tsor.shape() );
2472                     std::fill( ans.begin(), ans.end(), value_type{0} );
2473                     view_4d vo{ans.data(), batch_size, row, col, channel};
2474
2475                     for ( auto bs : range(batch_size) )
2476                     {
2477                         for ( auto ch : range( channel ) )
2478                         {
2479                             auto [row_shift, col_shift] = std::make_tuple( _shifts[ch][0],
_shifts[ch][1]);
2480
2481                             for ( auto r : range( row ) )
2482                             {
2483                                 if (r-row_shift>=0 && r-row_shift<row)
2484                                 {
2485                                     for ( auto c : range( col ) )
2486                                     {
2487                                         if (c-col_shift>=0 && c-col_shift<col )
2488                                             vo[bs][r][c][ch] =
2489                                                 vi[bs][r-row_shift][c-col_shift][ch];
2490                                     }
2491                                 }
2492                             }
2493                         }
2494                     }
2495                     return ans;
2496                 },
2497                 [=]<Tensor Tsor>( Tsor const&, Tsor const&, Tsor const& grad ) noexcept
2498                 {
2499                     typedef typename Tsor::value_type value_type;
2500                     std::vector<unsigned long> const& shape = grad.shape();
2501                     auto const[batch_size, row, col, channel] = std::make_tuple( shape[0],
shape[1], shape[2], shape[3] );
2502                     view_4d vi{grad.data(), batch_size, row, col, channel};
2503                     tensor<long> shifts = context_cast<tensor<long>>( shift_cache );
2504                     view_2d _shifts{shifts.data(), channel, 2};
2505
2506                     Tsor& ans = context_cast<Tsor>( backward_cache );
2507                     ans.resize( grad.shape() );
2508                     std::fill( ans.begin(), ans.end(), value_type{0} );
2509                     view_4d vo{ans.data(), batch_size, row, col, channel};

```

```

2510
2511         for ( auto bs : range(batch_size) )
2512         {
2513             for ( auto ch : range(channel) )
2514             {
2515                 auto [row_shift, col_shift] = std::make_tuple( _shifts[ch][0],
2516 _shifts[ch][1] );
2517                 for ( auto r : range(row) )
2518                 {
2519                     if (r+row_shift>=0 && r+row_shift<row)
2520                     {
2521                         for ( auto c : range(col) )
2522                         {
2523                             if (c+col_shift>=0 && c+col_shift<col )
2524                                 vo[bs][r][c][ch] =
2525 vi[bs][r+row_shift][c+col_shift][ch];
2526                             }
2527                         }
2528                     }
2529                 }
2530                 return ans;
2531             },
2532             "Sliding2D"
2533         )( ex );
2534     };
2535 }
2536 } // anonymous namespace
2537
2538 inline auto sliding_2d( unsigned long pixels ) noexcept
2539 {
2540     std::shared_ptr<std::any> shift_cache = std::make_shared<std::any>();
2541     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2542     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
2543
2544     return [=]<Expression Ex>( Ex const& ex ) noexcept
2545     {
2546         return cropping_2d( {pixels,} )( detailed_sliding_2d(pixels, shift_cache, forward_cache,
2547 backward_cache)( zero_padding_2d( {pixels,} )( ex ) ) );
2548     };
2549 }
2550 namespace
2551 {
2552     struct repeat_context
2553     {
2554         auto make_forward() const noexcept
2555         {
2556             return [] ( unsigned long repeats, unsigned long axis, std::shared_ptr<std::any>
2557 forward_cache ) noexcept
2558             {
2559                 return [=]<Tensor Tsor>( Tsor const& input ) noexcept
2560                 {
2561                     if ( 1UL == repeats ) return input;
2562                     unsigned long const ax = std::min( axis, input.shape().size()-1 );
2563
2564                     auto const& shape = input.shape();
2565                     unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
2566 1UL, [] ( unsigned long x, unsigned long y ){ return x*y; } );
2567                     unsigned long const iterations = std::accumulate( shape.begin(),
2568 shape.begin()+ax+1, 1UL, [] ( unsigned long x, unsigned long y ){ return x*y; } );
2569
2570                     // generate output tensor
2571                     std::vector<unsigned long> output_shape = input.shape();
2572                     output_shape[ax] *= repeats;
2573
2574                     Tsor& ans = context_cast<Tsor>( forward_cache );
2575                     ans.resize( output_shape );
2576
2577                     // create 2D and 3D view
2578                     view_2d v2{ input.data(), iterations, stride };
2579                     view_3d v3{ ans.data(), iterations, repeats, stride };
2580
2581                     // copy data
2582                     for ( auto it : range( iterations ) )
2583                         for ( auto re : range( repeats ) )
2584                             std::copy_n( v2[it], stride, v3[it][re] );
2585
2586                     return ans;
2587                 };
2588             };
2589         }
2590     };
2591     auto make_backward() const noexcept
2592     {
2593         return [] ( unsigned long repeats, unsigned long axis, std::shared_ptr<std::any>

```

```

backward_cache ) noexcept
2591     {
2592         return [=]<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr const& grad )
noexcept
2593     {
2594         if ( 1UL == repeats ) return grad;
2595         unsigned long const ax = std::min( axis, input.shape().size()-1 );
2596
2597         auto const& shape = input.shape();
2598         unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
1UL, []( unsigned long x, unsigned long y ){ return x*y; } );
2599         unsigned long const iterations = std::accumulate( shape.begin(),
shape.begin()+ax+1, 1UL, []( unsigned long x, unsigned long y ){ return x*y; } );
2600
2601         Tsr& ans = context_cast<Tsr>( backward_cache );
2602         ans.resize( input.shape() );
2603         ans.reset();
2604
2605         view_2d v2{ans.data(), iterations, stride };
2606         view_3d v3{ grad.data(), iterations, repeats, stride };
2607
2608         for ( auto id : range( iterations ) )
2609             for ( auto re : range( repeats ) )
2610                 for ( auto st : range( stride ) )
2611                     v2[id][st] += v3[id][re][st];
2612
2613         return ans;
2614     };
2615 };
2616 }
2617 }; //struct repeat_context
2618 } //anonymous namespace
2619
2620
2635 inline auto repeat( unsigned long repeats, unsigned long axis=-1 ) noexcept
2636 {
2637     better_assert( repeats > 0, "repeat: repeats can not be zero." );
2638
2639     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2640     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
2641
2642     return [repeats, axis, forward_cache, backward_cache]<Expression Ex>( Ex const& ex ) noexcept
2643     {
2644         return make_unary_operator
2645         (
2646             repeat_context{}.make_forward()( repeats, axis, forward_cache ),
2647             repeat_context{}.make_backward()( repeats, axis, backward_cache ),
2648             "Repeat",
2649             [=]( std::vector<unsigned long> const& shape ) noexcept
2650             {
2651                 std::vector<unsigned long> ans = shape;
2652                 if ( axis >= ans.size() ) axis = ans.size()-1;
2653                 ans[axis] *= repeats;
2654                 return ans;
2655             }
2656         )
2657         ( ex );
2658     };
2659 }
2660
2661 namespace
2662 {
2663     struct reduce_min_context
2664     {
2665         auto make_forward() const noexcept
2666         {
2667             return []( unsigned long axis, std::shared_ptr<std::any> forward_cache,
std::shared_ptr<std::any> index_cache ) noexcept
2668             {
2669                 return [=]<Tensor Tsr>( Tsr const& input ) noexcept
2670                 {
2671                     unsigned long const ax = std::min( axis, input.shape().size()-1 );
2672
2673                     // example: for an input tensor of shape ( 2, 3, 4, 5 ), and axis is 1
2674                     auto const& shape = input.shape(); // example: the shape is ( 2, 3, 4, 5 )
2675                     unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
2676                     1UL, []( unsigned long x, unsigned long y ){ return x*y; } ); // example: the stride is 20
2677                     unsigned long const iterations = std::accumulate( shape.begin(),
shape.begin()+ax+1, 1UL, []( unsigned long x, unsigned long y ){ return x*y; } ); // example: the
iterations is 2
2678                     unsigned long const scales = shape[ax]; // the elements in the dimension to
reduce. example: scales is 3
2679
2680                     // generate output tensor
2681                     std::vector<unsigned long> output_shape = input.shape(); // example:
temporarily being ( 2, 3, 4, 5 )

```

```

2682         std::copy( output_shape.begin()+ax+1, output_shape.end(),
2683 output_shape.begin()+ax ); // example: temporarily being ( 2, 4, 5, 5 )
2683         output_shape.resize( output_shape.size() - 1 ); // example: output_shape is (
2684 2, 4, 5 )
2684
2685         Tsor& ans = context_cast<Tsor>( forward_cache );
2686         ans.resize( output_shape ); // example: ans shape is ( 2, 4, 5 )
2687
2688         tensor<unsigned long>& index = context_cast<tensor<unsigned long>>( index_cache
2689 );
2689         index.resize( output_shape ); // example: index shape is ( 2, 4, 5 )
2690
2691         // create 2D and 3D view
2692         view_2d v2{ ans.data(), iterations, stride }; // example: viewing as a matrix
2693 of shape ( 2, 20 )
2693         view_2d v_index{ index.data(), iterations, stride }; // example: viewing as a
2694 matrix of ( 2, 20 )
2694         view_3d v3{ input.data(), iterations, scales, stride }; // example: viewing as
2695 a tube of ( 2, 3, 20 )
2695
2696         // reduce minimal elements along the selected axis
2697         for ( auto it : range( iterations ) ) // example: range (2)
2698             for ( auto st : range( stride ) ) // example: range (20)
2699             {
2700                 // reduce the minimal elements along the column of st
2701                 auto min_itor = std::min_element( v3[it].col_begin(st),
2702 v3[it].col_end(st) );
2702                 v2[it][st] = *min_itor;
2703
2704                 // record the minimal position offset with respect to the head of the
2705 column
2705                 unsigned long const offset = std::distance( v3[it].col_begin(st),
2706 min_itor );
2706                 v_index[it][st] = offset;
2707             }
2708
2709         return ans;
2710     };
2711 };
2712 }
2713
2714 auto make_backward() const noexcept
2715 {
2716     return []( unsigned long axis, std::shared_ptr<std::any> backward_cache,
2717 std::shared_ptr<std::any> index_cache ) noexcept
2718     {
2719         return [=]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor const& grad )
2720 noexcept
2721         {
2722             unsigned long const ax = std::min( axis, input.shape().size()-1 );
2723
2724             // example: for an input tensor of shape ( 2, 3, 4, 5 ), and axis is 1
2725             auto const& shape = input.shape(); // example: the shape is ( 2, 3, 4, 5 )
2726             unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
2727 1UL, []( unsigned long x, unsigned long y ){ return x*y; } ); // example: the stride is 20
2728             unsigned long const iterations = std::accumulate( shape.begin(),
2729 shape.begin()+ax, 1UL, []( unsigned long x, unsigned long y ){ return x*y; } ); // example: the
2730 iterations is 2
2731             unsigned long const scales = shape[ax]; // the elements in the dimension to
2732 reduce. example: scales is 3
2733
2734             std::vector<unsigned long> const& output_shape = grad.shape(); // example:
2735 output shape of ( 2, 4, 5 )
2736             tensor<unsigned long>& index = context_cast<tensor<unsigned long>>( index_cache
2737 );
2738             index.resize( output_shape ); // example: index shape is ( 2, 4, 5 )
2739
2740             Tsor& ans = context_cast<Tsor>( backward_cache );
2741             ans.resize( shape ); // example: ans shape is ( 2, 3, 4, 5 )
2742             ans.reset();
2743
2744             view_2d v_index{ index.data(), iterations, stride }; // example: viewing as a
2745 matrix of ( 2, 20 )
2746             view_3d v3{ ans.data(), iterations, scales, stride }; // example: view as a
2747 cube of ( 2, 3, 20 )
2748             view_2d v2{ grad.data(), iterations, stride }; // example: viewing as a matrix
2749 of ( 2, 20 )
2750
2751             for ( auto it : range( iterations ) ) // example: range (2)
2752                 for ( auto st : range( stride ) ) // example: range (20)
2753                 {
2754                     unsigned long const offset = v_index[it][st]; // get the offset from
2755 record
2756                     v3[it][offset][st] = v2[it][st]; // only the element at the minimal
2757 position has gradient back-propagated
2758                 }
2759         }
2760     }
2761 }

```

```

2747         return ans;
2748     };
2749 };
2750 }
2751 }; //struct reduce_min_context
2752 } //anonymous namespace
2753
2754
2755 inline auto reduce_min( unsigned long axis=-1 ) noexcept
2756 {
2757     std::shared_ptr<std::any> index_cache = std::make_shared<std::any>();
2758     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2759     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
2760
2761     return [axis, index_cache, forward_cache, backward_cache]<Expression Ex>( Ex const& ex )
2762     noexcept
2763     {
2764         return make_unary_operator
2765         (
2766             reduce_min_context{}.make_forward()( axis, forward_cache, index_cache ),
2767             reduce_min_context{}.make_backward()( axis, backward_cache, index_cache ),
2768             "ReduceMin",
2769             [=]( std::vector<unsigned long> const& shape ) noexcept
2770             {
2771                 std::vector<unsigned long> ans = shape;
2772                 if ( axis >= shape.size() ) axis = shape.size() - 1;
2773                 std::copy( ans.begin()+axis+1, ans.end(), ans.begin()+axis );
2774                 ans.resize( ans.size() - 1 );
2775                 return ans;
2776             }
2777         )
2778         ( ex );
2779     };
2780 }
2781
2782 namespace
2783 {
2784     struct reduce_max_context
2785     {
2786         auto make_forward() const noexcept
2787         {
2788             return [=]( unsigned long axis, std::shared_ptr<std::any> forward_cache,
2789             std::shared_ptr<std::any> index_cache ) noexcept
2790             {
2791                 return [=]<Tensor Tsor>( Tsor const& input ) noexcept
2792                 {
2793                     unsigned long const ax = std::min( axis, input.shape().size()-1 );
2794
2795                     // example: for an input tensor of shape ( 2, 3, 4, 5 ), and axis is 1
2796                     auto const& shape = input.shape(); // example: the shape is ( 2, 3, 4, 5 )
2797                     unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
2798                     1UL, [=]( unsigned long x, unsigned long y ){ return x*y; } ); // example: the stride is 20
2799                     unsigned long const iterations = std::accumulate( shape.begin(),
2800                     shape.begin()+ax, 1UL, [=]( unsigned long x, unsigned long y ){ return x*y; } ); // example: the
2801                     iterations is 2
2802                     unsigned long const scales = shape[ax]; // the elements in the dimension to
2803                     reduce. example: scales is 3
2804
2805                     // generate output tensor
2806                     std::vector<unsigned long> output_shape = input.shape(); // example:
2807                     temporarily being ( 2, 3, 4, 5 )
2808                     std::copy( output_shape.begin()+ax+1, output_shape.end(),
2809                     output_shape.begin()+ax ); // example: temporarily being ( 2, 4, 5, 5 )
2810                     output_shape.resize( output_shape.size() - 1 ); // example: output_shape is (
2811                     2, 4, 5 )
2812
2813                     Tsor& ans = context_cast<Tsor>( forward_cache );
2814                     ans.resize( output_shape ); // example: ans shape is ( 2, 4, 5 )
2815
2816                     tensor<unsigned long>& index = context_cast<tensor<unsigned long>>( index_cache
2817                     );
2818                     index.resize( output_shape ); // example: index shape is ( 2, 4, 5 )
2819
2820                     // create 2D and 3D view
2821                     view_2d v2{ ans.data(), iterations, stride }; // example: viewing as a matrix
2822                     of shape ( 2, 20 )
2823                     view_2d v_index{ index.data(), iterations, stride }; // example: viewing as a
2824                     matrix of ( 2, 20 )
2825                     view_3d v3{ input.data(), iterations, scales, stride }; // example: viewing as
2826                     a tube of ( 2, 3, 20 )
2827
2828                     // reduce maximal elements along the selected axis
2829                     for ( auto it : range( iterations ) ) // example: range (2)
2830                     {
2831                         for ( auto st : range( stride ) ) // example: range (20)
2832                         {

```

```

2834                                     // reduce the maximal elements along the column of st
2835                                     auto max_itor = std::max_element( v3[it].col_begin(st),
v3[it].col_end(st) );
2836                                     v2[it][st] = *max_itor;
2837
2838                                     // record the maximal position offset with respect to the head of the
column
2839                                     unsigned long const offset = std::distance( v3[it].col_begin(st),
max_itor );
2840                                     v_index[it][st] = offset;
2841                                     }
2842
2843                                     return ans;
2844                                     };
2845                                     };
2846                                     }
2847
2848                                     auto make_backward() const noexcept
2849                                     {
2850                                     return []( unsigned long axis, std::shared_ptr<std::any> backward_cache,
std::shared_ptr<std::any> index_cache ) noexcept
2851                                     {
2852                                     return [=]<Tensor Tsor>( Tsor const& input, Tsor const& , Tsor const& grad )
noexcept
2853                                     {
2854                                     unsigned long const ax = std::min( axis, input.shape().size()-1 );
2855
2856                                     // example: for an input tensor of shape ( 2, 3, 4, 5 ), and axis is 1
2857                                     auto const& shape = input.shape(); // example: the shape is ( 2, 3, 4, 5 )
2858                                     unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
1UL, []( unsigned long x, unsigned long y ){ return x*y; } ); // example: the stride is 20
2859                                     unsigned long const iterations = std::accumulate( shape.begin(),
shape.begin()+ax, 1UL, []( unsigned long x, unsigned long y ){ return x*y; } ); // example: the
iterations is 2
2860                                     unsigned long const scales = shape[ax]; // the elements in the dimension to
reduce. example: scales is 3
2861
2862                                     std::vector<unsigned long> const& output_shape = grad.shape(); // example:
output shape of ( 2, 4, 5 )
2863                                     tensor<unsigned long>& index = context_cast<tensor<unsigned long>>( index_cache
);
2864                                     index.resize( output_shape ); // example: index shape is ( 2, 4, 5 )
2865
2866                                     Tsor& ans = context_cast<Tsor>( backward_cache );
2867                                     ans.resize( shape ); // example: ans shape is ( 2, 3, 4, 5 )
2868                                     ans.reset();
2869
2870                                     view_2d v_index{ index.data(), iterations, stride }; // example: viewing as a
matrix of ( 2, 20 )
2871                                     view_3d v3{ ans.data(), iterations, scales, stride }; // example: view as a
cube of ( 2, 3, 20 )
2872                                     view_2d v2{ grad.data(), iterations, stride }; // example: viewing as a matrix
of ( 2, 20 )
2873
2874                                     for ( auto it : range( iterations ) ) // example: range( 2 )
2875                                     for ( auto st : range( stride ) ) // example: range( 20 )
2876                                     {
2877                                     unsigned long const offset = v_index[it][st]; // get the offset from
record
2878                                     v3[it][offset][st] = v2[it][st]; // only the element at the maximal
position has gradient back-propagated
2879                                     }
2880
2881                                     return ans;
2882                                     };
2883                                     };
2884                                     }
2885                                     }; //struct reduce_max_context
2886                                     } //anonymous namespace
2887
2888
2902                                     inline auto reduce_max( unsigned long axis=-1 ) noexcept
2903                                     {
2904                                     std::shared_ptr<std::any> index_cache = std::make_shared<std::any>();
2905                                     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
2906                                     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
2907
2908                                     return [axis, index_cache, forward_cache, backward_cache]<Expression Ex>( Ex const& ex )
noexcept
2909                                     {
2910                                     return make_unary_operator
2911                                     (
2912                                     reduce_max_context{}.make_forward()( axis, forward_cache, index_cache ),
2913                                     reduce_max_context{}.make_backward()( axis, backward_cache, index_cache ),
2914                                     "ReduceMax",
2915                                     [=]( std::vector<unsigned long> const& shape ) noexcept
2916                                     {

```



