

Lipnet

1.0.0

Generated by Doxygen 1.8.20



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>5</b>
2.1 Class List	5
<b>3 Class Documentation</b>	<b>11</b>
3.1 <code>lipnet::activation_t&lt; T, TYPE &gt;</code> Struct Template Reference	11
3.1.1 Detailed Description	11
3.1.2 Member Function Documentation	12
3.1.2.1 <code>derivative()</code>	12
3.1.2.2 <code>forward()</code>	12
3.2 <code>lipnet::adam_barrier_t_impl&lt; T, P, VAR, GRAD, feasibility_enabled &gt;</code> Struct Template Reference	13
3.2.1 Detailed Description	13
3.2.2 Constructor & Destructor Documentation	14
3.2.2.1 <code>adam_barrier_t_impl()</code>	14
3.2.3 Member Function Documentation	14
3.2.3.1 <code>run()</code>	14
3.3 <code>lipnet::adam_momentum_t_impl&lt; T, P, VAR, GRAD &gt;</code> Struct Template Reference	15
3.3.1 Detailed Description	16
3.3.2 Constructor & Destructor Documentation	16
3.3.2.1 <code>adam_momentum_t_impl()</code>	16
3.3.3 Member Function Documentation	16
3.3.3.1 <code>run()</code>	16
3.4 <code>lipnet::adam_projected_t_impl&lt; T, P, VAR, GRAD &gt;</code> Struct Template Reference	17
3.4.1 Detailed Description	17
3.4.2 Constructor & Destructor Documentation	18
3.4.2.1 <code>adam_projected_t_impl()</code>	18
3.4.3 Member Function Documentation	18
3.4.3.1 <code>project()</code>	18
3.4.3.2 <code>run()</code>	19
3.5 <code>lipnet::admm_optimizer_t_impl&lt; T, P, X, Z, DUAL &gt;</code> Struct Template Reference	19
3.5.1 Detailed Description	20
3.5.2 Constructor & Destructor Documentation	20
3.5.2.1 <code>admm_optimizer_t_impl()</code>	20
3.5.3 Member Function Documentation	21
3.5.3.1 <code>evaluate()</code>	21
3.5.3.2 <code>optimize1()</code>	21
3.5.3.3 <code>optimize2()</code>	22
3.5.3.4 <code>residual()</code>	22
3.5.3.5 <code>run()</code>	22
3.6 <code>lipnet::backpropagation_batch_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</code> Struct Template Reference	23
3.6.1 Detailed Description	24

3.6.2 Member Function Documentation	24
3.6.2.1 backward()	24
3.6.2.2 compute()	26
3.6.2.3 forward()	26
3.6.2.4 run()	27
3.7 lipnet::barrierfunction_t< T, N > Struct Template Reference	27
3.8 lipnet::barrierfunction_wot_t< T, N > Struct Template Reference	28
3.9 lipnet::calculate_lipschitz_t< T, N > Struct Template Reference	28
3.9.1 Detailed Description	29
3.10 lipnet::cholesky_diagentry< T, N, NARGS > Struct Template Reference	29
3.11 lipnet::cholesky_diagentry_impl< T, N, NS > Struct Template Reference	30
3.12 lipnet::cholesky_diagentry_impl< T, N > Struct Template Reference	30
3.13 lipnet::cholesky_subentry< T, NI, NO, RE, NARGS > Struct Template Reference	30
3.14 lipnet::cholesky_subentry_impl< T, NI, NO, NS > Struct Template Reference	30
3.15 lipnet::cholesky_subentry_impl< T, NI, NO > Struct Template Reference	31
3.16 lipnet::cholesky_topology< T, N > Struct Template Reference	31
3.17 lipnet::cross_entropy_t< T > Struct Template Reference	31
3.17.1 Detailed Description	31
3.17.2 Member Function Documentation	32
3.17.2.1 evaluate()	32
3.18 lipnet::data_container_t< T > Struct Template Reference	32
3.18.1 Detailed Description	33
3.19 lipnet::network_t< T, ATYPE, N >::data_serialization_t< saveing > Struct Template Reference	33
3.19.1 Detailed Description	34
3.20 lipnet::data_container_t< T >::data_t< saveing > Struct Template Reference	34
3.21 lipnet::decompos_diagentry< T, N, NARGS > Struct Template Reference	35
3.22 lipnet::decompos_diagentry_impl< T, N, NS > Struct Template Reference	35
3.23 lipnet::decompos_diagentry_impl< T, N > Struct Template Reference	35
3.24 lipnet::decompos_subentry< T, NI, NO, RE, NARGS > Struct Template Reference	35
3.25 lipnet::decompos_subentry_impl< T, NI, NO, NS > Struct Template Reference	36
3.26 lipnet::decompos_subentry_impl< T, NI, NO > Struct Template Reference	36
3.27 std::detail::detector< Default, AlwaysVoid, Op, Args > Struct Template Reference	36
3.28 std::detail::detector< Default, std::void_t< Op< Args... > >, Op, Args... > Struct Template Reference	36
3.29 lipnet::equation_system_t< V1, V2 > Struct Template Reference	37
3.29.1 Detailed Description	37
3.30 lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > > Struct Template Reference	37
3.30.1 Detailed Description	37
3.30.2 Member Function Documentation	38
3.30.2.1 solve()	38
3.31 lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticVector< T, N3, blaze::columnVector > > Struct Template Reference	38
3.31.1 Detailed Description	39

3.31.2 Member Function Documentation	39
3.31.2.1 solve()	39
3.32 lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD > Struct Template Reference	40
3.32.1 Detailed Description	40
3.32.2 Constructor & Destructor Documentation	41
3.32.2.1 fast_gradient_descent_t_impl()	41
3.32.3 Member Function Documentation	41
3.32.3.1 run()	41
3.33 lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t Struct Reference	42
3.33.1 Detailed Description	42
3.34 lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t Struct Reference	43
3.34.1 Detailed Description	43
3.35 lipnet::feasibility_t< IMPL, T, DIR > Struct Template Reference	44
3.35.1 Detailed Description	44
3.35.2 Member Function Documentation	44
3.35.2.1 operator>()	44
3.35.2.2 operator<<()	45
3.36 lipnet::feasibilitycheck_t< T, N > Struct Template Reference	45
3.37 lipnet::feasibilitycheck_wot_t< T, N > Struct Template Reference	46
3.38 lipnet::function_t< V > Struct Template Reference	46
3.38.1 Detailed Description	46
3.39 lipnet::function_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > Struct Template Reference	47
3.39.1 Detailed Description	47
3.40 lipnet::function_t< blaze::StaticVector< T, N, blaze::columnVector > > Struct Template Reference	47
3.40.1 Detailed Description	48
3.41 lipnet::function_t< layer_t< T, I, O > > Struct Template Reference	48
3.42 lipnet::function_t< liptrainweights_t< T, N... > > Struct Template Reference	48
3.43 lipnet::function_t< std::tuple< ARGS... > > Struct Template Reference	49
3.44 lipnet::generate_batch_data< T, B, N, NS > Struct Template Reference	49
3.44.1 Detailed Description	49
3.45 lipnet::generate_batch_data< T, B, N > Struct Template Reference	49
3.45.1 Detailed Description	50
3.46 lipnet::generate_batch_data_remove_first< T, B, N, NS > Struct Template Reference	50
3.46.1 Detailed Description	50
3.47 lipnet::generate_data< T, N, NS > Struct Template Reference	50
3.47.1 Detailed Description	51
3.48 lipnet::generate_data< T, N > Struct Template Reference	51
3.48.1 Detailed Description	51
3.49 lipnet::generate_data_remove_first< T, N, NS > Struct Template Reference	51
3.49.1 Detailed Description	52
3.50 lipnet::generator_t< V > Struct Template Reference	52

3.50.1 Detailed Description	52
3.51 <code>lipnet::generator_t&lt; blaze::StaticMatrix&lt; T, N1, N2, blaze::rowMajor &gt; &gt;</code> Struct Template Reference	52
3.51.1 Detailed Description	53
3.52 <code>lipnet::generator_t&lt; blaze::StaticVector&lt; T, N, blaze::columnVector &gt; &gt;</code> Struct Template Reference	53
3.52.1 Detailed Description	53
3.53 <code>lipnet::generator_t&lt; layer_t&lt; T, I, O &gt; &gt;</code> Struct Template Reference	54
3.54 <code>lipnet::generator_t&lt; liptrainweights_t&lt; T, N... &gt; &gt;</code> Struct Template Reference	54
3.55 <code>lipnet::generator_t&lt; parameter_decompo_t&lt; T, N... &gt; &gt;</code> Struct Template Reference	54
3.56 <code>lipnet::generator_t&lt; std::tuple&lt; ARGS... &gt; &gt;</code> Struct Template Reference	55
3.56.1 Detailed Description	55
3.57 <code>lipnet::gradient_descent_projected_t_impl&lt; T, P, VAR, GRAD &gt;</code> Struct Template Reference	55
3.57.1 Detailed Description	56
3.57.2 Constructor & Destructor Documentation	56
3.57.2.1 <code>gradient_descent_projected_t_impl()</code>	56
3.57.3 Member Function Documentation	57
3.57.3.1 <code>project()</code>	57
3.57.3.2 <code>run()</code>	57
3.58 <code>lipnet::helper_function_t&lt; V &gt;</code> Struct Template Reference	58
3.58.1 Member Data Documentation	58
3.58.1.1 <code>v1</code>	58
3.58.1.2 <code>v2</code>	58
3.59 <code>lipnet::helper_inner_t&lt; T, V1, V2 &gt;</code> Struct Template Reference	58
3.59.1 Member Data Documentation	58
3.59.1.1 <code>value</code>	59
3.60 <code>lipnet::helper_norm_t&lt; T, V &gt;</code> Struct Template Reference	59
3.60.1 Member Data Documentation	59
3.60.1.1 <code>value</code>	59
3.61 <code>image_t</code> Struct Reference	59
3.62 <code>lipnet::inverse_diagentry&lt; T, N, NARGS &gt;</code> Struct Template Reference	60
3.63 <code>lipnet::inverse_diagentry_impl&lt; T, N, NS &gt;</code> Struct Template Reference	60
3.64 <code>lipnet::inverse_diagentry_impl&lt; T, N &gt;</code> Struct Template Reference	60
3.65 <code>lipnet::inverse_subentry&lt; T, NI, NO, RE, NARGS &gt;</code> Struct Template Reference	61
3.66 <code>lipnet::inverse_subentry_impl&lt; T, NI, NO, NS &gt;</code> Struct Template Reference	61
3.67 <code>lipnet::inverse_subentry_impl&lt; T, NI, NO &gt;</code> Struct Template Reference	61
3.68 <code>lipnet::inverse_topology&lt; T, N &gt;</code> Struct Template Reference	61
3.69 <code>lipnet::join_tuples&lt; typename, typename &gt;</code> Struct Template Reference	62
3.69.1 Detailed Description	62
3.70 <code>lipnet::join_tuples&lt; std::tuple&lt; NEW... &gt;, std::tuple&lt; NEXT... &gt; &gt;</code> Struct Template Reference	62
3.70.1 Detailed Description	62
3.71 <code>lipnet::layer_t&lt; T, I, O &gt;</code> Struct Template Reference	63
3.71.1 Detailed Description	63
3.71.2 Constructor & Destructor Documentation	63

3.71.2.1 layer_t()	64
3.72 lipnet::linesearch_t< IMPL, T, DIRECTION > Struct Template Reference	64
3.72.1 Detailed Description	64
3.72.2 Member Function Documentation	65
3.72.2.1 operator>()	65
3.72.2.2 operator<<()	65
3.73 lipnet::lipcalc_parameter_t< T, N > Struct Template Reference	66
3.74 lipnet::liptrainweights_t< T, N > Struct Template Reference	66
3.75 lipnet::loader_t< T > Struct Template Reference	66
3.75.1 Detailed Description	66
3.75.2 Member Function Documentation	67
3.75.2.1 load()	67
3.76 lipnet::optimizer_t< T, P, IMPL, VARS >::main_statistics_t Struct Reference	67
3.76.1 Detailed Description	68
3.77 lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t Struct Reference	68
3.78 lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t Struct Reference	68
3.79 lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t Struct Reference	69
3.80 lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t Struct Reference	69
3.81 lipnet::backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t Struct Reference	69
3.82 lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t Struct Reference	70
3.83 lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t Struct Reference	70
3.84 lipnet::metainfo_t< IMPL > Struct Template Reference	71
3.84.1 Detailed Description	71
3.85 lipnet::mosek_projection_wot_t< T, N > Struct Template Reference	71
3.85.1 Detailed Description	72
3.85.2 Member Function Documentation	72
3.85.2.1 projection()	72
3.86 lipnet::network_data_t< T, IN, OUT > Struct Template Reference	73
3.86.1 Detailed Description	73
3.87 lipnet::network_libcalc_t< T, N > Struct Template Reference	73
3.87.1 Detailed Description	74
3.87.2 Member Function Documentation	74
3.87.2.1 solve()	74
3.88 lipnet::network_libtrain_enforcing_t< T, N > Struct Template Reference	74
3.88.1 Detailed Description	75
3.88.2 Member Function Documentation	75
3.88.2.1 train()	75
3.89 lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference	76
3.89.1 Detailed Description	77
3.89.2 Member Function Documentation	77
3.89.2.1 operator>()	77

3.89.3 Member Data Documentation	78
3.89.3.1 weights_bar	78
3.90 lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference	78
3.90.1 Detailed Description	79
3.90.2 Constructor & Destructor Documentation	79
3.90.2.1 network_problem_batch_l2_t()	79
3.90.3 Member Function Documentation	80
3.90.3.1 operator()()	80
3.91 lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference	80
3.91.1 Detailed Description	81
3.91.2 Member Function Documentation	81
3.91.2.1 operator()()	81
3.92 lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference	82
3.92.1 Detailed Description	83
3.92.2 Constructor & Destructor Documentation	83
3.92.2.1 network_problem_liptrain_enforcing_adam_t()	83
3.92.3 Member Function Documentation	83
3.92.3.1 loss()	84
3.92.3.2 optimize1()	84
3.92.3.3 optimize2()	85
3.92.3.4 residual()	85
3.93 lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference	86
3.93.1 Detailed Description	87
3.93.2 Constructor & Destructor Documentation	87
3.93.2.1 network_problem_log_barrier_t()	87
3.93.3 Member Function Documentation	87
3.93.3.1 operator()() [1/4]	88
3.93.3.2 operator()() [2/4]	88
3.93.3.3 operator()() [3/4]	88
3.93.3.4 operator()() [4/4]	89
3.93.3.5 run()	89
3.94 lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference	90
3.94.1 Detailed Description	91
3.94.2 Constructor & Destructor Documentation	91
3.94.2.1 network_problem_log_barrier_wot_t()	91
3.94.3 Member Function Documentation	92
3.94.3.1 operator()() [1/4]	92
3.94.3.2 operator()() [2/4]	92
3.94.3.3 operator()() [3/4]	93
3.94.3.4 operator()() [4/4]	93
3.94.3.5 run()	94



3.95	<a href="#">lipnet::network_problem_projection_t&lt; T, ATYPE, LOSS, BATCH, N &gt; Struct Template Reference</a>	94
3.95.1	<a href="#">Detailed Description</a>	95
3.95.2	<a href="#">Constructor &amp; Destructor Documentation</a>	96
3.95.2.1	<a href="#">network_problem_projection_t()</a>	96
3.95.3	<a href="#">Member Function Documentation</a>	96
3.95.3.1	<a href="#">operator&gt;()</a>	96
3.95.3.2	<a href="#">projection()</a>	97
3.96	<a href="#">lipnet::network_t&lt; T, ATYPE, N &gt; Struct Template Reference</a>	97
3.96.1	<a href="#">Detailed Description</a>	98
3.96.2	<a href="#">Member Function Documentation</a>	98
3.96.2.1	<a href="#">query()</a>	98
3.97	<a href="#">lipnet::network_topology&lt; T, NI, NO, NARGS &gt; Struct Template Reference</a>	99
3.97.1	<a href="#">Detailed Description</a>	99
3.98	<a href="#">lipnet::network_topology_impl&lt; T, NI, NO, NS &gt; Struct Template Reference</a>	99
3.98.1	<a href="#">Detailed Description</a>	99
3.99	<a href="#">lipnet::network_topology_impl&lt; T, NI, NO &gt; Struct Template Reference</a>	100
3.99.1	<a href="#">Detailed Description</a>	100
3.100	<a href="#">std::nonesuch Struct Reference</a>	100
3.101	<a href="#">lipnet::norm_t&lt; T, V &gt; Struct Template Reference</a>	100
3.101.1	<a href="#">Detailed Description</a>	100
3.102	<a href="#">lipnet::norm_t&lt; T, blaze::StaticMatrix&lt; T, N1, N2, blaze::rowMajor &gt; &gt; Struct Template Reference</a>	101
3.102.1	<a href="#">Detailed Description</a>	101
3.102.2	<a href="#">Member Function Documentation</a>	101
3.102.2.1	<a href="#">norm()</a>	102
3.103	<a href="#">lipnet::norm_t&lt; T, blaze::StaticVector&lt; T, N, blaze::columnVector &gt; &gt; Struct Template Reference</a>	102
3.103.1	<a href="#">Detailed Description</a>	102
3.103.2	<a href="#">Member Function Documentation</a>	103
3.103.2.1	<a href="#">norm()</a>	103
3.104	<a href="#">lipnet::norm_t&lt; T, layer_t&lt; T, I, O &gt; &gt; Struct Template Reference</a>	103
3.105	<a href="#">lipnet::norm_t&lt; T, lipcalc_parameter_t&lt; T, N... &gt; &gt; Struct Template Reference</a>	103
3.106	<a href="#">lipnet::norm_t&lt; T, liptrainweights_t&lt; T, N... &gt; &gt; Struct Template Reference</a>	104
3.107	<a href="#">lipnet::norm_t&lt; T, std::tuple&lt; ARGS... &gt; &gt; Struct Template Reference</a>	104
3.108	<a href="#">lipnet::optimizer_t&lt; T, P, IMPL, VARS &gt; Struct Template Reference</a>	104
3.108.1	<a href="#">Detailed Description</a>	105
3.108.2	<a href="#">Member Function Documentation</a>	105
3.108.2.1	<a href="#">operator&gt;()</a> [1/2]	106
3.108.2.2	<a href="#">operator&gt;()</a> [2/2]	106
3.108.2.3	<a href="#">run()</a>	107
3.109	<a href="#">lipnet::parameter_decompo_t&lt; T, N &gt; Struct Template Reference</a>	107
3.110	<a href="#">lipnet::adam_barrier_t_impl&lt; T, P, VAR, GRAD, feasibility_enabled &gt;::parameter_t Struct Reference</a>	108
3.110.1	<a href="#">Detailed Description</a>	108
3.111	<a href="#">lipnet::fast_gradient_descent_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t Struct Reference</a>	109

3.112	<a href="#">lipnet::adam_momentum_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t Struct Reference</a>	109
3.113	<a href="#">lipnet::admm_optimizer_t_impl&lt; T, P, X, Z, DUAL &gt;::parameter_t Struct Reference</a>	109
3.114	<a href="#">lipnet::adam_projected_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t Struct Reference</a>	110
3.115	<a href="#">lipnet::gradient_descent_projected_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t Struct Reference</a>	110
3.116	<a href="#">lipnet::parameter_tparam&lt; T, N, NARGS &gt; Struct Template Reference</a>	111
3.117	<a href="#">lipnet::parameter_tparam_impl&lt; T, N, NS &gt; Struct Template Reference</a>	111
3.118	<a href="#">lipnet::parameter_tparam_impl&lt; T, N, R &gt; Struct Template Reference</a>	111
3.119	<a href="#">lipnet::problem_t&lt; T, TYPE, IMPL, ARGS &gt; Struct Template Reference</a>	111
3.119.1	Detailed Description	111
3.120	<a href="#">lipnet::prod_t&lt; T, V1, V2 &gt; Struct Template Reference</a>	112
3.120.1	Detailed Description	112
3.121	<a href="#">lipnet::prod_t&lt; T, blaze::StaticMatrix&lt; T, N1, N2, blaze::rowMajor &gt;, blaze::StaticMatrix&lt; T, N3, N4, blaze::rowMajor &gt; &gt; Struct Template Reference</a>	112
3.121.1	Detailed Description	113
3.121.2	Member Function Documentation	113
3.121.2.1	inner()	113
3.121.2.2	outer()	114
3.122	<a href="#">lipnet::prod_t&lt; T, blaze::StaticVector&lt; T, N1, blaze::columnVector &gt;, blaze::StaticVector&lt; T, N2, blaze::columnVector &gt; &gt; Struct Template Reference</a>	114
3.122.1	Detailed Description	114
3.122.2	Member Function Documentation	115
3.122.2.1	inner()	115
3.122.2.2	outer()	115
3.123	<a href="#">lipnet::prod_t&lt; T, layer_t&lt; T, I1, O1 &gt;, layer_t&lt; T, I2, O2 &gt; &gt; Struct Template Reference</a>	116
3.124	<a href="#">lipnet::prod_t&lt; T, lipcalc_parameter_t&lt; T, N... &gt;, lipcalc_parameter_t&lt; T, N... &gt; &gt; Struct Template Reference</a>	116
3.125	<a href="#">lipnet::prod_t&lt; T, liptrainweights_t&lt; T, N... &gt;, liptrainweights_t&lt; T, N... &gt; &gt; Struct Template Reference</a>	116
3.126	<a href="#">lipnet::prod_t&lt; T, parameter_decompo_t&lt; T, N... &gt;, parameter_decompo_t&lt; T, N... &gt; &gt; Struct Template Reference</a>	117
3.127	<a href="#">lipnet::prod_t&lt; T, std::tuple&lt; ARGS1... &gt;, std::tuple&lt; ARGS2... &gt; &gt; Struct Template Reference</a>	117
3.128	<a href="#">lipnet::series_t&lt; T &gt; Struct Template Reference</a>	117
3.128.1	Detailed Description	117
3.129	<a href="#">lipnet::solve_function_helper&lt; P, VAR &gt; Struct Template Reference</a>	118
3.130	<a href="#">lipnet::squared_error_t&lt; T &gt; Struct Template Reference</a>	118
3.130.1	Detailed Description	118
3.130.2	Member Function Documentation	119
3.130.2.1	evaluate()	119
3.131	<a href="#">lipnet::statistics_helper Struct Reference</a>	119
3.131.1	Detailed Description	120
3.132	<a href="#">lipnet::gradient_descent_projected_t_impl&lt; T, P, VAR, GRAD &gt;::statistics_t Struct Reference</a>	120
3.132.1	Detailed Description	120
3.133	<a href="#">lipnet::admm_optimizer_t_impl&lt; T, P, X, Z, DUAL &gt;::statistics_t Struct Reference</a>	121