```

2917         std::vector<unsigned long> ans = shape;
2918         if ( axis >= shape.size() ) axis = shape.size() - 1;
2919         std::copy( ans.begin()+axis+1, ans.end(), ans.begin()+axis );
2920         ans.resize( ans.size() - 1 );
2921         return ans;
2922     }
2923     )
2924     ( ex );
2925 };
2926 }
2927
2928
2929 namespace
2930 {
2931     struct reduce_sum_context
2932     {
2933     {
2934         auto make_forward() const noexcept
2935         {
2936             return [] ( unsigned long axis, std::shared_ptr<std::any> forward_cache ) noexcept
2937             {
2938                 return [=]<Tensor Tsor>( Tsor const& input ) noexcept
2939                 {
2940                     typedef typename Tsor::value_type value_type;
2941
2942                     unsigned long const ax = std::min( axis, input.shape().size()-1 );
2943
2944                     // example: for an input tensor of shape ( 2, 3, 4, 5 ), and axis is 1
2945                     auto const& shape = input.shape(); // example: the shape is ( 2, 3, 4, 5 )
2946                     unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
1UL, [] ( unsigned long x, unsigned long y ){ return x*y; } ); // example: the stride is 20
2947                     unsigned long const iterations = std::accumulate( shape.begin(),
shape.begin()+ax, 1UL, [] ( unsigned long x, unsigned long y ){ return x*y; } ); // example: the
iterations is 2
2948                     unsigned long const scales = shape[ax]; // the elements in the dimension to
reduce. example: scales is 3
2949
2950                     // generate output tensor
2951                     std::vector<unsigned long> output_shape = input.shape(); // example:
temporarily being ( 2, 3, 4, 5 )
2952                     std::copy( output_shape.begin()+ax+1, output_shape.end(),
output_shape.begin()+ax ); // example: temporarily being ( 2, 4, 5, 5 )
2953                     output_shape.resize( output_shape.size() - 1 ); // example: output_shape is (
2, 4, 5 )
2954
2955                     Tsor& ans = context_cast<Tsor>( forward_cache );
2956                     ans.resize( output_shape ); // example: ans shape is ( 2, 4, 5 )
2957
2958                     // create 2D and 3D view
2959                     view_2d v2{ ans.data(), iterations, stride }; // example: viewing as a matrix
of shape ( 2, 20 )
2960                     view_3d v3{ input.data(), iterations, scales, stride }; // example: viewing as
a tube of ( 2, 3, 20 )
2961
2962                     // reduce sum along the selected axis
2963                     for ( auto it : range( iterations ) ) // example: range (2)
2964                         for ( auto st : range( stride ) ) // example: range (20)
2965                             v2[it][st] = std::accumulate( v3[it].col_begin(st), v3[it].col_end(st),
value_type{0} );
2966
2967                     return ans;
2968                 };
2969             };
2970         }
2971
2972         auto make_backward() const noexcept
2973         {
2974             return [] ( unsigned long axis, std::shared_ptr<std::any> backward_cache ) noexcept
2975             {
2976                 return [=]<Tensor Tsor>( Tsor const& input, Tsor const& , Tsor const& grad )
noexcept
2977                 {
2978                     unsigned long const ax = std::min( axis, input.shape().size()-1 );
2979
2980                     // example: for an input tensor of shape ( 2, 3, 4, 5 ), and axis is 1
2981                     auto const& shape = input.shape(); // example: the shape is ( 2, 3, 4, 5 )
2982                     unsigned long const stride = std::accumulate( shape.begin()+ax+1, shape.end(),
1UL, [] ( unsigned long x, unsigned long y ){ return x*y; } ); // example: the stride is 20
2983                     unsigned long const iterations = std::accumulate( shape.begin(),
shape.begin()+ax, 1UL, [] ( unsigned long x, unsigned long y ){ return x*y; } ); // example: the
iterations is 2
2984                     unsigned long const scales = shape[ax]; // the elements in the dimension to
reduce. example: scales is 3
2985
2986                     Tsor& ans = context_cast<Tsor>( backward_cache );
2987                     ans.resize( shape ); // example: ans shape is ( 2, 3, 4, 5 )
2988                     ans.reset();

```

```

2989
2990         view_3d v3{ ans.data(), iterations, scales, stride }; // example: view as a
cube of ( 2, 3, 20 )
2991         view_2d v2{ grad.data(), iterations, stride }; // example: viewing as a matrix
of ( 2, 20 )
2992
2993         for ( auto it : range( iterations ) ) // example: range( 2 )
2994             for ( auto st : range( stride ) ) // example: range( 20 )
2995                 std::fill( v3[it].col_begin( st ), v3[it].col_end( st ), v2[it][st] );
2996
2997         return ans;
2998     };
2999 };
3000 }
3001 }; //struct reduce_sum_context
3002 } //anonymous namespace
3003
3004
3018 inline auto reduce_sum( unsigned long axis ) noexcept
3019 {
3020     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3021     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3022
3023     return [axis, forward_cache, backward_cache]<Expression Ex>( Ex const& ex ) noexcept
3024     {
3025         return make_unary_operator
3026         (
3027             reduce_sum_context{}.make_forward()( axis, forward_cache ),
3028             reduce_sum_context{}.make_backward()( axis, backward_cache ),
3029             "ReduceSum",
3030             [=]( std::vector<unsigned long> const& shape ) noexcept
3031             {
3032                 std::vector<unsigned long> ans = shape;
3033                 if ( axis >= shape.size() ) axis = shape.size() - 1;
3034                 std::copy( ans.begin()+axis+1, ans.end(), ans.begin()+axis );
3035                 ans.resize( ans.size() - 1 );
3036                 return ans;
3037             }
3038         )
3039         ( ex );
3040     };
3041 }
3042
3043
3044
3045
3055 template <Expression Ex>
3056 auto constexpr abs( Ex const& ex ) noexcept
3057 {
3058     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3059     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3060     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3061     {
3062         Tsr& ans = context_cast<Tsr>( forward_cache );
3063         ans.resize( input.shape() );
3064         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::abs(x); } );
3065         return ans;
3066     },
3067     [backward_cache]<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr
const& grad ) noexcept
3068     {
3069         Tsr& ans = context_cast<Tsr>( backward_cache );
3070         ans.resize( input.shape() );
3071         for_each( input.begin(), input.end(), grad.begin(),
ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g * ((x > 0.0) ? 1.0 : ((x < 0.0) ? -1.0
: 0.0)); } );
3072         return ans;
3073     },
3074     "Abs"
3075     )( ex );
3076 };
3077
3078
3079
3080
3081
3082
3092 template <Expression Ex>
3093 auto constexpr acos( Ex const& ex ) noexcept
3094 {
3095     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3096     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3097     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3098     {
3099         Tsr& ans = context_cast<Tsr>( forward_cache );
3100         ans.resize( input.shape() );

```

```

3101         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3102 auto& v ) noexcept { v = std::acos(x); } );
3103         return ans;
3104     },
3105     [backward_cache]<Tensor T>( T const& input, T const&, T const& grad ) noexcept
3106     {
3107         T& ans = context_cast<T>( backward_cache );
3108         ans.resize( input.shape() );
3109         for_each( input.begin(), input.end(), grad.begin(),
3110 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = - g / std::sqrt(1.0-x*x); } );
3111         return ans;
3112     },
3113     "Acos"
3114 );
3115
3116 template <Expression Ex>
3117 auto constexpr acosh( Ex const& ex ) noexcept
3118 {
3119     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3120     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3121     return make_unary_operator( [forward_cache]<Tensor T>( T const& input ) noexcept
3122     {
3123         T& ans = context_cast<T>( forward_cache );
3124         ans.resize( input.shape() );
3125         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3126 auto& v ) noexcept { v = std::acosh(x); } );
3127         return ans;
3128     },
3129     [backward_cache]<Tensor T>( T const& input, T const&, T const& grad ) noexcept
3130     {
3131         T& ans = context_cast<T>( backward_cache );
3132         ans.resize( input.shape() );
3133         for_each( input.begin(), input.end(), grad.begin(),
3134 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / std::sqrt(x*x-1.0); } );
3135         return ans;
3136     },
3137     "Acosh"
3138 );
3139
3140 template <Expression Ex>
3141 auto constexpr asin( Ex const& ex ) noexcept
3142 {
3143     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3144     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3145     return make_unary_operator( [forward_cache]<Tensor T>( T const& input ) noexcept
3146     {
3147         T& ans = context_cast<T>( forward_cache );
3148         ans.resize( input.shape() );
3149         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3150 auto& v ) noexcept { v = std::asin(x); } );
3151         return ans;
3152     },
3153     [backward_cache]<Tensor T>( T const& input, T const&, T const& grad ) noexcept
3154     {
3155         T& ans = context_cast<T>( backward_cache );
3156         ans.resize( input.shape() );
3157         for_each( input.begin(), input.end(), grad.begin(),
3158 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / std::sqrt(1.0-x*x); } );
3159         return ans;
3160     },
3161     "Asin"
3162 );
3163
3164 template <Expression Ex>
3165 auto constexpr asinh( Ex const& ex ) noexcept
3166 {
3167     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3168     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3169     return make_unary_operator( [forward_cache]<Tensor T>( T const& input ) noexcept
3170     {
3171         T& ans = context_cast<T>( forward_cache );
3172         ans.resize( input.shape() );
3173         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3174 auto& v ) noexcept { v = std::asinh(x); } );
3175         return ans;
3176     },
3177     [backward_cache]<Tensor T>( T const& input, T const&, T const& grad ) noexcept
3178     {
3179         T& ans = context_cast<T>( backward_cache );
3180         ans.resize( input.shape() );
3181         for_each( input.begin(), input.end(), grad.begin(),
3182 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / std::sqrt(1.0+x*x); } );
3183         return ans;
3184     },
3185     "Asinh"
3186 );
3187
3188 template <Expression Ex>
3189 auto constexpr atanh( Ex const& ex ) noexcept
3190 {
3191     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3192     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3193     return make_unary_operator( [forward_cache]<Tensor T>( T const& input ) noexcept
3194     {
3195         T& ans = context_cast<T>( forward_cache );
3196         ans.resize( input.shape() );
3197         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3198 auto& v ) noexcept { v = std::atanh(x); } );
3199         return ans;
3200     },
3201     [backward_cache]<Tensor T>( T const& input, T const&, T const& grad ) noexcept
3202     {
3203         T& ans = context_cast<T>( backward_cache );
3204         ans.resize( input.shape() );
3205         for_each( input.begin(), input.end(), grad.begin(),

```

```

3206     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3207     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3208     return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3209     {
3210         Tzor& ans = context_cast<Tzor>( forward_cache );
3211         ans.resize( input.shape() );
3212         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3213 auto& v ) noexcept { v = std::asinh(x); } );
3214         return ans;
3215     },
3216     [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
3217 const& grad ) noexcept
3218     {
3219         Tzor& ans = context_cast<Tzor>( backward_cache );
3220         ans.resize( input.shape() );
3221         for_each( input.begin(), input.end(), grad.begin(),
3222 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / std::sqrt(1.0+x*x); } );
3223         return ans;
3224     },
3225     "Asinh"
3226 );
3227 };
3228
3229
3230
3231 template <Expression Ex>
3232 auto constexpr atan( Ex const& ex ) noexcept
3233 {
3234     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3235     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3236     return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3237     {
3238         Tzor& ans = context_cast<Tzor>( forward_cache );
3239         ans.resize( input.shape() );
3240         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3241 auto& v ) noexcept { v = std::atan(x); } );
3242         return ans;
3243     },
3244     [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
3245 const& grad ) noexcept
3246     {
3247         Tzor& ans = context_cast<Tzor>( backward_cache );
3248         ans.resize( input.shape() );
3249         for_each( input.begin(), input.end(), grad.begin(),
3250 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / (1.0+x*x); } );
3251         return ans;
3252     },
3253     "Atan"
3254 );
3255 };
3256
3257
3258
3259 template <Expression Ex>
3260 auto constexpr atanh( Ex const& ex ) noexcept
3261 {
3262     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3263     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3264     return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3265     {
3266         Tzor& ans = context_cast<Tzor>( forward_cache );
3267         ans.resize( input.shape() );
3268         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3269 auto& v ) noexcept { v = std::atanh(x); } );
3270         return ans;
3271     },
3272     [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
3273 const& grad ) noexcept
3274     {
3275         Tzor& ans = context_cast<Tzor>( backward_cache );
3276         ans.resize( input.shape() );
3277         for_each( input.begin(), input.end(), grad.begin(),
3278 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / (1-x*x); } );
3279         return ans;
3280     },
3281     "Atanh"
3282 );
3283 };
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301

```

```

3302
3303
3304
3314     template <Expression Ex>
3315     auto constexpr cbrt( Ex const& ex ) noexcept
3316     {
3317         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3318         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3319         return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
3320         {
3321             Tsor& ans = context_cast<Tsor>( forward_cache );
3322             ans.resize( input.shape() );
3323             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3324 auto& v ) noexcept { v = std::cbrt(x); } );
3325             return ans;
3326         },
3327         [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&
3328 output, Tsor const& grad ) noexcept
3329         {
3330             Tsor& ans = context_cast<Tsor>( backward_cache );
3331             ans.resize( input.shape() );
3332             for_each( input.begin(), input.end(), output.begin(),
3333 grad.begin(), ans.begin(), []( auto, auto o, auto g, auto& v ) noexcept { v = g / (3.0*o*o); } );
3334             return ans;
3335         },
3336         "Cbrt"
3337     )( ex );
3338 };
3339
3340
3341
3351     template <Expression Ex>
3352     auto constexpr ceil( Ex const& ex ) noexcept
3353     {
3354         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3355         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3356         return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
3357         {
3358             Tsor& ans = context_cast<Tsor>( forward_cache );
3359             ans.resize( input.shape() );
3360             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3361 auto& v ) noexcept { v = std::ceil(x); } );
3362             return ans;
3363         },
3364         [<Tensor Tsor>( Tsor const&, Tsor const&, Tsor const& grad )
3365 noexcept
3366         {
3367             return grad;
3368         },
3369         "Ceil"
3370     )( ex );
3371 };
3372
3373
3374
3385     template <Expression Ex>
3386     auto constexpr cos( Ex const& ex ) noexcept
3387     {
3388         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3389         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3390         return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
3391         {
3392             Tsor& ans = context_cast<Tsor>( forward_cache );
3393             ans.resize( input.shape() );
3394             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3395 auto& v ) noexcept { v = std::cos(x); } );
3396             return ans;
3397         },
3398         [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor
3399 const& grad ) noexcept
3400         {
3401             Tsor& ans = context_cast<Tsor>( backward_cache );
3402             ans.resize( input.shape() );
3403             for_each( input.begin(), input.end(), grad.begin(),
3404 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = - g * std::sin(x); } );
3405             return ans;
3406         },
3407         "Cos"
3408     )( ex );
3409 };
3410

```

```

3408
3409
3410
3411
3412
3422     template <Expression Ex>
3423     auto constexpr cosh( Ex const& ex ) noexcept
3424     {
3425         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3426         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3427         return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3428         {
3429             Tzor& ans = context_cast<Tzor>( forward_cache );
3430             ans.resize( input.shape() );
3431             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::cosh(x); } );
3432             return ans;
3433         },
3434         [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
const& grad ) noexcept
3435         {
3436             Tzor& ans = context_cast<Tzor>( backward_cache );
3437             ans.resize( input.shape() );
3438             for_each( input.begin(), input.end(), grad.begin(),
ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g * std::sinh(x); } );
3439             return ans;
3440         },
3441         "Cosh"
3442     )( ex );
3443 };
3444
3445
3446
3447
3448
3449
3459     template <Expression Ex>
3460     auto constexpr erf( Ex const& ex ) noexcept
3461     {
3462         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3463         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3464         return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3465         {
3466             Tzor& ans = context_cast<Tzor>( forward_cache );
3467             ans.resize( input.shape() );
3468             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::erf(x); } );
3469             return ans;
3470         },
3471         [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
const& grad ) noexcept
3472         {
3473             Tzor& ans = context_cast<Tzor>( backward_cache );
3474             ans.resize( input.shape() );
3475             for_each( input.begin(), input.end(), grad.begin(),
ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = typename
Tzor::value_type{1.12837916709551257389} * g * std::exp(-x*x); } );
3476             return ans;
3477         },
3478         "Erf"
3479     )( ex );
3480 };
3481
3482
3483
3484
3485
3486
3496     template <Expression Ex>
3497     auto constexpr erfc( Ex const& ex ) noexcept
3498     {
3499
3500         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3501         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3502         return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3503         {
3504             Tzor& ans = context_cast<Tzor>( forward_cache );
3505             ans.resize( input.shape() );
3506             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::erfc(x); } );
3507             return ans;
3508         },
3509         [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
const& grad ) noexcept
3510         {
3511             Tzor& ans = context_cast<Tzor>( backward_cache );
3512             ans.resize( input.shape() );

```

```

3513         for_each( input.begin(), input.end(), grad.begin(),
ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = typename
Tsr::value_type{-1.12837916709551257389} * g * std::exp(-x*x); } );
3514         return ans;
3515     },
3516     "Erfc"
3517 )( ex );
3518 };
3519
3520
3521
3522
3523
3524
3525 template <Expression Ex>
3526 auto constexpr exp( Ex const& ex ) noexcept
3527 {
3528     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3529     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3530     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3531     {
3532         Tsr& ans = context_cast<Tsr>( forward_cache );
3533         ans.resize( input.shape() );
3534         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::exp(x); } );
3535         return ans;
3536     },
3537     [backward_cache]<Tensor Tsr>( Tsr const& input, Tsr const&
output, Tsr const& grad ) noexcept
3538     {
3539         Tsr& ans = context_cast<Tsr>( backward_cache );
3540         ans.resize( input.shape() );
3541         for_each( input.begin(), input.end(), output.begin(),
grad.begin(), ans.begin(), []( auto, auto o, auto g, auto& v ) noexcept { v = g * o; } );
3542         return ans;
3543     },
3544     "Exp"
3545 )( ex );
3546 };
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562 template <Expression Ex>
3563 auto constexpr exp2( Ex const& ex ) noexcept
3564 {
3565     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3566     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3567     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3568     {
3569         Tsr& ans = context_cast<Tsr>( forward_cache );
3570         ans.resize( input.shape() );
3571         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::exp2(x); } );
3572         return ans;
3573     },
3574     [backward_cache]<Tensor Tsr>( Tsr const& input, Tsr const&
output, Tsr const& grad ) noexcept
3575     {
3576         Tsr& ans = context_cast<Tsr>( backward_cache );
3577         ans.resize( input.shape() );
3578         for_each( input.begin(), input.end(), output.begin(),
grad.begin(), ans.begin(), []( auto, auto o, auto g, auto& v ) noexcept { v = std::log(2.0) * g * o; }
);
3579         return ans;
3580     },
3581     "Exp2"
3582 )( ex );
3583 };
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599 template <Expression Ex>
3600 auto constexpr expm1( Ex const& ex ) noexcept
3601 {
3602     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3603     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3604     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3605     {
3606         Tsr& ans = context_cast<Tsr>( forward_cache );
3607         ans.resize( input.shape() );
3608         for_each( input.begin(), input.end(), ans.begin(), []( auto x,

```