3.133.1 Detailed Description	121
3.134 lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::statistics_t Struct Reference	121
3.134.1 Detailed Description	122
3.135 lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >::statistics_t Struct Reference	122
3.135.1 Detailed Description	122
3.136 lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >::statistics_t Struct Reference	123
3.136.1 Detailed Description	123
3.137 lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::statistics_t Struct Reference	123
3.137.1 Detailed Description	124
3.138 lipnet::optimizer_t< T, P, IMPL, VARS >::stats_type_exists< TT, U > Struct Template Reference	124
3.139 lipnet::statistics_helper::stats_type_exists< TT, U > Struct Template Reference	124
3.140 lipnet::optimizer_t< T, P, IMPL, VARS >::stats_type_exists< TT, typename void_t< typename TT↵ ::statistics_t >::type > Struct Template Reference	125
3.141 lipnet::statistics_helper::stats_type_exists< TT, typename void_t< typename TT::statistics_t >↵ ::type > Struct Template Reference	125
3.142 lipnet::network_t< T, ATYPE, N >::topology_serialization_t Struct Reference	125
3.142.1 Detailed Description	125
3.143 lipnet::data_container_t< T >::tuple_t< saveing > Struct Template Reference	126
3.144 lipnet::cholesky_topology< T, N >::type Struct Reference	126
3.145 lipnet::inverse_topology< T, N >::type Struct Reference	126
3.146 lipnet::data_container_t< T >::view_t< saveing > Struct Template Reference	127
3.147 lipnet::optimizer_t< T, P, IMPL, VARS >::void_t< TT > Struct Template Reference	127
3.147.1 Detailed Description	127
3.148 lipnet::statistics_helper::void_t< TT > Struct Template Reference	128
3.149 std::void_type Struct Reference	128
3.149.1 Detailed Description	128
<b>Bibliography</b>	<b>129</b>
<b>Index</b>	<b>131</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

lipnet::activation_t< T, TYPE > . . . . .	11
lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled > . . . . .	13
lipnet::adam_momentum_t_impl< T, P, VAR, GRAD > . . . . .	15
lipnet::adam_projected_t_impl< T, P, VAR, GRAD > . . . . .	17
lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL > . . . . .	19
lipnet::backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N > . . . . .	23
lipnet::backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N... > . . . . .	23
lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N > . . . . .	76
lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N > . . . . .	78
lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N > . . . . .	80
lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N > . . . . .	86
lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N > . . . . .	90
lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N > . . . . .	94
lipnet::barrierfunction_t< T, N > . . . . .	27
lipnet::barrierfunction_t< T, N... > . . . . .	27
lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N > . . . . .	86
lipnet::barrierfunction_wot_t< T, N > . . . . .	28
lipnet::barrierfunction_wot_t< T, N... > . . . . .	28
lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N > . . . . .	90
lipnet::calculate_lipschitz_t< T, N > . . . . .	28
lipnet::cholesky_diagentry< T, N, NARGS > . . . . .	29
lipnet::cholesky_diagentry< T, N... > . . . . .	29
lipnet::cholesky_diagentry_impl< T, N, NS > . . . . .	30
lipnet::cholesky_diagentry_impl< T, N > . . . . .	30
lipnet::cholesky_subentry< T, NI, NO, RE, NARGS > . . . . .	30
lipnet::cholesky_subentry< T, N... > . . . . .	30
lipnet::cholesky_subentry_impl< T, NI, NO, NS > . . . . .	30
lipnet::cholesky_subentry_impl< T, NI, NO > . . . . .	31
lipnet::cholesky_topology< T, N > . . . . .	31
lipnet::cross_entropy_t< T > . . . . .	31
lipnet::data_container_t< T > . . . . .	32
lipnet::network_t< T, ATYPE, N >::data_serialization_t< saveing > . . . . .	33
lipnet::data_container_t< T >::data_t< saveing > . . . . .	34
lipnet::decompos_diagentry< T, N, NARGS > . . . . .	35

lipnet::decompos_diagentry< T, N... > . . . . .	35
lipnet::decompos_diagentry_impl< T, N, NS > . . . . .	35
lipnet::decompos_diagentry_impl< T, N > . . . . .	35
lipnet::decompos_subentry< T, NI, NO, RE, NARGS > . . . . .	35
lipnet::decompos_subentry< T, N... > . . . . .	35
lipnet::decompos_subentry_impl< T, NI, NO, NS > . . . . .	36
lipnet::decompos_subentry_impl< T, NI, NO > . . . . .	36
std::detail::detector< Default, AlwaysVoid, Op, Args > . . . . .	36
std::detail::detector< Default, std::void_t< Op< Args... > >, Op, Args... > . . . . .	36
lipnet::equation_system_t< V1, V2 > . . . . .	37
lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > > . . . . .	37
lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticVector< T, N3, blaze::columnVector > > . . . . .	38
lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD > . . . . .	40
feasibility_t	
lipnet::feasibility_t< IMPL, T, DIR > . . . . .	44
lipnet::feasibilitycheck_t< T, N > . . . . .	45
lipnet::feasibilitycheck_t< T, N... > . . . . .	45
lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t . . . . .	42
lipnet::feasibilitycheck_wot_t< T, N > . . . . .	46
lipnet::feasibilitycheck_wot_t< T, N... > . . . . .	46
lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t . . . . .	43
lipnet::function_t< V > . . . . .	46
lipnet::function_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > . . . . .	47
lipnet::function_t< blaze::StaticVector< T, N, blaze::columnVector > > . . . . .	47
lipnet::function_t< layer_t< T, I, O > > . . . . .	48
lipnet::function_t< liptrainweights_t< T, N... > > . . . . .	48
lipnet::function_t< std::tuple< ARGS... > > . . . . .	49
lipnet::generate_batch_data< T, B, N, NS > . . . . .	49
lipnet::generate_batch_data< T, B, N > . . . . .	49
lipnet::generate_batch_data_remove_first< T, B, N, NS > . . . . .	50
lipnet::generate_data< T, N, NS > . . . . .	50
lipnet::generate_data< T, N > . . . . .	51
lipnet::generate_data_remove_first< T, N, NS > . . . . .	51
lipnet::generator_t< V > . . . . .	52
lipnet::generator_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > . . . . .	52
lipnet::generator_t< blaze::StaticVector< T, N, blaze::columnVector > > . . . . .	53
lipnet::generator_t< layer_t< T, I, O > > . . . . .	54
lipnet::generator_t< liptrainweights_t< T, N... > > . . . . .	54
lipnet::generator_t< parameter_decompo_t< T, N... > > . . . . .	54
lipnet::generator_t< std::tuple< ARGS... > > . . . . .	55
lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD > . . . . .	55
lipnet::helper_function_t< V > . . . . .	58
lipnet::helper_inner_t< T, V1, V2 > . . . . .	58
lipnet::helper_norm_t< T, V > . . . . .	59
image_t . . . . .	59
IMPL	
lipnet::optimizer_t< T, P, IMPL, VARS > . . . . .	104
lipnet::inverse_diagentry< T, N, NARGS > . . . . .	60
lipnet::inverse_diagentry< T, N... > . . . . .	60
lipnet::inverse_diagentry_impl< T, N, NS > . . . . .	60
lipnet::inverse_diagentry_impl< T, N > . . . . .	60
lipnet::inverse_subentry< T, NI, NO, RE, NARGS > . . . . .	61
lipnet::inverse_subentry< T, N... > . . . . .	61
lipnet::inverse_subentry_impl< T, NI, NO, NS > . . . . .	61
lipnet::inverse_subentry_impl< T, NI, NO > . . . . .	61
lipnet::inverse_topology< T, N > . . . . .	61

lipnet::join_tuples< typename, typename > . . . . .	62
lipnet::join_tuples< std::tuple< NEW... >, std::tuple< NEXT... > > . . . . .	62
lipnet::layer_t< T, I, O > . . . . .	63
linsearch_t	
lipnet::linsearch_t< IMPL, T, DIRECTION > . . . . .	64
lipnet::lipcalc_parameter_t< T, N > . . . . .	66
lipnet::liptrainweights_t< T, N > . . . . .	66
lipnet::liptrainweights_t< T, N... > . . . . .	66
lipnet::loader_t< T > . . . . .	66
metainfo_t	
lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t . . . . .	68
lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t . . . . .	69
lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t . . . . .	68
lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t . . . . .	69
lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t . . . . .	70
metainfo_t	
lipnet::metainfo_t< IMPL > . . . . .	71
lipnet::backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t . . . . .	69
lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t . . . . .	70
lipnet::mosek_projection_wot_t< T, N > . . . . .	71
lipnet::network_data_t< T, IN, OUT > . . . . .	73
lipnet::network_libcalc_t< T, N > . . . . .	73
lipnet::network_libtrain_enforcing_t< T, N > . . . . .	74
lipnet::network_t< T, ATYPE, N > . . . . .	97
lipnet::network_topology< T, NI, NO, NARGS > . . . . .	99
lipnet::network_topology< T, N... > . . . . .	99
lipnet::network_topology_impl< T, NI, NO, NS > . . . . .	99
lipnet::network_topology_impl< T, NI, NO > . . . . .	100
std::nonesuch . . . . .	100
lipnet::norm_t< T, V > . . . . .	100
lipnet::norm_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > . . . . .	101
lipnet::norm_t< T, blaze::StaticVector< T, N, blaze::columnVector > > . . . . .	102
lipnet::norm_t< T, layer_t< T, I, O > > . . . . .	103
lipnet::norm_t< T, lipcalc_parameter_t< T, N... > > . . . . .	103
lipnet::norm_t< T, liptrainweights_t< T, N... > > . . . . .	104
lipnet::norm_t< T, std::tuple< ARGS... > > . . . . .	104
lipnet::parameter_decompo_t< T, N > . . . . .	107
lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::parameter_t . . . . .	108
lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >::parameter_t . . . . .	109
lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >::parameter_t . . . . .	109
lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::parameter_t . . . . .	109
lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::parameter_t . . . . .	110
lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >::parameter_t . . . . .	110
lipnet::parameter_tparam< T, N, NARGS > . . . . .	111
lipnet::parameter_tparam< T, N... > . . . . .	111
lipnet::parameter_tparam_impl< T, N, NS > . . . . .	111
lipnet::parameter_tparam_impl< T, N, R > . . . . .	111
lipnet::problem_t< T, TYPE, IMPL, ARGS > . . . . .	111
lipnet::problem_t< T, problem_type::ADMM, network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, N... >, network_t< T, ATYPE, N... >::layer_t, network_t< T, ATYPE, N... >::layer_t, network_t< T, ATYPE, N... >::layer_t > . . . . .	111
lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N > . . . . .	82
lipnet::problem_t< T, problem_type::NONLINEAR, network_problem_batch_admm_t< T, ATYPE, LOSS, N... >, network_t< T, ATYPE, N... >::layer_t, network_t< T, ATYPE, N... >::layer_t > . . . . .	111
lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N > . . . . .	76
lipnet::problem_t< T, problem_type::NONLINEAR, network_problem_batch_l2_t< T, ATYPE, LOSS, N... >, network_t< T, ATYPE, N... >::layer_t, network_t< T, ATYPE, N... >::layer_t > . . . . .	111

lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N > . . . . .	78
lipnet::problem_t< T, problem_type::NONLINEAR, network_problem_batch_t< T, ATYPE, LOSS, N... >, network_t< T, ATYPE, N... >::layer_t, network_t< T, ATYPE, N... >::layer_t > . . . . .	111
lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N > . . . . .	80
lipnet::problem_t< T, problem_type::NONLINEAR, network_problem_log_barrier_t< T, ATYPE, LOSS, N... > > . . . . .	111
lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N > . . . . .	86
lipnet::problem_t< T, problem_type::NONLINEAR, network_problem_log_barrier_wot_t< T, ATYPE, LOSS, N... >, network_t< T, ATYPE, N... >::layer_t, network_t< T, ATYPE, N... >::layer_t > . . . . .	111
lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N > . . . . .	90
lipnet::problem_t< T, problem_type::NONLINEAR, network_problem_projection_t< T, ATYPE, LOSS, N... > > . . . . .	111
lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N > . . . . .	94
lipnet::prod_t< T, V1, V2 > . . . . .	112
lipnet::prod_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > > . . . . .	112
lipnet::prod_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > > . . . . .	114
lipnet::prod_t< T, layer_t< T, l1, O1 >, layer_t< T, l2, O2 > > . . . . .	116
lipnet::prod_t< T, lipcalc_parameter_t< T, N... >, lipcalc_parameter_t< T, N... > > . . . . .	116
lipnet::prod_t< T, liptrainweights_t< T, N... >, liptrainweights_t< T, N... > > . . . . .	116
lipnet::prod_t< T, parameter_decompo_t< T, N... >, parameter_decompo_t< T, N... > > . . . . .	117
lipnet::prod_t< T, std::tuple< ARGS1... >, std::tuple< ARGS2... > > . . . . .	117
lipnet::series_t< T > . . . . .	117
lipnet::solve_function_helper< P, VAR > . . . . .	118
lipnet::squared_error_t< T > . . . . .	118
lipnet::statistics_helper . . . . .	119
lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >::statistics_t . . . . .	120
lipnet::optimizer_t< T, P, IMPL, VARS >::main_statistics_t . . . . .	67
lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::statistics_t . . . . .	121
lipnet::admm_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::statistics_t . . . . .	121
lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >::statistics_t . . . . .	122
lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >::statistics_t . . . . .	123
lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::statistics_t . . . . .	123
lipnet::optimizer_t< T, P, IMPL, VARS >::stats_type_exists< TT, U > . . . . .	124
lipnet::statistics_helper::stats_type_exists< TT, U > . . . . .	124
lipnet::optimizer_t< T, P, IMPL, VARS >::stats_type_exists< TT, typename void_t< typename TT::statistics_t >::type > . . . . .	125
lipnet::statistics_helper::stats_type_exists< TT, typename void_t< typename TT::statistics_t >::type > . . . . .	125
lipnet::network_t< T, ATYPE, N >::topology_serialization_t . . . . .	125
lipnet::data_container_t< T >::tuple_t< saveing > . . . . .	126
lipnet::cholesky_topology< T, N >::type . . . . .	126
lipnet::inverse_topology< T, N >::type . . . . .	126
lipnet::data_container_t< T >::view_t< saveing > . . . . .	127
lipnet::data_container_t< T >::view_t< true > . . . . .	127
lipnet::optimizer_t< T, P, IMPL, VARS >::void_t< TT > . . . . .	127
lipnet::statistics_helper::void_t< TT > . . . . .	128
std::void_type . . . . .	128
lipnet::optimizer_t< T, P, IMPL, VARS >::main_statistics_t . . . . .	67



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">lipnet::activation_t&lt; T, TYPE &gt;</a>	
The <a href="#">activation_t</a> struct; implementation of the activation functions	11
<a href="#">lipnet::adam_barrier_t_impl&lt; T, P, VAR, GRAD, feasibility_enabled &gt;</a>	
Modified adam method for use with barrier functions; it follows the central path	13
<a href="#">lipnet::adam_momentum_t_impl&lt; T, P, VAR, GRAD &gt;</a>	
The Adam method. [5]	15
<a href="#">lipnet::adam_projected_t_impl&lt; T, P, VAR, GRAD &gt;</a>	
Modified Adam method. Projected Adam method. [5]	17
<a href="#">lipnet::admm_optimizer_t_impl&lt; T, P, X, Z, DUAL &gt;</a>	
Alternating Direction Method of Multipliers. ADMM [1]	19
<a href="#">lipnet::backpropagation_batch_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">backpropagation_batch_t</a> struct; implmentation of backtracking with batches	23
<a href="#">lipnet::barrierfunction_t&lt; T, N &gt;</a>	27
<a href="#">lipnet::barrierfunction_wot_t&lt; T, N &gt;</a>	28
<a href="#">lipnet::calculate_lipschitz_t&lt; T, N &gt;</a>	
Compute trivial lipschitz constant	28
<a href="#">lipnet::cholesky_diagentry&lt; T, N, NARGS &gt;</a>	29
<a href="#">lipnet::cholesky_diagentry_impl&lt; T, N, NS &gt;</a>	30
<a href="#">lipnet::cholesky_diagentry_impl&lt; T, N &gt;</a>	30
<a href="#">lipnet::cholesky_subentry&lt; T, NI, NO, RE, NARGS &gt;</a>	30
<a href="#">lipnet::cholesky_subentry_impl&lt; T, NI, NO, NS &gt;</a>	30
<a href="#">lipnet::cholesky_subentry_impl&lt; T, NI, NO &gt;</a>	31
<a href="#">lipnet::cholesky_topology&lt; T, N &gt;</a>	31
<a href="#">lipnet::cross_entropy_t&lt; T &gt;</a>	
The <a href="#">cross_entropy_t</a> struct; implementation of the cross entropy objective function	31
<a href="#">lipnet::data_container_t&lt; T &gt;</a>	
Trining data holder; <a href="#">data_container_t</a>	32
<a href="#">lipnet::network_t&lt; T, ATYPE, N &gt;::data_serialization_t&lt; saveing &gt;</a>	
Serialization helper struct	33
<a href="#">lipnet::data_container_t&lt; T &gt;::data_t&lt; saveing &gt;</a>	34
<a href="#">lipnet::decompos_diagentry&lt; T, N, NARGS &gt;</a>	35
<a href="#">lipnet::decompos_diagentry_impl&lt; T, N, NS &gt;</a>	35
<a href="#">lipnet::decompos_diagentry_impl&lt; T, N &gt;</a>	35
<a href="#">lipnet::decompos_subentry&lt; T, NI, NO, RE, NARGS &gt;</a>	35
<a href="#">lipnet::decompos_subentry_impl&lt; T, NI, NO, NS &gt;</a>	36

lipnet::decompos_subentry_impl< T, NI, NO > . . . . .	36
std::detail::detector< Default, AlwaysVoid, Op, Args > . . . . .	36
std::detail::detector< Default, std::void_t< Op< Args... > >, Op, Args... > . . . . .	36
lipnet::equation_system_t< V1, V2 > . . . . .	
The equation_system_t struct. Just a interface for all possible types. Solve a system of equations . . . . .	37
lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > > . . . . .	
The equation_system_t struct for blaze::StaticMatrix . . . . .	37
lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticVector< T, N3, blaze::columnVector > > . . . . .	
The equation_system_t struct for blaze::StaticMatrix and blaze::StaticVector . . . . .	38
lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD > . . . . .	
Gradient descent algorithm . . . . .	40
lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t . . . . .	
The feasibility_t struct. Implementation of feasibility check for this problem . . . . .	42
lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t . . . . .	
The feasibility_t struct. Implementation of feasibility check for this problem . . . . .	43
lipnet::feasibility_t< IMPL, T, DIR > . . . . .	
The feasibility_t struct. base feasibility struct (basically a placeholder class) . . . . .	44
lipnet::feasibilitycheck_t< T, N > . . . . .	45
lipnet::feasibilitycheck_wot_t< T, N > . . . . .	46
lipnet::function_t< V > . . . . .	
The function_t struct. Just a interface for all possible types. Apply function to tensor elementwise . . . . .	46
lipnet::function_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > . . . . .	
The function_t struct for blaze::StaticMatrix . . . . .	47
lipnet::function_t< blaze::StaticVector< T, N, blaze::columnVector > > . . . . .	
The function_t struct for blaze::StaticVector . . . . .	47
lipnet::function_t< layer_t< T, I, O > > . . . . .	48
lipnet::function_t< liptrainweights_t< T, N... > > . . . . .	48
lipnet::function_t< std::tuple< ARGS... > > . . . . .	49
lipnet::generate_batch_data< T, B, N, NS > . . . . .	
Helper struct for data . . . . .	49
lipnet::generate_batch_data< T, B, N > . . . . .	
Helper struct for data . . . . .	49
lipnet::generate_batch_data_remove_first< T, B, N, NS > . . . . .	
Helper struct for data . . . . .	50
lipnet::generate_data< T, N, NS > . . . . .	
Helper struct for data . . . . .	50
lipnet::generate_data< T, N > . . . . .	
Helper struct for data . . . . .	51
lipnet::generate_data_remove_first< T, N, NS > . . . . .	
Helper struct for data . . . . .	51
lipnet::generator_t< V > . . . . .	
The generator_t struct. Just a interface for all possible types. Instanciate tensor of type V . . . . .	52
lipnet::generator_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > . . . . .	
The generator_t struct for blaze::StaticMatrix . . . . .	52
lipnet::generator_t< blaze::StaticVector< T, N, blaze::columnVector > > . . . . .	
The generator_t struct for blaze::StaticVector . . . . .	53
lipnet::generator_t< layer_t< T, I, O > > . . . . .	54
lipnet::generator_t< liptrainweights_t< T, N... > > . . . . .	54
lipnet::generator_t< parameter_decompo_t< T, N... > > . . . . .	54
lipnet::generator_t< std::tuple< ARGS... > > . . . . .	
Generator_t implementation for std::tuple . . . . .	55
lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD > . . . . .	
Projected gradient descent algorithm . . . . .	55
lipnet::helper_function_t< V > . . . . .	58
lipnet::helper_inner_t< T, V1, V2 > . . . . .	58
lipnet::helper_norm_t< T, V > . . . . .	59
image_t . . . . .	59
lipnet::inverse_diagentry< T, N, NARGS > . . . . .	60

<a href="#">lipnet::inverse_diagentry_impl&lt; T, N, NS &gt;</a>	60
<a href="#">lipnet::inverse_diagentry_impl&lt; T, N &gt;</a>	60
<a href="#">lipnet::inverse_subentry&lt; T, NI, NO, RE, NARGS &gt;</a>	61
<a href="#">lipnet::inverse_subentry_impl&lt; T, NI, NO, NS &gt;</a>	61
<a href="#">lipnet::inverse_subentry_impl&lt; T, NI, NO &gt;</a>	61
<a href="#">lipnet::inverse_topology&lt; T, N &gt;</a>	61
<a href="#">lipnet::join_tuples&lt; typename, typename &gt;</a>	
Helper struct to join two tuples. (std::tuple)	62
<a href="#">lipnet::join_tuples&lt; std::tuple&lt; NEW... &gt;, std::tuple&lt; NEXT... &gt; &gt;</a>	
Implementation of <a href="#">join_tuples</a> struct to join two tuples. (std::tuple)	62
<a href="#">lipnet::layer_t&lt; T, I, O &gt;</a>	
The <a href="#">layer_t</a> struct; the layer implementation of each layer; contains the weight and the biases	63
<a href="#">lipnet::linesearch_t&lt; IMPL, T, DIRECTION &gt;</a>	
The <a href="#">linesearch_t</a> struct. base linesearch struct (basically a placeholder class)	64
<a href="#">lipnet::lipcalc_parameter_t&lt; T, N &gt;</a>	66
<a href="#">lipnet::liptrainweights_t&lt; T, N &gt;</a>	66
<a href="#">lipnet::loader_t&lt; T &gt;</a>	
Struct for loading matrix from csv file;	66
<a href="#">lipnet::optimizer_t&lt; T, P, IMPL, VARS &gt;::main_statistics_t</a>	
The <a href="#">main_statistics_t</a> struct	67
<a href="#">lipnet::network_problem_batch_t&lt; T, ATYPE, LOSS, BATCH, N &gt;::metainfo_t</a>	68
<a href="#">lipnet::network_problem_batch_admm_t&lt; T, ATYPE, LOSS, BATCH, N &gt;::metainfo_t</a>	68
<a href="#">lipnet::network_problem_batch_l2_t&lt; T, ATYPE, LOSS, BATCH, N &gt;::metainfo_t</a>	69
<a href="#">lipnet::network_problem_log_barrier_t&lt; T, ATYPE, LOSS, BATCH, N &gt;::metainfo_t</a>	69
<a href="#">lipnet::backpropagation_batch_t&lt; T, ATYPE, LOSS, BATCH, N &gt;::metainfo_t</a>	69
<a href="#">lipnet::network_problem_log_barrier_wot_t&lt; T, ATYPE, LOSS, BATCH, N &gt;::metainfo_t</a>	70
<a href="#">lipnet::network_problem_projection_t&lt; T, ATYPE, LOSS, BATCH, N &gt;::metainfo_t</a>	70
<a href="#">lipnet::metainfo_t&lt; IMPL &gt;</a>	
The <a href="#">metainfo_t</a> struct. Data holder type for data needed during the iterations	71
<a href="#">lipnet::mosek_projection_wot_t&lt; T, N &gt;</a>	
The <a href="#">mosek_projection_wot_t</a> struct. Compute the projection of the reference weights. It is conic program and will be solved with mosek (interior point method)	71
<a href="#">lipnet::network_data_t&lt; T, IN, OUT &gt;</a>	
The <a href="#">network_data_t</a> struct; training dataset	73
<a href="#">lipnet::network_libcalc_t&lt; T, N &gt;</a>	73
<a href="#">lipnet::network_libtrain_enforcing_t&lt; T, N &gt;</a>	
Network_libtrain_enforcing_; Implementaion of the second subproblem of the admm method	74
<a href="#">lipnet::network_problem_batch_admm_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">network_problem_batch_admm_t</a> struct. The problem implementation of admm neural network training in batches	76
<a href="#">lipnet::network_problem_batch_l2_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">network_problem_batch_l2_t</a> struct. The problem implementation of l2 neural network training in batches	78
<a href="#">lipnet::network_problem_batch_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">network_problem_batch_t</a> struct. The problem implementation of nominal neural network training in batches	80
<a href="#">lipnet::network_problem_liptrain_enforcing_adam_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">network_problem_liptrain_enforcing_adam_t</a> struct. The problem implementation of admm neural network training to enforce lipschitz bound	82
<a href="#">lipnet::network_problem_log_barrier_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">network_problem_log_barrier_t</a> struct. The problem implementation of barrier neural network training in batches	86
<a href="#">lipnet::network_problem_log_barrier_wot_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">network_problem_log_barrier_wot_t</a> struct. The problem implementation of barrier (without T) neural network training in batches	90
<a href="#">lipnet::network_problem_projection_t&lt; T, ATYPE, LOSS, BATCH, N &gt;</a>	
The <a href="#">network_problem_projection_t</a> struct. The problem implementation of projected neural network training in batches	94