```

    auto& v ) noexcept { v = std::expm1(x); } );
3618         return ans;
3619     },
3620     [backward_cache]<Tensor Tsr>( Tsr const& input, Tsr const&
output, Tsr const& grad ) noexcept
3621     {
3622         Tsr& ans = context_cast<Tsr>( backward_cache );
3623         ans.resize( input.shape() );
3624         for_each( input.begin(), input.end(), output.begin(),
grad.begin(), ans.begin(), []( auto, auto o, auto g, auto& v ) noexcept { v = g * (o+1.0); } );
3625         return ans;
3626     },
3627     "Expm1"
3628 ) ( ex );
3629 };
3630
3631
3632
3633
3634
3635
3645     template <Expression Ex>
3646     auto constexpr fabs( Ex const& ex ) noexcept
3647     {
3648         return abs( ex );
3649     };
3650
3651
3652
3653
3654
3655
3665     template <Expression Ex>
3666     auto constexpr floor( Ex const& ex ) noexcept
3667     {
3668         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3669         return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3670         {
3671             Tsr& ans = context_cast<Tsr>( forward_cache );
3672             ans.resize( input.shape() );
3673             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::floor(x); } );
3674             return ans;
3675         },
3676         []<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
noexcept
3677         {
3678             return grad;
3679         },
3680         "Floor"
3681 ) ( ex );
3682 };
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3702     template <Expression Ex>
3703     auto constexpr llrint( Ex const& ex ) noexcept
3704     {
3705         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3706         return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3707         {
3708             Tsr& ans = context_cast<Tsr>( forward_cache );
3709             ans.resize( input.shape() );
3710             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::llrint(x); } );
3711             return ans;
3712         },
3713         []<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
noexcept
3714         {
3715             return grad;
3716         },
3717         "Llrint"
3718 ) ( ex );
3719 };
3720
3721
3722
3723
3724

```



```

3725
3735     template <Expression Ex>
3736     auto constexpr llround( Ex const& ex ) noexcept
3737     {
3738         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3739         return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
3740         {
3741             Tsor& ans = context_cast<Tsor>( forward_cache );
3742             ans.resize( input.shape() );
3743             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3744 auto& v ) noexcept { v = std::llround(x); } );
3745             return ans;
3746         }<Tensor Tsor>( Tsor const&, Tsor const&, Tsor const& grad )
3747         {
3748             return grad;
3749         },
3750         "Llround"
3751     )( ex );
3752 };
3753
3754
3755
3756
3757
3758
3768     template <Expression Ex>
3769     auto constexpr log( Ex const& ex ) noexcept
3770     {
3771         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3772         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3773         return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
3774         {
3775             Tsor& ans = context_cast<Tsor>( forward_cache );
3776             ans.resize( input.shape() );
3777             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3778 auto& v ) noexcept { v = std::log(x); } );
3779             return ans;
3780         },
3781         [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor
3782 const& grad ) noexcept
3783         {
3784             Tsor& ans = context_cast<Tsor>( backward_cache );
3785             ans.resize( input.shape() );
3786             for_each( input.begin(), input.end(), grad.begin(),
3787 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / x; } );
3788             return ans;
3789         },
3790         "Log"
3791     )( ex );
3792 };
3793
3794
3795
3805     template <Expression Ex>
3806     auto constexpr log10( Ex const& ex ) noexcept
3807     {
3808         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3809         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3810         return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
3811         {
3812             Tsor& ans = context_cast<Tsor>( forward_cache );
3813             ans.resize( input.shape() );
3814             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3815 auto& v ) noexcept { v = std::log10(x); } );
3816             return ans;
3817         },
3818         [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&, Tsor
3819 const& grad ) noexcept
3820         {
3821             Tsor& ans = context_cast<Tsor>( backward_cache );
3822             ans.resize( input.shape() );
3823             for_each( input.begin(), input.end(), grad.begin(),
3824 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / (2.30258509299404568402*x); } );
3825             return ans;
3826         },
3827         "Log10"
3828     )( ex );
3829 };
3830

```

```

3831
3832
3842     template <Expression Ex>
3843     auto constexpr loglp( Ex const& ex ) noexcept
3844     {
3845         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3846         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3847         return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3848         {
3849             Tzor& ans = context_cast<Tzor>( forward_cache );
3850             ans.resize( input.shape() );
3851             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::loglp(x); } );
3852             return ans;
3853         },
3854         [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
const& grad ) noexcept
3855         {
3856             Tzor& ans = context_cast<Tzor>( backward_cache );
3857             ans.resize( input.shape() );
3858             for_each( input.begin(), input.end(), grad.begin(),
ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / x; } );
3859             return ans;
3860         },
3861         "Loglp"
3862     )( ex );
3863 };
3864
3865
3866
3867
3868
3869
3879     template <Expression Ex>
3880     auto constexpr log2( Ex const& ex ) noexcept
3881     {
3882         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3883         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3884         return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3885         {
3886             Tzor& ans = context_cast<Tzor>( forward_cache );
3887             ans.resize( input.shape() );
3888             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::log2(x); } );
3889             return ans;
3890         },
3891         [backward_cache]<Tensor Tzor>( Tzor const& input, Tzor const&, Tzor
const& grad ) noexcept
3892         {
3893             Tzor& ans = context_cast<Tzor>( backward_cache );
3894             ans.resize( input.shape() );
3895             for_each( input.begin(), input.end(), grad.begin(),
ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g / (0.69314718055994530942*x); } );
3896             return ans;
3897         },
3898         "Log2"
3899     )( ex );
3900 };
3901
3902
3903
3913     template <Expression Ex>
3914     auto constexpr lrint( Ex const& ex ) noexcept
3915     {
3916         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3917         std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
3918         return make_unary_operator( [forward_cache]<Tensor Tzor>( Tzor const& input ) noexcept
3919         {
3920             Tzor& ans = context_cast<Tzor>( forward_cache );
3921             ans.resize( input.shape() );
3922             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::lrint(x); } );
3923             return ans;
3924         },
3925         [backward_cache]<Tensor Tzor>( Tzor const&, Tzor const&, Tzor
const& grad ) noexcept
3926         {
3927             return grad;
3928         },
3929         "Lrint"
3930     )( ex );
3931 };
3932
3933
3934
3935
3936

```

```

3937
3947     template <Expression Ex>
3948     auto constexpr lround( Ex const& ex ) noexcept
3949     {
3950         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3951         return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3952         {
3953             Tsr& ans = context_cast<Tsr>( forward_cache );
3954             ans.resize( input.shape() );
3955             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3956 auto& v ) noexcept { v = std::lround(x); } );
3957             return ans;
3958         },
3959         [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
3960         {
3961             return grad;
3962         },
3963         "Lround"
3964     )( ex );
3965 };
3966
3967
3968
3969
3970
3980     template <Expression Ex>
3981     auto constexpr nearbyint( Ex const& ex ) noexcept
3982     {
3983         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
3984         return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
3985         {
3986             Tsr& ans = context_cast<Tsr>( forward_cache );
3987             ans.resize( input.shape() );
3988             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
3989 auto& v ) noexcept { v = std::nearbyint(x); } );
3990             return ans;
3991         },
3992         [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
3993         {
3994             return grad;
3995         },
3996         "Nearbyint"
3997     )( ex );
3998 };
3999
4000
4001
4002
4003
4013     template <Expression Ex>
4014     auto constexpr rint( Ex const& ex ) noexcept
4015     {
4016         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4017         return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
4018         {
4019             Tsr& ans = context_cast<Tsr>( forward_cache );
4020             ans.resize( input.shape() );
4021             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
4022 auto& v ) noexcept { v = std::rint(x); } );
4023             return ans;
4024         },
4025         [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
4026         {
4027             return grad;
4028         },
4029         "Rint"
4030     )( ex );
4031 };
4032
4033
4034
4035
4036
4046     template <Expression Ex>
4047     auto constexpr round( Ex const& ex ) noexcept
4048     {
4049         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4050         return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
4051         {
4052             Tsr& ans = context_cast<Tsr>( forward_cache );
4053             ans.resize( input.shape() );

```

```

4054         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
4055 auto& v ) noexcept { v = std::round(x); } );
4056         return ans;
4057     },
4058     [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
4059     noexcept
4060     {
4061         return grad;
4062     },
4063     "Round"
4064     )( ex );
4065 };
4066
4067
4068
4069
4070 template <Expression Ex>
4071 auto constexpr sin( Ex const& ex ) noexcept
4072 {
4073     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4074     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
4075     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
4076     {
4077         Tsr& ans = context_cast<Tsr>( forward_cache );
4078         ans.resize( input.shape() );
4079         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
4080 auto& v ) noexcept { v = std::sin(x); } );
4081         return ans;
4082     },
4083     [backward_cache]<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr
4084     const& grad ) noexcept
4085     {
4086         Tsr& ans = context_cast<Tsr>( backward_cache );
4087         ans.resize( input.shape() );
4088         for_each( input.begin(), input.end(), grad.begin(),
4089 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g * std::cos(x); } );
4090         return ans;
4091     },
4092     "Sin"
4093     )( ex );
4094 };
4095
4096
4097
4098
4099
4100
4101
4102
4103
4104
4105
4106
4107 template <Expression Ex>
4108 auto constexpr sinh( Ex const& ex ) noexcept
4109 {
4110     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4111     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
4112     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
4113     {
4114         Tsr& ans = context_cast<Tsr>( forward_cache );
4115         ans.resize( input.shape() );
4116         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
4117 auto& v ) noexcept { v = std::sinh(x); } );
4118         return ans;
4119     },
4120     [backward_cache]<Tensor Tsr>( Tsr const& input, Tsr const&, Tsr
4121     const& grad ) noexcept
4122     {
4123         Tsr& ans = context_cast<Tsr>( backward_cache );
4124         ans.resize( input.shape() );
4125         for_each( input.begin(), input.end(), grad.begin(),
4126 ans.begin(), []( auto x, auto g, auto& v ) noexcept { v = g * std::cosh(x); } );
4127         return ans;
4128     },
4129     "Sinh"
4130     )( ex );
4131 };
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144 template <Expression Ex>
4145 auto constexpr sqrt( Ex const& ex ) noexcept
4146 {
4147     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4148     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
4149     return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
4150     {

```

```

4160         Tsor& ans = context_cast<Tsor>( forward_cache );
4161         ans.resize( input.shape() );
4162         auto& v ) noexcept { v = std::sqrt(x); } );
4163         return ans;
4164     },
4165     [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&
4166     output, Tsor const& grad ) noexcept
4167     {
4168         Tsor& ans = context_cast<Tsor>( backward_cache );
4169         ans.resize( input.shape() );
4170         for_each( input.begin(), input.end(), output.begin(),
4171         grad.begin(), ans.begin(), []( auto, auto o, auto g, auto& v ) noexcept { v = g / (o+o); } );
4172         return ans;
4173     },
4174     "Sqrt"
4175 );
4176
4177
4178
4179
4180
4181 template <Expression Ex>
4182 auto constexpr tan( Ex const& ex ) noexcept
4183 {
4184     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4185     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
4186     return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
4187     {
4188         Tsor& ans = context_cast<Tsor>( forward_cache );
4189         ans.resize( input.shape() );
4190         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
4191         auto& v ) noexcept { v = std::tan(x); } );
4192         return ans;
4193     },
4194     [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&
4195     output, Tsor const& grad ) noexcept
4196     {
4197         Tsor& ans = context_cast<Tsor>( backward_cache );
4198         ans.resize( input.shape() );
4199         for_each( input.begin(), input.end(), output.begin(),
4200         grad.begin(), ans.begin(), []( auto x, auto o, auto g, auto& v ) noexcept { v = g * (1.0+o*o); } );
4201         return ans;
4202     },
4203     "Tan"
4204 );
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218 template <Expression Ex>
4219 auto constexpr tanh( Ex const& ex ) noexcept
4220 {
4221     std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4222     std::shared_ptr<std::any> backward_cache = std::make_shared<std::any>();
4223     return make_unary_operator( [forward_cache]<Tensor Tsor>( Tsor const& input ) noexcept
4224     {
4225         Tsor& ans = context_cast<Tsor>( forward_cache );
4226         ans.resize( input.shape() );
4227         for_each( input.begin(), input.end(), ans.begin(), []( auto x,
4228         auto& v ) noexcept { v = std::tanh(x); } );
4229         return ans;
4230     },
4231     [backward_cache]<Tensor Tsor>( Tsor const& input, Tsor const&
4232     output, Tsor const& grad ) noexcept
4233     {
4234         Tsor& ans = context_cast<Tsor>( backward_cache );
4235         ans.resize( input.shape() );
4236         for_each( input.begin(), input.end(), output.begin(),
4237         grad.begin(), ans.begin(), []( auto, auto o, auto g, auto& v ) noexcept { v = g * (1.0-o*o); } );
4238         return ans;
4239     },
4240     "Tanh"
4241 );
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255

```

```

4256
4257
4258
4268     template <Expression Ex>
4269     auto constexpr trunc( Ex const& ex ) noexcept
4270     {
4271         std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
4272         return make_unary_operator( [forward_cache]<Tensor Tsr>( Tsr const& input ) noexcept
4273         {
4274             Tsr& ans = context_cast<Tsr>( forward_cache );
4275             ans.resize( input.shape() );
4276             for_each( input.begin(), input.end(), ans.begin(), []( auto x,
auto& v ) noexcept { v = std::trunc(x); } );
             return ans;
         },
         [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad )
noexcept
         {
             return grad;
         },
         "Trunc"
         )( ex );
4285     };
4286
4287
4288
4301     template< Expression Lhs_Expression, Variable Rhs_Expression >
4302     auto constexpr assign( Lhs_Expression const& lhs_ex, Rhs_Expression const& rhs_ex ) noexcept
4303     {
4304         return make_binary_operator( [<Tensor Tsr>( Tsr const& lhs_tensor, Tsr& rhs_tensor )
noexcept
         {
4305             rhs_tensor.reshape( lhs_tensor.shape() );
4306             std::copy( lhs_tensor.begin(), lhs_tensor.end(),
rhs_tensor.begin() );
4307             return lhs_tensor;
4308         },
4309         [<Tensor Tsr>( Tsr const& lhs_input, Tsr const& rhs_input,
Tsr const&, Tsr const& ) noexcept
4310         {
4311             return std::make_tuple( zeros_like( lhs_input ), zeros_like(
rhs_input ) );
4312         },
4313         "Assign"
         )( lhs_ex, rhs_ex );
4316     };
4317
4318
4319
4331     template< Expression Ex>
4332     auto poisson(Ex const& ex ) noexcept
4333     {
4334         return make_unary_operator
4335         (
4336             [=]<Tensor Tsr>( Tsr const& tsr ) noexcept
4337             {
4338                 return poisson( tsr );
4339             },
4340             [<Tensor Tsr>( Tsr const&, Tsr const&, Tsr const& grad ) noexcept
4341             {
4342                 return grad;
4343             },
4344             "Poisson"
         )(ex);
4346     }
4347
4348
4349 } // namespace ceras
4350
4351 #endif // IPKVWSJOCMGVVRASCBLPYHFBCHRIVEXYBOMMDAKFAUDFYVYOOOISLRXJNUJKPJEVMLDPRDSNM
4352

```

9.25 /home/feng/workspace/github.repo/ceras/include/optimizer.hpp File Reference

```

#include "../config.hpp"
#include "../operation.hpp"
#include "../place_holder.hpp"
#include "../variable.hpp"

```

```
#include "../session.hpp"
#include "../utils/color.hpp"
#include "../utils/debug.hpp"
#include "../utils/id.hpp"
#include "../utils/enable_shared.hpp"
#include "../utils/fmt.hpp"
```

Classes

- struct [ceras::sgd](#)< Loss, T >
- struct [ceras::adagrad](#)< Loss, T >
- struct [ceras::rmsprop](#)< Loss, T >
- struct [ceras::adadelat](#)< Loss, T >
- struct [ceras::adam](#)< Loss, T >
- struct [ceras::gradient_descent](#)< Loss, T >

Namespaces

- namespace [ceras](#)

Typedefs

- template<typename Loss , typename T >
using [ceras::ada_grad](#) = adagrad< Loss, T >
- template<typename Loss , typename T >
using [ceras::rms_prop](#) = rmsprop< Loss, T >
- template<typename Loss , typename T >
using [ceras::ada_delta](#) = adadelat< Loss, T >

Variables

- auto [ceras::Adam](#)
- auto [ceras::SGD](#)
- auto [ceras::Adagrad](#)
- auto [ceras::RMSprop](#)
- auto [ceras::Adadelat](#)

9.26 optimizer.hpp

[Go to the documentation of this file.](#)

```
1 #ifndef XNRPSJMCYFXBDGNRJAWDNDIYQNGNXMRVLEHGNQWILKMTGNOVHODLLXCCNIMUUFQSMOIYHDUD
2 #define XNRPSJMCYFXBDGNRJAWDNDIYQNGNXMRVLEHGNQWILKMTGNOVHODLLXCCNIMUUFQSMOIYHDUD
3
4 #include "../config.hpp"
5 #include "../operation.hpp"
6 #include "../place_holder.hpp"
7 #include "../variable.hpp"
8 #include "../session.hpp"
9 #include "../utils/color.hpp"
10 #include "../utils/debug.hpp"
11 #include "../utils/id.hpp"
12 #include "../utils/enable_shared.hpp"
13 #include "../utils/fmt.hpp"
```

```

14
15 namespace ceras
16 {
17
18     // sgd:
19     //     - loss:
20     //     - batch_size:
21     //     - learning_rate:
22     //     - momentum:
23     //     - decay: should be very small, such as 1.0e-8
24     //     - nesterov:
25     //
26     template< typename Loss, typename T >
27     struct sgd : enable_id<sgd<Loss, T>, "sgd optimizer">, enable_shared<sgd<Loss, T>
28     {
29         typedef tensor< T > tensor_type;
30
31         Loss&      loss_;
32         T          learning_rate_;
33         T          momentum_;
34         T          decay_;
35         bool       nesterov_;
36         unsigned long iterations_;
37
38         sgd(Loss& loss, std::size_t batch_size, T learning_rate=1.0e-1, T momentum=0.0, T decay=0.0, bool
nesterov=false) noexcept :
39             loss_(loss), learning_rate_(learning_rate), momentum_(std::max(T{0}, momentum)),
40             decay_(std::max(T{0}, decay)), nesterov_(nesterov), iterations_{0}
41         {
42             better_assert( batch_size >= 1, "batch_size must be positive, but got: ", batch_size );
43             learning_rate_ /= static_cast<T>( batch_size );
44         }
45
46         void forward()
47         {
48             loss_.backward( ones<T>( {1, } ) );
49             learning_rate_ /= ( 1.0 + decay_ * iterations_ );
50             auto& ss = get_default_session<tensor_type>();
51             for ( auto [id, v] : ss.variables_ )
52             {
53                 if (v.trainable_)
54                 {
55                     auto& data = v.data();
56                     auto& gradient = v.gradient();
57                     auto& contexts = v.contexts();
58                     if ( contexts.empty() ) // create context
59                         contexts.push_back( zeros_like( data ) );
60                     auto& moments = contexts[0];
61                     for_each( moments.begin(), moments.end(), gradient.begin(), [this]( T& m, T g ) { m
*= (*this).momentum_; m -= (*this).learning_rate_ * g; } );
62                     if (!nesterov_) for_each( moments.begin(), moments.end(), data.begin(),
gradient.begin(), [this]( T m, T& v, T g ) { v += (*this).momentum_ * m - (*this).learning_rate_ * g;
} );
63                     else data += moments;
64                 }
65                 gradient.reset(); // clear variable gradient
66             }
67             ++iterations_;
68         } //sgd::forward
69     }; //sgd
70
71     template< typename Loss, typename T >
72     struct adagrad : enable_id<adagrad<Loss, T>, "adagrad optimizer">, enable_shared<adagrad<Loss,T>
73     {
74         typedef tensor< T > tensor_type;
75
76         Loss&      loss_;
77         T          learning_rate_;
78         T          decay_;
79         unsigned long iterations_;
80
81         adagrad(Loss& loss, std::size_t batch_size, T learning_rate=1.0e-1, T decay=0.0) noexcept :
82             loss_(loss), learning_rate_(learning_rate), decay_(std::max(T{0}, decay)), iterations_{0}
83         {
84             better_assert( batch_size >= 1, "batch_size must be positive, but got: ", batch_size );
85             learning_rate_ /= static_cast<T>( batch_size );
86         }
87
88         void forward()
89         {
90             loss_.backward( ones<T>( {1, } ) );
91
92             learning_rate_ /= ( 1.0 + decay_ * iterations_ );
93
94             auto& ss = get_default_session<tensor_type>();//.get();
95             for ( auto [id, v] : ss.variables_ )

```



```

96         {
97             if (v.trainable_)
98             {
99                 auto& data = v.data();
100                 auto& gradient = v.gradient();
101                 auto& contexts = v.contexts();
102                 if ( contexts.empty() ) // create context
103                     contexts.push_back( zeros_like( data ) );
104                 //contexts.push_back( std::make_shared<tensor_type>( zeros_like( data ) ) );
105                 auto& moments = contexts[0];
106
107                 for_each( moments.begin(), moments.end(), gradient.begin(), []( T& m, T g ) { m +=
g*g; } );
108
109                 for_each( data.begin(), data.end(), gradient.begin(), moments.begin(), [this]( T& d,
T g, T m ) { d -= (*this).learning_rate_ * g / (eps + std::sqrt(m)); } );
110
111                 gradient.reset(); // clear variable gradient
112             }
113         }
114         ++iterations_;
115     } //forward
116 }; //adagrad
117
118 template< typename Loss, typename T >
119 using ada_grad = adagrad<Loss, T>;
120
121 template< typename Loss, typename T >
122 struct rmsprop : enable_id< rmsprop< Loss, T >, "rmsprop optimizer" >, enable_shared<rmsprop<Loss,
T>
123 {
124     typedef tensor< T > tensor_type;
125
126     Loss&      loss_;
127     T          learning_rate_;
128     T          rho_;
129     T          decay_;
130     unsigned long iterations_;
131
132     rmsprop(Loss& loss, std::size_t batch_size, T learning_rate=1.0e-1, T rho=0.9, T decay=0.0)
noexcept :
133         loss_(loss), learning_rate_(learning_rate), rho_(rho), decay_(std::max(T{0}, decay)),
iterations_{0}
134     {
135         better_assert( batch_size >= 1, "batch_size must be positive, but got:  ", batch_size );
136         learning_rate_ /= static_cast<T>( batch_size );
137     }
138
139     void forward()
140     {
141         loss_.backward( ones<T>( {1, } ) );
142
143         learning_rate_ /= ( 1.0 + decay_ * iterations_ );
144
145         auto& ss = get_default_session<tensor_type>(); //get();
146         for ( auto [id, v] : ss.variables_ )
147         {
148             if (v.trainable_)
149             {
150                 auto& data = v.data();
151                 auto& gradient = v.gradient();
152                 auto& contexts = v.contexts();
153                 if ( contexts.empty() ) // create context
154                     contexts.push_back( zeros_like( data ) );
155                 //contexts.push_back( std::make_shared<tensor_type>( zeros_like( data ) ) );
156                 auto& moments = contexts[0];
157
158                 if ( iterations_ == 0 )
159                     for_each( moments.begin(), moments.end(), gradient.begin(), [this]( T& m, T g )
{ m = g*g; } );
160                 else
161                     for_each( moments.begin(), moments.end(), gradient.begin(), [this]( T& m, T g )
{ m *= (*this).rho_; m += g*g*(1.0-(*this).rho_); } );
162
163                 for_each( data.begin(), data.end(), gradient.begin(), moments.begin(), [this]( T& d,
T g, T m ) { d -= (*this).learning_rate_ * g / (eps + std::sqrt(m)); } );
164
165                 gradient.reset(); // clear variable gradient
166             }
167         }
168         ++iterations_;
169     } //forward
170 }; //rmsprop
171
172 template< typename Loss, typename T >
173 using rms_prop = rmsprop< Loss, T >;
174

```

```

175     template< typename Loss, typename T >
176     struct adadelta : enable_id< adadelta< Loss, T >, "adadelta optimizer" >,
enable_shared<adadelta<Loss, T>
177     {
178         typedef tensor< T > tensor_type;
179
180         Loss&      loss_;
181         T          rho_;
182         T          learning_rate_;
183         unsigned long iterations_;
184
185         adadelta(Loss& loss, std::size_t batch_size, T rho=0.9) noexcept : loss_(loss), rho_(rho),
iterations_{0}
186         {
187             better_assert( batch_size >= 1, "batch_size must be positive, but got: ", batch_size );
188             learning_rate_ = T{1} / static_cast<T>( batch_size );
189         }
190
191         void forward()
192         {
193             loss_.backward( ones<T>( {1, } ) );
194
195             auto& ss = get_default_session<tensor_type>(); // .get();
196             for ( auto [id, v] : ss.variables_ )
197             {
198                 if (v.trainable_)
199                 {
200                     auto& data = v.data();
201                     auto& gradient = v.gradient();
202                     auto& contexts = v.contexts();
203                     if ( contexts.empty() ) // create context
204                     {
205                         //contexts.push_back( std::make_shared<tensor_type>( zeros_like( data ) ) );
206                         //contexts.push_back( std::make_shared<tensor_type>( zeros_like( data ) ) );
207                         contexts.push_back( zeros_like( data ) );
208                         contexts.push_back( zeros_like( data ) );
209                     }
210                     auto& moments = contexts[0];
211                     auto& delta = contexts[0];
212
213                     /*
214                     if (iterations_==0)
215                     {
216                         for_each( moments.begin(), moments.end(), gradient.begin(), []( T& m, T g ) { m += g*g; } );
217                         for_each( delta.begin(), delta.end(), gradient.begin(), []( T& d, T g ) { d += g*g; } );
218                     }
219                     else
220                     {
221                         // m = rho * m + (1-rho) * g * g;
222                         for_each( moments.begin(), moments.end(), gradient.begin(), [this]( T& m, T g ) { m *= (*this).rho_; m
+= g*g*(1.0-(*this).rho_); } );
223                     }
224                     */
225
226                     for_each( moments.begin(), moments.end(), gradient.begin(), [this]( T& m, T g ) { m
*= (*this).rho_; m += g*g*(1.0-(*this).rho_); } );
227
228                     // g_ = \sqrt{ (delta+eps) / (m+eps) }
229                     for_each( gradient.begin(), gradient.end(), delta.begin(), moments.begin(), [this](
T& g, T d, T m ){ g *= (*this).learning_rate_ * std::sqrt( (d+eps)/(m+eps)); } );
230                     // x = x - g_
231                     data -= gradient;
232                     // delta = rho * delta + (1-rho) * g_ * g_
233                     /*
234                     if (iterations_!=0)
235                     */
236                     for_each( delta.begin(), delta.end(), gradient.begin(), [this]( T& d, T g ) { d *=
(*this).rho_; d += (1.0-(*this).rho_) * g * g; } );
237
238                     gradient.reset(); // clear variable gradient
239                 }
240             }
241             ++iterations_;
242         } // forward
243     }; // adadelta
244
245     template< typename Loss, typename T >
246     using ada_delta = adadelta< Loss, T >;
247
248     template< typename Loss, typename T >
249     struct adam : enable_id< adam< Loss, T >, "adam optimizer" >, enable_shared<adam<Loss, T>
250     {
251         typedef tensor< T > tensor_type;
252
253         Loss&      loss_;
254         T          learning_rate_;
255         T          beta_1_;

```

```

256         T          beta_2_;
257         bool        amsgrad_;
258         unsigned long iterations_;
259
260         adam(Loss& loss, std::size_t batch_size, T learning_rate=1.0e-1, T beta_1=0.9, T beta_2=0.999,
bool amsgrad=false) noexcept :
261             loss_{loss}, learning_rate_{learning_rate}, beta_1_{beta_1}, beta_2_{beta_2}, amsgrad_{
amsgrad }, iterations_{0}
262         {
263             better_assert( batch_size >= 1, "batch_size must be positive, but got: ", batch_size );
264             learning_rate_ /= static_cast<T>( batch_size );
265         }
266
267         void forward()
268         {
269             loss_.backward( ones<T>( {1, } ) );
270             auto& ss = get_default_session<tensor_type>();//.get();
271             for ( auto [id, v] : ss.variables_ )
272             {
273                 if (v.trainable_)
274                 {
275                     auto& data = v.data();
276                     auto& gradient = v.gradient();
277                     auto& contexts = v.contexts();
278                     if ( contexts.empty() ) // create context
279                     {
280                         //contexts.push_back( std::make_shared<tensor_type>( zeros_like( data ) ) );
281                         //contexts.push_back( std::make_shared<tensor_type>( zeros_like( data ) ) );
282                         contexts.push_back( zeros_like( data ) );
283                         contexts.push_back( zeros_like( data ) );
284                     }
285                     auto& m = contexts[0];
286                     auto& v = contexts[1];
287
288                     T const b_beta_1 = beta_1_;
289                     T const b_beta_2 = beta_2_;
290
291                     for_each( m.begin(), m.end(), gradient.begin(), [b_beta_1](T& m_, T g_){ m_ *=
b_beta_1; m_ += g_*(1.0-b_beta_1); } );
292
293                     for_each( v.begin(), v.end(), gradient.begin(), [b_beta_2](T& v_, T g_){ v_ *=
b_beta_2; v_ += g_*(1.0-b_beta_2); } );
294
295                     T lr = learning_rate_ * std::sqrt( 1.0 - std::pow(beta_2_, iterations_+1) ) / ( 1.0
- std::pow(beta_1_, iterations_+1) );
296
297                     if ( iterations_ > 1 )
298                     for_each( data.begin(), data.end(), m.begin(), v.begin(), [lr]( T& d_, T m_, T
v_ ){ d_ -= lr * m_ / (eps+std::sqrt(v_)); } );
299                     else
300                     for_each( data.begin(), data.end(), gradient.begin(), [this]( T& d_, T g_ ){ d_
-= (*this).learning_rate_ * g_; } );
301
302                     gradient.reset(); // clear variable gradient
303                     // TODO: enabling amsgrad
304                 }
305             } //loop of variables
306             ++iterations_;
307         } //adam::forward
308     }; // adam
309
310
311
312     // Example usage:
313     //
314     // //session ss;
315     // auto& ss = get_default_session<tensor<float>>();
316     // auto loss = ...;
317     // auto optimizer = gradient{ loss, 1.0e-3f };
318     // for i = 1 : 1000
319     //     ss.run( loss, batch_size )
320     //     ss.run( optimizer )
321     //
322     template< typename Loss, typename T >
323     struct gradient_descent : enable_id< gradient_descent< Loss, T >, "gradient_descent optimizer" >,
enable_shared<gradient_descent<Loss, T>
324     {
325         typedef tensor< T > tensor_type;
326         Loss& loss_;
327         T learning_rate_;
328         T momentum_;
329
330         gradient_descent(Loss& loss, std::size_t batch_size, T learning_rate=1.0e-3, T momentum=0.0)
noexcept : loss_(loss), learning_rate_(learning_rate), momentum_(momentum)
331         {
332             learning_rate_ /= static_cast<T>( batch_size ); // fix for batch size
333         }

```

```

334
335     void forward()
336     {
337         // update the gradient in the loss
338         loss_.backward( ones<T>( {1, } ) );
339         //update variables
340         auto& ss = get_default_session<tensor_type>(); //get();
341         for ( auto& [id, v] : ss.variables_ )
342         {
343             if (v.trainable_)
344             {
345                 //v.data() -= learning_rate_ * (v.gradient());
346                 //
347                 auto& gradient = v.gradient();
348                 has a nan value." );
349                 better_assert( !has_nan(gradient), "gradient_descent error, tensor with id ", id, "
350                 v.data() -= learning_rate_ * gradient;
351                 if (0)
352                 {
353                     std::ofstream ofs( fmt::format("./debug/weight_{}.txt", id) );
354                     ofs << v.gradient() << std::endl;
355                     ofs.close();
356                     better_assert( false, "stop here!" );
357                 }
358                 gradient.reset(); // clear variable gradient
359             }
360         }
361     };
362
363 // TODO: adamax, nadam, ftrl
364
365 //
366 // optimizers interfaces
367 //
368
369 inline auto Adam = [] ( auto ... args )
370 {
371     return [=]<Expression Ex>( Ex& loss )
372     {
373         return adam{loss, args...};
374     };
375 };
376
377 inline auto SGD = [] ( auto ... args )
378 {
379     return [=]<Expression Ex>( Ex& loss )
380     {
381         return sgd{loss, args...};
382     };
383 };
384
385 inline auto Adagrad = [] ( auto ... args )
386 {
387     return [=]<Expression Ex>( Ex& loss )
388     {
389         return adagrad{loss, args...};
390     };
391 };
392
393 inline auto RMSprop = [] ( auto ... args )
394 {
395     return [=]<Expression Ex>( Ex& loss )
396     {
397         return rmsprop{loss, args...};
398     };
399 };
400
401 inline auto Adadelta = [] ( auto ... args )
402 {
403     return [=]<Expression Ex>( Ex& loss )
404     {
405         return adadelta{loss, args...};
406     };
407 };
408
409 //namespace ceras
410 #endif//XNRPSJMCYFXBDGNRJAWDNDIYQNGNXMRVLEHGNQWILKMTGHGNOVHODLLXCCNIMUUFQSMOIYHDUD
411

```

9.27 /home/feng/workspace/github.repo/ceras/include/place_holder.hpp File Reference

```
#include "../includes.hpp"
#include "../tensor.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/debug.hpp"
#include "../utils/id.hpp"
#include "../utils/enable_shared.hpp"
#include "../utils/state.hpp"
```

Classes

- struct [ceras::place_holder_state< Tsor >](#)
- struct [ceras::place_holder< Tsor >](#)
- struct [ceras::is_place_holder< T >](#)
- struct [ceras::is_place_holder< place_holder< Tsor > >](#)

Namespaces

- namespace [ceras](#)

Concepts

- concept [ceras::Place_Holder](#)

Functions

- [template<Place_Holder Ph>](#)
[bool ceras::operator==](#) (Ph const &lhs, Ph const &rhs)
- [template<Place_Holder Ph>](#)
[bool ceras::operator!=](#) (Ph const &lhs, Ph const &rhs)
- [template<Place_Holder Ph>](#)
[bool ceras::operator<](#) (Ph const &lhs, Ph const &rhs)
- [template<Place_Holder Ph>](#)
[bool ceras::operator>](#) (Ph const &lhs, Ph const &rhs)
- [template<Place_Holder Ph>](#)
[bool ceras::operator<=](#) (Ph const &lhs, Ph const &rhs)
- [template<Place_Holder Ph>](#)
[bool ceras::operator>=](#) (Ph const &lhs, Ph const &rhs)

Variables

- [template<class T >](#)
[constexpr bool ceras::is_place_holder_v](#) = [is_place_holder<T>::value](#)

9.28 place_holder.hpp

[Go to the documentation of this file.](#)

```

1  #ifndef JPSBQEEADUPURARCCBVXOLXVMQHTNWCQWXUKKHCOTWFGOGXSODKEEYLSSTFGTVXNBROLKKEJM
2  #define JPSBQEEADUPURARCCBVXOLXVMQHTNWCQWXUKKHCOTWFGOGXSODKEEYLSSTFGTVXNBROLKKEJM
3
4  #include "../includes.hpp"
5  #include "../tensor.hpp"
6  #include "../utils/better_assert.hpp"
7  #include "../utils/debug.hpp"
8  #include "../utils/id.hpp"
9  #include "../utils/enable_shared.hpp"
10 #include "../utils/state.hpp"
11
12 namespace ceras
13 {
14
15     template< Tensor Tsor >
16     struct place_holder_state
17     {
18         Tsor data_;
19         std::vector< unsigned long> shape_hint_;
20     };
21
22     template< Tensor Tsor >
23     struct place_holder : enable_id< place_holder<Tsor>, "PlaceHolder" >,
24         enable_shared_state<place_holder<Tsor>, place_holder_state<Tsor>
25     {
26         typedef Tsor tensor_type;
27
28         place_holder( place_holder const& other) = default;
29         place_holder( place_holder && other) = default;
30         place_holder& operator = ( place_holder const& other) = default;
31         place_holder& operator = ( place_holder && other) = default;
32
33         place_holder()
34         {
35             (*this).state_ = std::make_shared<place_holder_state<Tsor>>();
36             ((*this).state_).shape_hint_ = std::vector< unsigned long >{ {-1UL}, };
37         }
38
39         place_holder( std::vector<unsigned long> const& shape_hint )
40         {
41             (*this).state_ = std::make_shared<place_holder_state<Tsor>>();
42             ((*this).state_).shape_hint_ = shape_hint;
43             if ( shape_hint[0] != -1UL )
44             {
45                 auto & si = ((*this).state_).shape_hint_;
46                 si.insert( si.begin(), 1UL );
47             }
48         }
49
50         void bind( Tsor data )
51         {
52             better_assert( (*this).state_, "Error with empty state." );
53             ((*this).state_).data_ = data;
54             ((*this).state_).shape_hint_ = data.shape();
55         }
56
57         Tsor const forward()const
58         {
59             better_assert( (*this).state_, "Error with empty state." );
60             better_assert( !((*this).state_).data_.empty(), "Error with empty tensor." );
61             return ((*this).state_).data_;
62         }
63
64         void reset() noexcept
65         {
66             ((*this).state_).data_ = Tsor{};
67             ((*this).state_).shape_hint_ = std::vector<unsigned long>{{-1UL}};
68         }
69
70         void backward( auto ) const noexcept { }
71
72         void shape( std::vector< unsigned long> const& shape_hint ) noexcept
73         {
74             ((*this).state_).shape_hint_ = shape_hint;
75         }
76
77         std::vector<unsigned long> shape() const noexcept
78         {
79             if ( ! ((*this).state_).data_.empty() )
80                 return ((*this).state_).data_.shape();
81
82             //debug_log( fmt::format("calculating the shape for place_holder with id {} ", (*this).id())
83 );

```

```

82         //debug_log( fmt::format("got {} ", ((*this).state_).shape_hint_ ) );
83         return ((*this).state_).shape_hint_;
84     }
85
86 };
87
88 template< typename T >
89 struct is_place_holder : std::false_type {};
90
91 template< Tensor Tsor >
92 struct is_place_holder< place_holder< Tsor > > : std::true_type {};
93
94 template< class T >
95 inline constexpr bool is_place_holder_v = is_place_holder<T>::value;
96
97 template< typename T >
98 concept Place_Holder = is_place_holder_v<T>;
99
100 template< Place_Holder Ph >
101 bool operator == ( Ph const& lhs, Ph const& rhs )
102 {
103     return lhs.id() == rhs.id();
104 }
105
106 template< Place_Holder Ph >
107 bool operator != ( Ph const& lhs, Ph const& rhs )
108 {
109     return lhs.id() != rhs.id();
110 }
111
112 template< Place_Holder Ph >
113 bool operator < ( Ph const& lhs, Ph const& rhs )
114 {
115     return lhs.id() < rhs.id();
116 }
117
118 template< Place_Holder Ph >
119 bool operator > ( Ph const& lhs, Ph const& rhs )
120 {
121     return lhs.id() < rhs.id();
122 }
123
124 template< Place_Holder Ph >
125 bool operator <= ( Ph const& lhs, Ph const& rhs )
126 {
127     return lhs.id() <= rhs.id();
128 }
129
130 template< Place_Holder Ph >
131 bool operator >= ( Ph const& lhs, Ph const& rhs )
132 {
133     return lhs.id() >= rhs.id();
134 }
135
136 } //namespace ceras
137
138 #endif//JPSBQEEADUPURARCCBVXOLXVMQHTNWCQWXUKKHCOTWFGOGXSODKEEYLSSTFGTVXNBROLKKEJM
139

```

9.29 /home/feng/workspace/github.repo/ceras/include/recurrent.hpp File Reference