<a href="#">lipnet::network_t&lt; T, ATYPE, N &gt;</a>	
The <a href="#">network_t</a> struct; neural network implementation	97
<a href="#">lipnet::network_topology&lt; T, NI, NO, NARGS &gt;</a>	99
<a href="#">lipnet::network_topology_impl&lt; T, NI, NO, NS &gt;</a>	
Newtork layer holder and creator struct; helper struct to create compile time layers in stack memory -> performance	99
<a href="#">lipnet::network_topology_impl&lt; T, NI, NO &gt;</a>	100
<a href="#">std::nonesuch</a>	100
<a href="#">lipnet::norm_t&lt; T, V &gt;</a>	
The <a href="#">norm_t</a> struct. Just a interface for all possible types. Compute norm of argument	100
<a href="#">lipnet::norm_t&lt; T, blaze::StaticMatrix&lt; T, N1, N2, blaze::rowMajor &gt; &gt;</a>	
The <a href="#">norm_t</a> struct for blaze::StaticMatrix	101
<a href="#">lipnet::norm_t&lt; T, blaze::StaticVector&lt; T, N, blaze::columnVector &gt; &gt;</a>	
The <a href="#">norm_t</a> struct for blaze::StaticVector	102
<a href="#">lipnet::norm_t&lt; T, layer_t&lt; T, I, O &gt; &gt;</a>	103
<a href="#">lipnet::norm_t&lt; T, lipcalc_parameter_t&lt; T, N... &gt; &gt;</a>	103
<a href="#">lipnet::norm_t&lt; T, liptrainweights_t&lt; T, N... &gt; &gt;</a>	104
<a href="#">lipnet::norm_t&lt; T, std::tuple&lt; ARGS... &gt; &gt;</a>	104
<a href="#">lipnet::optimizer_t&lt; T, P, IMPL, VARS &gt;</a>	
The <a href="#">optimizer_t</a> struct. On instantiation a class with the implementation as base class will be created	104
<a href="#">lipnet::parameter_decompo_t&lt; T, N &gt;</a>	107
<a href="#">lipnet::adam_barrier_t_impl&lt; T, P, VAR, GRAD, feasibility_enabled &gt;::parameter_t</a>	
The <a href="#">parameter_t</a> struct; all meta parameters for optimisation	108
<a href="#">lipnet::fast_gradient_descent_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t</a>	109
<a href="#">lipnet::adam_momentum_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t</a>	109
<a href="#">lipnet::admm_optimizer_t_impl&lt; T, P, X, Z, DUAL &gt;::parameter_t</a>	109
<a href="#">lipnet::adam_projected_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t</a>	110
<a href="#">lipnet::gradient_descent_projected_t_impl&lt; T, P, VAR, GRAD &gt;::parameter_t</a>	110
<a href="#">lipnet::parameter_tparam&lt; T, N, NARGS &gt;</a>	111
<a href="#">lipnet::parameter_tparam_impl&lt; T, N, NS &gt;</a>	111
<a href="#">lipnet::parameter_tparam_impl&lt; T, N, R &gt;</a>	111
<a href="#">lipnet::problem_t&lt; T, TYPE, IMPL, ARGS &gt;</a>	
The <a href="#">problem_t</a> struct; base problem struct (basically a placeholder class)	111
<a href="#">lipnet::prod_t&lt; T, V1, V2 &gt;</a>	
The <a href="#">prod_t</a> struct. Just a interface for all possible types. Compute inner/outer/... products	112
<a href="#">lipnet::prod_t&lt; T, blaze::StaticMatrix&lt; T, N1, N2, blaze::rowMajor &gt;, blaze::StaticMatrix&lt; T, N3, N4, blaze::rowMajor &gt; &gt;</a>	
The <a href="#">prod_t</a> struct for blaze::StaticMatrix	112
<a href="#">lipnet::prod_t&lt; T, blaze::StaticVector&lt; T, N1, blaze::columnVector &gt;, blaze::StaticVector&lt; T, N2, blaze::columnVector &gt; &gt;</a>	
The <a href="#">prod_t</a> struct for blaze::StaticVector	114
<a href="#">lipnet::prod_t&lt; T, layer_t&lt; T, I1, O1 &gt;, layer_t&lt; T, I2, O2 &gt; &gt;</a>	116
<a href="#">lipnet::prod_t&lt; T, lipcalc_parameter_t&lt; T, N... &gt;, lipcalc_parameter_t&lt; T, N... &gt; &gt;</a>	116
<a href="#">lipnet::prod_t&lt; T, liptrainweights_t&lt; T, N... &gt;, liptrainweights_t&lt; T, N... &gt; &gt;</a>	116
<a href="#">lipnet::prod_t&lt; T, parameter_decompo_t&lt; T, N... &gt;, parameter_decompo_t&lt; T, N... &gt; &gt;</a>	117
<a href="#">lipnet::prod_t&lt; T, std::tuple&lt; ARGS1... &gt;, std::tuple&lt; ARGS2... &gt; &gt;</a>	117
<a href="#">lipnet::series_t&lt; T &gt;</a>	
The <a href="#">series_t</a> struct. Base struct for logging	117
<a href="#">lipnet::solve_function_helper&lt; P, VAR &gt;</a>	118
<a href="#">lipnet::squared_error_t&lt; T &gt;</a>	
The <a href="#">squared_error_t</a> struct; implementation of the squarred error objective function	118
<a href="#">lipnet::statistics_helper</a>	
The <a href="#">statistics_helper</a> struct. Helper function to disable logging for performance reasons if it is desired	119
<a href="#">lipnet::gradient_descent_projected_t_impl&lt; T, P, VAR, GRAD &gt;::statistics_t</a>	
Problem specific implementation of <a href="#">statistics_t</a>	120
<a href="#">lipnet::admm_optimizer_t_impl&lt; T, P, X, Z, DUAL &gt;::statistics_t</a>	
Problem specific implementation of <a href="#">statistics_t</a>	121

lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::statistics_t	
Problem specific implementation of <a href="#">statistics_t</a> . . . . .	121
lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >::statistics_t	
Problem specific implementation of <a href="#">statistics_t</a> . . . . .	122
lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >::statistics_t	
Problem specific implementation of <a href="#">statistics_t</a> . . . . .	123
lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::statistics_t	
Problem specific implementation of <a href="#">statistics_t</a> . . . . .	123
lipnet::optimizer_t< T, P, IMPL, VARS >::stats_type_exists< TT, U > . . . . .	124
lipnet::statistics_helper::stats_type_exists< TT, U > . . . . .	124
lipnet::optimizer_t< T, P, IMPL, VARS >::stats_type_exists< TT, typename void_t< typename TT::statistics_t >::type >	
125	
lipnet::statistics_helper::stats_type_exists< TT, typename void_t< typename TT::statistics_t >::type > .	125
lipnet::network_t< T, ATYPE, N >::topology_serialization_t	
Serialization helper struct . . . . .	125
lipnet::data_container_t< T >::tuple_t< saveing > . . . . .	126
lipnet::cholesky_topology< T, N >::type . . . . .	126
lipnet::inverse_topology< T, N >::type . . . . .	126
lipnet::data_container_t< T >::view_t< saveing > . . . . .	127
lipnet::optimizer_t< T, P, IMPL, VARS >::void_t< TT >	
Type holder . . . . .	127
lipnet::statistics_helper::void_t< TT > . . . . .	128
std::void_type	
Void type. Holdes nothing . . . . .	128



## Chapter 3

# Class Documentation

### 3.1 lipnet::activation\_t< T, TYPE > Struct Template Reference

The [activation\\_t](#) struct; implementation of the activation functions.

```
#include <activation.hpp>
```

#### Public Types

- `template<typename TT, size_t O, size_t I>`  
using **matrix\_t** = blaze::StaticMatrix< TT, O, I, blaze::columnMajor >
- `template<typename TT, size_t N>`  
using **vector\_t** = blaze::StaticVector< TT, N, blaze::columnVector >

#### Static Public Member Functions

- `template<size_t N, size_t BATCH = 1>`  
static auto [forward](#) (const auto &val)  
*evaluate activation function*
- `template<size_t N, size_t BATCH = 1>`  
static auto [derivative](#) (const auto &val)  
*derivative of activation function*

#### 3.1.1 Detailed Description

```
template<typename T, atype_t TYPE>  
struct lipnet::activation_t< T, TYPE >
```

The [activation\\_t](#) struct; implementation of the activation functions.

#### Template Parameters

<i>T</i>	numerical value type
<i>TYPE</i>	choose the activation type

### 3.1.2 Member Function Documentation

#### 3.1.2.1 derivative()

```
template<typename T , atype_t TYPE>
template<size_t N, size_t BATCH = 1>
static auto lipnet::activation_t< T, TYPE >::derivative (
    const auto & val ) [inline], [static]
```

derivative of activation function

##### Template Parameters

<i>N</i>	input dimension
<i>BATCH</i>	batch size

##### Parameters

<i>val</i>	input vector
------------	--------------

##### Returns

output vector

#### 3.1.2.2 forward()

```
template<typename T , atype_t TYPE>
template<size_t N, size_t BATCH = 1>
static auto lipnet::activation_t< T, TYPE >::forward (
    const auto & val ) [inline], [static]
```

evaluate activation function

##### Template Parameters

<i>N</i>	input dimension
<i>BATCH</i>	batch size

##### Parameters

<i>val</i>	input vector
------------	--------------



### Returns

output vector

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

$$\sigma(x) = \tanh(x)$$

$$\sigma(x) = x$$

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

- lipnet/include/lipnet/network/activation.hpp

## 3.2 lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled > Struct Template Reference

Modified adam method for use with barrier functions; it follows the central path.

```
#include <adam_barrier.hpp>
```

### Classes

- struct [parameter\\_t](#)  
*The [parameter\\_t](#) struct; all meta parameters for optimisation.*
- struct [statistics\\_t](#)  
*problem specific implementation of [statistics\\_t](#)*

### Public Member Functions

- void **unpack** (std::tuple< GRAD, T > &&t, GRAD &dx, T &fx) const
- [adam\\_barrier\\_t\\_impl](#) ([parameter\\_t](#) &&param=[parameter\\_t](#){(size\_t) 5e5,(size\_t) 5, 1e-10, 1e-8, 300, 1.0, 0.02, 0.9, 0.999, 5.0, 0.5, 0.5, 1e-8})  
*Default constructor.*
- template<bool stats\_enabled = false, bool problem\_stats\_exists = statistics\_helper::stats\_type\_exists<P>::value>  
std::tuple< VAR, T > **run** (P &prob, VAR &&x, typename std::conditional< stats\_enabled, [statistics\\_t](#), [std::void\\_type](#) >::type &stats) const  
*The run method. Implementation of the optimisation algorithm. Modified Adam-method.*

### Public Attributes

- [parameter\\_t](#) param  
*variables to optimize*

#### 3.2.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD, bool feasibility_enabled = false>
struct lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >
```

Modified adam method for use with barrier functions; it follows the central path.

## Template Parameters

<i>T</i>	numerical value type
<i>P</i>	problem type
<i>VAR</i>	variable type
<i>GRAD</i>	gradient type
<i>feasibility_enabled</i>	set this value to true if you want to enable feasibility checking

## 3.2.2 Constructor & Destructor Documentation

### 3.2.2.1 adam\_barrier\_t\_impl()

```
template<typename T , typename P , typename VAR , typename GRAD , bool feasibility_enabled =
false>
lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::adam_barrier_t_impl (
    parameter_t && param = parameter_t{ (size_t) 5e5, (size_t) 5, 1e-10, 1e-8, 300, 1.0, 0.02, 0.9, 0
) [inline], [explicit]
```

Default constructor.

## Parameters

<i>hyperparameter</i>	of optimisation. Init hyperparameters with (size_t) 5e5, (size_t) 5, 1e-10, 1e-8, 300, 1.0, 0.02, 0.9, 0.999, 5.0, 0.5, 0.5, 1e-8
-----------------------	---

## 3.2.3 Member Function Documentation

### 3.2.3.1 run()

```
template<typename T , typename P , typename VAR , typename GRAD , bool feasibility_enabled =
false>
template<bool stats_enabled = false, bool problem_stats_exists = statistics_helper::stats_↔
type_exists<P>::value>
std::tuple<VAR,T> lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::run (
    P & prob,
    VAR && x,
    typename std::conditional< stats_enabled, statistics_t, std::void_type >::type &
stats ) const [inline]
```

The run method. Implementation of the optimisation algorithm. Modified Adam-method.

## Template Parameters

<code>stats_enabled</code>	enable/disable logging
----------------------------	------------------------

## Parameters

<code>prob</code>	problem
<code>x</code>	start variable / initial variable / start point
<code>stats</code>	statistics holder [5]

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

- lipnet/include/lipnet/optimizer/adam\_barrier.hpp

### 3.3 lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD > Struct Template Reference

The Adam method. [5].

```
#include <adam_momentum.hpp>
```

## Classes

- struct [parameter\\_t](#)
- struct [statistics\\_t](#)  
*problem specific implementation of [statistics\\_t](#)*

## Public Types

- typedef std::function< bool(const T &, const VAR &, const GRAD &)> **criterion\_t**

## Public Member Functions

- void **unpack** (std::tuple< GRAD, T > &&t, GRAD &dx, T &fx) const
- [adam\\_momentum\\_t\\_impl](#) ([parameter\\_t](#) &&param=[parameter\\_t](#){(size\_t) 5e4, 1e-10, 1e-4, 0.02, 0.9, 0.999, 1e-8}, [criterion\\_t](#) &&c=[ ](const T &, const VAR &, const GRAD &){return true;})  
*Default constructor.*
- template<bool stats\_enabled = false>  
std::tuple< VAR, T > **run** (P &prob, VAR &&x, typename std::conditional< stats\_enabled, [statistics\\_t](#), [std::void\\_type](#) >::type &stats) const  
*The run method. Implementation of the optimisation algorithm. Adam-method.*

## Public Attributes

- [parameter\\_t](#) **param**  
*variables to optimize*
- [criterion\\_t](#) **criterion**  
*custom stopping criterion*

### 3.3.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>
struct lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >
```

The Adam method. [5].

#### Template Parameters

<i>T</i>	numerical value type
<i>P</i>	problem type
<i>VAR</i>	variable type
<i>GRAD</i>	gradient type

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 adam\_momentum\_t\_impl()

```
template<typename T , typename P , typename VAR , typename GRAD >
lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >::adam_momentum_t_impl (
    parameter_t && param = parameter_t( (size_t) 5e4, 1e-10, 1e-4, 0.02, 0.9, 0.999, 1e-8},
    criterion_t && c = [] (const T&, const VAR&, const GRAD&) {return true;} ) [inline],
[explicit]
```

Default constructor.

#### Parameters

<i>hyperparameter</i>	of optimisation. Init hyperparameters with (size_t) 5e4, 1e-10, 1e-4, 0.02, 0.9, 0.999, 1e-8
-----------------------	--

### 3.3.3 Member Function Documentation

#### 3.3.3.1 run()

```
template<typename T , typename P , typename VAR , typename GRAD >
template<bool stats_enabled = false>
std::tuple<VAR,T> lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >::run (
    P & prob,
    VAR && x,
    typename std::conditional< stats_enabled, statistics_t, std::void_type >::type &
    stats ) const [inline]
```

The run method. Implementation of the optimisation algorithm. Adam-method.

## Template Parameters

<code>stats_enabled</code>	enable/disable logging
----------------------------	------------------------

## Parameters

<code>prob</code>	problem
<code>x</code>	start variable / initial variable / start point
<code>stats</code>	statistics holder [5]

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

- lipnet/include/lipnet/optimizer/adam\_momentum.hpp

### 3.4 lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD > Struct Template Reference

Modified Adam method. Projected Adam method. [5].

```
#include <adam_projected.hpp>
```

## Classes

- struct [parameter\\_t](#)
- struct [statistics\\_t](#)  
*problem specific implementation of [statistics\\_t](#)*

## Public Member Functions

- void **unpack** (std::tuple< GRAD, T > &&t, GRAD &dx, T &fx) const
- auto **project** (const P &prob, VAR &&var) const  
*The project method. Call projection method of problem.*
- **adam\_projected\_t\_impl** ([parameter\\_t](#) &&param=[parameter\\_t](#){(size\_t) 1e4, 1e-7, 1e-8, 300, 0.02, 0.9, 0.999, 1e-8 })  
*Default constructor.*
- template<bool stats\_enabled = false>  
std::tuple< VAR, T > **run** (P &prob, VAR &&x, typename std::conditional< stats\_enabled, [statistics\\_t](#), [std::void\\_type](#) >::type &stats) const  
*The run method. Implementation of the optimisation algorithm. Adam-method.*

## Public Attributes

- [parameter\\_t](#) param  
*variables to optimize*

#### 3.4.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>
struct lipnet::adam_projected_t_impl< T, P, VAR, GRAD >
```

Modified Adam method. Projected Adam method. [5].

## Template Parameters

<i>T</i>	numerical value type
<i>P</i>	problem type
<i>VAR</i>	variable type
<i>GRAD</i>	gradient type

## 3.4.2 Constructor &amp; Destructor Documentation

## 3.4.2.1 adam\_projected\_t\_impl()

```
template<typename T , typename P , typename VAR , typename GRAD >
lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::adam_projected_t_impl (
    parameter_t && param = parameter_t((size_t) 1e4, 1e-7, 1e-8, 300, 0.02, 0.9, 0.999, 1e-8 )
) [inline], [explicit]
```

Default constructor.

## Parameters

<i>hyperparameter</i>	of optimisation. Init hyperparameters with (size_t) 1e4, 1e-7, 1e-8, 300, 0.02, 0.9, 0.999, 1e-8
-----------------------	--

## 3.4.3 Member Function Documentation

## 3.4.3.1 project()

```
template<typename T , typename P , typename VAR , typename GRAD >
auto lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::project (
    const P & prob,
    VAR && var ) const [inline]
```

The project method. Call projection method of problem.

## Parameters

<i>prob</i>	problem
<i>var</i>	current variables; will be projected to feasible set

## 3.4.3.2 run()

```
template<typename T , typename P , typename VAR , typename GRAD >
template<bool stats_enabled = false>
std::tuple<VAR,T> lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::run (
    P & prob,
    VAR && x,
    typename std::conditional< stats_enabled, statistics_t, std::void_type >::type &
    stats ) const [inline]
```

The run method. Implementation of the optimisation algorithm. Adam-method.

## Template Parameters

<code>stats_enabled</code>	enable/disable logging
----------------------------	------------------------

## Parameters

<code>prob</code>	problem
<code>x</code>	start variable / initial variable / start point
<code>stats</code>	statistics holder [5]

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

- lipnet/include/lipnet/optimizer/adam\_projected.hpp

## 3.5 lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL > Struct Template Reference

Alternating Direction Method of Multipliers. ADMM [1].

```
#include <admm_optimizer.hpp>
```

## Classes

- struct [parameter\\_t](#)
- struct [statistics\\_t](#)  
problem specific implementation of [statistics\\_t](#)

## Public Member Functions

- DUAL [residual](#) (const P &prob, const X &x, const Z &z) const  
compute residual  $Ax + Bz - c$  [1]
- X [optimize1](#) (const P &prob, const X &x, const Z &z, const DUAL &d) const  
optimize first subproblem.  $\arg \min_x L_v(x, z^t, y^t)$  [1]
- Z [optimize2](#) (const P &prob, const X &x, const Z &z, const DUAL &d) const  
optimize second subproblem.  $\arg \min_z L_v(x^{t+1}, z, y^t)$  [1]

- T [evaluate](#) (const P &prob, const X &x, const Z &z) const  
*evaluate augmented lagrangian*
- [admm\\_optimizer\\_t\\_impl](#) ([parameter\\_t](#) &&param=[parameter\\_t](#){(size\_t) 1e4, 2, 1e-1})  
*Default constructor.*
- template<bool stats\_enabled = false>  
std::tuple< X, Z, T > [run](#) (P &prob, X &&x, Z &&z, typename std::conditional< stats\_enabled, [statistics\\_t](#), [std::void\\_type](#) >::type &stats) const  
*The run method. Implementation of the optimisation algorithm. Adam-method.*

## Public Attributes

- [parameter\\_t](#) param  
*variables to optimize*

### 3.5.1 Detailed Description

```
template<typename T, typename P, typename X, typename Z, typename DUAL>
struct lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >
```

Alternating Direction Method of Multipliers. ADMM [1].

Template Parameters

<i>T</i>	numerical value type
<i>P</i>	problem type
<i>X</i>	first variable type
<i>Z</i>	second variable type
<i>DUAL</i>	dual variable type

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 admm\_optimizer\_t\_impl()

```
template<typename T , typename P , typename X , typename Z , typename DUAL >
lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::admm_optimizer_t_impl (
    parameter\_t && param = parameter\_t{ (size_t) 1e4, 2, 1e-1} ) [inline], [explicit]
```

Default constructor.

Parameters

<i>hyperparameter</i>	of optimisation. Init hyperparameters with (size_t) 1e4, 2, 1e-1
-----------------------	--



### 3.5.3 Member Function Documentation

#### 3.5.3.1 evaluate()

```
template<typename T , typename P , typename X , typename Z , typename DUAL >
T lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::evaluate (
    const P & prob,
    const X & x,
    const Z & z ) const [inline]
```

evaluate augmented lagrangian

##### Parameters

<i>prob</i>	problem
<i>x</i>	variable
<i>z</i>	variable

##### Returns

loss/objectiv

#### 3.5.3.2 optimize1()

```
template<typename T , typename P , typename X , typename Z , typename DUAL >
X lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::optimize1 (
    const P & prob,
    const X & x,
    const Z & z,
    const DUAL & d ) const [inline]
```

optimize first subproblem.  $\arg \min_x L_v(x, z^t, y^t)$  [1]

##### Parameters

<i>prob</i>	problem
<i>x</i>	variable
<i>z</i>	const variable
<i>d</i>	dual variable

##### Returns

optimal point x

### 3.5.3.3 optimize2()

```
template<typename T , typename P , typename X , typename Z , typename DUAL >
Z lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::optimize2 (
    const P & prob,
    const X & x,
    const Z & z,
    const DUAL & d ) const [inline]
```

optimize second subproblem.  $\arg \min_z L_v(x^{t+1}, z, y^t)$  [1]

#### Parameters

<i>prob</i>	problem
<i>x</i>	const variable
<i>z</i>	variable
<i>d</i>	dual variable

#### Returns

optimal point z

### 3.5.3.4 residual()

```
template<typename T , typename P , typename X , typename Z , typename DUAL >
DUAL lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::residual (
    const P & prob,
    const X & x,
    const Z & z ) const [inline]
```

compute residual  $Ax + Bz - c$  [1]

#### Parameters

<i>prob</i>	problem
<i>x</i>	variable
<i>z</i>	variable

#### Returns

residual

### 3.5.3.5 run()

```
template<typename T , typename P , typename X , typename Z , typename DUAL >
template<bool stats_enabled = false>
```

```
std::tuple<X,Z,T> lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::run (
    P & prob,
    X && x,
    Z && z,
    typename std::conditional< stats_enabled, statistics_t, std::void_type >::type &
    stats ) const [inline]
```

The run method. Implementation of the optimisation algorithm. Adam-method.

#### Template Parameters

<i>stats_enabled</i>	enable/disable logging
----------------------	------------------------

#### Parameters

<i>prob</i>	problem
<i>x</i>	start variable / initial variable / start point (first variable)
<i>z</i>	start variable / initial variable / start point (second variable)
<i>stats</i>	statistics holder [5]

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

- lipnet/include/lipnet/optimizer/admm\_optimizer.hpp

## 3.6 lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference

The [backpropagation\\_batch\\_t](#) struct; implmentation of backtracking with batches.

```
#include <backpropagation.hpp>
```

### Classes

- struct [metainfo\\_t](#)

### Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef std::integral\_constant< size\_t, sizeof...(N) -1 > **L**
- typedef std::integral\_constant< size\_t,(N+...) > **NL**
- typedef std::integer\_sequence< size\_t, N... > **DIMS**
- typedef [network\\_t](#)< T, ATYPE, N... >::layer\_t **variable\_t**
- typedef [generate\\_batch\\_data\\_remove\\_first](#)< T, BATCH, N... >::type **zdata\_t**
- typedef [generate\\_batch\\_data](#)< T, BATCH, N... >::type **xdata\_t**

## Public Member Functions

- **backpropagation\_batch\_t** (LOSS< T > &&l, [network\\_data\\_t](#)< T, at< 0, N... >(), at< L::value, N... >() > &&data)
- void **run** (const variable\_t &var, [metainfo\\_t](#) &info, variable\_t &gradient, T &objective) const  
*run function; compute backpropagation*
- void **compute** (const variable\_t &var, variable\_t &gradient, T &objective) const  
*run function; compute backpropagation*
- void **forward** (const variable\_t &layers, xdata\_t &x, zdata\_t &z) const  
*forward function; compute forwardpropagation*
- void **backward** (const variable\_t &layers, variable\_t &gradient, xdata\_t &x, zdata\_t &delta, zdata\_t &z) const  
*backward function; compute backpropagation*

## Public Attributes

- [network\\_data\\_t](#)< T, at< 0, N... >(), at< L::value, N... >() > **training\_data**
- LOSS< T > **loss**

### 3.6.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N >
```

The [backpropagation\\_batch\\_t](#) struct; implmentation of backtracking with batches.