```

#include "../constant.hpp"
#include "../session.hpp"
#include "../operation.hpp"
#include "../activation.hpp"
#include "../layer.hpp"

```

Namespaces

- namespace [ceras](#)

Functions

- auto `ceras::lstm` (`std::unsigned long units`) `noexcept`

9.29.1 Variable Documentation

9.29.1.1 `units_`

`unsigned long units_`

9.30 `recurrent.hpp`

[Go to the documentation of this file.](#)

```

1 #ifndef QGIVVEUESCDFTUPFLDJSNBGBWHLCAILBAKEBNCOGGOBHFQUQOTGENHFEHKQDGCDEVWQSMJOQL
2 #define QGIVVEUESCDFTUPFLDJSNBGBWHLCAILBAKEBNCOGGOBHFQUQOTGENHFEHKQDGCDEVWQSMJOQL
3
4 #include "constant.hpp"
5 #include "session.hpp"
6 #include "operation.hpp"
7 #include "activation.hpp"
8 #include "layer.hpp"
9
10 namespace ceras
11 {
12
13     namespace
14     {
15         struct lstm_context
16         {
17             unsigned long units_;
18         };
19
20
21
22         template< Expression Ex, Expression Ey >
23         auto copy_state( Ex const& ex, Ey const& ey ) noexcept
24         {
25             std::shared_ptr<std::any> forward_cache = std::make_shared<std::any>();
26             return make_binary_operator( [forward_cache]<Tensor Tsor>( Tsor const&, Tsor const&
rhs_tensor ) noexcept
27                                     {
28                     Tsor& ans = context_cast<Tsor>( forward_cache );
29                     ans.resize( rhs_tensor.shape() ); // note: when batch_size
differs, the shape migh change
30                     std::copy( rhs_tensor.begin(), rhs_tensor.end(), ans.begin()
);
31                     return ans;
32                 },
33                 [<Tensor Tsor>( Tsor const& lhs_input, Tsor const& rhs_input,
Tsor const&, Tsor const& ) noexcept
34                 {
35                     auto z = zeros_like( lhs_input );
36                     return std::make_tuple( z, z );
37                 },
38                 "CopyState"
39             )( lhs_ex, rhs_ex );
40     }
41
42 } // anonymous namespace
43
44
45 inline auto lstm( std::unsigned long units ) noexcept
46 {
47     std::shared_ptr<std::any> short_term_memory = std::make_shared<std::any>();
48     std::shared_ptr<std::any> long_term_memory = std::make_shared<std::any>();
49
50     return [=]<Expression Ex>( Ex const& ex ) noexcept
51     {
52         variable h_; // previous h

```



```

53         varialbe c_; // previous c
54
55         auto hx = concatenate( -1 )( h_, ex );
56         auto f = sigmoid( Dense( units, units+units )( hx ) );
57         auto i = sigmoid( Dense( units, units+units )( hx ) );
58         auto ca = tanh( Dense( units, units+units )( hx ) );
59         auto c = hadamard_product( f, c_ ) + hadamard_product( i, ca );
60
61         auto o = sigmoid( Dense( units, units+units )( hx ) );
62         auto h = hadamard_product( o, tanh( c ) );
63
64         auto reserve_h = zeros_like( assign( h_, h ) );
65         auto reserve_c = zeros_like( assign( c_, c ) );
66
67         return o + reserve_h + reserve_c; // 'reserve_h' and 'reserve_c' are zeros. They are here
        just to prevent 'reserve_h' and 'reserve_c' from being optimized out.
68     };
69 };
70
71
72
73
74 } // namespace ceras
75
76 #endif // QGIVVEUESCDFJTUPFLDJSNBGBWHLCAILBAKEBNCOGGOBHFQUQOTGENHFEHKQDGCDEVWQSMJOQL
77

```

9.31 /home/feng/workspace/github.repo/ceras/include/session.hpp File Reference

```

#include "../includes.hpp"
#include "../tensor.hpp"
#include "../place_holder.hpp"
#include "../variable.hpp"
#include "../utils/singleton.hpp"
#include "../utils/debug.hpp"
#include "../utils/lzw.hpp"
#include "../utils/fmt.hpp"

```

Classes

- struct [ceras::ceras_private::session< Tsor >](#)

Namespaces

- namespace [ceras](#)
- namespace [ceras::ceras_private](#)

Functions

- template<Tensor Tsor>
ceras_private::session< Tsor > & [ceras::get_default_session](#) ()
Get the default global session.
- template<Tensor Tsor>
auto & [ceras::bind](#) (place_holder< Tsor > &p_holder, Tsor const &value)
Bind a tensor to a place holder.
- template<typename Operation >
auto [ceras::run](#) (Operation &op)
Run an expression.

9.32 session.hpp

[Go to the documentation of this file.](#)

```

1  #ifndef NRFLVKIAQLDTRLNHHBYUJJAMYCRCKLQTDSDKQSALHQGURGGKBSIGGVWXBKSHQGPAUDLPUBBQ
2  #define NRFLVKIAQLDTRLNHHBYUJJAMYCRCKLQTDSDKQSALHQGURGGKBSIGGVWXBKSHQGPAUDLPUBBQ
3
4  #include "../includes.hpp"
5  #include "../tensor.hpp"
6  #include "../place_holder.hpp"
7  #include "../variable.hpp"
8  #include "../utils/singleton.hpp"
9  #include "../utils/debug.hpp"
10 #include "../utils/lzw.hpp"
11 #include "../utils/fmt.hpp"
12
13 namespace ceras
14 {
15
16     namespace ceras_private
17     {
18
19         template< Tensor Tsor >
20         struct session
21         {
22             typedef place_holder<Tsor> place_holder_type;
23             typedef variable<Tsor> variable_type;
24             typedef variable_state<Tsor> variable_state_type;
25
26             std::vector<place_holder_type> place_holders_;
27             std::map<int, variable_type> variables_;
28
29             session() { }
30
31             session( session const& ) = delete;
32             session( session&& ) = default;
33             session& operator=( session const& ) = delete;
34             session& operator=( session&& ) = default;
35
36             session& rebind( place_holder_type& p_holder, Tsor const& value )
37             {
38                 p_holder.bind( value );
39                 return *this;
40             }
41
42             session& bind( place_holder_type& p_holder, Tsor const& value )
43             {
44                 p_holder.bind( value );
45                 place_holders_.emplace_back( p_holder );
46                 return *this;
47             }
48
49             session& remember( variable_type const& v )
50             {
51                 if ( variables_.find( v.id_ ) == variables_.end() )
52                     variables_.insert( {v.id_, v} );
53                 return *this;
54             }
55
56             template< typename Operation >
57             auto run( Operation& op )const
58             {
59                 return op.forward();
60             }
61
62             // register variables associated to the op to this session
63             // usually being called before restoring a session from a file
64             template< typename Operation >
65             void tap( Operation& op )const
66             {
67                 run( op );
68             }
69
70             void deserialize( std::string const& file_path )
71             {
72                 restore( file_path );
73             }
74
75             void serialize( std::string const& file_path )const
76             {
77                 save( file_path );
78             }
79
80             void save( std::string const& file_path )const
81             {
82                 // find a tmp file

```

```

83         std::string const& tmp_file_path = file_path + std::string(".tmp");
84
85         // save original to tmp file
86         save_original( tmp_file_path );
87
88         // compress tmp file to file_path
89         {
90             std::ifstream ifs( tmp_file_path, std::ios_base::binary );
91             std::ofstream ofs( file_path, std::ios_base::binary );
92             lzw::compress( ifs, ofs );
93         }
94
95         // remove original
96         std::remove( tmp_file_path.c_str() );
97     }
98
99     void restore( std::string const& file_path )
100     {
101         // find a tmp file
102         std::string const& tmp_file_path = file_path + std::string(".tmp");
103
104         // uncompress tmp file
105         {
106             std::ifstream ifs( file_path, std::ios_base::binary );
107             std::ofstream ofs( tmp_file_path, std::ios_base::binary );
108             lzw::decompress( ifs, ofs );
109         }
110
111         // restore original from tmp file to file_path
112         restore_original( tmp_file_path );
113
114         // remove tmp file
115         //std::remove( tmp_file_path );
116         std::remove( tmp_file_path.c_str() );
117     }
118
119     void save_original( std::string const& file_path ) const
120     {
121         std::ofstream ofs( file_path );
122         better_assert( ofs.good(), "failed to open file ", file_path );
123
124         // save id
125         for ( auto const& [id, v] : variables_ )
126         {
127             ofs << id << " ";
128         }
129         ofs << "\n";
130
131         // save tensors
132         for ( auto const& [id, v] : variables_ )
133         {
134             write_tensor( ofs, v.data() );
135         }
136
137         ofs.close();
138     }
139
140     void restore_original( std::string const& file_path )
141     {
142         std::ifstream ifs( file_path );
143         better_assert( ifs.good(), "failed to open file ", file_path );
144
145         // get list of ids from the 1st line
146         std::vector<int> ids;
147         {
148             std::string str_ids;
149             std::getline( ifs, str_ids );
150             std::stringstream ss( str_ids );
151             std::copy( std::istream_iterator<int>( ss ), std::istream_iterator<int>(),
152                 std::back_inserter( ids ) );
153         }
154
155         // restore each of the tensor, ignoring their gradients
156         for ( auto id : ids )
157         {
158             auto itor = variables_.find( id );
159             better_assert( itor != variables_.end(), "Error: unknown variable to load, the id is ",
160                 id );
161
162             auto [_id, _var] = *itor;
163             read_tensor( ifs, _var.data() );
164         }
165         ifs.close();
166     }
167     ~session()

```

```

168         {
169             for ( auto& p_holder : place_holders_ )
170                 p_holder.reset();
171             place_holders_.clear();
172             variables_.clear();
173             singleton<session<Tsor>*>::instance() = nullptr;
174         }
175     }; // session
176
177 } //namespace ceras_private
178
179 template< Tensor Tsor >
180 ceras_private::session<Tsor>& get_default_session()
181 {
182     return singleton<ceras_private::session<Tsor>*>::instance();
183 }
184
185 template< Tensor Tsor >
186 auto& bind( place_holder<Tsor>& p_holder, Tsor const& value )
187 {
188     auto& ss = get_default_session<Tsor>();
189     ss.bind( p_holder, value );
190     return ss;
191 }
192
193 template< typename Operation >
194 auto run( Operation& op )
195 {
196     typedef typename Operation::tensor_type tensor_type;
197     auto ss = get_default_session<tensor_type>();
198     return ss.run( op );
199 }
200
201 } //namespace ceras
202
203 #endif //NRFLVKIAQLDTRLNHHBYUJJAMYCRCKLQTDSDKQSALHQGURGGKBSIGGVWXBKSHQGP AUDLPUBBQ
204
205

```

9.33 /home/feng/workspace/github.repo/ceras/include/tensor.hpp File Reference

```

#include "../backend/cblas.hpp"
#include "../backend/cuda.hpp"
#include "../config.hpp"
#include "../includes.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/buffered_allocator.hpp"
#include "../utils/debug.hpp"
#include "../utils/fmt.hpp"
#include "../utils/for_each.hpp"
#include "../utils/id.hpp"
#include "../utils/range.hpp"
#include "../utils/stride_iterator.hpp"
#include "../utils/view.hpp"
#include "../tensor.tcc"

```

Classes

- struct [ceras::tensor< T, Allocator >](#)
- struct [ceras::is_tensor< T >](#)
- struct [ceras::is_tensor< tensor< T, A > >](#)

Namespaces

- namespace [ceras](#)

Concepts

- concept [ceras::Tensor](#)

Typedefs

- template<typename T >
using [ceras::default_allocator](#) = buffered_allocator< T, 256 >

Functions

- template<typename T , typename A = default_allocator<T>>
constexpr tensor< T, A > [ceras::as_tensor](#) (T val) noexcept

Variables

- static unsigned long [ceras::random_seed](#) = std::chrono::system_clock::now().time_since_epoch().count()
Random seed for the tensor library.
- static std::mt19937 [ceras::random_generator](#) {random_seed}
- template<class T >
constexpr bool [ceras::is_tensor_v](#) = is_tensor<T>::value

9.34 tensor.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef HQKGLAXWWVFBFHQNHBTQJKGUFTPCQPTPXDVNOSBDJIBHITCEKDISJYNAMCPLJDURURDAISFV
2 #define HQKGLAXWWVFBFHQNHBTQJKGUFTPCQPTPXDVNOSBDJIBHITCEKDISJYNAMCPLJDURURDAISFV
3
4 #include "../backend/cblas.hpp"
5 #include "../backend/cuda.hpp"
6 #include "../config.hpp"
7 #include "../includes.hpp"
8 #include "../utils/better_assert.hpp"
9 #include "../utils/buffered_allocator.hpp"
10 #include "../utils/debug.hpp"
11 #include "../utils/fmt.hpp"
12 #include "../utils/for_each.hpp"
13 #include "../utils/id.hpp"
14 #include "../utils/range.hpp"
15 #include "../utils/stride_iterator.hpp"
16 #include "../utils/view.hpp"
17
18 namespace ceras
19 {
20     static unsigned long random_seed = std::chrono::system_clock::now().time_since_epoch().count();
21
22     // static random number random_generator
23     static std::mt19937 random_generator{random_seed};
24
25     template< typename T >
26     using default_allocator = buffered_allocator<T, 256>;
27     //using default_allocator = std::allocator<T>;
28
29     template< typename T, typename Allocator = default_allocator<T> >
30     struct tensor : enable_id<tensor<T, Allocator>, "Tensor">
31     {

```

```

41     typedef T value_type;
42     typedef Allocator allocator;
43     typedef std::vector<T, Allocator> vector_type;
44     typedef std::shared_ptr<vector_type> shared_vector;
45     typedef tensor self_type;
46
47     std::vector<unsigned long> shape_;
48     unsigned long memory_offset_;
49     shared_vector vector_;
50
51     tensor() : shape_{std::vector<unsigned long>{}}, memory_offset_{0},
52     vector_{std::make_shared<vector_type>()} { }
53
54     constexpr tensor( std::vector<unsigned long> const& shape, std::initializer_list<T> init, const
55     Allocator& alloc = Allocator() ) : shape_{shape}, memory_offset_{0},
56     vector_{std::make_shared<vector_type>(init, alloc)}
57     {
58         better_assert( (*vector_).size() == std::accumulate( shape_.begin(), shape_.end(), 1UL,
59     [](auto x, auto y){ return x+y; } ), "Expecting vector has same size as the shape indicates." );
60     }
61
62     constexpr tensor( std::vector<unsigned long> const& shape ) : shape_{shape}, memory_offset_{0},
63     vector_{std::make_shared<vector_type>(std::accumulate(shape_.begin(), shape_.end(), 1UL, [](auto x,
64     auto y){return x+y; } ), T{0})}{}
65
66     constexpr tensor( std::vector<unsigned long> const& shape, T init ) : shape_{shape},
67     memory_offset_{0},
68     vector_{std::make_shared<vector_type>(std::accumulate(shape_.begin(), shape_.end(), 1UL, [](auto x,
69     auto y){return x+y; } ), T{0})}{}
70
71     constexpr tensor( std::vector<unsigned long> const& shape, T init ) : shape_{shape},
72     memory_offset_{0},
73     vector_{std::make_shared<vector_type>(std::accumulate(shape_.begin(), shape_.end(), 1UL, [](auto x,
74     auto y){return x+y; } ), T{0})}{}
75     {
76         std::fill( begin(), end(), init );
77     }
78
79     constexpr tensor( tensor const& other, unsigned long memory_offset ) noexcept : shape_{
80     other.shape_ }, memory_offset_{ memory_offset }, vector_{ other.vector_ } {}
81
82     constexpr tensor( self_type const& other ) noexcept : shape_{ other.shape_ }, memory_offset_{
83     other.memory_offset_ }
84     {
85         vector_ = other.vector_;
86         (*this).id_ = other.id_;
87     }
88
89     constexpr tensor( self_type && other ) noexcept : shape_{ other.shape_ }, memory_offset_{
90     other.memory_offset_ }
91     {
92         vector_ = other.vector_;
93         (*this).id_ = other.id_;
94     }
95
96     constexpr self_type& operator = ( self_type const& other ) noexcept
97     {
98         shape_ = other.shape_;
99         memory_offset_ = other.memory_offset_;
100        vector_ = other.vector_;
101        (*this).id_ = other.id_;
102        return *this;
103    }
104
105    constexpr self_type& operator = ( self_type && other ) noexcept
106    {
107        shape_ = other.shape_;
108        memory_offset_ = other.memory_offset_;
109        vector_ = other.vector_;
110        (*this).id_ = other.id_;
111        return *this;
112    }
113
114    constexpr auto begin() noexcept
115    {
116        return data();
117    }
118
119    constexpr auto begin() const noexcept
120    {
121        return data();
122    }
123
124    constexpr auto cbegin() const noexcept
125    {
126        return begin();
127    }
128
129    constexpr auto end() noexcept
130    {

```

```

152         return begin() + size();
153     }
154
155     constexpr auto end() const noexcept
156     {
157         return begin() + size();
158     }
159
160     constexpr auto cend() const noexcept
161     {
162         return end();
163     }
164
165     constexpr auto rbegin() noexcept
166     {
167         return make_reverse_iterator( end() );
168     }
169
170     constexpr auto rbegin() const noexcept
171     {
172         return make_reverse_iterator( end() );
173     }
174
175     constexpr auto crbegin() const noexcept
176     {
177         return make_reverse_iterator( cend() );
178     }
179
180     constexpr auto rend() noexcept
181     {
182         return make_reverse_iterator( begin() );
183     }
184
185     constexpr auto rend() const noexcept
186     {
187         return make_reverse_iterator( begin() );
188     }
189
190     constexpr auto crend() const noexcept
191     {
192         return make_reverse_iterator( cbegin() );
193     }
194
195     constexpr auto front()
196     {
197         return (*vector_).front();
198     }
199
200     constexpr auto front()const
201     {
202         return (*vector_).front();
203     }
204
205     constexpr auto back()
206     {
207         return (*vector_).back();
208     }
209
210     constexpr auto back()const
211     {
212         return (*vector_).back();
213     }
214
215     constexpr unsigned long size() const noexcept
216     {
217         if ( !vector_ ) return 0;
218         return (*vector_).size() - memory_offset_;
219         //return std::accumulate( shape_.begin(), shape_.end(), 1UL, [](unsigned long x, unsigned
220 long y){return x*y;} );
221     }
222
223     [[nodiscard]] constexpr bool empty() const noexcept
224     {
225         return cbegin() == cend();
226     }
227
228     constexpr self_type& reset( T val = T{} )
229     {
230         std::fill_n( data(), size(), val );
231         return *this;
232     }