#### Template Parameters

<i>T</i>	numerical type
<i>ATYPE</i>	activation function type
<i>LOSS</i>	loss function type
<i>BATCH</i>	batch size
<i>N</i>	network topology

### 3.6.2 Member Function Documentation

#### 3.6.2.1 backward()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
void lipnet::backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N >::backward (
    const variable_t & layers,
    variable_t & gradient,
    xdata_t & x,
```

```
zdata_t & delta,  
zdata_t & z ) const [inline]
```

backward function; compute backpropagation

**Parameters**

<i>layers</i>	weights and biases at each layer
<i>gradient</i>	gradient with respect to the weights and biases
<i>x</i>	
<i>delta</i>	gradients with respect to the layer inputs
<i>z</i>	

**3.6.2.2 compute()**

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO←
SS, size_t BATCH, size_t ... N>
void lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >::compute (
    const variable_t & var,
    variable_t & gradient,
    T & objective ) const [inline]
```

run function; compute backpropagation

**Parameters**

<i>var</i>	current position
<i>info</i>	optimisation metainfo which are needed during the iterations
<i>gradient</i>	the computed gradients; the return value
<i>objective</i>	the loss at the current position

**3.6.2.3 forward()**

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO←
SS, size_t BATCH, size_t ... N>
void lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >::forward (
    const variable_t & layers,
    xdata_t & x,
    zdata_t & z ) const [inline]
```

forward function; compute forwardpropagation

**Parameters**

<i>layers</i>	weights and biases at each layer
<i>x</i>	
<i>z</i>	

## 3.6.2.4 run()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO←
SS, size_t BATCH, size_t ... N>
void lipnet::backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N >::run (
    const variable_t & var,
    metainfo_t & info,
    variable_t & gradient,
    T & objective ) const [inline]
```

run function; compute backpropagation

## Parameters

<i>var</i>	current position
<i>info</i>	optimisation metainfo which are needed during the iterations
<i>gradient</i>	the computed gradients; the return value
<i>objective</i>	the loss at the current position

## See also

[compute\( const variable\\_t& var, variable\\_t& gradient, T& objective \) const](#)

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

- [lipnet/include/lipnet/network/backpropagation.hpp](#)

## 3.7 lipnet::barrierfunction\_t&lt; T, N &gt; Struct Template Reference

## Public Types

- `template<size_t NN>`  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- `template<size_t NN1, size_t NN2>`  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- `typedef blaze::IdentityMatrix< T > eye`
- `typedef cholesky\_topology< T, N... >::type cholesky_t`
- `typedef inverse\_topology< T, N... >::type inverse_t`
- `typedef network\_topology< T, N... >::type weights_t`
- `typedef parameter\_tparam< T, N... >::type tparam_t`
- `typedef liptrainweights\_t< T, N... > variable_t`
- `typedef std::integral_constant< size_t, sizeof...(N) -2 > LN`
- `typedef std::integral_constant< size_t, sizeof...(N) -1 > L`

## Public Member Functions

- **barrierfunction\_t** (const T lipschitz=70.0)
- auto **compute** (const [variable\\_t](#) &var, [variable\\_t](#) &gradient, const T &gamma) const
- `template<bool numeric_stability = true, typename kondition = std::ratio<1,100>, typename = typename std::enable_if<kondition::den != 0>::type>`  
[cholesky\\_t](#) **chol** (const T lipschitz, const [variable\\_t](#) &var) const
- [inverse\\_t](#) **inv** (const [cholesky\\_t](#) &val) const

## Public Attributes

- **T lipschitz**

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

- lipnet/include/lipnet/lipschitz/barrier.hpp

## 3.8 lipnet::barrierfunction\_wot\_t< T, N > Struct Template Reference

### Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef blaze::IdentityMatrix< T > **eye**
- typedef [cholesky\\_topology](#)< T, N... >::type **cholesky\_t**
- typedef [inverse\\_topology](#)< T, N... >::type **inverse\_t**
- typedef [network\\_topology](#)< T, N... >::type **variable\_t**
- typedef [parameter\\_tparam](#)< T, N... >::type **tparam\_t**
- typedef std::integral\_constant< size\_t, sizeof...(N) -2 > **LN**
- typedef std::integral\_constant< size\_t, sizeof...(N) -1 > **L**

### Public Member Functions

- **barrierfunction\_wot\_t** (tparam\_t &&tmat, const T lipschitz=70.0)
- auto **compute** (const variable\_t &var, variable\_t &gradient, const T &gamma) const
- [cholesky\\_t](#) **chol** (const T lipschitz, const variable\_t &weights, const tparam\_t &tparam) const
- [inverse\\_t](#) **inv** (const [cholesky\\_t](#) &val) const

### Public Attributes

- **T lipschitz**
- tparam\_t **tparam**

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

- lipnet/include/lipnet/lipschitz/barrier\_wot.hpp

## 3.9 lipnet::calculate\_lipschitz\_t< T, N > Struct Template Reference

compute trivial lipschitz constant

```
#include <trivial.hpp>
```



## Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef [network\\_topology](#)< T, N... >::type **variable\_t**

## Static Public Member Functions

- static T **trivial\_lipschitz** (const variable\_t &var)

### 3.9.1 Detailed Description

```
template<typename T, size_t ... N>
struct lipnet::calculate_lipschitz_t< T, N >
```

compute trivial lipschitz constant

Template Parameters

<i>T</i>	numerical value type
<i>N</i>	network topology <a href="#">[3]</a>

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

- lipnet/include/lipnet/lipschitz/trivial.hpp

## 3.10 lipnet::cholesky\_diagentry< T, N, NARGS > Struct Template Reference

## Public Types

- typedef [cholesky\\_diagentry\\_impl](#)< T, NARGS... >::type **next**
- typedef [join\\_tuples](#)< std::tuple< T >, next >::type **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

### 3.11 `lipnet::cholesky_diagentry_impl< T, N, NS >` Struct Template Reference

#### Public Types

- typedef `cholesky_diagentry_impl< T, NS... >::type` **next**
- typedef `join_tuples< std::tuple< blaze::LowerMatrix< blaze::StaticMatrix< T, N, N > > >, next >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/topology.hpp`

### 3.12 `lipnet::cholesky_diagentry_impl< T, N >` Struct Template Reference

#### Public Types

- typedef `std::tuple< blaze::LowerMatrix< blaze::StaticMatrix< T, N, N > > >` **type**

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

- `lipnet/include/lipnet/lipschitz/topology.hpp`

### 3.13 `lipnet::cholesky_subentry< T, NI, NO, RE, NARGS >` Struct Template Reference

#### Public Types

- typedef `cholesky_subentry_impl< T, NI, NO, RE, NARGS... >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/topology.hpp`

### 3.14 `lipnet::cholesky_subentry_impl< T, NI, NO, NS >` Struct Template Reference

#### Public Types

- typedef `cholesky_subentry_impl< T, NO, NS... >::type` **next**
- typedef `join_tuples< std::tuple< blaze::StaticMatrix< T, NO, NI > >, next >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/topology.hpp`

## 3.15 lipnet::cholesky\_subentry\_impl< T, NI, NO > Struct Template Reference

### Public Types

- typedef std::tuple< blaze::StaticMatrix< T, NO, NI > > **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

## 3.16 lipnet::cholesky\_topology< T, N > Struct Template Reference

### Classes

- struct [type](#)

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

- lipnet/include/lipnet/lipschitz/topology.hpp

## 3.17 lipnet::cross\_entropy\_t< T > Struct Template Reference

The [cross\\_entropy\\_t](#) struct; implementation of the cross entropy objective function.

```
#include <loss.hpp>
```

### Public Types

- template<typename TT, size\_t O, size\_t I>  
using **matrix\_t** = blaze::StaticMatrix< TT, O, I, blaze::columnMajor >
- template<typename TT, size\_t N>  
using **vector\_t** = blaze::StaticVector< TT, N, blaze::columnVector >

### Public Member Functions

- template<size\_t N, size\_t BATCH = 0, typename std::enable\_if<!(BATCH<=0), int >::type = 0>  
T [evaluate](#) (const matrix\_t< T, N, BATCH > &target, const matrix\_t< T, N, BATCH > &data) const  
*The evaluate function; compute loss.*

### 3.17.1 Detailed Description

```
template<typename T>
struct lipnet::cross_entropy_t< T >
```

The [cross\\_entropy\\_t](#) struct; implementation of the cross entropy objective function.

$$\mathcal{L}(x, y) = \frac{\sum [x == y] \exp -x}{\sum \exp -x}$$

## Template Parameters

<i>T</i>	numerical value type
<i>TYPE</i>	choose the activation type

### 3.17.2 Member Function Documentation

#### 3.17.2.1 evaluate()

```
template<typename T >
template<size_t N, size_t BATCH = 0, typename std::enable_if<!(BATCH<=0), int >::type = 0>
T lipnet::cross_entropy_t< T >::evaluate (
    const matrix_t< T, N, BATCH > & target,
    const matrix_t< T, N, BATCH > & data ) const [inline]
```

The evaluate function; compute loss.

## Template Parameters

<i>N</i>	input dimension type
<i>BATCH</i>	batch size

## Parameters

<i>target</i>	real value
<i>estimated</i>	value

## Returns

loss

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

- lipnet/include/lipnet/network/loss.hpp

## 3.18 lipnet::data\_container\_t< T > Struct Template Reference

training data holder; [data\\_container\\_t](#)

```
#include <container.hpp>
```

## Classes

- struct [data\\_t](#)
- struct [tuple\\_t](#)
- struct [view\\_t](#)

## Public Types

- using **matrix\_t** = blaze::DynamicMatrix< T, blaze::rowMajor >

## Public Attributes

- matrix\_t **x**
- matrix\_t **y**

### 3.18.1 Detailed Description

```
template<typename T>
struct lipnet::data_container_t< T >
```

training data holder; [data\\_container\\_t](#)

#### Template Parameters

<i>T</i>	numerical value type
----------	----------------------

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

- lipnet/include/lipnet/loader/container.hpp

## 3.19 lipnet::network\_t< T, ATYPE, N >::data\_serialization\_t< saveing > Struct Template Reference

serialization helper struct

```
#include <network.hpp>
```

## Public Types

- using **value\_t** = typename std::conditional< saveing, const layer\_t, layer\_t >::type
- using **seq\_t** = std::make\_integer\_sequence< size\_t, L::value >

## Public Member Functions

- `template<class Archive , size_t ... INTS>`  
void **serialize\_impl** (Archive &ar, const std::integer\_sequence< size\_t, INTS... > &)
- `template<class Archive >`  
void **serialize** (Archive &ar)

## Public Attributes

- `value_t` & **layersdata**

### 3.19.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, size_t ... N>
template<bool saveing = true>
struct lipnet::network_t< T, ATYPE, N >::data_serialization_t< saveing >
```

serialization helper struct

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

- `lipnet/include/lipnet/network/network.hpp`

## 3.20 lipnet::data\_container\_t< T >::data\_t< saveing > Struct Template Reference

### Public Types

- using **value\_t** = typename std::conditional< saveing, const matrix\_t, matrix\_t >::type

### Public Member Functions

- `template<class Archive >`  
void **serialize** (Archive &ar)

### Public Attributes

- `value_t` & **x**
- `value_t` & **y**

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

- `lipnet/include/lipnet/loader/container.hpp`

### 3.21 `lipnet::decompos_diagentry< T, N, NARGS >` Struct Template Reference

#### Public Types

- typedef `decompos_diagentry_impl< T, NARGS... >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/decompos.hpp`

### 3.22 `lipnet::decompos_diagentry_impl< T, N, NS >` Struct Template Reference

#### Public Types

- typedef `decompos_diagentry_impl< T, NS... >::type` **next**
- typedef `join_tuples< std::tuple< blaze::StaticMatrix< T, N, N > >, next >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/decompos.hpp`

### 3.23 `lipnet::decompos_diagentry_impl< T, N >` Struct Template Reference

#### Public Types

- typedef `std::tuple< blaze::StaticMatrix< T, N, N > >` **type**

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

- `lipnet/include/lipnet/lipschitz/decompos.hpp`

### 3.24 `lipnet::decompos_subentry< T, NI, NO, RE, NARGS >` Struct Template Reference

#### Public Types

- typedef `decompos_subentry_impl< T, NI, NO, RE, NARGS... >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/decompos.hpp`

### 3.25 `lipnet::decompos_subentry_impl< T, NI, NO, NS >` Struct Template Reference

#### Public Types

- typedef `decompos_subentry_impl< T, NO, NS... >::type` **next**
- typedef `join_tuples< std::tuple< blaze::StaticMatrix< T, NO, NI > >, next >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/decompos.hpp`

### 3.26 `lipnet::decompos_subentry_impl< T, NI, NO >` Struct Template Reference

#### Public Types

- typedef `std::tuple< blaze::StaticMatrix< T, NO, NI > >` **type**

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

- `lipnet/include/lipnet/lipschitz/decompos.hpp`

### 3.27 `std::detail::detector< Default, AlwaysVoid, Op, Args >` Struct Template Reference

#### Public Types

- using **value\_t** = `std::false_type`
- using **type** = `Default`

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

- `lipnet/include/lipnet/traits.hpp`

### 3.28 `std::detail::detector< Default, std::void_t< Op< Args... > >, Op, Args... >` Struct Template Reference

#### Public Types

- using **value\_t** = `std::true_type`
- using **type** = `Op< Args... >`

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

- `lipnet/include/lipnet/traits.hpp`



## 3.29 `lipnet::equation_system_t< V1, V2 >` Struct Template Reference

The `equation_system_t` struct. Just a interface for all possible types. Solve a system of equations.

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

### 3.29.1 Detailed Description

```
template<typename V1, typename V2>
struct lipnet::equation_system_t< V1, V2 >
```

The `equation_system_t` struct. Just a interface for all possible types. Solve a system of equations.

$$Ax = b$$

#### Template Parameters

<i>V1</i>	tensor type of first argument
<i>V2</i>	tensot type of second argument

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

- `lipnet/include/lipnet/variable.hpp`

## 3.30 `lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >` Struct Template Reference

The `equation_system_t` struct for `blaze::StaticMatrix`.

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

### Static Public Member Functions

- static auto `solve` (const `blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >` &A, const `blaze::StaticMatrix< T, N3, N4, blaze::rowMajor >` &B)

*The solve method. Solve system of equations.  $AX = A$ .*

### 3.30.1 Detailed Description

```
template<typename T, size_t N1, size_t N2, size_t N3, size_t N4>
struct lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >
```

The `equation_system_t` struct for `blaze::StaticMatrix`.

## Template Parameters

<i>T</i>	numerical value type
<i>N1</i>	row dimension of first argument
<i>N2</i>	column dimension of first argument
<i>N3</i>	row dimension of second argument
<i>N4</i>	column dimension of second argument

## See also

[lipnet::equation\\_system\\_t](#) [6]

## 3.30.2 Member Function Documentation

## 3.30.2.1 solve()

```
template<typename T , size_t N1, size_t N2, size_t N3, size_t N4>
static auto lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >,
blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >::solve (
    const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > & A,
    const blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > & B ) [inline], [static]
```

The solve method. Solve system of equations.  $AX = A$ .

## Parameters

<i>A</i>	matrix <i>A</i> (first argument)
<i>B</i>	matrix <i>A</i> (second argument)

## Returns

matrix *X*

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

- lipnet/include/lipnet/tensor.hpp

### 3.31 lipnet::equation\_system\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticVector< T, N3, blaze::columnVector > > Struct Template Reference

The [equation\\_system\\_t](#) struct for blaze::StaticMatrix and blaze::StaticVector.

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

## Static Public Member Functions

- static auto [solve](#) (const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > &A, const blaze::StaticVector< T, N3, blaze::columnVector > &B)

*The solve method. Solve system of equations.  $Ax = b$ .*

### 3.31.1 Detailed Description

```
template<typename T, size_t N1, size_t N2, size_t N3>
struct lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticVector< T, N3, blaze::columnVector > >
```

The [equation\\_system\\_t](#) struct for blaze::StaticMatrix and blaze::StaticVector.

#### Template Parameters

<i>T</i>	numerical value type
<i>N1</i>	row dimension of first argument
<i>N2</i>	column dimension of first argument
<i>N3</i>	dimension of second argument

See also

[lipnet::equation\\_system\\_t](#) [6]

### 3.31.2 Member Function Documentation

#### 3.31.2.1 solve()

```
template<typename T , size_t N1, size_t N2, size_t N3>
static auto lipnet::equation_system_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >,
blaze::StaticVector< T, N3, blaze::columnVector > >::solve (
    const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > & A,
    const blaze::StaticVector< T, N3, blaze::columnVector > & B ) [inline], [static]
```

The solve method. Solve system of equations.  $Ax = b$ .

#### Parameters

<i>A</i>	matrix <i>A</i> (first argument)
<i>B</i>	vector <i>b</i> (second argument)

#### Returns

vector *x*

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

- `lipnet/include/lipnet/tensor.hpp`

### 3.32 `lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >` Struct Template Reference

gradient descent algorithm.

```
#include <fast_gradient_descent.hpp>
```

#### Classes

- struct [parameter\\_t](#)
- struct [statistics\\_t](#)  
*problem specific implementation of [statistics\\_t](#)*

#### Public Member Functions

- void **unpack** (std::tuple< GRAD, T > &&t, GRAD &dx, T &fx) const
- [fast\\_gradient\\_descent\\_t\\_impl](#) ([parameter\\_t](#) &&param=[parameter\\_t](#){0.001, 1e-8})  
*Default constructor.*
- template<bool stats\_enabled = false>  
std::tuple< VAR, T > **run** (P &prob, VAR &&x, typename std::conditional< stats\_enabled, [statistics\\_t](#), [std::void\\_type](#) >::type &stats) const  
*The run method. Implementation of the optimisation algorithm.*

#### Public Attributes

- [parameter\\_t](#) param  
*variables to optimize*

#### 3.32.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>
struct lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >
```

gradient descent algorithm.

##### Template Parameters

<i>T</i>	numerical value type
<i>P</i>	problem type
<i>VAR</i>	variable type
<i>GRAD</i>	gradient type

## 3.32.2 Constructor & Destructor Documentation

### 3.32.2.1 fast\_gradient\_descent\_t\_impl()

```
template<typename T , typename P , typename VAR , typename GRAD >
lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >::fast_gradient_descent_t_impl (
    parameter_t && param = parameter_t{0.001, 1e-8} ) [inline], [explicit]
```

Default constructor.

Parameters

<i>hyperparameter</i>	of optimisation. Init hyperparameters with 0.001, 1e-8
-----------------------	--

## 3.32.3 Member Function Documentation

### 3.32.3.1 run()

```
template<typename T , typename P , typename VAR , typename GRAD >
template<bool stats_enabled = false>
std::tuple<VAR,T> lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >::run (
    P & prob,
    VAR && x,
    typename std::conditional< stats_enabled, statistics_t, std::void_type >::type &
    stats ) const [inline]
```

The run method. Implementation of the optimisation algorithm.

Template Parameters

<i>stats_enabled</i>	enable/disable logging
----------------------	------------------------

Parameters

<i>prob</i>	problem
<i>x</i>	start variable / inital variable / start point
<i>stats</i>	statistics holder

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

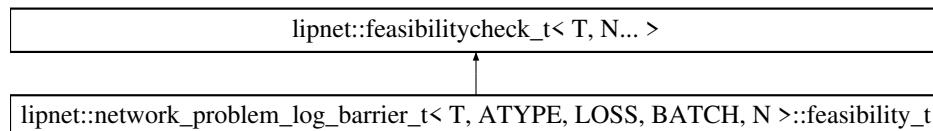
- lipnet/include/lipnet/optimizer/fast\_gradient\_descent.hpp

### 3.33 lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >::feasibility\_t Struct Reference

The [feasibility\\_t](#) struct. Implementation of feasibility check for this problem.

```
#include <nn_problem_liptrain_barrier.hpp>
```

Inheritance diagram for lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >::feasibility\_t:



#### Public Member Functions

- void **init** (const T r, const [variable\\_t](#) &p)
- void **run** (const [variable\\_t](#) &dir)

#### Public Attributes

- [variable\\_t](#) pos
- T step
- T rho

#### Additional Inherited Members

##### 3.33.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t
```

The [feasibility\\_t](#) struct. Implementation of feasibility check for this problem.

See also

[lipnet::feasibilitycheck\\_t](#)

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

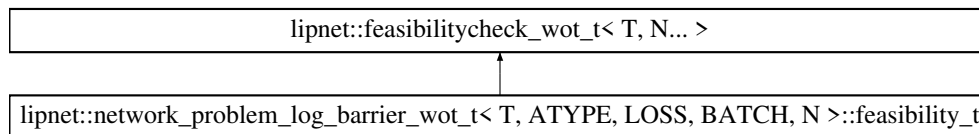
- lipnet/include/lipnet/problem/nn\_problem\_liptrain\_barrier.hpp

### 3.34 lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >::feasibility\_t Struct Reference

The [feasibility\\_t](#) struct. Implementation of feasibility check for this problem.

```
#include <nn_problem_liptrain_barrier_wot.hpp>
```

Inheritance diagram for lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >::feasibility\_t:



#### Public Member Functions

- void **init** (typename [self\\_barrier\\_t::cholesky\\_t](#) &&l, const typename self\_barrier\_t::tparam\_t &t)
- void **run** (const variable\_t &dir)

#### Public Attributes

- [self\\_barrier\\_t::cholesky\\_t](#) **L**
- T **step**
- std::optional< std::reference\_wrapper< const typename self\_barrier\_t::tparam\_t > > **Tparam**

#### Additional Inherited Members

##### 3.34.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::feasibility_t
```

The [feasibility\\_t](#) struct. Implementation of feasibility check for this problem.

See also

[lipnet::feasibilitycheck\\_wot\\_t](#)

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

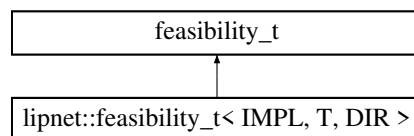
- lipnet/include/lipnet/problem/nn\_problem\_liptrain\_barrier\_wot.hpp

### 3.35 lipnet::feasibility\_t< IMPL, T, DIR > Struct Template Reference

The `feasibility_t` struct. base feasibility struct (basically a placeholder class)

```
#include <problem.hpp>
```

Inheritance diagram for `lipnet::feasibility_t< IMPL, T, DIR >`:



#### Public Member Functions

- `T operator() () const`  
*compute max stepsize for problem specific constraint.*
- `void operator<< (const DIR &dir)`  
*set direction for evaluation.*

#### 3.35.1 Detailed Description

```
template<typename IMPL, typename T, typename DIR>
struct lipnet::feasibility_t< IMPL, T, DIR >
```

The `feasibility_t` struct. base feasibility struct (basically a placeholder class)

##### Template Parameters

<i>IMPL</i>	problem type
<i>T</i>	numerical value type
<i>DIR</i>	variable type

#### 3.35.2 Member Function Documentation

##### 3.35.2.1 operator()()

```
template<typename IMPL , typename T , typename DIR >
T lipnet::feasibility_t< IMPL, T, DIR >::operator() () const [inline]
```

compute max stepsize for problem specific constraint.

$$\hat{\alpha} = \max_{\alpha} \alpha \quad \text{s.t.} \quad [x_k - \alpha \Delta x] \text{ is feasible}$$



## 3.35.2.2 operator&lt;&lt;()

```
template<typename IMPL , typename T , typename DIR >
void lipnet::feasibility_t< IMPL, T, DIR >::operator<< (
    const DIR & dir ) [inline]
```

set direction for evaluation.

## Parameters

<i>dir</i>	direction
	$\Delta x \quad x_{k+1} = x_k - \alpha \Delta x$

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

- lipnet/include/lipnet/problem.hpp

## 3.36 lipnet::feasibilitycheck\_t&lt; T, N &gt; Struct Template Reference

## Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef blaze::IdentityMatrix< T > **eye**
- typedef **cholesky\_topology**< T, N... >::type **cholesky\_t**
- typedef **inverse\_topology**< T, N... >::type **inverse\_t**
- typedef **network\_topology**< T, N... >::type **weight\_t**
- typedef **parameter\_tparam**< T, N... >::type **tparam\_t**
- typedef **liptrainweights\_t**< T, N... > **variable\_t**
- typedef std::integral\_constant< size\_t,(N+...) > **NN**
- typedef std::integral\_constant< size\_t, sizeof...(N) -1 > **L**

## Public Member Functions

- template<typename kondition = std::ratio<2,1>, typename = typename std::enable\_if<kondition::den != 0>::type>  
T **compute** (const **variable\_t** &pos, const **variable\_t** &gradient, const T rho) const

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

- lipnet/include/lipnet/lipschitz/feasibility.hpp

### 3.37 `lipnet::feasibilitycheck_wot_t< T, N >` Struct Template Reference

#### Public Types

- `template<size_t NN>`  
using **vector\_t** = `blaze::StaticVector< T, NN, blaze::columnVector >`
- `template<size_t NN1, size_t NN2>`  
using **matrix\_t** = `blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >`
- `typedef blaze::IdentityMatrix< T >` **eye**
- `typedef cholesky_topology< T, N... >::type` **cholesky\_t**
- `typedef inverse_topology< T, N... >::type` **inverse\_t**
- `typedef network_topology< T, N... >::type` **variable\_t**
- `typedef parameter_tparam< T, N... >::type` **tparam\_t**
- `typedef std::integral_constant< size_t,(N+...) >` **NN**
- `typedef std::integral_constant< size_t, sizeof...(N) -1 >` **L**

#### Public Member Functions

- `T compute` (`const tparam_t &tparam`, `const cholesky_t &var`, `const variable_t &gradient`) `const`

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

- `lipnet/include/lipnet/lipschitz/feasibility.hpp`

### 3.38 `lipnet::function_t< V >` Struct Template Reference

The `function_t` struct. Just a interface for all possible types. Apply function to tensor elementwise.

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

#### 3.38.1 Detailed Description

```
template<typename V>
struct lipnet::function_t< V >
```

The `function_t` struct. Just a interface for all possible types. Apply function to tensor elementwise.

##### Template Parameters

<code>V</code>	tensor type of argument
----------------	-------------------------

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

- `lipnet/include/lipnet/variable.hpp`

### 3.39 lipnet::function\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > Struct Template Reference

The [function\\_t](#) struct for blaze::StaticMatrix.

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

#### Static Public Member Functions

- static auto [trans](#) (const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > &m)  
*transpose matrix  $M^T$*

#### 3.39.1 Detailed Description

```
template<typename T, size_t N1, size_t N2>
struct lipnet::function_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >
```

The [function\\_t](#) struct for blaze::StaticMatrix.

##### Template Parameters

<i>T</i>	numerical value type
<i>N1</i>	row dimension of argument
<i>N2</i>	column dimension of argument

See also

[lipnet::function\\_t \[6\]](#)

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

- lipnet/include/lipnet/tensor.hpp

### 3.40 lipnet::function\_t< blaze::StaticVector< T, N, blaze::columnVector > > Struct Template Reference

The [function\\_t](#) struct for blaze::StaticVector.

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

#### Static Public Member Functions

- static auto [trans](#) (const blaze::StaticVector< T, N, blaze::columnVector > &vec)  
*transpose vector  $v^T$*
- static auto [square](#) (const blaze::StaticVector< T, N, blaze::columnVector > &vec)  
*square vector elementwise*
- static auto [sqrt](#) (const blaze::StaticVector< T, N, blaze::columnVector > &vec)  
*take square root of vector elementwise*

### 3.40.1 Detailed Description

```
template<typename T, size_t N>
struct lipnet::function_t< blaze::StaticVector< T, N, blaze::columnVector > >
```

The [function\\_t](#) struct for blaze::StaticVector.

#### Template Parameters

<i>T</i>	numerical value type
<i>N</i>	dimension of argument

See also

[lipnet::function\\_t \[6\]](#)

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

- [lipnet/include/lipnet/tensor.hpp](#)

## 3.41 lipnet::function\_t< layer\_t< T, I, O > > Struct Template Reference

### Static Public Member Functions

- static auto **square** (const [layer\\_t](#)< T, I, O > &m)
- static auto **sqrt** (const [layer\\_t](#)< T, I, O > &m)

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

- [lipnet/include/lipnet/network/layer.hpp](#)

## 3.42 lipnet::function\_t< liptrainweights\_t< T, N... > > Struct Template Reference

### Static Public Member Functions

- static auto **square** (const [liptrainweights\\_t](#)< T, N... > &m)
- static auto **sqrt** (const [liptrainweights\\_t](#)< T, N... > &m)

### Public Attributes

- decltype([liptrainweights\\_t](#)< T, N... >::W) typedef **arg1\_t**
- decltype([liptrainweights\\_t](#)< T, N... >::t) typedef **arg2\_t**

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

- [lipnet/include/lipnet/lipschitz/barrier.hpp](#)

### 3.43 lipnet::function\_t< std::tuple< ARGS... > > Struct Template Reference

#### Static Public Member Functions

- template<size\_t ... INTS>  
static auto **square\_impl** (const std::tuple< ARGS... > &m, std::integer\_sequence< size\_t, INTS... >)
- static auto **square** (const std::tuple< ARGS... > &m)
- template<size\_t ... INTS>  
static auto **sqrt\_impl** (const std::tuple< ARGS... > &m, std::integer\_sequence< size\_t, INTS... >)
- static auto **sqrt** (const std::tuple< ARGS... > &m)

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

- lipnet/include/lipnet/tuple.hpp

### 3.44 lipnet::generate\_batch\_data< T, B, N, NS > Struct Template Reference

helper struct for data

```
#include <topology.hpp>
```

#### Public Types

- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef [generate\\_batch\\_data](#)< T, B, NS... >::type **next**
- typedef [join\\_tuples](#)< std::tuple< matrix\_t< N, B > >, next >::type **type**

#### 3.44.1 Detailed Description

```
template<typename T, size_t B, size_t N, size_t ... NS>  
struct lipnet::generate_batch_data< T, B, N, NS >
```

helper struct for data

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

- lipnet/include/lipnet/network/topology.hpp

### 3.45 lipnet::generate\_batch\_data< T, B, N > Struct Template Reference

helper struct for data

```
#include <topology.hpp>
```

## Public Types

- `template<size_t NN1, size_t NN2>`  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- `typedef std::tuple< matrix_t< N, B > > type`

### 3.45.1 Detailed Description

```
template<typename T, size_t B, size_t N>
struct lipnet::generate_batch_data< T, B, N >
```

helper struct for data

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

- `lipnet/include/lipnet/network/topology.hpp`

## 3.46 lipnet::generate\_batch\_data\_remove\_first< T, B, N, NS > Struct Template Reference

helper struct for data

```
#include <topology.hpp>
```

## Public Types

- `typedef generate\_batch\_data< T, B, NS... >::type type`

### 3.46.1 Detailed Description

```
template<typename T, size_t B, size_t N, size_t ... NS>
struct lipnet::generate_batch_data_remove_first< T, B, N, NS >
```

helper struct for data

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

- `lipnet/include/lipnet/network/topology.hpp`

## 3.47 lipnet::generate\_data< T, N, NS > Struct Template Reference

helper struct for data

```
#include <topology.hpp>
```

## Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- typedef generate\_data< T, NS... >::type **next**
- typedef join\_tuples< std::tuple< vector\_t< N > >, next >::type **type**

### 3.47.1 Detailed Description

```
template<typename T, size_t N, size_t ... NS>
struct lipnet::generate_data< T, N, NS >
```

helper struct for data

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

- lipnet/include/lipnet/network/topology.hpp

## 3.48 lipnet::generate\_data< T, N > Struct Template Reference

helper struct for data

```
#include <topology.hpp>
```

## Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- typedef std::tuple< vector\_t< N > > **type**

### 3.48.1 Detailed Description

```
template<typename T, size_t N>
struct lipnet::generate_data< T, N >
```

helper struct for data

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

- lipnet/include/lipnet/network/topology.hpp

## 3.49 lipnet::generate\_data\_remove\_first< T, N, NS > Struct Template Reference

helper struct for data

```
#include <topology.hpp>
```

## Public Types

- typedef [generate\\_data](#)< T, NS..., 0 >::type **type**

### 3.49.1 Detailed Description

```
template<typename T, size_t N, size_t ... NS>
struct lipnet::generate_data_remove_first< T, N, NS >
```

helper struct for data

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

- lipnet/include/lipnet/network/topology.hpp

## 3.50 lipnet::generator\_t< V > Struct Template Reference

The [generator\\_t](#) struct. Just a interface for all possible types. Instantiate tensor of type V.

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

### 3.50.1 Detailed Description

```
template<typename V>
struct lipnet::generator_t< V >
```

The [generator\\_t](#) struct. Just a interface for all possible types. Instantiate tensor of type V.

Template Parameters

V	tensor type to create
---	-----------------------

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

- lipnet/include/lipnet/variable.hpp

## 3.51 lipnet::generator\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > Struct Template Reference

The [generator\\_t](#) struct for blaze::StaticMatrix.

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



## Static Public Member Functions

- static auto [make](#) (const T &val)  
*uniform distribution constructor  $\sim \mathcal{U}(-val, val)$*
- static auto [unifrom](#) (const T &val)  
*uniform distribution constructor  $\sim \mathcal{U}(-val, val)$*
- static auto [identity](#) ()  
*identity constructor  $I$*

### 3.51.1 Detailed Description

```
template<typename T, size_t N1, size_t N2>
struct lipnet::generator_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >
```

The [generator\\_t](#) struct for blaze::StaticMatrix.

Template Parameters

<i>T</i>	numerical value type
<i>N1</i>	row dimension of return matrix
<i>N2</i>	column dimension of return matrix

See also

[lipnet::generator\\_t](#) [6]

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

- lipnet/include/lipnet/tensor.hpp

## 3.52 lipnet::generator\_t< blaze::StaticVector< T, N, blaze::columnVector > > Struct Template Reference

The [generator\\_t](#) struct for blaze::StaticVector.

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

## Static Public Member Functions

- static auto [make](#) (const T &val)  
*uniform distribution constructor  $\sim \mathcal{U}(-val, val)$*
- static auto [unifrom](#) (const T &val)  
*uniform distribution constructor  $\sim \mathcal{U}(-val, val)$*

### 3.52.1 Detailed Description

```
template<typename T, size_t N>
struct lipnet::generator_t< blaze::StaticVector< T, N, blaze::columnVector > >
```

The [generator\\_t](#) struct for blaze::StaticVector.

## Template Parameters

$T$	numerical value type
$N$	dimension of return vector

## See also

[lipnet::generator\\_t \[6\]](#)

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

- `lipnet/include/lipnet/tensor.hpp`

### 3.53 `lipnet::generator_t< layer_t< T, I, O > >` Struct Template Reference

#### Static Public Member Functions

- static `layer_t< T, I, O >` **make** (T val)

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

- `lipnet/include/lipnet/network/layer.hpp`

### 3.54 `lipnet::generator_t< liptrainweights_t< T, N... > >` Struct Template Reference

#### Static Public Member Functions

- static `liptrainweights_t< T, N... >` **make** (T val, T uni)

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

- `lipnet/include/lipnet/lipschitz/barrier.hpp`

### 3.55 `lipnet::generator_t< parameter_decompo_t< T, N... > >` Struct Template Reference

#### Static Public Member Functions

- static auto **make** (const T &init)

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

- `lipnet/include/lipnet/lipschitz/decompos.hpp`

## 3.56 lipnet::generator\_t< std::tuple< ARGS... > > Struct Template Reference

[generator\\_t](#) implementation for std::tuple

```
#include <topology.hpp>
```

### Static Public Member Functions

- `template<typename T >`  
`static std::tuple< ARGS... > make (T val)`

#### 3.56.1 Detailed Description

```
template<typename ... ARGS>
struct lipnet::generator_t< std::tuple< ARGS... > >
```

[generator\\_t](#) implementation for std::tuple

See also

[lipnet::generator\\_t](#)

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

- `lipnet/include/lipnet/network/topology.hpp`

## 3.57 lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD > Struct Template Reference

projected gradient descent algorithm.

```
#include <gradient_descent_projected.hpp>
```

### Classes

- struct [parameter\\_t](#)
- struct [statistics\\_t](#)  
*problem specific implementation of [statistics\\_t](#)*

## Public Member Functions

- void **unpack** (std::tuple< GRAD, T > &&t, GRAD &dx, T &fx) const
- auto **project** (const P &prob, VAR &&var) const  
*The project method. Call projection method of problem.*
- **gradient\_descent\_projected\_t\_impl** (parameter\_t &&param=parameter\_t{(size\_t) 5e5, 1e-6, 0.001, 1e-8})  
*Default constructor.*
- template<bool stats\_enabled = false>  
std::tuple< VAR, T > **run** (P &prob, VAR &&x, typename std::conditional< stats\_enabled, statistics\_t, std::void\_type >::type &stats) const  
*The run method. Implementation of the optimisation algorithm.*

## Public Attributes

- **parameter\_t** param  
*variables to optimize*

### 3.57.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>
struct lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >
```

projected gradient descent algorithm.

#### Template Parameters

<i>T</i>	numerical value type
<i>P</i>	problem type
<i>VAR</i>	variable type
<i>GRAD</i>	gradient type

### 3.57.2 Constructor & Destructor Documentation

#### 3.57.2.1 gradient\_descent\_projected\_t\_impl()

```
template<typename T , typename P , typename VAR , typename GRAD >
lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >::gradient_descent_projected_t_impl
(
    parameter_t && param = parameter_t{ (size_t) 5e5, 1e-6, 0.001, 1e-8} ) [inline],
[explicit]
```

Default constructor.

## Parameters

<i>hyperparameter</i>	of optimisation. Init hyperparameters with (size_t) 5e3, 1e-6, 0.001, 1e-8
-----------------------	--

### 3.57.3 Member Function Documentation

#### 3.57.3.1 project()

```
template<typename T , typename P , typename VAR , typename GRAD >
auto lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >::project (
    const P & prob,
    VAR && var ) const [inline]
```

The project method. Call projection method of problem.

## Parameters

<i>prob</i>	problem
<i>var</i>	current variables; will be projected to feasible set

#### 3.57.3.2 run()

```
template<typename T , typename P , typename VAR , typename GRAD >
template<bool stats_enabled = false>
std::tuple<VAR,T> lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >::run (
    P & prob,
    VAR && x,
    typename std::conditional< stats_enabled, statistics_t, std::void_type >::type &
    stats ) const [inline]
```

The run method. Implementation of the optimisation algorithm.

## Template Parameters

<i>stats_enabled</i>	enable/disable logging
----------------------	------------------------

## Parameters

<i>prob</i>	problem
<i>x</i>	start variable / initial variable / start point
<i>stats</i>	statistics holder

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

- `lipnet/include/lipnet/optimizer/gradient_descent_projected.hpp`

## 3.58 `lipnet::helper_function_t< V >` Struct Template Reference

### Static Public Attributes

- constexpr static bool `v1`
- constexpr static bool `v2`
- constexpr static bool `value` = `v1 && v2`

### 3.58.1 Member Data Documentation

#### 3.58.1.1 `v1`

```
template<typename V >
constexpr static bool lipnet::helper_function_t< V >::v1 [inline], [static], [constexpr]
```

##### Initial value:

```
= std::is_invocable_r<V,
    decltype(&function_t<V>::square), const V&::value
```

#### 3.58.1.2 `v2`

```
template<typename V >
constexpr static bool lipnet::helper_function_t< V >::v2 [inline], [static], [constexpr]
```

##### Initial value:

```
= std::is_invocable_r<V,
    decltype(&function_t<V>::sqrt), const V&::value
```

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

- `lipnet/include/lipnet/variable.hpp`

## 3.59 `lipnet::helper_inner_t< T, V1, V2 >` Struct Template Reference

### Static Public Attributes

- constexpr static bool `value`

### 3.59.1 Member Data Documentation

### 3.59.1.1 value

```
template<typename T , typename V1 , typename V2 >
constexpr static bool lipnet::helper_inner_t< T, V1, V2 >::value [inline], [static], [constexpr]
```

#### Initial value:

```
= std::is_invocable_r<T,
    decltype(&prod_t<T,V1,V2>::inner), const V1&, const V2&>::value
```

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

- lipnet/include/lipnet/variable.hpp

## 3.60 lipnet::helper\_norm\_t< T, V > Struct Template Reference

### Static Public Attributes

- constexpr static bool **value**

### 3.60.1 Member Data Documentation

#### 3.60.1.1 value

```
template<typename T , typename V >
constexpr static bool lipnet::helper_norm_t< T, V >::value [inline], [static], [constexpr]
```

#### Initial value:

```
= std::is_invocable_r<T,
    decltype(&norm_t<T,V>::norm), const V&>::value
```

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

- lipnet/include/lipnet/variable.hpp

## 3.61 image\_t Struct Reference

### Public Types

- typedef blaze::StaticVector< double, 3, blaze::columnVector > **pixel\_t**

### Public Member Functions

- **image\_t** (size\_t w, size\_t h)
- pixel\_t & **operator()** (const size\_t &x, const size\_t &y)

## Public Attributes

- `size_t width`
- `size_t height`
- `std::vector< pixel_t > data`

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

- `lipnet/src/plotting_objectivsurface.cpp`

## 3.62 `lipnet::inverse_diagentry< T, N, NARGS >` Struct Template Reference

### Public Types

- typedef `inverse_diagentry_impl< T, N, NARGS... >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/topology.hpp`

## 3.63 `lipnet::inverse_diagentry_impl< T, N, NS >` Struct Template Reference

### Public Types

- typedef `inverse_diagentry_impl< T, NS... >::type` **next**
- typedef `join_tuples< std::tuple< blaze::SymmetricMatrix< blaze::StaticMatrix< T, N, N > > >, next >::type` **type**

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

- `lipnet/include/lipnet/lipschitz/topology.hpp`

## 3.64 `lipnet::inverse_diagentry_impl< T, N >` Struct Template Reference

### Public Types

- typedef `std::tuple< blaze::SymmetricMatrix< blaze::StaticMatrix< T, N, N > > >` **type**

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

- `lipnet/include/lipnet/lipschitz/topology.hpp`



### 3.65 lipnet::inverse\_subentry< T, NI, NO, RE, NARGS > Struct Template Reference

#### Public Types

- typedef [inverse\\_subentry\\_impl](#)< T, NI, NO, RE, NARGS... >::type **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

### 3.66 lipnet::inverse\_subentry\_impl< T, NI, NO, NS > Struct Template Reference

#### Public Types

- typedef [inverse\\_subentry\\_impl](#)< T, NO, NS... >::type **next**
- typedef [join\\_tuples](#)< std::tuple< blaze::StaticMatrix< T, NO, NI > >, next >::type **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

### 3.67 lipnet::inverse\_subentry\_impl< T, NI, NO > Struct Template Reference

#### Public Types

- typedef std::tuple< blaze::StaticMatrix< T, NO, NI > > **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

### 3.68 lipnet::inverse\_topology< T, N > Struct Template Reference

#### Classes

- struct [type](#)

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

- lipnet/include/lipnet/lipschitz/topology.hpp

### 3.69 lipnet::join\_tuples< typename, typename > Struct Template Reference

Helper struct to join two tuples. (std::tuple)

```
#include <tuple.hpp>
```

#### 3.69.1 Detailed Description

```
template<typename, typename>
struct lipnet::join_tuples< typename, typename >
```

Helper struct to join two tuples. (std::tuple)

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

- lipnet/include/lipnet/tuple.hpp

### 3.70 lipnet::join\_tuples< std::tuple< NEW... >, std::tuple< NEXT... > > Struct Template Reference

Implementation of [join\\_tuples](#) struct to join two tuples. (std::tuple)

```
#include <tuple.hpp>
```

#### Public Types

- typedef std::tuple< NEW..., NEXT... > **type**

#### 3.70.1 Detailed Description

```
template<typename... NEW, typename... NEXT>
struct lipnet::join_tuples< std::tuple< NEW... >, std::tuple< NEXT... > >
```

Implementation of [join\\_tuples](#) struct to join two tuples. (std::tuple)

See also

[lipnet::join\\_tuples](#)

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

- lipnet/include/lipnet/tuple.hpp

## 3.71 lipnet::layer\_t< T, I, O > Struct Template Reference

The [layer\\_t](#) struct; the layer implementation of each layer; contains the weight and the biases.

```
#include <layer.hpp>
```

### Public Types

- typedef std::array< T, I \* O > **weight\_array\_t**
- typedef std::array< T, O > **bias\_array\_t**
- typedef blaze::StaticMatrix< T, O, I, blaze::columnMajor > **MT**
- typedef blaze::StaticVector< T, O, blaze::columnVector > **VT**

### Public Member Functions

- **layer\_t** (MT &&w, VT &&b)
- [layer\\_t](#) (const T &var)  
*The [layer\\_t](#) constructor; initialize weight and bias with random values.*
- template<class Archive >  
void [serialize](#) (Archive &ar)  
*serialize [layer\\_t](#)*

### Public Attributes

- MT **weight**
- VT **bias**

#### 3.71.1 Detailed Description

```
template<typename T, size_t I, size_t O>
struct lipnet::layer_t< T, I, O >
```

The [layer\\_t](#) struct; the layer implementation of each layer; contains the weight and the biases.

#### Template Parameters

<i>T</i>	numerical value type
<i>I</i>	input dimension
<i>O</i>	output dimension

#### 3.71.2 Constructor & Destructor Documentation

### 3.71.2.1 layer\_t()

```
template<typename T , size_t I, size_t O>
lipnet::layer_t< T, I, O >::layer_t (
    const T & var ) [inline], [explicit]
```

The `layer_t` constructor; initialize weight and bias with random values.

#### Parameters

<code>var</code>	some kind of variance
------------------	-----------------------

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

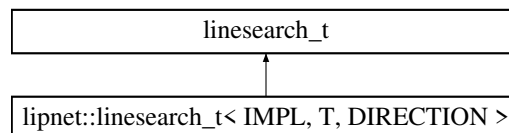
- `lipnet/include/lipnet/network/layer.hpp`

## 3.72 lipnet::linesearch\_t< IMPL, T, DIRECTION > Struct Template Reference

The `linesearch_t` struct. base linesearch struct (basically a placeholder class)

```
#include <problem.hpp>
```

Inheritance diagram for `lipnet::linesearch_t< IMPL, T, DIRECTION >`:



### Public Member Functions

- `T operator()` (const T val) const  
*evaluate function with stepsize val.*
- `void operator<<` (const DIRECTION &dir)  
*set direction for evaluation.*

### 3.72.1 Detailed Description

```
template<typename IMPL, typename T, typename DIRECTION>
struct lipnet::linesearch_t< IMPL, T, DIRECTION >
```

The `linesearch_t` struct. base linesearch struct (basically a placeholder class)

## Template Parameters

<i>IMPL</i>	problem type
<i>T</i>	numerical value type
<i>DIRECTION</i>	variable type

## 3.72.2 Member Function Documentation

## 3.72.2.1 operator&gt;()

```
template<typename IMPL , typename T , typename DIRECTION >
T lipnet::linesearch_t< IMPL, T, DIRECTION >::operator() (
    const T val ) const [inline]
```

evaluate function with stepsize val.

## Parameters

<i>val</i>	stepsize $\alpha \quad x_{k+1} = x_k - \alpha \Delta x$
------------	--

## 3.72.2.2 operator&lt;&lt;()

```
template<typename IMPL , typename T , typename DIRECTION >
void lipnet::linesearch_t< IMPL, T, DIRECTION >::operator<< (
    const DIRECTION & dir ) [inline]
```

set direction for evaluation.

## Parameters

<i>dir</i>	direction $\Delta x \quad x_{k+1} = x_k - \alpha \Delta x$
------------	---

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

- lipnet/include/lipnet/problem.hpp

### 3.73 `lipnet::lipcalc_parameter_t< T, N >` Struct Template Reference

#### Public Attributes

- `T rho`
- `blaze::StaticVector< T, sum< sizeof...(N) - 1, N... > - at< 0, N... > > tmat`

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

- `lipnet/include/lipnet/lipschitz/parameter.hpp`

### 3.74 `lipnet::liptrainweights_t< T, N >` Struct Template Reference

#### Public Member Functions

- `template<class Archive >`  
`void save (Archive &ar) const`
- `template<class Archive >`  
`void load (Archive &ar)`

#### Public Attributes

- `network_topology< T, N... >::type W`
- `parameter_tparam< T, N... >::type t`

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

- `lipnet/include/lipnet/lipschitz/barrier.hpp`

### 3.75 `lipnet::loader_t< T >` Struct Template Reference

struct for loading matrix from csv file;

```
#include <loader.hpp>
```

#### Public Types

- `typedef blaze::DynamicMatrix< T, blaze::rowMajor > dmatrix_t`

#### Static Public Member Functions

- `static std::optional< dmatrix_t > load (const std::string &path)`  
*load matrix from csv file;*

#### 3.75.1 Detailed Description

```
template<typename T>
struct lipnet::loader_t< T >
```

struct for loading matrix from csv file;

## Template Parameters

<i>T</i>	numerical value type [7]
----------	--------------------------

## 3.75.2 Member Function Documentation

## 3.75.2.1 load()

```
template<typename T >
static std::optional<dmatrix_t> lipnet::loader_t< T >::load (
    const std::string & path ) [inline], [static]
```

load matrix from csv file;

## Parameters

<i>path</i>	path to file on filesystem
-------------	----------------------------

## Returns

matrix

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

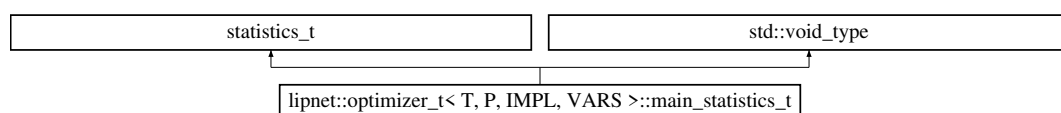
- lipnet/include/lipnet/loader/loader.hpp

## 3.76 lipnet::optimizer\_t&lt; T, P, IMPL, VARS &gt;::main\_statistics\_t Struct Reference

The [main\\_statistics\\_t](#) struct.

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

Inheritance diagram for lipnet::optimizer\_t< T, P, IMPL, VARS >::main\_statistics\_t:



## Public Member Functions

- template<class Archive >  
void **serialize** (Archive &archive)

## Public Attributes

- `std::chrono::milliseconds` **duration**

### 3.76.1 Detailed Description

```
template<typename T, typename P, typename IMPL, typename ... VARS>
struct lipnet::optimizer_t< T, P, IMPL, VARS >::main_statistics_t
```

The [main\\_statistics\\_t](#) struct.

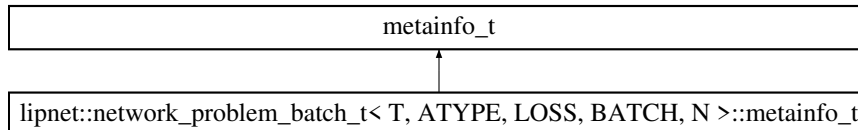
Just contains a variable to, which stores the computation time to solve the problem. The variable stores its value in milliseconds.

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

- `lipnet/include/lipnet/optimizer.hpp`

## 3.77 `lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t` Struct Reference

Inheritance diagram for `lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t`:

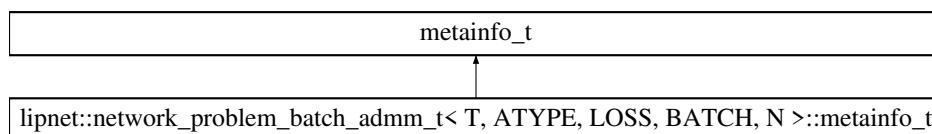


The documentation for this struct was generated from the following file:

- `lipnet/include/lipnet/problem/nn_problem_batch.hpp`

## 3.78 `lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t` Struct Reference

Inheritance diagram for `lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t`:



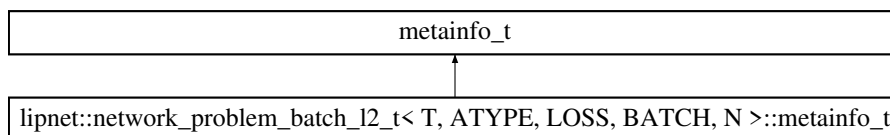
The documentation for this struct was generated from the following file:

- `lipnet/include/lipnet/problem/nn_problem_batch_admm.hpp`



### 3.79 lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t Struct Reference

Inheritance diagram for lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t:

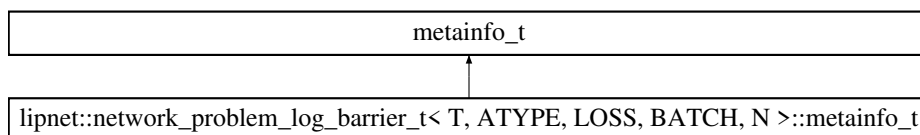


The documentation for this struct was generated from the following file:

- lipnet/include/lipnet/problem/nn\_problem\_batch\_l2.hpp

### 3.80 lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t Struct Reference

Inheritance diagram for lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t:

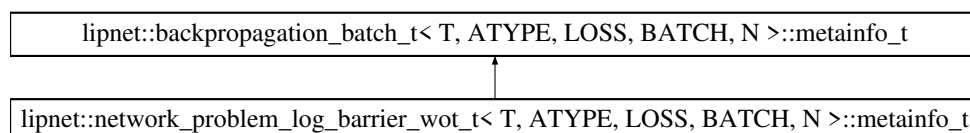


The documentation for this struct was generated from the following file:

- lipnet/include/lipnet/problem/nn\_problem\_liptrain\_barrier.hpp

### 3.81 lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t Struct Reference

Inheritance diagram for lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t:



## Public Attributes

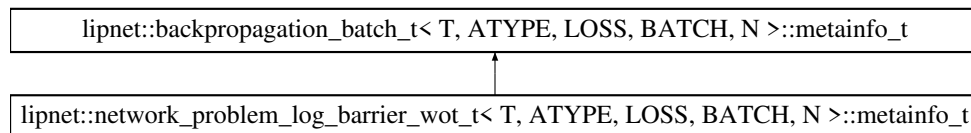
- `size_t iter`

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

- `lipnet/include/lipnet/network/backpropagation.hpp`

### 3.82 `lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t` Struct Reference

Inheritance diagram for `lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t`:



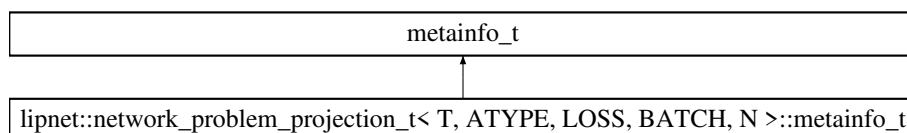
## Additional Inherited Members

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

- `lipnet/include/lipnet/problem/nn_problem_liptrain_barrier_wot.hpp`

### 3.83 `lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t` Struct Reference

Inheritance diagram for `lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >::metainfo_t`:



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

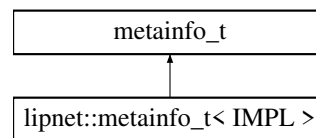
- `lipnet/include/lipnet/problem/nn_problem_liptrain_projection.hpp`

## 3.84 lipnet::metainfo\_t< IMPL > Struct Template Reference

The [metainfo\\_t](#) struct. Data holder type for data needed during the iterations.