```

```

289
293     constexpr unsigned long ndim() const noexcept
294     {
295         return shape_.size();
296     }
297
301     constexpr std::vector<unsigned long> const& shape() const noexcept
302     {
303         return shape_;
304     }
305
306
310     constexpr self_type& deep_copy( self_type const& other )
311     {
312         (*this).resize( other.shape() );
313         std::copy_n( other.data(), size(), (*this).data() );
314         return *this;
315     }
316
317     constexpr self_type const deep_copy()const
318 {
319     self_type ans{ shape_ };
320     std::copy_n( data(), size(), ans.data() );
321     return ans;
322 }
323
324     constexpr self_type const copy()const
325 {
326     return deep_copy();
327 }
328
329     // 1-D view
330     constexpr value_type& operator[]( unsigned long idx )
331     {
332         return *(data()+idx);
333     }
334
335     // 1-D view
336     constexpr value_type const& operator[]( unsigned long idx )const
337 {
338     return *(data()+idx);
339 }
340
344     constexpr self_type& resize( std::vector< unsigned long > const& new_shape )
345     {
346         unsigned long const new_size = std::accumulate( new_shape.begin(), new_shape.end(), 1UL,
347 [] (auto x, auto y){ return x*y; } );
348         if ( (*this).size() != new_size )
349         {
350             (*vector_).resize(new_size);
351             memory_offset_ = 0UL;
352         }
353         (*this).shape_ = new_shape;
354         return *this;
355     }
356
365     constexpr self_type& reshape( std::vector<unsigned long> const& new_shape )
366     {
367         std::vector<unsigned long> _new_shape = new_shape;
368         if ( *(_new_shape.rbegin()) == static_cast<unsigned long>( -1 ) )
369             *(_new_shape.rbegin()) = (*this).size() / std::accumulate( _new_shape.begin(),
370 _new_shape.end()-1, 1UL, [] ( unsigned long x, unsigned long y ){ return x*y; } );
371
372         unsigned long const new_size = std::accumulate( _new_shape.begin(), _new_shape.end(), 1UL,
373 [] (auto x, auto y){ return x*y; } );
374         if ( (*this).size() != new_size ) return resize( _new_shape );
375
376         better_assert( (*this).size() == new_size, "reshape: expecting same size, but the original
377 size is ", (*this).size(), ", and the new size is ", new_size );
378         (*this).shape_ = _new_shape;
379         return *this;
380     }
381
382     //mapping a smaller tensor on a larger one
383     constexpr self_type& shrink_to( std::vector< unsigned long > const& new_shape )
384     {
385         unsigned long const new_size = std::accumulate( new_shape.begin(), new_shape.end(), 1UL,
386 [] (auto x, auto y){ return x*y; } );
387         better_assert( (*this).size() >= new_size, "reshape: expecting smaller size, but the
388 original size is ", (*this).size(), ", and the new size is ", new_size );
389         (*this).shape_ = new_shape;
390         return *this;
391     }
392
393     //adjust the memory offset
394     constexpr self_type& creep_to( unsigned long new_memory_offset )
395     {

```



```

391         (*this).memory_offset_ = new_memory_offset;
392         return *this;
393     }
394
395     constexpr value_type* data() noexcept
396     {
397         return (*vector_).data() + memory_offset_;
398     }
399
400     constexpr const value_type* data() const noexcept
401     {
402         return (*vector_).data() + memory_offset_;
403     }
404
405     template< typename Function >
406     constexpr self_type& map( Function const& f )
407     {
408         for_each( (*this).data(), (*this).data()+(*this).size(), [&f]( auto& v ){ f(v); } );
409         return *this;
410     }
411
412     constexpr self_type& operator += ( self_type const& other )
413     {
414         //better_assert( shape() == other.shape(), "Error with tensor::operator += : Shape
415 mismatch! -- current shape is ", shape(), " and other tensor shape is ", other.shape() );
416         better_assert( shape() == other.shape(), fmt::format("Error with tensor::operator += :
417 Shape mismatch! This shape is {}, while other shape is {}.", shape(), other.shape() ) );
418         std::transform( data(), data()+size(), other.data(), data(), []( auto x, auto y ){ return
419 x+y; } );
420         return *this;
421     }
422
423     constexpr self_type& operator += ( value_type x )
424     {
425         for_each( data(), data()+size(), [x]( value_type& v ){ v += x; } );
426         return *this;
427     }
428
429     constexpr self_type& operator -= ( self_type const& other )
430     {
431         better_assert( shape() == other.shape(), "Error with tensor::operator -=: Shape not match!"
432 );
433         std::transform( data(), data()+size(), other.data(), data(), []( auto x, auto y ){ return
434 x-y; } );
435         return *this;
436     }
437
438     constexpr self_type& operator -= ( value_type x )
439     {
440         for_each( data(), data()+size(), [x]( auto& v ){ v -= x; } );
441         return *this;
442     }
443
444     constexpr self_type& operator *= ( self_type const& other )
445     {
446         better_assert( shape() == other.shape(), "Shape not match!" );
447         std::transform( data(), data()+size(), other.data(), data(), []( auto x, auto y ){ return
448 x*y; } );
449         return *this;
450     }
451
452     constexpr self_type& operator *= ( value_type x )
453     {
454         for_each( data(), data()+size(), [x]( auto& v ){ v *= x; } );
455         return *this;
456     }
457
458     constexpr self_type& operator /= ( self_type const& other )
459     {
460         better_assert( shape() == other.shape(), "Shape not match!" );
461         std::transform( data(), data()+size(), other.data(), data(), []( auto x, auto y ){ return
462 x/y; } );
463         return *this;
464     }
465
466     constexpr self_type& operator /= ( value_type x )
467     {
468         for_each( data(), data()+size(), [x]( auto& v ){ v /= x; } );
469         return *this;
470     }
471
472     constexpr self_type const operator - ()const
473     {
474         self_type ans = (*this).deep_copy();
475         for_each( ans.data(), ans.data()+size(), []( auto& v ){ v = -v; } );
476         return ans;
477     }
478
479     constexpr self_type const operator - ()const
480     {
481         self_type ans = (*this).deep_copy();
482         for_each( ans.data(), ans.data()+size(), []( auto& v ){ v = -v; } );
483         return ans;
484     }
485
486 {
487     self_type ans = (*this).deep_copy();
488     for_each( ans.data(), ans.data()+size(), []( auto& v ){ v = -v; } );
489     return ans;
490 }

```

```

491
492     constexpr value_type as_scalar() const noexcept
493     {
494         better_assert( size() == 1, "Expecting tensor has a single value, but got ", size() );
495         return *begin();
496     }
497
498     template< typename U >
499     constexpr auto as_type() const noexcept
500     {
501         tensor<U, typename std::allocator_traits<Allocator>::rebind_alloc<U> ans( (*this).shape() );
502         std::copy( (*this).begin(), (*this).end(), ans.begin() );
503         return ans;
504     }
505
506     tensor slice( unsigned long m, unsigned long n ) const noexcept
507     {
508         better_assert( m < n, "starting dimension larger than then ending dimension." );
509         better_assert( !shape_.empty(), "Cannot slice an empty tensor." );
510
511         unsigned long first_dim = shape_[0];
512         better_assert( n <= first_dim, "this tensor only has ", first_dim, " at the first dimension,
too small for n = ", n );
513
514         unsigned long rest_dims = std::accumulate( shape_.begin()+1, shape_.end(), 1UL, []( auto x,
auto y ){ return x*y; } );
515
516         tensor ans = *this;
517         ans.shape_[0] = n - m;
518         ans.memory_offset_ = rest_dims * m + memory_offset_;
519         return ans;
520     }
521 };
522
523
524 template <typename T, typename A=default_allocator<T> >
525 constexpr tensor<T, A> as_tensor( T val ) noexcept
526 {
527     tensor<T, A> ans{ {1,} };
528     ans[0] = val;
529     return ans;
530 }
531
532 template< typename T >
533 struct is_tensor : std::false_type {};
534
535 template< typename T, typename A >
536 struct is_tensor< tensor< T, A > : std::true_type {};
537
538 template< class T >
539 inline constexpr bool is_tensor_v = is_tensor<T>::value;
540
541 template< typename T >
542 concept Tensor = is_tensor_v<T>;
543
544
545 } // namespace ceras
546
547 #include "../tensor.tcc"
548
549 #endif//HQKGLAXWWVFBFHQNBVTQJGUFTPCQPTPXDVNOSBDJIBHITCEKDISJYNAMCPLJDURURDAISFV
550

```

9.35 /home/feng/workspace/github.repo/ceras/include/value.hpp File Reference

```

#include "../includes.hpp"
#include "../tensor.hpp"
#include "../utils/id.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/enable_shared.hpp"

```

Classes

- struct `ceras::value< T >`

- struct `ceras::is_value< T >`
- struct `ceras::is_value< value< T > >`
- struct `ceras::tensor_deduction< L, R >`

Namespaces

- namespace `ceras`

Concepts

- concept `ceras::Value`

Variables

- template<class T >
constexpr bool `ceras::is_value_v` = `is_value<T>::value`

9.36 value.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef VALUE_HPP_INCLUDED_DS9P8IU4LKJASDOIPUY498YAFKASHFAS9F8Y4OKHDAFSIUOHASDFFS
2 #define VALUE_HPP_INCLUDED_DS9P8IU4LKJASDOIPUY498YAFKASHFAS9F8Y4OKHDAFSIUOHASDFFS
3
4 #include "includes.hpp"
5 #include "tensor.hpp"
6 #include "utils/id.hpp"
7 #include "utils/better_assert.hpp"
8 #include "utils/enable_shared.hpp"
9
10 namespace ceras
11 {
12
13     template< typename T > requires std::floating_point<T>
14     struct value : enable_id< value<T>, "Value" >
15     {
16         typedef T value_type;
17         typedef tensor<value_type> tensor_type;
18         value_type data_;
19
20         value() = delete;
21         value( value_type v ) noexcept : enable_id<value<T>, "Value">{}, data_{ v } {}
22         value( value const& ) noexcept = default;
23         value( value && ) noexcept = default;
24         value& operator=( value const& ) noexcept = default;
25         value& operator=( value && ) noexcept = default;
26
27         void backward( auto ) noexcept { }
28
29         template< Tensor Tsor >
30         Tsor const forward( Tsor const& refer )const
31     {
32         Tsor ans = ones_like( refer ); // cast it to a tensor
33         ans *= data_;
34         return ans;
35     }
36
37     std::vector<unsigned long> shape() const noexcept
38     {
39         return std::vector<unsigned long>{ {-1UL}, };
40     }
41
42 }; //struct value
43
44 template< typename T >
45 struct is_value : std::false_type {};
46
47 template< typename T >

```

```

48     struct is_value< value< T > > : std::true_type {};
49
50     template< class T >
51     inline constexpr bool is_value_v = is_value<T>::value;
52
53     template< typename T >
54     concept Value = is_value_v<T>;
55
56
57     // for tensor_type deduction in a binary operator
58     template< typename L, typename R >
59     struct tensor_deduction
60     {
61         using op_type = std::conditional<is_value_v<L>, R, L::type>;
62         using tensor_type = std::remove_cv_t<decltype(std::declval<op_type>().forward())>;
63     };
64
65
66 } // namespace ceras
67
68 #endif // VALUE_HPP_INCLUDED_DS9P8IU4LKJASDOIPUY498YAFKASHFAS9F8Y4OKHDAFSIUOHASDFFS
69

```

9.37 /home/feng/workspace/github.repo/ceras/include/variable.hpp File Reference

```

#include "../includes.hpp"
#include "../tensor.hpp"
#include "../utils/id.hpp"
#include "../utils/debug.hpp"
#include "../config.hpp"
#include "../utils/enable_shared.hpp"
#include "../utils/state.hpp"

```

Classes

- struct [ceras::variable_state< Tsor >](#)
- struct [ceras::regularizer< Float >](#)
- struct [ceras::variable< Tsor >](#)
- struct [ceras::is_variable< T >](#)
- struct [ceras::is_variable< variable< Tsor > >](#)

Namespaces

- namespace [ceras](#)
- namespace [ceras::ceras_private](#)

Concepts

- concept [ceras::Variable](#)

Functions

- template<Tensor Tsor>
[ceras_private::session< Tsor > & ceras::get_default_session \(\)](#)
Get the default global session.
- template<Variable Var>
[bool ceras::operator== \(Var const &lhs, Var const &rhs\) noexcept](#)

Variables

- `template<class T >`
`constexpr bool ceras::is_variable_v = is_variable<T>::value`

9.38 variable.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef QVETFLYKDJJLDBPAMVBWUGPWXIAIGMXUDVOFGQIHUHVOTBAWEMPJQEWJQIGGSTUCNDHLUYL
2 #define QVETFLYKDJJLDBPAMVBWUGPWXIAIGMXUDVOFGQIHUHVOTBAWEMPJQEWJQIGGSTUCNDHLUYL
3
4 #include "../includes.hpp"
5 #include "../tensor.hpp"
6 #include "../utils/id.hpp"
7 #include "../utils/debug.hpp"
8 #include "../config.hpp"
9 #include "../utils/enable_shared.hpp"
10 #include "../utils/state.hpp"
11
12 namespace ceras
13 {
14
15     namespace ceras_private
16     {
17         template< Tensor Tsor >
18         struct session;
19     }
20
21     template< Tensor Tsor >
22     ceras_private::session<Tsor>& get_default_session();
23
24     template< Tensor Tsor >
25     struct variable_state
26     {
27         Tsor data_;
28         Tsor gradient_;
29         std::vector<Tsor> contexts_;
30     };
31
32     template< typename Float > requires std::floating_point<Float>
33     struct regularizer
34     {
35         typedef Float value_type;
36         value_type l1_;
37         value_type l2_;
38         bool synchronized_;
39
40         constexpr regularizer( value_type l1=0.0, value_type l2=0.0, bool synchronized=false ) noexcept :
41             l1_{l1}, l2_{l2}, synchronized_{synchronized} {}
42     };
43
44     template< Tensor Tsor >
45     struct variable : enable_id<variable<Tsor>, "Variable">
46     {
47         typedef Tsor tensor_type;
48         typedef typename tensor_type::value_type value_type;
49
50         std::shared_ptr<variable_state<tensor_type>> state_;
51         regularizer<value_type> regularizer_;
52         bool trainable_;
53
54         variable( tensor_type const& data, value_type l1=value_type{0}, value_type l2=value_type{0}, bool
55             trainable=true ) : enable_id<variable<tensor_type>, "Variable">{}, regularizer_{l1, l2, true},
56             trainable_{trainable}
57         {
58             (*this).state_ = std::make_shared<variable_state<tensor_type>>();
59             ((*this).state_).data_ = data;
60             ((*this).state_).gradient_ = tensor_type{ data.shape() };
61
62             auto& ss = get_default_session<tensor_type>();
63             ss.remember( *this );
64         }
65
66         //variable() = delete;
67         variable() noexcept {}
68         variable( variable const& other ) = default;
69         variable( variable && ) = default;
70         variable& operator=( variable&& ) = default;
71         variable& operator=( variable const& other) = default;
72
73     };
74
75     }
76
77 }
```

```

70     tensor_type const forward() noexcept// const
71     {
72         auto& state = *((*this).state_);
73
74         if ( learning_phase == 1 )
75         {
76             typedef typename tensor_type::value_type value_type;
77             state.gradient_.reset( value_type{0} );
78             regularizer_.synchronized_ = false; // mark changes
79         }
80         return state.data_;
81     }
82
83     void backward( auto const& grad ) noexcept
84     {
85         if (!trainable_) return;
86
87         auto& state = *((*this).state_);
88         {
89             if (state.gradient_.shape() != state.data_.shape())
90                 state.gradient_.resize( state.data_.shape() );
91         }
92         state.gradient_ += grad; // collecting all the gradients from its children nodes, will be
called multiple times in a single backward pass
93
94         // apply regularizers
95         if (!(regularizer_.synchronized_)) // in case of multiple invoke of this method in a same
backward pass
96         {
97             if ( regularizer_.l1_ >= eps ) // l1 regularizer
98             {
99                 value_type const factor = regularizer_.l1_;
100                 for_each( state.data_.begin(), state.data_.end(), state.gradient_.begin(), [factor](
value_type d, value_type& g ){ g += (d >= value_type{0}) ? factor : -factor; } );
101             }
102             if ( regularizer_.l2_ >= eps ) // l2 regularizer
103             {
104                 value_type const factor = regularizer_.l2_;
105                 for_each( state.data_.begin(), state.data_.end(), state.gradient_.begin(), [factor](
value_type d, value_type& g ){ g += value_type{2} * d * factor; } );
106             }
107             regularizer_.synchronized_ = true;
108         }
109     }
110
111     std::vector<std::size_t> shape() const noexcept
112     {
113         auto& state = *((*this).state_);
114         //debug_log( fmt::format("calculating the shape of variable with id {}, got {}",
115 (*this).id(), state.data_.shape() ) );
116         return state.data_.shape();
117     }
118
119     std::vector<tensor_type>& contexts()
120     {
121         auto& state = *((*this).state_);
122         return state.contexts_;
123     }
124
125     std::vector<tensor_type> contexts()const
126     {
127         auto& state = *((*this).state_);
128         return state.contexts_;
129     }
130
131     tensor_type& data()
132     {
133         auto& state = *((*this).state_);
134         return state.data_;
135     }
136
137     tensor_type data()const
138     {
139         auto& state = *((*this).state_);
140         return state.data_;
141     }
142
143     tensor_type& gradient()
144     {
145         auto& state = *((*this).state_);
146         return state.gradient_;
147     }
148
149     tensor_type gradient()const
150     {
151         auto& state = *((*this).state_);

```

```

152         return state.gradient_;
153     }
154
155     void reset()
156     {
157         data().reset();
158         gradient().reset();
159     }
160
161     /*
162 void reset_states()
163 {
164 if ( stateful_ )
165 reset();
166 }
167 */
168
169 bool trainable() const noexcept { return trainable_; }
170 void trainable( bool t ) { trainable_ = t; }
171 /*
172 bool stateful() const noexcept { return stateful_; }
173 void stateful( bool s ){ stateful_ = s; }
174 */
175
176 }; //struct variable
177
178 template< typename T >
179 struct is_variable : std::false_type {};
180
181 template< Tensor Tsor >
182 struct is_variable< variable<Tsor> > : std::true_type {};
183
184 template< class T >
185 inline constexpr bool is_variable_v = is_variable<T>::value;
186
187 template< typename T >
188 concept Variable = is_variable_v<T>;
189
190 template< Variable Var >
191 bool operator == ( Var const& lhs, Var const& rhs ) noexcept
192 {
193     return lhs.id_ == rhs.id_;
194 }
195
196 } //namespace ceras
197
198 #endif//QVETFLYKDJJLDBPAMVBWUGPWXIAIGMXUDVOFGQIHUHVOTBAWEMPJQEWJQIGGSTUCNDHLUYL
199

```

9.39 /home/feng/workspace/github.repo/ceras/include/xmodel.hpp File Reference

```

#include "../includes.hpp"
#include "../operation.hpp"
#include "../place_holder.hpp"
#include "../session.hpp"
#include "../tensor.hpp"
#include "../utils/better_assert.hpp"
#include "../utils/context_cast.hpp"
#include "../utils/tqdm.hpp"
#include "../utils/list.hpp"
#include "../utils/debug.hpp"

```

Classes

- struct [ceras::model< Ex, Ph >](#)

Namespaces

- namespace [ceras](#)