```
#include <problem.hpp>
```

Inheritance diagram for lipnet::metainfo\_t< IMPL >:



### 3.84.1 Detailed Description

```
template<typename IMPL>
struct lipnet::metainfo_t< IMPL >
```

The [metainfo\\_t](#) struct. Data holder type for data needed during the iterations.

Template Parameters

<i>IMPL</i>	problem type
-------------	--------------

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

- lipnet/include/lipnet/problem.hpp

## 3.85 lipnet::mosek\_projection\_wot\_t< T, N > Struct Template Reference

The [mosek\\_projection\\_wot\\_t](#) struct. Compute the projection of the reference weights. It is conic program and will be solved with mosek (interior point method)

```
#include <mosek_projection_wot.hpp>
```

### Public Types

- `template<size_t NN>`  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- `template<size_t NN1, size_t NN2>`  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- `typedef std::integral_constant< size_t, sizeof...(N) - 1 > L`
- `typedef std::integral_constant< size_t, (N+...) > NL`
- `typedef std::integer_sequence< size_t, N... > DIMS`
- `typedef std::integral_constant< size_t, sum_from_to< 1, L::value, N... > > n`
- `typedef network\_t< T, identity\_activation\_t, N... >::layer\_t variable_t`

## Static Public Member Functions

- `template<size_t R, size_t C>`  
`static fusion::Matrix::t map (const matrix_t< R, C > &mat)`  
*map input argument to mosek parameter type*
- `static variable_t projection (const T lipschitz, variable_t &&ref, const T &tinitval)`  
*Compute projection of weights into feasible set.*

### 3.85.1 Detailed Description

```
template<typename T, size_t ... N>
struct lipnet::mosek_projection_wot_t< T, N >
```

The [mosek\\_projection\\_wot\\_t](#) struct. Compute the projection of the reference weights. It is conic program and will be solved with mosek (interior point method)

$$\begin{aligned} @f[ \arg \min_{W, \eta} \quad & \eta \quad \text{ s.t. } \quad \chi(\Psi^2, W) \leq 0 \quad \\ & \left[ \eta \ ; \ ; \ ; \ \mathrm{fl}(W - W_{\mathrm{ref}}) \right] \leq \mathcal{Q} \ 0 \ @f] \end{aligned}$$

#### Template Parameters

<i>T</i>	numerical value type
<i>N</i>	network topology

### 3.85.2 Member Function Documentation

#### 3.85.2.1 [projection\(\)](#)

```
template<typename T , size_t ... N>
static variable_t lipnet::mosek\_projection\_wot\_t< T, N >::projection (
    const T lipschitz,
    variable_t && ref,
    const T & tinitval ) [inline], [static]
```

Compute projection of weights into feasible set.

#### Parameters

<i>lipschitz</i>	lipschitz integral_constant
<i>ref</i>	reference weights; computed during gradient descent step
<i>tinitval</i>	hyperparameter T of chi matrix

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

- `lipnet/include/lipnet/extern/mosek_projection_wot.hpp`

## 3.86 lipnet::network\_data\_t< T, IN, OUT > Struct Template Reference

The [network\\_data\\_t](#) struct; training dataset.

```
#include <backpropagation.hpp>
```

### Public Attributes

- blaze::DynamicMatrix< T, blaze::rowMajor > **idata**
- blaze::DynamicMatrix< T, blaze::rowMajor > **tdata**

### 3.86.1 Detailed Description

```
template<typename T, size_t IN, size_t OUT>
struct lipnet::network_data_t< T, IN, OUT >
```

The [network\\_data\\_t](#) struct; training dataset.

#### Template Parameters

<i>T</i>	numerical value type
<i>IN</i>	input dimension
<i>OUT</i>	output dimension

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

- lipnet/include/lipnet/network/backpropagation.hpp

## 3.87 lipnet::network\_libcalc\_t< T, N > Struct Template Reference

```
#include <nn_libcalc.hpp>
```

### Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef std::integral\_constant< size\_t, sizeof...(N) -1 > **L**
- typedef std::integral\_constant< size\_t,(N+...) > **NL**
- typedef std::integer\_sequence< size\_t, N... > **DIMS**
- typedef [network\\_topology](#)< T, N... >::type **variable\_t**

### Static Public Member Functions

- static std::tuple< T, vector\_t< sum\_from\_to< 1, L::value, N... >> > [solve](#) (const variable\_t &var)  
*solve sdp; via mosek; via interior point method*

### 3.87.1 Detailed Description

```
template<typename T, size_t ... N>
struct lipnet::network_libcalc_t< T, N >
```

@breif calculate lipschitz constant of neural network via conic program (SDP)

$$\arg \min_{\Psi, T} \Psi^2 \quad \text{s.t.} \chi(\Psi^2, W) \succeq 0$$

#### Template Parameters

<i>T</i>	numerical value type
<i>N</i>	network topology <a href="#">[2]</a>

### 3.87.2 Member Function Documentation

#### 3.87.2.1 solve()

```
template<typename T , size_t ... N>
static std::tuple<T, vector_t<sum_from_to<1, L::value, N...>>> > lipnet::network_libcalc_t<
T, N >::solve (
    const variable_t & var ) [inline], [static]
```

solve sdp; via mosek; via interior point method

#### Parameters

<i>var</i>	network weights <a href="#">[2]</a>
------------	-------------------------------------

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

- lipnet/include/lipnet/extern/nn\_libcalc.hpp

## 3.88 lipnet::network\_libtrain\_enforcing\_t< T, N > Struct Template Reference

network\_libtrain\_enforcing\_; Implementaion of the second subproblem of the admm method

```
#include <nn_liptrain_enforcing.hpp>
```

## Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef std::integral\_constant< size\_t, sizeof...(N) -1 > **L**
- typedef std::integral\_constant< size\_t,(N+...) > **NL**
- typedef std::integer\_sequence< size\_t, N... > **DIMS**
- typedef std::integral\_constant< size\_t, sum\_from\_to< 1, L::value, N... > > **n**
- typedef [network\\_t](#)< T, [identity\\_activation\\_t](#), N... >::[layer\\_t](#) **variable\_t**

## Static Public Member Functions

- static variable\_t [train](#) (const T lipschitz, const T mu, const variable\_t &Rvar, const vector\_t< n::value > &SDT, const variable\_t &dual)  
*solve second admm subproblem*

### 3.88.1 Detailed Description

```
template<typename T, size_t ... N>
struct lipnet::network_libtrain_enforcing_t< T, N >
```

[network\\_libtrain\\_enforcing\\_](#); Implementaion of the second subproblem of the admm method

$$\arg \min_{\tilde{W}, \eta} \quad \text{tr}(Y(W - \tilde{W})) + \frac{v}{2}\eta \quad \text{s.t.} \quad \chi(\Psi^2, \tilde{W}) \succeq 0 \quad \left[ \eta \quad \text{fl}(W - \tilde{W}) \right] \succeq_{\mathcal{Q}} 0$$

#### Template Parameters

<i>T</i>	numerical value type
<i>N</i>	network topology

### 3.88.2 Member Function Documentation

#### 3.88.2.1 train()

```
template<typename T , size_t ... N>
static variable_t lipnet::network\_libtrain\_enforcing\_t< T, N >::train (
    const T lipschitz,
    const T mu,
    const variable_t & Rvar,
    const vector_t< n::value > & SDT,
    const variable_t & dual ) [inline], [static]
```

solve second admm subproblem

## Parameters

<i>lipschitz</i>	lipschitz constant
<i>mu</i>	admm hyperparameter; augmented lagrange multipliers
<i>Rvar</i>	reference weights $W$
<i>SDT</i>	hyperparameter $T$ of matrix $\chi(\Psi^2, W)$
<i>dual</i>	dual variable

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

- `lipnet/include/lipnet/extern/nn_liptrain_enforcing.hpp`

### 3.89 `lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >` Struct Template Reference

The `network_problem_batch_admm_t` struct. The problem implementation of admm neural network training in batches.

```
#include <nn_problem_batch_admm.hpp>
```

Inheritance diagram for `lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >`:



## Classes

- struct `metainfo_t`

## Public Types

- `typedef std::integral_constant< size_t, sizeof...(N) - 1 > L`
- `typedef std::integral_constant< size_t, (N+...) > NL`
- `typedef std::integer_sequence< size_t, N... > DIMS`
- `typedef backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N... > self_back_t`
- `typedef self_back_t::variable_t variable_t`

## Public Member Functions

- `network_problem_batch_admm_t (LOSS< T > &&l, network_data_t< T, at< 0, N... >(), at< L::value, N... >() > &&data, const T rho, const variable_t &dualvariable, const variable_t &weights_bar)`
- `std::tuple< variable_t, T > operator() (const variable_t &var, metainfo_t &info) const`

*The operator () function. compute gradient.*



## Public Attributes

- const variable\_t & [dualvariable](#)  
*dual variable*
- const variable\_t & [weights\\_bar](#)
- const T [rho](#)  
*admm hyperparameter*

### 3.89.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >
```

The [network\\_problem\\_batch\\_admm\\_t](#) struct. The problem implementation of admm neural network training in batches.

$$\nabla_{W,b}\mathcal{L}(f_{W,b}) + L_v(W, \tilde{W}, y)$$

#### Template Parameters

<i>T</i>	Base numeric type (eg. double, float, ...).
<i>ATYPE</i>	Activation type of this neural network.
<i>LOSS</i>	Objectiv function type of this neural network
<i>BATCH</i>	Const integer value specifying the batch size.
<i>N</i>	Neural network topology. Array of postive integer values specifying the number of neurons at each layer.

### 3.89.2 Member Function Documentation

#### 3.89.2.1 operator>()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >↵
::operator() (
    const variable_t & var,
    metainfo_t & info ) const [inline]
```

The operator () function. compute gradient.

#### Parameters

<i>var</i>	current position
------------	------------------

## Returns

gradient and loss at specified position

### 3.89.3 Member Data Documentation

#### 3.89.3.1 weights\_bar

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
const variable_t& lipnet::network_problem_batch_admm_t< T, ATYPE, LOSS, BATCH, N >::weights_bar
```

weights and biases variable x

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

- lipnet/include/lipnet/problem/nn\_problem\_batch\_admm.hpp

## 3.90 lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference

The [network\\_problem\\_batch\\_l2\\_t](#) struct. The problem implementation of l2 neural network training in batches.

```
#include <nn_problem_batch_l2.hpp>
```

Inheritance diagram for lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N >:



## Classes

- struct [metainfo\\_t](#)

## Public Types

- typedef [backpropagation\\_batch\\_t](#)< T, ATYPE, LOSS, BATCH, N... > **self\_back\_t**
- typedef self\_back\_t::variable\_t **variable\_t**

## Public Member Functions

- [network\\_problem\\_batch\\_l2\\_t](#) (LOSS< T > &&l, [network\\_data\\_t](#)< T, at< 0, N... >(), at< self\_back\_t::L...::value, N... >() > &&data, const T rho=1.0)  
[network\\_problem\\_batch\\_l2\\_t](#); default constructor
- std::tuple< variable\_t, T > [operator\(\)](#) (const variable\_t &var, [metainfo\\_t](#) &info) const  
*The operator () function. compute gradient.*

## Public Attributes

- const T **rho** = 1.0

### 3.90.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N >
```

The [network\\_problem\\_batch\\_l2\\_t](#) struct. The problem implementation of l2 neural network training in batches.

$$\nabla_{W,b} \mathcal{L}(f_{W,b}) + \frac{\rho}{2} \|W\|^2 + \frac{\rho}{2} \|b\|^2$$

#### Template Parameters

<i>T</i>	Base numeric type (eg. double, float, ...).
<i>ATYPE</i>	Activation type of this neural network.
<i>LOSS</i>	Objectiv function type of this neural network
<i>BATCH</i>	Const integer value specifying the batch size.
<i>N</i>	Neural network topology. Array of postive integer values specifying the number of neurons at each layer.

### 3.90.2 Constructor & Destructor Documentation

#### 3.90.2.1 network\_problem\_batch\_l2\_t()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N >::network_problem_batch_l2_t (
    LOSS< T > && l,
    network_data_t< T, at< 0, N... >(), at< self_back_t::L::value, N... >() > &&
    data,
    const T rho = 1.0 ) [inline], [explicit]
```

[network\\_problem\\_batch\\_l2\\_t](#); default constructor

#### Parameters

<i>l</i>	loss object
<i>data</i>	traing data
<i>rho</i>	hyperparameter of L2 regularisation

### 3.90.3 Member Function Documentation

#### 3.90.3.1 operator()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_batch_l2_t< T, ATYPE, LOSS, BATCH, N >::operator() (
    const variable_t & var,
    metainfo_t & info ) const [inline]
```

The operator () function. compute gradient.

##### Parameters

<i>var</i>	Current position
------------	------------------

##### Returns

Gradient and loss at specified position

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

- lipnet/include/lipnet/problem/nn\_problem\_batch\_l2.hpp

## 3.91 lipnet::network\_problem\_batch\_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference

The [network\\_problem\\_batch\\_t](#) struct. The problem implementation of nominal neural network training in batches.

```
#include <nn_problem_batch.hpp>
```

Inheritance diagram for lipnet::network\_problem\_batch\_t< T, ATYPE, LOSS, BATCH, N >:



## Classes

- struct [metainfo\\_t](#)

## Public Types

- typedef [backpropagation\\_batch\\_t](#)< T, ATYPE, LOSS, BATCH, N... > **self\_back\_t**
- typedef self\_back\_t::variable\_t **variable\_t**

## Public Member Functions

- `std::tuple< variable_t, T > operator()` (const variable\_t &var, [metainfo\\_t](#) &info) const

*The operator () function. compute gradient.*

## Additional Inherited Members

### 3.91.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N >
```

The [network\\_problem\\_batch\\_t](#) struct. The problem implementation of nominal neural network training in batches.

$$\nabla_{W,b}\mathcal{L}(f_{W,b})$$

#### Template Parameters

<i>T</i>	Base numeric type (eg. double, float, ...).
<i>ATYPE</i>	Activation type of this neural network.
<i>LOSS</i>	Objectiv function type of this neural network
<i>BATCH</i>	Const integer value specifying the batch size.
<i>N</i>	Neural network topology. Array of postive integer values specifying the number of neurons at each layer.

### 3.91.2 Member Function Documentation

#### 3.91.2.1 operator()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_batch_t< T, ATYPE, LOSS, BATCH, N >::operator()
(
    const variable_t & var,
    metainfo_t & info ) const [inline]
```

The operator () function. compute gradient.

#### Parameters

<i>var</i>	current position
------------	------------------

**Returns**

gradient and loss at specified position

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

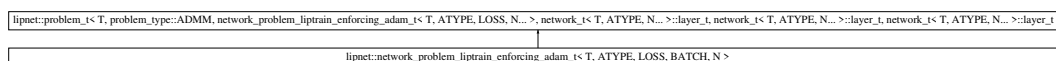
- `lipnet/include/lipnet/problem/nn_problem_batch.hpp`

### 3.92 `lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >` Struct Template Reference

The `network_problem_liptrain_enforcing_adam_t` struct. The problem implementation of admm neural network training to enforce lipschitz bound.

```
#include <nn_problem_liptrain_admm.hpp>
```

Inheritance diagram for `lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >`:

**Public Types**

- `template<size_t NN>`  
using **vector\_t** = `blaze::StaticVector< T, NN, blaze::columnVector >`
- `template<size_t NN1, size_t NN2>`  
using **matrix\_t** = `blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >`
- `typedef std::integral_constant< size_t, sizeof...(N) - 1 > L`
- `typedef std::integral_constant< size_t, (N+...) > NL`
- `typedef std::integer_sequence< size_t, N... > DIMS`
- `typedef network_t< T, ATYPE, N... >::layer_t variable_t`

**Public Member Functions**

- `network_problem_liptrain_enforcing_adam_t` (const `network_data_t< T, at< 0, N... >()`, at< `L::value, N... >()` > &&data, const `T lip=70.0`)  
*network\_problem\_liptrain\_enforcing\_adam\_t; default constructor*
- `variable_t residual` (const `variable_t &x`, const `variable_t &z`) const  
*The residual method; compute residual.*
- `variable_t optimize1` (const `T rho`, const `variable_t &var`, const `variable_t &varbar`, const `variable_t &dvar`) const  
*optimize first subproblem; with nominell training; adam method*
- `variable_t optimize2` (const `T rho`, const `variable_t &var`, const `variable_t &varbar`, const `variable_t &dvar`) const  
*optimize second variable; conic programm; mosek; interior point method*
- `T loss` (const `T rho`, const `variable_t &var`, const `variable_t &varbar`) const  
*compute lipschitz constant; mosek; interior point method;*

## Public Attributes

- [network\\_data\\_t](#)< T, at< 0, N... >(), at< L::value, N... >() > **training\_data**
- const T **lipschitz**

### 3.92.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
struct lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >
```

The [network\\_problem\\_liptrain\\_enforcing\\_adam\\_t](#) struct. The problem implementation of admm neural network training to enforce lipschitz bound.

#### Template Parameters

<i>T</i>	Base numeric type (eg. double, float, ...).
<i>ATYPE</i>	Activation type of this neural network.
<i>LOSS</i>	Objectiv function type of this neural network
<i>BATCH</i>	Const integer value specifying the batch size.
<i>N</i>	Neural network topology. Array of postive integer values specifying the number of neurons at each layer.

### 3.92.2 Constructor & Destructor Documentation

#### 3.92.2.1 network\_problem\_liptrain\_enforcing\_adam\_t()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↔
SS, size_t BATCH, size_t ... N>
lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >::network_problem_liptrain_enfor
(
    const network\_data\_t< T, at< 0, N... >(), at< L::value, N... >() > && data,
    const T lip = 70.0 ) [inline], [explicit]
```

[network\\_problem\\_liptrain\\_enforcing\\_adam\\_t](#); default constructor

#### Parameters

<i>data</i>	training data
<i>lip</i>	lipschitz constant

### 3.92.3 Member Function Documentation

### 3.92.3.1 loss()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↔
SS, size_t BATCH, size_t ... N>
T lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >::loss (
    const T rho,
    const variable_t & var,
    const variable_t & varbar ) const [inline]
```

compute lipschitz constant; mosek; interior point method;

#### Parameters

<i>rho</i>	admm hyperparameter; augmented lagrange multiplier
<i>var</i>	first const variable
<i>varbar</i>	second const variable

#### Returns

lipschitz constant

### 3.92.3.2 optimize1()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↔
SS, size_t BATCH, size_t ... N>
variable_t lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >↔
::optimize1 (
    const T rho,
    const variable_t & var,
    const variable_t & varbar,
    const variable_t & dvar ) const [inline]
```

optimize first subproblem; with nominell training; adam method

$$\text{@f[ \arg \min_{\{W,b\}} L_v(W,b,\tilde{W},Y) \text{ @f}]}$$

#### Parameters

<i>rho</i>	admm hyperparameter; augmented lagrange multiplier
<i>var</i>	variable to optimize
<i>varbar</i>	second const variable
<i>dvar</i>	dual variable

#### Returns

optimal point var



### 3.92.3.3 optimize2()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
variable_t lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >↵
::optimize2 (
    const T rho,
    const variable_t & var,
    const variable_t & varbar,
    const variable_t & dvar ) const [inline]
```

optimize second variable; conic programm; mosek; interior point method

$$@f[\arg \min_{\{\tilde{W}\}} L_v(W,b,\tilde{W},Y) @f]$$

#### Parameters

<i>rho</i>	admm hyperparameter; augmented lagrange multiplier
<i>var</i>	first const variable
<i>varbar</i>	variable
<i>dvar</i>	dual variable

#### Returns

optimal point varvar

### 3.92.3.4 residual()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
variable_t lipnet::network_problem_liptrain_enforcing_adam_t< T, ATYPE, LOSS, BATCH, N >↵
::residual (
    const variable_t & x,
    const variable_t & z ) const [inline]
```

The residual method; compute residual.

#### Parameters

<i>x</i>	variable
<i>z</i>	variable

#### Returns

residual

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

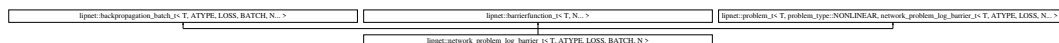
- lipnet/include/lipnet/problem/nn\_problem\_liptrain\_admm.hpp

### 3.93 lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N > Struct Template Reference

The [network\\_problem\\_log\\_barrier\\_t](#) struct. The problem implementation of barrier neural network training in batches.

```
#include <nn_problem_liptrain_barrier.hpp>
```

Inheritance diagram for lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >:



## Classes

- struct [feasibility\\_t](#)  
The *feasibility\_t* struct. Implementation of feasibility check for this problem.
- struct [metainfo\\_t](#)

## Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef std::integral\_constant< size\_t, sizeof...(N) - 1 > **L**
- typedef std::integral\_constant< size\_t, (N+...) > **NL**
- typedef std::integral\_constant< size\_t, (N+...) - at< 0, N... > ) - at< L::value, N... > ) > **TN**
- typedef [backpropagation\\_batch\\_t](#)< T, ATYPE, LOSS, BATCH, N... > **self\_back\_t**
- typedef [barrierfunction\\_t](#)< T, N... > **self\_barrier\_t**
- typedef [self\\_barrier\\_t::variable\\_t](#) **variable\_t**

## Public Member Functions

- [network\\_problem\\_log\\_barrier\\_t](#) (LOSS< T > &&l, [network\\_data\\_t](#)< T, at< 0, N... >(), at< L::value, N... >() > &&data, const T lipschitz=70.0)  
*network\_problem\_log\_barrier\_t; default constructor*
- std::tuple< [variable\\_t](#), T > [operator\(\)](#) (const [variable\\_t](#) &var, [metainfo\\_t](#) &info, [feasibility\\_t](#) &line, T &gamma) const  
*compute gradients*
- std::tuple< [variable\\_t](#), T > [operator\(\)](#) (const [variable\\_t](#) &var, [metainfo\\_t](#) &info, [feasibility\\_t](#) &line) const
- std::tuple< [variable\\_t](#), T > [operator\(\)](#) (const [variable\\_t](#) &var, [metainfo\\_t](#) &info, const T &gamma) const
- std::tuple< [variable\\_t](#), T > [operator\(\)](#) (const [variable\\_t](#) &var, [metainfo\\_t](#) &info) const
- template<bool feasibility\_enabled = false, bool gamma\_enabled = false>  
std::tuple< [variable\\_t](#), T > [run](#) (const [variable\\_t](#) &var, [metainfo\\_t](#) &info, typename std::conditional< feasibility\_enabled, [feasibility\\_t](#), std::void\_type >::type &feasibility, typename std::conditional< gamma\_enabled, T, std::void\_type >::type level) const  
*compute gradient of objectiv function*

## Additional Inherited Members

### 3.93.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >
```

The [network\\_problem\\_log\\_barrier\\_t](#) struct. The problem implementation of barrier neural network training in batches.

$$\nabla_{W,b} \mathcal{L}(f_{W,b}) - \rho \log \det(\chi(\Psi^2, W))$$

#### Template Parameters

<i>T</i>	Base numeric type (eg. double, float, ...).
<i>ATYPE</i>	Activation type of this neural network.
<i>LOSS</i>	Objective function type of this neural network
<i>BATCH</i>	Const integer value specifying the batch size.
<i>N</i>	Neural network topology. Array of positive integer values specifying the number of neurons at each layer.