9.40 xmodel.hpp

[Go to the documentation of this file.](#)

```

1 #ifndef BPLYFMIFNNWSGMLLEKBJMAJDBRSPHHRAYMOHTWSTCMNMFSLLYNQTTCCAQXKXSLMSLKESHRA
2 #define BPLYFMIFNNWSGMLLEKBJMAJDBRSPHHRAYMOHTWSTCMNMFSLLYNQTTCCAQXKXSLMSLKESHRA
3
4 #include "../includes.hpp"
5 #include "../operation.hpp"
6 #include "../place_holder.hpp"
7 #include "../session.hpp"
8 #include "../tensor.hpp"
9 #include "../utils/better_assert.hpp"
10 #include "../utils/context_cast.hpp"
11 #include "../utils/tqdm.hpp"
12 #include "../utils/list.hpp"
13 #include "../utils/debug.hpp"
14
15 namespace ceras
16 {
17
18     template< List Input_List, List Output_List >
19     struct model
20     {
21         Input_List input_layer_;
22         Output_List output_layer_;
23
24         constexpr model( Input_List const& input_layer, Output_List const& output_layer ) :
25             input_layer_( input_layer ), output_layer_( output_layer ) {}
26
27         auto constexpr input()const { return input_layer_; }
28
29         auto constexpr output()const { return output_layer_; }
30
31         template< Tensor Tsor>
32         auto constexpr predict( Tsor const& input_tensor ) const // in case of only one input layer
33         {
34             //constexpr debug<length( input_layer_ )> = 0;
35
36             better_assert( length( input_layer_ ) == 1, "Expecting the model only have a single input
37 layer, but get ", length( input_layer_ ) );
38
39             auto& s = get_default_session<Tsor>();
40             s.bind( car(input_layer_), input_tensor );
41
42             learning_phase = 0; // for different behaviours in normalization and drop-out layers
43
44             if constexpr( length(output_layer_) == 1 )
45             {
46                 // if the model has a single output layer, return a tensor.
47                 Tsor ans = s.run( car(output_layer_) );
48                 learning_phase = 1; // restore learning phase
49                 return ans;
50             }
51             else
52             {
53                 // if the model has multiple output layer, return a tuple of tensors
54                 auto ans = map( [&s]<Expression Ex>( Ex const& ex ){ return s.run(ex); }, output_layer_
55 );
56                 learning_phase = 1; // restore learning phase
57                 return ans.as_tuple();
58             }
59         }
60
61         template< List Tsor_List >
62         auto predict( Tsor_List const& input_tensor )const
63         {
64             static_assert( length(input_tensor) == length(input_layer_), "Expecting same number of input
65 layers" );
66
67             if constexpr( length(input_tensor) == 1 )
68             {
69                 // case of single input layer
70                 return predict( car(input_tensor) );
71             }
72         }
73     };
74
75     template< List Tsor_List >
76     auto predict( Tsor_List const& input_tensor )const
77     {
78         static_assert( length(input_tensor) == length(input_layer_), "Expecting same number of input
79 layers" );
80
81         if constexpr( length(input_tensor) == 1 )
82         {
83             // case of single input layer
84             return predict( car(input_tensor) );
85         }
86     }
87 }
88
89 
```



```

92         else
93         {
94             typedef typename std::remove_cv_t<decltype( car(input_tensor) )> tensor_type;
95             auto& s = get_default_session<tensor_type>();
96             learning_phase = 0; // supress training behaviours in Dropout and BN layers
97
98             map
99             (
100                 [&s]<List List_PHolder_Tensor>( List_PHolder_Tensor const& list_of_layer_and_tensor
101 ) // invokes on sub-list of input_layer-input-tensor
102                 {
103                     list_of_layer_and_tensor
104                     (
105                         [&s]<Place_Holder Ph, Tensor Tsor>(Ph input_layer, Tsor input_tensor) //
106 invokes with an input layer (a place holder) and an input tensor
107                         {
108                             s.bind( input_layer, input_tensor );
109                         }
110                     ),
111                     zip( input_layer_, input_tensor ) // pairing input layer and input tensor as a list
112 of list -> [ [input_layer_0, input_tensor_0], [input_layer_1, input_tensor_1], ..., [input_layer_n,
113 input_tensor_n] ]
114                 );
115
116                 if constexpr( length(output_layer_) == 1 )
117                 {
118                     // if the model has a single output layer, return a tensor.
119                     tensor_type ans = s.run( car(output_layer_) );
120                     learning_phase = 1; // restore learning phase
121                     return ans;
122                 }
123                 else
124                 {
125                     // if the model has multiple output layer, return a tuple of tensors
126                     auto ans = map( [&s]<Expression Ex>( Ex const& ex ){ return s.run(ex); },
127 output_layer_ );
128                     learning_phase = 1; // restore learning phase
129                     return ans.as_tuple();
130                 }
131             }
132         }; // struct model
133
134
135 } // namespace ceras
136
137 #endif//BPLYFMIFNNWSGMLLEKBJMAJDBRSPHHRAYMOHTWSTCMNMFSLLYNQTTCCAQXKXSLMSLKESHASL

```


Index

/home/feng/workspace/github.repo/ceras/include/activation_ops.h
135, 137
operation.hpp, 175

/home/feng/workspace/github.repo/ceras/include/ceras.hpp
143
ceras, 19

/home/feng/workspace/github.repo/ceras/include/complex_operation.hpp
144, 146
ceras, 19

/home/feng/workspace/github.repo/ceras/include/config.hpp
148
Adadelta
ceras, 48

/home/feng/workspace/github.repo/ceras/include/constant.hpp
148, 149
Adadelta
ceras::adadelta< Loss, T >, 62

/home/feng/workspace/github.repo/ceras/include/dataset.hpp
150
Adagrad
ceras, 48

/home/feng/workspace/github.repo/ceras/include/includes.hpp
152, 154
Adagrad
ceras::adagrad< Loss, T >, 64

/home/feng/workspace/github.repo/ceras/include/layer.hpp
154, 156
Adam
ceras, 49

/home/feng/workspace/github.repo/ceras/include/loss.hpp
159, 160
adam
ceras::adam< Loss, T >, 66

/home/feng/workspace/github.repo/ceras/include/metric.hpp
163, 164
Add
ceras, 20

/home/feng/workspace/github.repo/ceras/include/model.hpp
164, 165
allocator
ceras::tensor< T, Allocator >, 110

/home/feng/workspace/github.repo/ceras/include/operation.hpp
168, 194
rmsgrad
ceras::adam< Loss, T >, 66

/home/feng/workspace/github.repo/ceras/include/optimizer.hpp
242, 243
amp
ceras, 20

/home/feng/workspace/github.repo/ceras/include/place_holder.hpp
249, 250
keras.hpp, operation.hpp, 175

/home/feng/workspace/github.repo/ceras/include/recurrent.hpp
251, 252
ampin
operation.hpp, 176

/home/feng/workspace/github.repo/ceras/include/session.hpp
253, 254
ps_scalar
ceras::tensor< T, Allocator >, 112

/home/feng/workspace/github.repo/ceras/include/tensor.hpp
256, 257
pas_tensor
ceras, 21

/home/feng/workspace/github.repo/ceras/include/value.hpp
262, 263
pas_type
ceras::tensor< T, Allocator >, 112

/home/feng/workspace/github.repo/ceras/include/variable.hpp
264, 265
ppin
operation.hpp, 176

/home/feng/workspace/github.repo/ceras/include/xmodel.hpp
267, 268
psinh
operation.hpp, 176

~session
assign
operation.hpp, 176

 ceras::ceras_private::session< Tsor >, 102
 atan
 operation.hpp, 177

abs
ceras, 20
atan2
operation.hpp, 177

abs_loss
ceras, 20
atanh
operation.hpp, 177

acos
average_pooling_2d
operation.hpp, 175

- operation.hpp, 177
- AveragePooling2D
 - ceras, 21
- back
 - ceras::tensor< T, Allocator >, 112, 113
- backward
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 69
 - ceras::constant< Tzor >, 77
 - ceras::place_holder< Tzor >, 94
 - ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 123
 - ceras::value< T >, 127
 - ceras::variable< Tzor >, 130
- backward_action_
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 70
 - ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 124
- batch_normalization
 - operation.hpp, 177
- BatchNormalization
 - ceras, 21
- begin
 - ceras::tensor< T, Allocator >, 113
- beta_1_
 - ceras::adam< Loss, T >, 66
- beta_2_
 - ceras::adam< Loss, T >, 67
- binary_accuracy
 - ceras, 22
- binary_cross_entropy_loss
 - ceras, 22
- binary_operator
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 69
- binary_step
 - ceras, 22
- BinaryCrossEntropy
 - ceras, 49
- BinaryCrossentropy
 - ceras, 49
- bind
 - ceras, 22
 - ceras::ceras_private::session< Tzor >, 102
 - ceras::place_holder< Tzor >, 95
- CategoricalCrossEntropy
 - ceras, 50
- CategoricalCrossentropy
 - ceras, 49
- cbegin
 - ceras::tensor< T, Allocator >, 113
- cbrt
 - operation.hpp, 178
- ceil
 - operation.hpp, 178
- cend
 - ceras::tensor< T, Allocator >, 113
- ceras, 11
 - abs, 20
 - abs_loss, 20
 - ada_delta, 19
 - ada_grad, 19
 - Adadelata, 48
 - Adagrad, 48
 - Adam, 49
 - Add, 20
 - arg, 20
 - as_tensor, 21
 - AveragePooling2D, 21
 - BatchNormalization, 21
 - binary_accuracy, 22
 - binary_cross_entropy_loss, 22
 - binary_step, 22
 - BinaryCrossEntropy, 49
 - BinaryCrossentropy, 49
 - bind, 22
 - CategoricalCrossEntropy, 50
 - CategoricalCrossentropy, 49
 - computation_graph, 23
 - Concatenate, 23
 - conj, 23
 - Conv2D, 24
 - crelu, 25
 - cross_entropy, 25
 - cross_entropy_loss, 26
 - default_allocator, 19
 - Dense, 26
 - divide, 27
 - Dropout, 27
 - elementwise_multiply, 27
 - elementwise_product, 27
 - ELU, 28
 - elu, 28
 - exponential, 28
 - Flatten, 29
 - gaussian, 29
 - gelu, 29
 - get_default_session, 29
 - hadamard_product, 30
 - hard_sigmoid, 30
 - heaviside_step, 30
 - Hinge, 50
 - hinge_loss, 30
 - imag, 31
 - Input, 31
 - inverse, 31
 - is_binary_operator_v, 50
 - is_complex_v, 50
 - is_constant_v, 50

- is_place_holder_v, 51
- is_tensor_v, 51
- is_unary_operator_v, 51
- is_value_v, 51
- is_variable_v, 51
- leaky_relu, 31
- LeakyReLU, 32
- lisht, 32
- lstm, 32
- MAE, 51
- mae, 32
- make_binary_operator, 32
- make_compiled_model, 33
- make_trainable, 33
- make_unary_operator, 33
- MaxPooling2D, 33
- mean, 33
- mean_absolute_error, 34
- mean_reduce, 34
- mean_squared_error, 34
- mean_squared_logarithmic_error, 34
- MeanAbsoluteError, 52
- MeanSquaredError, 52
- minus, 34
- mish, 35
- MSE, 52
- mse, 35
- Multiply, 35
- negative, 35
- negative_relu, 35
- norm, 36
- operator!=, 36
- operator<, 40
- operator<=, 40
- operator>, 40
- operator>=, 41
- operator*, 36, 37
- operator+, 37, 38
- operator-, 38, 39
- operator/, 39
- operator==, 40
- plus, 41
- polar, 41
- prelu, 41
- random_generator, 53
- random_seed, 53
- real, 42
- reduce_mean, 42
- reduce_sum, 42
- ReLU, 43
- relu, 42
- relu6, 43
- replace_placeholder_with_expression, 43
- Reshape, 43
- rms_prop, 19
- RMSprop, 53
- run, 44
- selu, 44
- SGD, 53
- sigmoid, 44
- silu, 45
- soft_sign, 45
- Softmax, 45
- softmax, 45
- softplus, 45
- softsign, 46
- square, 46
- squared_loss, 47
- Subtract, 47
- sum_reduce, 47
- swish, 47
- tanh_shrink, 48
- unit_step, 48
- UpSampling2D, 48
- ceras::adadelta< Loss, T >, 61
 - adadelta, 62
 - forward, 62
 - iterations_, 62
 - learning_rate_, 62
 - loss_, 62
 - rho_, 63
 - tensor_type, 61
- ceras::adagrad< Loss, T >, 63
 - adagrad, 64
 - decay_, 64
 - forward, 64
 - iterations_, 64
 - learning_rate_, 64
 - loss_, 65
 - tensor_type, 63
- ceras::adam< Loss, T >, 65
 - adam, 66
 - amsgrad_, 66
 - beta_1_, 66
 - beta_2_, 67
 - forward, 66
 - iterations_, 67
 - learning_rate_, 67
 - loss_, 67
 - tensor_type, 66
- ceras::Binary_Operator, 57
- ceras::binary_operator< Lhs_Operator, Rhc_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 67
 - backward, 69
 - backward_action_, 70
 - binary_operator, 69
 - forward, 69
 - forward_action_, 70
 - lhs_input_data_, 70
 - lhs_op_, 70
 - output_data_, 70
 - output_shape_calculator_, 70
 - rhs_input_data_, 71
 - rhs_op_, 71
 - shape, 69

tensor_type, 68
 ceras::ceras_private, 54
 ceras::ceras_private::session< Tsr >, 100
 ~session, 102
 bind, 102
 deserialize, 102
 operator=, 103
 place_holder_type, 101
 place_holders_, 105
 rebind, 103
 remember, 103
 restore, 103
 restore_original, 103
 run, 104
 save, 104
 save_original, 104
 serialize, 104
 session, 102
 tap, 104
 variable_state_type, 101
 variable_type, 101
 variables_, 105
 ceras::compiled_model< Model, Optimizer, Loss >, 71
 compiled_model, 72
 compiled_optimizer_, 75
 evaluate, 72
 fit, 73
 ground_truth_place_holder_, 75
 input_place_holder_, 75
 io_layer_type, 72
 loss_, 75
 model_, 75
 operator(), 73
 optimizer_, 75
 optimizer_type, 72
 predict, 74
 train_on_batch, 74
 trainable, 74
 ceras::Complex, 57
 ceras::complex< Real_Ex, Imag_Ex >, 76
 imag_, 76
 real_, 76
 ceras::Constant, 58
 ceras::constant< Tsr >, 76
 backward, 77
 constant, 77
 data_, 78
 forward, 78
 shape, 78
 tensor_type, 77
 ceras::dataset, 54
 ceras::dataset::fashion_mnist, 54
 load_data, 54
 ceras::dataset::mnist, 55
 load_data, 55
 ceras::Expression, 58
 ceras::gradient_descent< Loss, T >, 78
 forward, 79
 gradient_descent, 79
 learning_rate_, 79
 loss_, 80
 momentum_, 80
 tensor_type, 79
 ceras::identity_output_shape_calculator, 80
 operator(), 80, 81
 ceras::is_binary_operator< binary_operator< Lhs_Operator, Rh_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >, 82
 ceras::is_binary_operator< T >, 81
 ceras::is_complex< complex< Real_Ex, Imag_Ex > >, 82
 ceras::is_complex< T >, 82
 ceras::is_constant< constant< Tsr > >, 83
 ceras::is_constant< T >, 83
 ceras::is_place_holder< place_holder< Tsr > >, 84
 ceras::is_place_holder< T >, 83
 ceras::is_tensor< T >, 84
 ceras::is_tensor< tensor< T, A > >, 84
 ceras::is_unary_operator< T >, 85
 ceras::is_unary_operator< unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator > >, 85
 ceras::is_value< T >, 85
 ceras::is_value< value< T > >, 86
 ceras::is_variable< T >, 86
 ceras::is_variable< variable< Tsr > >, 86
 ceras::model< Ex, Ph >, 87
 compile, 89
 expression_, 92
 input, 89
 input_layer_, 92
 input_layer_type, 88
 load_weights, 89
 model, 88
 operator(), 90
 output, 90
 output_layer_, 93
 output_layer_type, 88
 place_holder_, 93
 predict, 90, 91
 save_weights, 92
 summary, 92
 trainable, 92
 ceras::Operator, 58
 ceras::Place_Holder, 58
 ceras::place_holder< Tsr >, 93
 backward, 94
 bind, 95
 forward, 95
 operator=, 95
 place_holder, 94
 reset, 95
 shape, 95, 96
 tensor_type, 94
 ceras::place_holder_state< Tsr >, 96

- data_, 96
- shape_hint_, 96
- ceras::regularizer< Float >, 97
 - l1_, 97
 - l2_, 98
 - regularizer, 97
 - synchronized_, 98
 - value_type, 97
- ceras::rmsprop< Loss, T >, 98
 - decay_, 99
 - forward, 99
 - iterations_, 100
 - learning_rate_, 100
 - loss_, 100
 - rho_, 100
 - rmsprop, 99
 - tensor_type, 99
- ceras::sgd< Loss, T >, 105
 - decay_, 106
 - forward, 106
 - iterations_, 107
 - learning_rate_, 107
 - loss_, 107
 - momentum_, 107
 - nesterov_, 107
 - sgd, 106
 - tensor_type, 106
- ceras::Tensor, 59
- ceras::tensor< T, Allocator >, 108
 - allocator, 110
 - as_scalar, 112
 - as_type, 112
 - back, 112, 113
 - begin, 113
 - cbegin, 113
 - cend, 113
 - copy, 113
 - crbegin, 114
 - creep_to, 114
 - crend, 114
 - data, 114
 - deep_copy, 114, 115
 - empty, 115
 - end, 115
 - front, 115
 - map, 116
 - memory_offset_, 120
 - ndim, 116
 - operator*=: 116
 - operator+=, 116
 - operator-, 117
 - operator-=, 117
 - operator/=: 117
 - operator=, 117, 118
 - operator[], 118
 - rbegin, 118
 - rend, 118, 119
 - reset, 119
 - reshape, 119
 - resize, 119
 - self_type, 110
 - shape, 119
 - shape_, 120
 - shared_vector, 110
 - shrink_to, 120
 - size, 120
 - slice, 120
 - tensor, 111, 112
 - value_type, 110
 - vector_, 121
 - vector_type, 110
- ceras::tensor_deduction< L, R >, 121
 - op_type, 121
 - tensor_type, 121
- ceras::Unary_Operator, 59
- ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 122
 - backward, 123
 - backward_action_, 124
 - forward, 123
 - forward_action_, 124
 - input_data_, 124
 - op_, 124
 - output_data_, 124
 - output_shape_calculator_, 125
 - shape, 123
 - tensor_type, 123
 - unary_operator, 123
- ceras::Value, 59
- ceras::value< T >, 125
 - backward, 127
 - data_, 128
 - forward, 127
 - operator=, 127
 - shape, 127
 - tensor_type, 126
 - value, 126
 - value_type, 126
- ceras::Variable, 59
- ceras::variable< Tsor >, 128
 - backward, 130
 - contexts, 130
 - data, 130
 - forward, 131
 - gradient, 131
 - operator=, 131
 - regularizer_, 132
 - reset, 131
 - shape, 131
 - state_, 132
 - tensor_type, 129
 - trainable, 132
 - trainable_, 132
 - value_type, 129
 - variable, 129, 130

- ceras::variable_state< T_{src} >, 133
 - contexts_, 133
 - data_, 133
 - gradient_, 133
- clip
 - operation.hpp, 178
- compile
 - ceras::model< Ex, Ph >, 89
- compiled_model
 - ceras::compiled_model< Model, Optimizer, Loss >, 72
- compiled_optimizer_
 - ceras::compiled_model< Model, Optimizer, Loss >, 75
- computation_graph
 - ceras, 23
- concat
 - operation.hpp, 178
- Concatenate
 - ceras, 23
- concatenate
 - operation.hpp, 179
- conj
 - ceras, 23
- constant
 - ceras::constant< T_{src} >, 77
- contexts
 - ceras::variable< T_{src} >, 130
- contexts_
 - ceras::variable_state< T_{src} >, 133
- Conv2D
 - ceras, 24
- conv2d
 - operation.hpp, 179
- copy
 - ceras::tensor< T, Allocator >, 113
- cos
 - operation.hpp, 179
- cosh
 - operation.hpp, 179
- crbegin
 - ceras::tensor< T, Allocator >, 114
- creep_to
 - ceras::tensor< T, Allocator >, 114
- crelu
 - ceras, 25
- crend
 - ceras::tensor< T, Allocator >, 114
- cropping_2d
 - operation.hpp, 180
- cross_entropy
 - ceras, 25
- cross_entropy_loss
 - ceras, 26
- data
 - ceras::tensor< T, Allocator >, 114
 - ceras::variable< T_{src} >, 130
- data_
 - ceras::constant< T_{src} >, 78
 - ceras::place_holder_state< T_{src} >, 96
 - ceras::value< T >, 128
 - ceras::variable_state< T_{src} >, 133
- decay_
 - ceras::adagrad< Loss, T >, 64
 - ceras::rmsprop< Loss, T >, 99
 - ceras::sgd< Loss, T >, 106
- deep_copy
 - ceras::tensor< T, Allocator >, 114, 115
- default_allocator
 - ceras, 19
- Dense
 - ceras, 26
- deserialize
 - ceras::ceras_private::session< T_{src} >, 102
- divide
 - ceras, 27
- drop_out
 - operation.hpp, 180
- Dropout
 - ceras, 27
- dropout
 - operation.hpp, 180
- elementwise_multiply
 - ceras, 27
- elementwise_product
 - ceras, 27
- ELU
 - ceras, 28
- elu
 - ceras, 28
- empty
 - ceras::tensor< T, Allocator >, 115
- end
 - ceras::tensor< T, Allocator >, 115
- equal
 - operation.hpp, 180
- erf
 - operation.hpp, 181
- erfc
 - operation.hpp, 181
- evaluate
 - ceras::compiled_model< Model, Optimizer, Loss >, 72
- exp
 - operation.hpp, 181
- exp2
 - operation.hpp, 182
- expand_dims
 - operation.hpp, 182
- expm1
 - operation.hpp, 182
- exponential
 - ceras, 28
- expression_
 - ceras::model< Ex, Ph >, 92