### 3.93.2 Constructor & Destructor Documentation

#### 3.93.2.1 network\_problem\_log\_barrier\_t()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::network_problem_log_barrier_t
(
    LOSS< T > && l,
    network_data_t< T, at< 0, N... >(), at< L::value, N... >() > && data,
    const T lipschitz = 70.0 ) [inline], [explicit]
```

[network\\_problem\\_log\\_barrier\\_t](#); default constructor

#### Parameters

<i>l</i>	loss object
<i>data</i>	training data
<i>lipschitz</i>	lipschitz constant

### 3.93.3 Member Function Documentation

**3.93.3.1 operator>() [1/4]**

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >↵
::operator() (
    const variable_t & var,
    metainfo_t & info ) const [inline]
```

**See also**

[run](#)( const variable\_t& var, [metainfo\\_t](#) &info, typename std::conditional<feasibility\_enabled, feasibility\_t, std↵  
::void\_type >::type &feasibility, typename std::conditional<gamma\_enabled, T, std::void\_type >::type level )  
const

**3.93.3.2 operator>() [2/4]**

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >↵
::operator() (
    const variable_t & var,
    metainfo_t & info,
    const T & gamma ) const [inline]
```

**See also**

[run](#)( const variable\_t& var, [metainfo\\_t](#) &info, typename std::conditional<feasibility\_enabled, feasibility\_t, std↵  
::void\_type >::type &feasibility, typename std::conditional<gamma\_enabled, T, std::void\_type >::type level )  
const

**3.93.3.3 operator>() [3/4]**

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >↵
::operator() (
    const variable_t & var,
    metainfo_t & info,
    feasibility_t & line ) const [inline]
```

**See also**

[run](#)( const variable\_t& var, [metainfo\\_t](#) &info, typename std::conditional<feasibility\_enabled, feasibility\_t, std↵  
::void\_type >::type &feasibility, typename std::conditional<gamma\_enabled, T, std::void\_type >::type level )  
const

## 3.93.3.4 operator() [4/4]

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::operator() (
    const variable_t & var,
    metainfo_t & info,
    feasibility_t & line,
    T & gamma ) const [inline]
```

compute gradients

## Parameters

<i>var</i>	variable
<i>info</i>	metainfo
<i>line</i>	feasibility check
<i>gamma</i>	hyperparameter

## Returns

gradients

## See also

[run](#)( const variable\_t& var, [metainfo\\_t](#) &info, typename std::conditional<feasibility\_enabled, feasibility\_t, std::void\_type >::type &feasibility, typename std::conditional<gamma\_enabled, T, std::void\_type >::type level ) const

## 3.93.3.5 run()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
template<bool feasibility_enabled = false, bool gamma_enabled = false>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_t< T, ATYPE, LOSS, BATCH, N >::run (
    const variable_t & var,
    metainfo_t & info,
    typename std::conditional< feasibility_enabled, feasibility_t, std::void_type >::type & feasibility,
    typename std::conditional< gamma_enabled, T, std::void_type >::type level ) const
[inline]
```

compute gradient of objectiv function

## Template Parameters

<i>feasibility_enabled</i>	enable/disable feasibility checking
<i>gamma_enabled</i>	enable/disable set init hyperparameter gamma

## Parameters

<i>var</i>	variable
<i>info</i>	metainfo
<i>line</i>	feasibility check
<i>level</i>	hyperparameter

## Returns

gradients

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

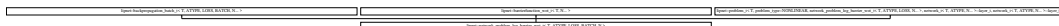
- `lipnet/include/lipnet/problem/nn_problem_liptrain_barrier.hpp`

### 3.94 `lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >` Struct Template Reference

The `network_problem_log_barrier_wot_t` struct. The problem implementation of barrier (without T) neural network training in batches.

```
#include <nn_problem_liptrain_barrier_wot.hpp>
```

Inheritance diagram for `lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >`:



## Classes

- struct `feasibility_t`  
The `feasibility_t` struct. Implementation of feasibility check for this problem.
- struct `metainfo_t`

## Public Types

- `template<size_t NN>`  
using `vector_t` = `blaze::StaticVector< T, NN, blaze::columnVector >`
- `template<size_t NN1, size_t NN2>`  
using `matrix_t` = `blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >`
- `typedef std::integral_constant< size_t, sizeof...(N) - 1 > L`
- `typedef std::integral_constant< size_t, (N+...) > NL`
- `typedef std::integral_constant< size_t, (N+...) - at< 0, N... > ) - at< L::value, N... > > TN`
- `typedef backpropagation_batch_t< T, ATYPE, LOSS, BATCH, N... > self_back_t`
- `typedef barrierfunction_wot_t< T, N... > self_barrier_t`
- `typedef self_barrier_t::tparam_t param_t`
- `typedef self_back_t::variable_t variable_t`

## Public Member Functions

- [network\\_problem\\_log\\_barrier\\_wot\\_t](#) (LOSS< T > &&l, [network\\_data\\_t](#)< T, at< 0, N... >(), at< L::value, N... >() > &&data, param\_t &&tparam, const T lipschitz=70.0)  
*network\_problem\_log\_barrier\_wot\_t; default constructor*
- std::tuple< variable\_t, T > [operator\(\)](#) (const variable\_t &var, [metainfo\\_t](#) &info, [feasibility\\_t](#) &line, T &gamma) const  
*compute gradients*
- std::tuple< variable\_t, T > [operator\(\)](#) (const variable\_t &var, [metainfo\\_t](#) &info, [feasibility\\_t](#) &line) const
- std::tuple< variable\_t, T > [operator\(\)](#) (const variable\_t &var, [metainfo\\_t](#) &info, const T &gamma) const
- std::tuple< variable\_t, T > [operator\(\)](#) (const variable\_t &var, [metainfo\\_t](#) &info) const
- template<bool feasibility\_enabled = false, bool gamma\_enabled = false>  
std::tuple< variable\_t, T > [run](#) (const variable\_t &var, [metainfo\\_t](#) &info, typename std::conditional< feasibility\_enabled, [feasibility\\_t](#), [std::void\\_type](#) >::type &feasibility, typename std::conditional< gamma\_enabled, T, [std::void\\_type](#) >::type level) const  
*compute gradient of objectiv function*

## Additional Inherited Members

### 3.94.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
```

```
struct lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >
```

The [network\\_problem\\_log\\_barrier\\_wot\\_t](#) struct. The problem implementation of barrier (without T) neural network training in batches.

$$\nabla_{W,b} \mathcal{L}(f_{W,b}) - \rho \log \det(\chi(\Psi^2, W))$$

#### Template Parameters

<i>T</i>	Base numeric type (eg. double, float, ...).
<i>ATYPE</i>	Activation type of this neural network.
<i>LOSS</i>	Objectiv function type of this neural network
<i>BATCH</i>	Const integer value specifying the batch size.
<i>N</i>	Neural network topology. Array of postive integer values specifying the number of neurons at each layer.

### 3.94.2 Constructor & Destructor Documentation

#### 3.94.2.1 network\_problem\_log\_barrier\_wot\_t()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t ... N>
```

```

lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N >::network_problem_log_barrier_wot_t
(
    LOSS< T > && l,
    network_data_t< T, at< 0, N... >(), at< L::value, N... >() > && data,
    param_t && tparam,
    const T lipschitz = 70.0 ) [inline], [explicit]

```

[network\\_problem\\_log\\_barrier\\_wot\\_t](#); default constructor

#### Parameters

<i>l</i>	loss object
<i>data</i>	training data
<i>tparam</i>	T hyperparameter from $\chi(\Psi^2, W)$
<i>lipschitz</i>	lipschitz constant

### 3.94.3 Member Function Documentation

#### 3.94.3.1 operator>() [1/4]

```

template<typename T , template< typename > typename ATYPE, template< typename > typename LO↔
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N
>::operator() (
    const variable_t & var,
    metainfo_t & info ) const [inline]

```

#### See also

[run](#)( const variable\_t& var, [metainfo\\_t](#) &info, typename std::conditional<feasibility\_enabled, feasibility\_t, std↔  
::void\_type >::type &feasibility, typename std::conditional<gamma\_enabled, T, std::void\_type >::type level )  
const

#### 3.94.3.2 operator>() [2/4]

```

template<typename T , template< typename > typename ATYPE, template< typename > typename LO↔
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N
>::operator() (
    const variable_t & var,
    metainfo_t & info,
    const T & gamma ) const [inline]

```

#### See also

[run](#)( const variable\_t& var, [metainfo\\_t](#) &info, typename std::conditional<feasibility\_enabled, feasibility\_t, std↔  
::void\_type >::type &feasibility, typename std::conditional<gamma\_enabled, T, std::void\_type >::type level )  
const



**3.94.3.3 operator>() [3/4]**

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N
>::operator() (
    const variable_t & var,
    metainfo_t & info,
    feasibility_t & line ) const [inline]
```

**See also**

```
run( const variable_t& var, metainfo_t &info, typename std::conditional<feasibility_enabled, feasibility_t, std↵
::void_type >::type &feasibility, typename std::conditional<gamma_enabled, T, std::void_type >::type level )
const
```

**3.94.3.4 operator>() [4/4]**

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N
>::operator() (
    const variable_t & var,
    metainfo_t & info,
    feasibility_t & line,
    T & gamma ) const [inline]
```

**compute gradients****Parameters**

<i>var</i>	variable
<i>info</i>	metainfo
<i>line</i>	feasibility check
<i>gamma</i>	hyperparameter

**Returns**

gradients

**See also**

```
run( const variable_t& var, metainfo_t &info, typename std::conditional<feasibility_enabled, feasibility_t, std↵
::void_type >::type &feasibility, typename std::conditional<gamma_enabled, T, std::void_type >::type level )
const
```

### 3.94.3.5 run()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↔
SS, size_t BATCH, size_t ... N>
template<bool feasibility_enabled = false, bool gamma_enabled = false>
std::tuple<variable_t,T> lipnet::network_problem_log_barrier_wot_t< T, ATYPE, LOSS, BATCH, N
>::run (
    const variable_t & var,
    metainfo_t & info,
    typename std::conditional< feasibility_enabled, feasibility_t, std::void_type >↔
::type & feasibility,
    typename std::conditional< gamma_enabled, T, std::void_type >::type level ) const
[inline]
```

compute gradient of objectiv function

#### Template Parameters

<i>feasibility_enabled</i>	enable/disable feasibility checking
<i>gamma_enabled</i>	enable/disable set init hyperparameter gamma

#### Parameters

<i>var</i>	variable
<i>info</i>	metainfo
<i>line</i>	feasibility check
<i>level</i>	hyperparameter

#### Returns

gradients

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

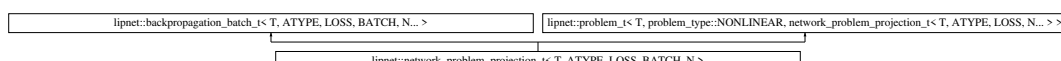
- lipnet/include/lipnet/problem/nn\_problem\_liptrain\_barrier\_wot.hpp

## 3.95 lipnet::network\_problem\_projection\_t< T, ATYPE, LOSS, BATCH, N > > Struct Template Reference

The [network\\_problem\\_projection\\_t](#) struct. The problem implementation of projected neural network training in batches.

```
#include <nn_problem_liptrain_projection.hpp>
```

Inheritance diagram for lipnet::network\_problem\_projection\_t< T, ATYPE, LOSS, BATCH, N >:



## Classes

- struct [metainfo\\_t](#)

## Public Types

- template<size\_t NN>  
using **vector\_t** = blaze::StaticVector< T, NN, blaze::columnVector >
- template<size\_t NN1, size\_t NN2>  
using **matrix\_t** = blaze::StaticMatrix< T, NN1, NN2, blaze::rowMajor >
- typedef std::integral\_constant< size\_t, sizeof...(N) -1 > **L**
- typedef std::integral\_constant< size\_t,(N+...) > **QN**
- typedef std::integral\_constant< size\_t,(N+...) - at< 0, N... > ) - at< L::value, N... > ) > **TN**
- typedef std::integer\_sequence< size\_t, N... > **DIMS**
- typedef [backpropagation\\_batch\\_t](#)< T, ATYPE, LOSS, BATCH, N... > **self\_back\_t**
- typedef self\_back\_t::variable\_t **variable\_t**

## Public Member Functions

- [network\\_problem\\_projection\\_t](#) (LOSS< T > &&l, [network\\_data\\_t](#)< T, at< 0, N... >(), at< L::value, N... >()  
> &&data, const T &lip=70.0, const T &tparam=100.0)  
*[network\\_problem\\_projection\\_t](#); default constructor*
- std::tuple< variable\_t, T > [operator\(\)](#) (const variable\_t &var, [metainfo\\_t](#) &info) const  
*compute gradient of objectiv function linke nominell training*
- variable\_t [projection](#) (variable\_t &&var) const  
*The projection method. Compute projection.*

## Public Attributes

- T **lipschitz**
- T **tparaminit**

### 3.95.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, template< typename > typename LOSS, size_t BATCH, size_t
... N>
struct lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >
```

The [network\\_problem\\_projection\\_t](#) struct. The problem implementation of projected neural network training in batches.

$$\nabla_{W,b}\mathcal{L}(f_{W,b})$$

#### Template Parameters

<i>T</i>	Base numeric type (eg. double, float, ...).
<i>ATYPE</i>	Activation type of this neural network.
<i>LOSS</i>	Objectiv function type of this neural network
<i>BATCH</i>	Const integer value specifying the batch size.
<i>N</i>	Neural network topology. Array of postive integer values specifying the number of neurons at each layer.

## 3.95.2 Constructor & Destructor Documentation

### 3.95.2.1 network\_problem\_projection\_t()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >::network_problem_projection_t
(
    LOSS< T > && l,
    network_data_t< T, at< 0, N... >(), at< L::value, N... >() > && data,
    const T & lip = 70.0,
    const T & tparam = 100.0 ) [inline], [explicit]
```

network\_problem\_projection\_t; default constructor

#### Parameters

<i>l</i>	loss object
<i>data</i>	tarining data
<i>lip</i>	lipschitz constant
<i>tparam</i>	T hyperparameter from $\chi(\Psi^2, W)$

## 3.95.3 Member Function Documentation

### 3.95.3.1 operator>()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LO↵
SS, size_t BATCH, size_t ... N>
std::tuple<variable_t,T> lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >↵
::operator() (
    const variable_t & var,
    metainfo_t & info ) const [inline]
```

compute gradient of objectiv function linke nominell training

#### Parameters

<i>var</i>	variable
<i>info</i>	metainfo

#### Returns

gradients

### 3.95.3.2 projection()

```
template<typename T , template< typename > typename ATYPE, template< typename > typename LOSS,
size_t BATCH, size_t ... N>
variable_t lipnet::network_problem_projection_t< T, ATYPE, LOSS, BATCH, N >::projection (
    variable_t && var ) const [inline]
```

The projection method. Compute projection.

$$\min ||W - \tilde{W}||^2 \quad \text{s.t.} \quad \chi(\Psi^2, W) \succeq 0$$

See also

[lipnet::mosek\\_projection\\_wot\\_t](#)

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

- lipnet/include/lipnet/problem/nn\_problem\_liptrain\_projection.hpp

## 3.96 lipnet::network\_t< T, ATYPE, N > Struct Template Reference

The [network\\_t](#) struct; neural network implementation.

```
#include <network.hpp>
```

### Classes

- struct [data\\_serialization\\_t](#)  
*serialization helper struct*
- struct [topology\\_serialization\\_t](#)  
*serialization helper struct*

### Public Types

- typedef [network\\_topology](#)< T, N... >::type **layer\_t**
- typedef std::integral\_constant< size\_t, sizeof...(N) -1 > **L**
- typedef std::integral\_constant< size\_t,(N+...) > **NL**
- typedef std::integer\_sequence< size\_t, N... > **DIMS**
- typedef blaze::StaticVector< T, at< L::value, N... >), blaze::columnVector > **outvec\_t**
- typedef blaze::StaticVector< T, at< 0, N... >), blaze::columnVector > **invec\_t**

### Public Member Functions

- outvec\_t [query](#) (const invec\_t &input) const  
*query the neural network*
- template<class Archive >  
void [save](#) (Archive &ar) const  
*serialize network*
- template<class Archive >  
void [load](#) (Archive &ar)  
*deserialize network*

## Public Attributes

- layer\_t [layers](#)  
*weights and biases*

### 3.96.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, size_t ... N>
struct lipnet::network_t< T, ATYPE, N >
```

The [network\\_t](#) struct; neural network implementation.

Template Parameters

<i>T</i>	numerical value type
<i>ATYPE</i>	activation function type
<i>N</i>	network topology

### 3.96.2 Member Function Documentation

#### 3.96.2.1 query()

```
template<typename T , template< typename > typename ATYPE, size_t ... N>
outvec_t lipnet::network_t< T, ATYPE, N >::query (
    const invec_t & input ) const [inline]
```

query the neural network

$$z_l = W_l x_l \quad x_{l+1} = \sigma(z_l) \quad \dots$$

Parameters

<i>input</i>	vector
--------------	--------

Returns

output vector

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

- lipnet/include/lipnet/network/network.hpp

## 3.97 lipnet::network\_topology< T, NI, NO, NARGS > Struct Template Reference

```
#include <topology.hpp>
```

### Public Types

- typedef [network\\_topology\\_impl](#)< T, NI, NO, NARGS... >::type **type**

#### 3.97.1 Detailed Description

```
template<typename T, size_t NI, size_t NO, size_t ... NARGS>
struct lipnet::network_topology< T, NI, NO, NARGS >
```

See also

[network\\_topology\\_impl](#)

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

- lipnet/include/lipnet/network/topology.hpp

## 3.98 lipnet::network\_topology\_impl< T, NI, NO, NS > Struct Template Reference

nework layer holder and creator struct; helper struct to create compile time layers in stack memory -> performance

```
#include <topology.hpp>
```

### Public Types

- typedef [network\\_topology\\_impl](#)< T, NO, NS... >::type **next**
- typedef [join\\_tuples](#)< std::tuple< [layer\\_t](#)< T, NI, NO > >, next >::type **type**

#### 3.98.1 Detailed Description

```
template<typename T, size_t NI, size_t NO, size_t ... NS>
struct lipnet::network_topology_impl< T, NI, NO, NS >
```

nework layer holder and creator struct; helper struct to create compile time layers in stack memory -> performance

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

- lipnet/include/lipnet/network/topology.hpp

### 3.99 lipnet::network\_topology\_impl< T, NI, NO > Struct Template Reference

```
#include <topology.hpp>
```

#### Public Types

- typedef std::tuple< [layer\\_t](#)< T, NI, NO > > **type**

#### 3.99.1 Detailed Description

```
template<typename T, size_t NI, size_t NO>
struct lipnet::network_topology_impl< T, NI, NO >
```

See also

[network\\_topology\\_impl](#)

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

- lipnet/include/lipnet/network/topology.hpp

### 3.100 std::nonesuch Struct Reference

#### Public Member Functions

- **nonesuch** ([nonesuch](#) const &)=delete
- void **operator=** ([nonesuch](#) const &)=delete

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

- lipnet/include/lipnet/traits.hpp

### 3.101 lipnet::norm\_t< T, V > Struct Template Reference

The [norm\\_t](#) struct. Just a interface for all possible types. Compute norm of argument.

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

#### 3.101.1 Detailed Description

```
template<typename T, typename V>
struct lipnet::norm_t< T, V >
```

The [norm\\_t](#) struct. Just a interface for all possible types. Compute norm of argument.

$$||V||_2$$

.



## Template Parameters

$T$	numerical value type
$V$	tensor type of argument

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

- lipnet/include/lipnet/variable.hpp

## 3.102 lipnet::norm\_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > > Struct Template Reference

The [norm\\_t](#) struct for blaze::StaticMatrix.

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

### Static Public Member Functions

- static  $T$  [norm](#) (const blaze::StaticMatrix<  $T$ ,  $N1$ ,  $N2$ , blaze::rowMajor > & $m$ )  
*The norm method. Compute norm of vector  $m$ .  $||m||_{2-ind.}$ .*

### 3.102.1 Detailed Description

```
template<typename T, size_t N1, size_t N2>
struct lipnet::norm_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >
```

The [norm\\_t](#) struct for blaze::StaticMatrix.

## Template Parameters

$T$	numerical value type
$N1$	row dimension of argument
$N2$	column dimension of argument

See also

[lipnet::norm\\_t](#) [6]

### 3.102.2 Member Function Documentation

### 3.102.2.1 norm()

```
template<typename T , size_t N1, size_t N2>
static T lipnet::norm_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >::norm (
    const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > & m ) [inline], [static]
```

The norm method. Compute norm of vector  $m$ .  $\|m\|_{2-ind}$ .

#### Parameters

$m$	input matrix
-----	--------------

#### Returns

norm of matrix  $m$

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

- lipnet/include/lipnet/tensor.hpp

## 3.103 lipnet::norm\_t< T, blaze::StaticVector< T, N, blaze::columnVector > > Struct Template Reference

The `norm_t` struct for `blaze::StaticVector`.

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

### Static Public Member Functions

- static T `norm` (const blaze::StaticVector< T, N, blaze::columnVector > &m)  
*The norm method. Compute norm of vector  $m$ .  $\|m\|_{2}$ .*

### 3.103.1 Detailed Description

```
template<typename T, size_t N>
struct lipnet::norm_t< T, blaze::StaticVector< T, N, blaze::columnVector > >
```

The `norm_t` struct for `blaze::StaticVector`.

#### Template Parameters

$T$	numerical value type
$N$	dimension of argument

See also

[lipnet::norm\\_t \[6\]](#)

## 3.103.2 Member Function Documentation

### 3.103.2.1 norm()

```
template<typename T , size_t N>
static T lipnet::norm_t< T, blaze::StaticVector< T, N, blaze::columnVector > >::norm (
    const blaze::StaticVector< T, N, blaze::columnVector > & m ) [inline], [static]
```

The norm method. Compute norm of vector m.  $\|m\|_2$ .

Parameters

<i>m</i>	input vector
----------	--------------

Returns

norm of vector m

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

- lipnet/include/lipnet/tensor.hpp

## 3.104 lipnet::norm\_t< T, layer\_t< T, I, O > > Struct Template Reference

### Static Public Member Functions

- static T **norm** (const [layer\\_t](#)< T, I, O > &m)

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

- lipnet/include/lipnet/network/layer.hpp

## 3.105 lipnet::norm\_t< T, lipcalc\_parameter\_t< T, N... > > Struct Template Reference

### Static Public Member Functions

- static T **norm** (const [lipcalc\\_parameter\\_t](#)< T, N... > &m)

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

- lipnet/include/lipnet/lipschitz/parameter.hpp

### 3.106 `lipnet::norm_t< T, liptrainweights_t< T, N... > >` Struct Template Reference

#### Static Public Member Functions

- static `T norm` (const `liptrainweights_t< T, N... > &m`)

#### Public Attributes

- `decltype(liptrainweights_t< T, N... >::W)` typedef `arg1_t`
- `decltype(liptrainweights_t< T, N... >::t)` typedef `arg2_t`

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

- `lipnet/include/lipnet/lipschitz/barrier.hpp`

### 3.107 `lipnet::norm_t< T, std::tuple< ARGS... > >` Struct Template Reference

#### Static Public Member Functions

- `template<size_t ... INTS>`  
static `T norm_impl` (const `std::tuple< ARGS... > &m`, `std::integer_sequence< size_t, INTS... >`)
- static `T norm` (const `std::tuple< ARGS... > &m`)

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

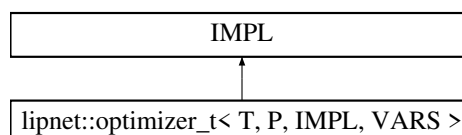
- `lipnet/include/lipnet/tuple.hpp`

### 3.108 `lipnet::optimizer_t< T, P, IMPL, VARS >` Struct Template Reference

The `optimizer_t` struct. On instantiation a class with the implementation as base class will be created.

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

Inheritance diagram for `lipnet::optimizer_t< T, P, IMPL, VARS >`:



## Classes

- struct [main\\_statistics\\_t](#)  
The *main\_statistics\_t* struct.
- struct [stats\\_type\\_exists](#)
- struct [stats\\_type\\_exists](#)< TT, typename void\_t< typename TT::statistics\_t >::type >
- struct [void\\_t](#)  
Type holder.

## Public Types

- typedef [stats\\_type\\_exists](#)< P >::type **statistics\_problem\_t**

## Public Member Functions

- template<bool stats\_enabled = false>  
std::tuple< VARS..., T > [run](#) (P &prob, VARS &&...vars, typename std::conditional< stats\_enabled, [main\\_statistics\\_t](#), std::void\_type >::type &stats) const  
The *mainoptimization* function.
- std::tuple< VARS..., T > [operator\(\)](#) (P &prob, VARS &&...vars, [main\\_statistics\\_t](#) &stats) const  
The *operator()* function. A wrapper for *run(P &prob, VARS&& ...vars , typename std::conditional<stats\_enabled, main\_statistics\_t, std::void\_type >::type &stats)* with statistics enabled.
- std::tuple< VARS..., T > [operator\(\)](#) (P &prob, VARS &&...vars) const  
The *operator()* function. A wrapper for *run(P &prob, VARS&& ...vars , typename std::conditional<stats\_enabled, main\_statistics\_t, std::void\_type >::type &stats)* with statistics disabled.

### 3.108.1 Detailed Description

```
template<typename T, typename P, typename IMPL, typename ... VARS>
struct lipnet::optimizer_t< T, P, IMPL, VARS >
```

The [optimizer\\_t](#) struct. On instantiation a class with the implementation as base class will be created.

#### Template Parameters

<i>T</i>	The numeric base type (e.g. double, float, ...)
<i>P</i>	The problem struct, which should be solved (e.g. lasso_problem, ...)
<i>IMPL</i>	The implementation of the solver, which should be used
<i>VARS</i>	Parameterpack of all type the implementation needs to solve the problem (e.g. VAR, GRADIENT, DUAL, ...)

### 3.108.2 Member Function Documentation

**3.108.2.1 operator>() [1/2]**

```
template<typename T , typename P , typename IMPL , typename ... VARS>
std::tuple<VARS...,T> lipnet::optimizer\_t< T, P, IMPL, VARS >::operator() (
    P & prob,
    VARS &&... vars ) const [inline]
```

The operator() function. A wrapper for run(P &prob, VARS&& ...vars , typename std::conditional<stats\_enabled, main\_statistics\_t, std::void\_type >::type &stats) with statistics disabled.

**Parameters**

<i>prob</i>	
<i>vars</i>	
<i>stats</i>	

**Returns**

Optimal value and optimal loss

**See also**

[run](#)( P &prob, VARS&& ...vars , typename std::conditional<stats\_enabled, [main\\_statistics\\_t](#), [std::void\\_type](#) >::type &stats )

**3.108.2.2 operator>() [2/2]**

```
template<typename T , typename P , typename IMPL , typename ... VARS>
std::tuple<VARS...,T> lipnet::optimizer\_t< T, P, IMPL, VARS >::operator() (
    P & prob,
    VARS &&... vars,
    main\_statistics\_t & stats ) const [inline]
```

The operator() function. A wrapper for run(P &prob, VARS&& ...vars , typename std::conditional<stats\_enabled, main\_statistics\_t, std::void\_type >::type &stats) with statistics enabled.

**Parameters**

<i>prob</i>	
<i>vars</i>	
<i>stats</i>	

**Returns**

Optimal value and optimal loss

See also

[run](#)( P &prob, VARS&& ...vars , typename std::conditional<stats\_enabled, [main\\_statistics\\_t](#), std::void\_type >::type &stats )

### 3.108.2.3 run()

```
template<typename T , typename P , typename IMPL , typename ... VARS>
template<bool stats_enabled = false>
std::tuple<VARS...,T> lipnet::optimizer_t< T, P, IMPL, VARS >::run (
    P & prob,
    VARS &&... vars,
    typename std::conditional< stats_enabled, main\_statistics\_t, std::void\_type >::type & stats ) const [inline]
```

The main optimization function.

#### Template Parameters

<i>stats_enabled</i>	Boolean value to decide if you want to create a statistic about this optimization process.
----------------------	--

#### Parameters

<i>prob</i>	The problem variable
<i>vars</i>	The initial values over which you want to optimize
<i>stats</i>	The statistics struct if you want to create statistics or just a void_type if not.

#### Returns

Optimal value and optimal loss

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

- [lipnet/include/lipnet/optimizer.hpp](#)

## 3.109 lipnet::parameter\_decompo\_t< T, N > Struct Template Reference

### Public Attributes

- [decompos\\_subentry](#)< T, N... >::type **subdiagonals**
- [decompos\\_diagentry](#)< T, N... >::type **diagonals**

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

- [lipnet/include/lipnet/lipschitz/decompos.hpp](#)

### 3.110 lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled >::parameter\_t Struct Reference

The [parameter\\_t](#) struct; all meta parameters for optimisation.