- fabs
 - operation.hpp, 182
- fit
 - ceras::compiled_model< Model, Optimizer, Loss >, 73
- Flatten
 - ceras, 29
- flatten
 - operation.hpp, 183
- floor
 - operation.hpp, 183
- forward
 - ceras::adadelta< Loss, T >, 62
 - ceras::adagrad< Loss, T >, 64
 - ceras::adam< Loss, T >, 66
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 69
 - ceras::constant< T >, 78
 - ceras::gradient_descent< Loss, T >, 79
 - ceras::place_holder< T >, 95
 - ceras::rmsprop< Loss, T >, 99
 - ceras::sgd< Loss, T >, 106
 - ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 123
 - ceras::value< T >, 127
 - ceras::variable< T >, 131
- forward_action_
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 70
 - ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 124
- front
 - ceras::tensor< T, Allocator >, 115
- gaussian
 - ceras, 29
- gelu
 - ceras, 29
- general_conv2d
 - operation.hpp, 183
- get_default_session
 - ceras, 29
- gradient
 - ceras::variable< T >, 131
- gradient_
 - ceras::variable_state< T >, 133
- gradient_descent
 - ceras::gradient_descent< Loss, T >, 79
- ground_truth_place_holder_
 - ceras::compiled_model< Model, Optimizer, Loss >, 75
- hadamard_product
 - ceras, 30
- hard_sigmoid
 - ceras, 30
- heaviside_step
 - ceras, 30
- Hinge
 - ceras, 50
- hinge_loss
 - ceras, 30
- hypot
 - operation.hpp, 183
- identity
 - operation.hpp, 184
- imag
 - ceras, 31
- imag_
 - ceras::complex< Real_Ex, Imag_Ex >, 76
- img2col
 - operation.hpp, 184
- includes.hpp
 - STB_IMAGE_IMPLEMENTATION, 153
 - STB_IMAGE_RESIZE_IMPLEMENTATION, 153
 - STB_IMAGE_WRITE_IMPLEMENTATION, 153
- Input
 - ceras, 31
- input
 - ceras::model< Ex, Ph >, 89
- input_data_
 - ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 124
- input_layer_
 - ceras::model< Ex, Ph >, 92
- input_layer_type
 - ceras::model< Ex, Ph >, 88
- input_place_holder_
 - ceras::compiled_model< Model, Optimizer, Loss >, 75
- inverse
 - ceras, 31
- io_layer_type
 - ceras::compiled_model< Model, Optimizer, Loss >, 72
- is_binary_operator_v
 - ceras, 50
- is_complex_v
 - ceras, 50
- is_constant_v
 - ceras, 50
- is_place_holder_v
 - ceras, 51
- is_tensor_v
 - ceras, 51
- is_unary_operator_v
 - ceras, 51
- is_value_v
 - ceras, 51
- is_variable_v
 - ceras, 51
- iterations_

- ceras::adadelta< Loss, T >, 62
 - ceras::adagrad< Loss, T >, 64
 - ceras::adam< Loss, T >, 67
 - ceras::rmsprop< Loss, T >, 100
 - ceras::sgd< Loss, T >, 107
- l1_
 - ceras::regularizer< Float >, 97
- l2_
 - ceras::regularizer< Float >, 98
- leaky_relu
 - ceras, 31
- LeakyReLU
 - ceras, 32
- learning_rate_
 - ceras::adadelta< Loss, T >, 62
 - ceras::adagrad< Loss, T >, 64
 - ceras::adam< Loss, T >, 67
 - ceras::gradient_descent< Loss, T >, 79
 - ceras::rmsprop< Loss, T >, 100
 - ceras::sgd< Loss, T >, 107
- lhs_input_data_
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 70
- lhs_op_
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 70
- lisht
 - ceras, 32
- llrint
 - operation.hpp, 184
- llround
 - operation.hpp, 184
- load_data
 - ceras::dataset::fashion_mnist, 54
 - ceras::dataset::mnist, 55
- load_weights
 - ceras::model< Ex, Ph >, 89
- log
 - operation.hpp, 185
- log10
 - operation.hpp, 185
- log1p
 - operation.hpp, 185
- log2
 - operation.hpp, 185
- loss_
 - ceras::adadelta< Loss, T >, 62
 - ceras::adagrad< Loss, T >, 65
 - ceras::adam< Loss, T >, 67
 - ceras::compiled_model< Model, Optimizer, Loss >, 75
 - ceras::gradient_descent< Loss, T >, 80
 - ceras::rmsprop< Loss, T >, 100
 - ceras::sgd< Loss, T >, 107
- llrint
 - operation.hpp, 186
- llround
 - operation.hpp, 186
- lstm
 - ceras, 32
- MAE
 - ceras, 51
- mae
 - ceras, 32
- make_binary_operator
 - ceras, 32
- make_compiled_model
 - ceras, 33
- make_trainable
 - ceras, 33
- make_unary_operator
 - ceras, 33
- map
 - ceras::tensor< T, Allocator >, 116
- max_pooling_2d
 - operation.hpp, 186
- maximum
 - operation.hpp, 186
- MaxPooling2D
 - ceras, 33
- mean
 - ceras, 33
- mean_absolute_error
 - ceras, 34
- mean_reduce
 - ceras, 34
- mean_squared_error
 - ceras, 34
- mean_squared_logarithmic_error
 - ceras, 34
- MeanAbsoluteError
 - ceras, 52
- MeanSquaredError
 - ceras, 52
- memory_offset_
 - ceras::tensor< T, Allocator >, 120
- minimum
 - operation.hpp, 186
- minus
 - ceras, 34
- mish
 - ceras, 35
- model
 - ceras::model< Ex, Ph >, 88
- model_
 - ceras::compiled_model< Model, Optimizer, Loss >, 75
- momentum_
 - ceras::gradient_descent< Loss, T >, 80
 - ceras::sgd< Loss, T >, 107
- MSE
 - ceras, 52
- mse
 - ceras, 35

- Multiply
 - ceras, [35](#)
- ndim
 - ceras::tensor< T, Allocator >, [116](#)
- nearbyint
 - operation.hpp, [187](#)
- negative
 - ceras, [35](#)
- negative_relu
 - ceras, [35](#)
- nesterov_
 - ceras::sgd< Loss, T >, [107](#)
- norm
 - ceras, [36](#)
- normalization_batch
 - operation.hpp, [187](#)
- ones_like
 - operation.hpp, [187](#)
- op_
 - ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, [124](#)
- op_type
 - ceras::tensor_deduction< L, R >, [121](#)
- operation.hpp
 - abs, [175](#)
 - acos, [175](#)
 - acosh, [175](#)
 - argmax, [175](#)
 - argmin, [176](#)
 - asin, [176](#)
 - asinh, [176](#)
 - assign, [176](#)
 - atan, [177](#)
 - atan2, [177](#)
 - atanh, [177](#)
 - average_pooling_2d, [177](#)
 - batch_normalization, [177](#)
 - cbrt, [178](#)
 - ceil, [178](#)
 - clip, [178](#)
 - concat, [178](#)
 - concatenate, [179](#)
 - conv2d, [179](#)
 - cos, [179](#)
 - cosh, [179](#)
 - cropping_2d, [180](#)
 - drop_out, [180](#)
 - dropout, [180](#)
 - equal, [180](#)
 - erf, [181](#)
 - erfc, [181](#)
 - exp, [181](#)
 - exp2, [182](#)
 - expand_dims, [182](#)
 - expm1, [182](#)
 - fabs, [182](#)
 - flatten, [183](#)
 - floor, [183](#)
 - general_conv2d, [183](#)
 - hypot, [183](#)
 - identity, [184](#)
 - img2col, [184](#)
 - llrint, [184](#)
 - llround, [184](#)
 - log, [185](#)
 - log10, [185](#)
 - log1p, [185](#)
 - log2, [185](#)
 - lrrint, [186](#)
 - lround, [186](#)
 - max_pooling_2d, [186](#)
 - maximum, [186](#)
 - minimum, [186](#)
 - nearbyint, [187](#)
 - normalization_batch, [187](#)
 - ones_like, [187](#)
 - poisson, [187](#)
 - random_normal_like, [188](#)
 - reduce_max, [188](#)
 - reduce_min, [189](#)
 - reduce_sum, [189](#)
 - repeat, [189](#)
 - reshape, [190](#)
 - rint, [190](#)
 - round, [190](#)
 - sign, [190](#)
 - sin, [191](#)
 - sinh, [191](#)
 - sliding_2d, [191](#)
 - sqr, [194](#)
 - sqrt, [192](#)
 - tan, [192](#)
 - tanh, [192](#)
 - transpose, [192](#)
 - trunc, [193](#)
 - up_sampling_2d, [193](#)
 - upsampling_2d, [193](#)
 - y, [194](#)
 - zero_padding_2d, [193](#)
 - zeros_like, [194](#)
- operator!=
 - ceras, [36](#)
- operator<
 - ceras, [40](#)
- operator<=
 - ceras, [40](#)
- operator>
 - ceras, [40](#)
- operator>=
 - ceras, [41](#)
- operator*
 - ceras, [36](#), [37](#)
- operator*=
 - ceras::tensor< T, Allocator >, [116](#)

operator()
 ceras::compiled_model< Model, Optimizer, Loss
 >, 73
 ceras::identity_output_shape_calculator, 80, 81
 ceras::model< Ex, Ph >, 90
 operator+
 ceras, 37, 38
 operator+=
 ceras::tensor< T, Allocator >, 116
 operator-
 ceras, 38, 39
 ceras::tensor< T, Allocator >, 117
 operator-=
 ceras::tensor< T, Allocator >, 117
 operator/
 ceras, 39
 operator/=
 ceras::tensor< T, Allocator >, 117
 operator=
 ceras::ceras_private::session< Tsor >, 103
 ceras::place_holder< Tsor >, 95
 ceras::tensor< T, Allocator >, 117, 118
 ceras::value< T >, 127
 ceras::variable< Tsor >, 131
 operator==
 ceras, 40
 operator[]
 ceras::tensor< T, Allocator >, 118
 optimizer_
 ceras::compiled_model< Model, Optimizer, Loss
 >, 75
 optimizer_type
 ceras::compiled_model< Model, Optimizer, Loss
 >, 72
 output
 ceras::model< Ex, Ph >, 90
 output_data_
 ceras::binary_operator< Lhs_Operator, Rhs_Operator,
 Forward_Action, Backward_Action, Out-
 put_Shape_Calculator >, 70
 ceras::unary_operator< Operator, Forward_Action,
 Backward_Action, Output_Shape_Calculator
 >, 124
 output_layer_
 ceras::model< Ex, Ph >, 93
 output_layer_type
 ceras::model< Ex, Ph >, 88
 output_shape_calculator_
 ceras::binary_operator< Lhs_Operator, Rhs_Operator,
 Forward_Action, Backward_Action, Out-
 put_Shape_Calculator >, 70
 ceras::unary_operator< Operator, Forward_Action,
 Backward_Action, Output_Shape_Calculator
 >, 125
 place_holder
 ceras::place_holder< Tsor >, 94
 place_holder_
 ceras::model< Ex, Ph >, 93
 place_holder_type
 ceras::ceras_private::session< Tsor >, 101
 place_holders_
 ceras::ceras_private::session< Tsor >, 105
 plus
 ceras, 41
 poisson
 operation.hpp, 187
 polar
 ceras, 41
 predict
 ceras::compiled_model< Model, Optimizer, Loss
 >, 74
 ceras::model< Ex, Ph >, 90, 91
 prelu
 ceras, 41
 random_generator
 ceras, 53
 random_normal_like
 operation.hpp, 188
 random_seed
 ceras, 53
 rbegin
 ceras::tensor< T, Allocator >, 118
 real
 ceras, 42
 real_
 ceras::complex< Real_Ex, Imag_Ex >, 76
 rebind
 ceras::ceras_private::session< Tsor >, 103
 recurrent.hpp
 units_, 252
 reduce_max
 operation.hpp, 188
 reduce_mean
 ceras, 42
 reduce_min
 operation.hpp, 189
 reduce_sum
 ceras, 42
 operation.hpp, 189
 regularizer
 ceras::regularizer< Float >, 97
 regularizer_
 ceras::variable< Tsor >, 132
 ReLU
 ceras, 43
 relu
 ceras, 42
 relu6
 ceras, 43
 remember
 ceras::ceras_private::session< Tsor >, 103
 rend
 ceras::tensor< T, Allocator >, 118, 119
 repeat
 operation.hpp, 189
 replace_placeholder_with_expression

- ceras, [43](#)
- reset
 - ceras::place_holder< T, Allocator >, [95](#)
 - ceras::tensor< T, Allocator >, [119](#)
 - ceras::variable< T, Allocator >, [131](#)
- Reshape
 - ceras, [43](#)
- reshape
 - ceras::tensor< T, Allocator >, [119](#)
 - operation.hpp, [190](#)
- resize
 - ceras::tensor< T, Allocator >, [119](#)
- restore
 - ceras::ceras_private::session< T, Allocator >, [103](#)
- restore_original
 - ceras::ceras_private::session< T, Allocator >, [103](#)
- rho_
 - ceras::adadelta< Loss, T >, [63](#)
 - ceras::rmsprop< Loss, T >, [100](#)
- rhs_input_data_
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, [71](#)
- rhs_op_
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, [71](#)
- rint
 - operation.hpp, [190](#)
- rms_prop
 - ceras, [19](#)
- RMSprop
 - ceras, [53](#)
- rmsprop
 - ceras::rmsprop< Loss, T >, [99](#)
- round
 - operation.hpp, [190](#)
- run
 - ceras, [44](#)
 - ceras::ceras_private::session< T, Allocator >, [104](#)
- save
 - ceras::ceras_private::session< T, Allocator >, [104](#)
- save_original
 - ceras::ceras_private::session< T, Allocator >, [104](#)
- save_weights
 - ceras::model< Ex, Ph >, [92](#)
- self_type
 - ceras::tensor< T, Allocator >, [110](#)
- selu
 - ceras, [44](#)
- serialize
 - ceras::ceras_private::session< T, Allocator >, [104](#)
- session
 - ceras::ceras_private::session< T, Allocator >, [102](#)
- SGD
 - ceras, [53](#)
- sgd
 - ceras::sgd< Loss, T >, [106](#)
- shape
 - ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, [69](#)
 - ceras::constant< T, Allocator >, [78](#)
 - ceras::place_holder< T, Allocator >, [95](#), [96](#)
 - ceras::tensor< T, Allocator >, [119](#)
 - ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, [123](#)
 - ceras::value< T >, [127](#)
 - ceras::variable< T, Allocator >, [131](#)
- shape_
 - ceras::tensor< T, Allocator >, [120](#)
- shape_hint_
 - ceras::place_holder_state< T, Allocator >, [96](#)
- shared_vector
 - ceras::tensor< T, Allocator >, [110](#)
- shrink_to
 - ceras::tensor< T, Allocator >, [120](#)
- sigmoid
 - ceras, [44](#)
- sign
 - operation.hpp, [190](#)
- sign_
 - ceras, [45](#)
- sin
 - operation.hpp, [191](#)
- sinh
 - operation.hpp, [191](#)
- size
 - ceras::tensor< T, Allocator >, [120](#)
- slice
 - ceras::tensor< T, Allocator >, [120](#)
- sliding_2d
 - operation.hpp, [191](#)
- soft_sign
 - ceras, [45](#)
- Softmax
 - ceras, [45](#)
- softmax
 - ceras, [45](#)
- softplus
 - ceras, [45](#)
- softsign
 - ceras, [46](#)
- sqr
 - operation.hpp, [194](#)
- sqrt
 - operation.hpp, [192](#)
- square
 - ceras, [46](#)
- squared_loss
 - ceras, [47](#)
- state_
 - ceras::variable< T, Allocator >, [132](#)
- STB_IMAGE_IMPLEMENTATION
 - includes.hpp, [153](#)

STB_IMAGE_RESIZE_IMPLEMENTATION
 includes.hpp, 153
 STB_IMAGE_WRITE_IMPLEMENTATION
 includes.hpp, 153
 Subtract
 ceras, 47
 sum_reduce
 ceras, 47
 summary
 ceras::model< Ex, Ph >, 92
 swish
 ceras, 47
 synchronized_
 ceras::regularizer< Float >, 98

 tan
 operation.hpp, 192
 tanh
 operation.hpp, 192
 tank_shrink
 ceras, 48
 tap
 ceras::ceras_private::session< Tzor >, 104
 tensor
 ceras::tensor< T, Allocator >, 111, 112
 tensor_type
 ceras::adadelta< Loss, T >, 61
 ceras::adagrad< Loss, T >, 63
 ceras::adam< Loss, T >, 66
 ceras::binary_operator< Lhs_Operator, Rhs_Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 68
 ceras::constant< Tzor >, 77
 ceras::gradient_descent< Loss, T >, 79
 ceras::place_holder< Tzor >, 94
 ceras::rmsprop< Loss, T >, 99
 ceras::sgd< Loss, T >, 106
 ceras::tensor_deduction< L, R >, 121
 ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 123
 ceras::value< T >, 126
 ceras::variable< Tzor >, 129
 train_on_batch
 ceras::compiled_model< Model, Optimizer, Loss >, 74
 trainable
 ceras::compiled_model< Model, Optimizer, Loss >, 74
 ceras::model< Ex, Ph >, 92
 ceras::variable< Tzor >, 132
 trainable_
 ceras::variable< Tzor >, 132
 transpose
 operation.hpp, 192
 trunc
 operation.hpp, 193

 unary_operator
 ceras::unary_operator< Operator, Forward_Action, Backward_Action, Output_Shape_Calculator >, 123
 unit_step
 ceras, 48
 units_
 recurrent.hpp, 252
 up_sampling_2d
 operation.hpp, 193
 UpSampling2D
 ceras, 48
 upsampling_2d
 operation.hpp, 193

 value
 ceras::value< T >, 126
 value_type
 ceras::regularizer< Float >, 97
 ceras::tensor< T, Allocator >, 110
 ceras::value< T >, 126
 ceras::variable< Tzor >, 129
 variable
 ceras::variable< Tzor >, 129, 130
 variable_state_type
 ceras::ceras_private::session< Tzor >, 101
 variable_type
 ceras::ceras_private::session< Tzor >, 101
 variables_
 ceras::ceras_private::session< Tzor >, 105

 ceras::tensor< T, Allocator >, 121
 vector_type
 ceras::tensor< T, Allocator >, 110

 y
 operation.hpp, 194

 zero_padding_2d
 operation.hpp, 193
 zeros_like
 operation.hpp, 194