```
#include <adam_barrier.hpp>
```

#### Public Attributes

- `size_t max_iter`
- `size_t cpsteps`  
*maximal iterations (default = 5e5)*
- `T diff`  
*central path steps (default = 5)*
- `T threshold`  
*stopping criterion loss difference (default = 1e-10)*
- `size_t window`  
*stopping criterion window threshold (default = 1e-8)*
- `T gamma`  
*stopping criterion window size (default = 300)*
- `T alpha`  
*barriere factor (default = 1)*
- `T beta1`  
*stepsize (default = 0.02)*
- `T beta2`  
*adam meta parameter beta1 (default = 0.9)*
- `T beta3`  
*adam meta parameter beta2 (default = 0.999)*
- `T alphadec`  
*meta parameter loss difference decrease factor (default = 5.0)*
- `T gammadec`  
*meta parameter stepsize decrease factor (default = 0.5)*
- `T eps`  
*meta parameter gamma decrease factor (default = 0.5)*

#### 3.110.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD, bool feasibility_enabled = false>
struct lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::parameter_t
```

The [parameter\\_t](#) struct; all meta parameters for optimisation.

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

- lipnet/include/lipnet/optimizer/adam\_barrier.hpp



### 3.111 lipnet::fast\_gradient\_descent\_t\_impl< T, P, VAR, GRAD >::parameter\_t Struct Reference

#### Public Attributes

- T **gamma**
- T **eps**  
*stepsize (default = 0.001)*

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

- lipnet/include/lipnet/optimizer/fast\_gradient\_descent.hpp

### 3.112 lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD >::parameter\_t Struct Reference

#### Public Attributes

- size\_t **max\_iter**
- T **diff**  
*max iterations (default = 5e5)*
- T **graddiff**  
*stopping criterion loss difference (default = 1e-10)*
- T **alpha**  
*stopping criterion gradient norm (default = 1e-4)*
- T **beta1**  
*stepsize (default = 0.02)*
- T **beta2**  
*adam meta parameter beta1 (default = 0.9)*
- T **eps**  
*adam meta parameter beta2 (default = 0.999)*

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

- lipnet/include/lipnet/optimizer/adam\_momentum.hpp

### 3.113 lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >::parameter\_t Struct Reference

#### Public Attributes

- size\_t **max\_iter**
- T **rho**  
*max iterations (default = 1e4)*
- T **eps**  
*admm hyperparameter (augmented lagrange multiplier parameter) (default = 2)*

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

- lipnet/include/lipnet/optimizer/admm\_optimizer.hpp

### 3.114 lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >::parameter\_t Struct Reference

#### Public Attributes

- size\_t **max\_iter**
- T **diff**  
*max iterations (default = 5e5)*
- T **threshold**  
*stopping criterion loss difference (default = 1e-10)*
- size\_t **window**  
*stopping criterion window threshold (default = 1e-8)*
- T **alpha**  
*stopping criterion window size (default = 300)*
- T **beta1**  
*stepsize (default = 0.02)*
- T **beta2**  
*adam meta parameter beta1 (default = 0.9)*
- T **eps**  
*adam meta parameter beta2 (default = 0.999)*

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

- lipnet/include/lipnet/optimizer/adam\_projected.hpp

### 3.115 lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD >::parameter\_t Struct Reference

#### Public Attributes

- size\_t **max\_iter**
- T **diff**  
*max iterations (default = 5e5)*
- T **gamma**  
*stopping criterion loss difference (default = 1e-6)*
- T **eps**  
*stepsize (default = 0.001)*

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

- lipnet/include/lipnet/optimizer/gradient\_descent\_projected.hpp

## 3.116 lipnet::parameter\_tparam< T, N, NARGS > Struct Template Reference

### Public Types

- typedef [parameter\\_tparam\\_impl](#)< T, NARGS... >::type **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

## 3.117 lipnet::parameter\_tparam\_impl< T, N, NS > Struct Template Reference

### Public Types

- typedef [parameter\\_tparam\\_impl](#)< T, NS... >::type **next**
- typedef [join\\_tuples](#)< std::tuple< blaze::StaticVector< T, N, blaze::columnVector > >, next >::type **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

## 3.118 lipnet::parameter\_tparam\_impl< T, N, R > Struct Template Reference

### Public Types

- typedef std::tuple< blaze::StaticVector< T, N, blaze::columnVector > > **type**

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

- lipnet/include/lipnet/lipschitz/topology.hpp

## 3.119 lipnet::problem\_t< T, TYPE, IMPL, ARGS > Struct Template Reference

The [problem\\_t](#) struct; base problem struct (basically a placeholder class)

```
#include <problem.hpp>
```

### 3.119.1 Detailed Description

```
template<typename T, problem_type TYPE, typename IMPL, typename ... ARGS>
struct lipnet::problem_t< T, TYPE, IMPL, ARGS >
```

The [problem\\_t](#) struct; base problem struct (basically a placeholder class)

## Template Parameters

<i>T</i>	numerical value type
<i>TYPE</i>	problem class
<i>IMPL</i>	actual problem struct
<i>ARGS</i>	problem specific types (passthrough)

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

- lipnet/include/lipnet/problem.hpp

### 3.120 lipnet::prod\_t< T, V1, V2 > Struct Template Reference

The [prod\\_t](#) struct. Just a interface for all possible types. Compute inner/outer/... products.

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

#### 3.120.1 Detailed Description

```
template<typename T, typename V1, typename V2>
struct lipnet::prod_t< T, V1, V2 >
```

The [prod\\_t](#) struct. Just a interface for all possible types. Compute inner/outer/... products.

$$V_1 V_2^T; \quad V_1^T V_2; \quad \dots$$

.

## Template Parameters

<i>T</i>	numerical value type
<i>V1</i>	tensor type of first argument
<i>V2</i>	tensor type of second argument

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

- lipnet/include/lipnet/variable.hpp

### 3.121 lipnet::prod\_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > > Struct Template Reference

The [prod\\_t](#) struct for blaze::StaticMatrix.

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

## Static Public Member Functions

- static `T inner` (`const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > &m1`, `const blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > &m2`)  
*The inner method. Implementation of the inner product of blaze::StaticVector type.  $m_1^T m_2$ .*
- static `auto outer` (`const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > &m1`, `const blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > &m2`)  
*The outer method. Implementation of the outer product of blaze::StaticMatrix type.*

### 3.121.1 Detailed Description

```
template<typename T, size_t N1, size_t N2, size_t N3, size_t N4>
struct lipnet::prod_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >
```

The `prod_t` struct for `blaze::StaticMatrix`.

#### Template Parameters

<i>T</i>	numerical value type
<i>N1</i>	row dimension of first argument
<i>N2</i>	column dimension of first argument
<i>N3</i>	row dimension of second argument
<i>N4</i>	column dimension of second argument

See also

[lipnet::prod\\_t](#) [6]

### 3.121.2 Member Function Documentation

#### 3.121.2.1 inner()

```
template<typename T , size_t N1, size_t N2, size_t N3, size_t N4>
static T lipnet::prod_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >::inner (
    const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > & m1,
    const blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > & m2 ) [inline], [static]
```

The inner method. Implementation of the inner product of `blaze::StaticVector` type.  $m_1^T m_2$ .

#### Parameters

<i>m1</i>	first argument ( <code>blaze::StaticVector&lt;T,N1,blaze::columnVector&gt;</code> )
<i>m2</i>	second argument ( <code>blaze::StaticVector&lt;T,N2,blaze::columnVector&gt;</code> )

**Returns**

inner product of m1 and m2

**3.121.2.2 outer()**

```
template<typename T , size_t N1, size_t N2, size_t N3, size_t N4>
static auto lipnet::prod_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >::outer (
    const blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > & m1,
    const blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > & m2 ) [inline], [static]
```

The outer method. Implementation of the outer product of blaze::StaticMatrix type.

**Parameters**

<i>m1</i>	first argument (blaze::StaticMatrix<T,N1,N2,blaze::rowMajor>)
<i>m2</i>	second argument (blaze::StaticMatrix<T,N3,N4,blaze::rowMajor>)

**Returns**

kronecker product of m1 and m2

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

- lipnet/include/lipnet/tensor.hpp

## 3.122 lipnet::prod\_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > > Struct Template Reference

The [prod\\_t](#) struct for blaze::StaticVector.

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

**Static Public Member Functions**

- static T [inner](#) (const blaze::StaticVector< T, N1, blaze::columnVector > &m1, const blaze::StaticVector< T, N2, blaze::columnVector > &m2)  
*The inner method. Implementation of the inner product of blaze::StaticVector type.  $m_1^T m_2$ .*
- static auto [outer](#) (const blaze::StaticVector< T, N1, blaze::columnVector > &m1, const blaze::StaticVector< T, N2, blaze::columnVector > &m2)  
*The outer method. Implementation of the outer product of blaze::StaticVector type.  $m_1 m_2^T$ .*

**3.122.1 Detailed Description**

```
template<typename T, size_t N1, size_t N2>
struct lipnet::prod_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > >
```

The [prod\\_t](#) struct for blaze::StaticVector.

## Template Parameters

<i>T</i>	numerical value type
<i>N1</i>	dimension of first argument
<i>N2</i>	dimension of second argument

See also

[lipnet::prod\\_t](#) [6]

## 3.122.2 Member Function Documentation

### 3.122.2.1 inner()

```
template<typename T , size_t N1, size_t N2>
static T lipnet::prod_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > >::inner (
    const blaze::StaticVector< T, N1, blaze::columnVector > & m1,
    const blaze::StaticVector< T, N2, blaze::columnVector > & m2 ) [inline], [static]
```

The inner method. Implementation of the inner product of blaze::StaticVector type.  $m_1^T m_2$ .

## Parameters

<i>m1</i>	first argument (blaze::StaticVector<T,N1,blaze::columnVector>)
<i>m2</i>	second argument (blaze::StaticVector<T,N2,blaze::columnVector>)

## Returns

inner product of m1 and m2

### 3.122.2.2 outer()

```
template<typename T , size_t N1, size_t N2>
static auto lipnet::prod_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > >::outer (
    const blaze::StaticVector< T, N1, blaze::columnVector > & m1,
    const blaze::StaticVector< T, N2, blaze::columnVector > & m2 ) [inline], [static]
```

The outer method. Implementation of the outer product of blaze::StaticVector type.  $m_1 m_2^T$ .

## Parameters

<i>m1</i>	first argument (blaze::StaticVector<T,N1,blaze::columnVector>)
<i>m2</i>	second argument (blaze::StaticVector<T,N2,blaze::columnVector>)

**Returns**

outer product of m1 and m2

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

- `lipnet/include/lipnet/tensor.hpp`

### 3.123 `lipnet::prod_t< T, layer_t< T, I1, O1 >, layer_t< T, I2, O2 > >` Struct Template Reference

**Static Public Member Functions**

- static T **inner** (const `layer_t< T, I1, O1 >` &m1, const `layer_t< T, I2, O2 >` &m2)

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

- `lipnet/include/lipnet/network/layer.hpp`

### 3.124 `lipnet::prod_t< T, lipcalc_parameter_t< T, N... >, lipcalc_parameter_t< T, N... > >` Struct Template Reference

**Static Public Member Functions**

- static T **inner** (const `lipcalc_parameter_t< T, N... >` &m1, const `lipcalc_parameter_t< T, N... >` &m2)

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

- `lipnet/include/lipnet/lipschitz/parameter.hpp`

### 3.125 `lipnet::prod_t< T, liptrainweights_t< T, N... >, liptrainweights_t< T, N... > >` Struct Template Reference

**Static Public Member Functions**

- static T **inner** (const `liptrainweights_t< T, N... >` &m1, const `liptrainweights_t< T, N... >` &m2)

**Public Attributes**

- `decltype(liptrainweights_t< T, N... >::W)` typedef **arg1\_t**
- `decltype(liptrainweights_t< T, N... >::t)` typedef **arg2\_t**

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

- `lipnet/include/lipnet/lipschitz/barrier.hpp`



### 3.126 lipnet::prod\_t< T, parameter\_decompo\_t< T, N... >, parameter\_decompo\_t< T, N... > > Struct Template Reference

#### Static Public Member Functions

- static T **inner** (const [parameter\\_decompo\\_t](#)< T, N... > &m1, const [parameter\\_decompo\\_t](#)< T, N... > &m2)

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

- lipnet/include/lipnet/lipschitz/decompos.hpp

### 3.127 lipnet::prod\_t< T, std::tuple< ARGS1... >, std::tuple< ARGS2... > > Struct Template Reference

#### Static Public Member Functions

- template<size\_t... INTS>  
static T **inner\_impl** (const std::tuple< ARGS1... > &m1, const std::tuple< ARGS2... > &m2, std::integer↵\_sequence< size\_t, INTS... >)
- static T **inner** (const std::tuple< ARGS1... > &m1, const std::tuple< ARGS2... > &m2)

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

- lipnet/include/lipnet/tuple.hpp

### 3.128 lipnet::series\_t< T > Struct Template Reference

The [series\\_t](#) struct. Base struct for logging.

```
#include <statistics.hpp>
```

#### Public Member Functions

- **series\_t** (const size\_t size=0)
- T & **operator()** (const size\_t index)
- [series\\_t](#)< T > & **operator**<< (const T point)

#### Public Attributes

- std::vector< T > **data**

#### 3.128.1 Detailed Description

```
template<typename T>
struct lipnet::series_t< T >
```

The [series\\_t](#) struct. Base struct for logging.

## Template Parameters

<i>T</i>	numerical type
----------	----------------

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

- lipnet/include/lipnet/statistics.hpp

### 3.129 lipnet::solve\_function\_helper< P, VAR > Struct Template Reference

#### Public Types

- template<typename T >  
using **member\_solve\_t** = decltype(std::declval< T >().solve(std::declval< const VAR & >()))

#### Static Public Attributes

- constexpr static bool **value** = std::is\_detected<member\_solve\_t, P>::value

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

- lipnet/include/lipnet/optimizer.hpp

### 3.130 lipnet::squared\_error\_t< T > Struct Template Reference

The [squared\\_error\\_t](#) struct; implementation of the squared error objective function.

```
#include <loss.hpp>
```

#### Public Types

- template<typename TT, size\_t O, size\_t I>  
using **matrix\_t** = blaze::StaticMatrix< TT, O, I, blaze::columnMajor >
- template<typename TT, size\_t N>  
using **vector\_t** = blaze::StaticVector< TT, N, blaze::columnVector >

#### Public Member Functions

- template<size\_t N, size\_t BATCH = 0, typename std::enable\_if<!(BATCH<=0), int >::type = 0>  
T **evaluate** (const matrix\_t< T, N, BATCH > &target, const matrix\_t< T, N, BATCH > &data) const  
*The evaluate function; compute loss.*

#### 3.130.1 Detailed Description

```
template<typename T>
struct lipnet::squared_error_t< T >
```

The [squared\\_error\\_t](#) struct; implementation of the squared error objective function.

## Template Parameters

<i>T</i>	numerical value type
<i>TYPE</i>	choose the activation type

## 3.130.2 Member Function Documentation

## 3.130.2.1 evaluate()

```
template<typename T >
template<size_t N, size_t BATCH = 0, typename std::enable_if<!(BATCH<=0), int >::type = 0>
T lipnet::squared_error_t< T >::evaluate (
    const matrix_t< T, N, BATCH > & target,
    const matrix_t< T, N, BATCH > & data ) const [inline]
```

The evaluate function; compute loss.

$$\mathcal{L}(x, y) = (x - y)^{\top} (x - y)$$

## Template Parameters

<i>N</i>	input dimension type
<i>BATCH</i>	batch size

## Parameters

<i>target</i>	real value
<i>estimated</i>	value

## Returns

loss

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

- lipnet/include/lipnet/network/loss.hpp

## 3.131 lipnet::statistics\_helper Struct Reference

The [statistics\\_helper](#) struct. Helper function to disable logging for performance reasons if it is desired.

```
#include <statistics.hpp>
```

## Classes

- struct [stats\\_type\\_exists](#)
- struct [stats\\_type\\_exists](#)< TT, typename void\_t< typename TT::statistics\_t >::type >
- struct [void\\_t](#)

### 3.131.1 Detailed Description

The [statistics\\_helper](#) struct. Helper function to disable logging for performance reasons if it is desired.

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

- `lipnet/include/lipnet/statistics.hpp`

## 3.132 `lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >::statistics_t` Struct Reference

problem specific implementation of [statistics\\_t](#)

```
#include <gradient_descent_projected.hpp>
```

### Public Member Functions

- `template<class Archive >`  
`void serialize (Archive &archive)`

### Public Attributes

- `series\_t< T > loss`

### 3.132.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>
struct lipnet::gradient_descent_projected_t_impl< T, P, VAR, GRAD >::statistics_t
```

problem specific implementation of [statistics\\_t](#)

See also

lipnet [statistics\\_t](#) [4]

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

- `lipnet/include/lipnet/optimizer/gradient_descent_projected.hpp`

### 3.133 lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >::statistics\_t Struct Reference

problem specific implementation of [statistics\\_t](#)

```
#include <admm_optimizer.hpp>
```

#### Public Member Functions

- template<class Archive >  
void **serialize** (Archive &archive)

#### Public Attributes

- [series\\_t](#)< T > **loss**

#### 3.133.1 Detailed Description

```
template<typename T, typename P, typename X, typename Z, typename DUAL>  
struct lipnet::admm_optimizer_t_impl< T, P, X, Z, DUAL >::statistics_t
```

problem specific implementation of [statistics\\_t](#)

See also

lipnet [statistics\\_t](#) [4]

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

- lipnet/include/lipnet/optimizer/admm\_optimizer.hpp

### 3.134 lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled >::statistics\_t Struct Reference

problem specific implementation of [statistics\\_t](#)

```
#include <adam_barrier.hpp>
```

#### Public Member Functions

- template<class Archive >  
void **serialize** (Archive &archive)

## Public Attributes

- [series\\_t](#)< T > **loss**

### 3.134.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD, bool feasibility_enabled = false>
struct lipnet::adam_barrier_t_impl< T, P, VAR, GRAD, feasibility_enabled >::statistics_t
```

problem specific implementation of [statistics\\_t](#)

See also

lipnet [statistics\\_t](#) [4]

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

- lipnet/include/lipnet/optimizer/adam\_barrier.hpp

## 3.135 lipnet::fast\_gradient\_descent\_t\_impl< T, P, VAR, GRAD >::statistics\_t Struct Reference

problem specific implementation of [statistics\\_t](#)

```
#include <fast_gradient_descent.hpp>
```

## Public Member Functions

- template<class Archive >  
void **serialize** (Archive &archive)

## Public Attributes

- [series\\_t](#)< T > **loss**

### 3.135.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>
struct lipnet::fast_gradient_descent_t_impl< T, P, VAR, GRAD >::statistics_t
```

problem specific implementation of [statistics\\_t](#)

See also

lipnet [statistics\\_t](#) [4]

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

- lipnet/include/lipnet/optimizer/fast\_gradient\_descent.hpp

## 3.136 lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD >::statistics\_t Struct Reference

problem specific implementation of [statistics\\_t](#)

```
#include <adam_momentum.hpp>
```

### Public Member Functions

- `template<class Archive >`  
`void serialize (Archive &archive)`

### Public Attributes

- `series\_t< T > loss`

#### 3.136.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>  
struct lipnet::adam_momentum_t_impl< T, P, VAR, GRAD >::statistics_t
```

problem specific implementation of [statistics\\_t](#)

See also

lipnet [statistics\\_t](#) [4]

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

- `lipnet/include/lipnet/optimizer/adam_momentum.hpp`

## 3.137 lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >::statistics\_t Struct Reference

problem specific implementation of [statistics\\_t](#)

```
#include <adam_projected.hpp>
```

### Public Member Functions

- `template<class Archive >`  
`void serialize (Archive &archive)`

## Public Attributes

- [series\\_t](#)< T > **loss**

### 3.137.1 Detailed Description

```
template<typename T, typename P, typename VAR, typename GRAD>
struct lipnet::adam_projected_t_impl< T, P, VAR, GRAD >::statistics_t
```

problem specific implementation of [statistics\\_t](#)

See also

lipnet [statistics\\_t](#) [4]

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

- lipnet/include/lipnet/optimizer/adam\_projected.hpp

## 3.138 lipnet::optimizer\_t< T, P, IMPL, VARS >::stats\_type\_exists< TT, U > Struct Template Reference

### Public Types

- enum { **value** = 0 }
- typedef [std::void\\_type](#) **type**

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

- lipnet/include/lipnet/optimizer.hpp

## 3.139 lipnet::statistics\_helper::stats\_type\_exists< TT, U > Struct Template Reference

### Public Types

- enum { **value** = 0 }
- typedef [std::void\\_type](#) **type**

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

- lipnet/include/lipnet/statistics.hpp



### 3.140 lipnet::optimizer\_t< T, P, IMPL, VARS >::stats\_type\_exists< TT, typename void\_t< typename TT::statistics\_t >::type > Struct Template Reference

#### Public Types

- enum { **value** = 1 }
- typedef TT::statistics\_t **type**

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

- lipnet/include/lipnet/optimizer.hpp

### 3.141 lipnet::statistics\_helper::stats\_type\_exists< TT, typename void\_t< typename TT::statistics\_t >::type > Struct Template Reference

#### Public Types

- enum { **value** = 1 }
- typedef TT::statistics\_t **type**

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

- lipnet/include/lipnet/statistics.hpp

### 3.142 lipnet::network\_t< T, ATYPE, N >::topology\_serialization\_t Struct Reference

serialization helper struct

```
#include <network.hpp>
```

#### Public Member Functions

- template<class Archive >  
void **serialize** (Archive &ar)

#### 3.142.1 Detailed Description

```
template<typename T, template< typename > typename ATYPE, size_t ... N>  
struct lipnet::network_t< T, ATYPE, N >::topology_serialization_t
```

serialization helper struct

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

- lipnet/include/lipnet/network/network.hpp

### 3.143 lipnet::data\_container\_t< T >::tuple\_t< saveing > Struct Template Reference

#### Public Member Functions

- template<class Archive >  
void **serialize** (Archive &ar)

#### Public Attributes

- [view\\_t< saveing > x](#)
- [view\\_t< saveing > y](#)

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

- lipnet/include/lipnet/loader/container.hpp

### 3.144 lipnet::cholesky\_topology< T, N >::type Struct Reference

#### Public Attributes

- [cholesky\\_diagentry< T, N... >::type D](#)
- [cholesky\\_subentry< T, N... >::type L](#)

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

- lipnet/include/lipnet/lipschitz/topology.hpp

### 3.145 lipnet::inverse\_topology< T, N >::type Struct Reference

#### Public Attributes

- [inverse\\_diagentry< T, N... >::type P](#)
- [inverse\\_subentry< T, N... >::type K](#)

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

- lipnet/include/lipnet/lipschitz/topology.hpp

## 3.146 lipnet::data\_container\_t< T >::view\_t< saveing > Struct Template Reference

### Public Types

- using **refer\_t** = decltype(blaze::row(std::declval< typename std::conditional< saveing, const matrix\_t, matrix\_t >::type >(), std::declval< int >()))
- using **item\_t** = typename std::conditional< saveing, const T, T >::type

### Public Member Functions

- template<class Archive >  
void **serialize** (Archive &ar)

### Public Attributes

- **refer\_t value**

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

- lipnet/include/lipnet/loader/container.hpp

## 3.147 lipnet::optimizer\_t< T, P, IMPL, VARS >::void\_t< TT > Struct Template Reference

Type holder.

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

### Public Types

- typedef void **type**

### 3.147.1 Detailed Description

```
template<typename T, typename P, typename IMPL, typename ... VARS>
template<class TT>
struct lipnet::optimizer_t< T, P, IMPL, VARS >::void_t< TT >
```

Type holder.

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

- lipnet/include/lipnet/optimizer.hpp

### 3.148 lipnet::statistics\_helper::void\_t< TT > Struct Template Reference

#### Public Types

- typedef void **type**

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

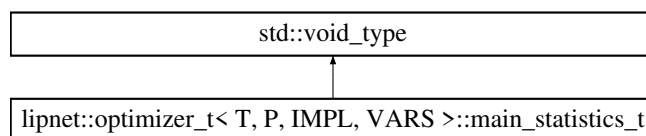
- lipnet/include/lipnet/statistics.hpp

### 3.149 std::void\_type Struct Reference

void type. Holdes nothing.

```
#include <traits.hpp>
```

Inheritance diagram for std::void\_type:



#### 3.149.1 Detailed Description

void type. Holdes nothing.

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

- lipnet/include/lipnet/traits.hpp

# Bibliography

- [1] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011. 5, 19, 20, 21, 22
- [2] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. 2019. cite arxiv:1906.04893. 74
- [3] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, pages 1–24, 2020. 29
- [4] W. Shane Grant and Randolph (2017) Voorhies. cereal - a c++11 library for serialization, 2020. <http://uscilab.github.io/cereal/>. 120, 121, 122, 123, 124
- [5] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 5, 15, 16, 17, 19, 23
- [6] Georg Hager Klaus Iglberger. Blaze, a high performance c++ math library, 2020. <https://bitbucket.org/blaze-lib/blaze/src/master/>. 38, 39, 47, 48, 53, 54, 101, 103, 113, 115
- [7] Keichi Takahashi Pranav. csv2, 2020. <https://github.com/p-ranav/csv2>. 67



# Index

adam\_barrier\_t\_impl  
lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled >, 14

adam\_momentum\_t\_impl  
lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD >, 16

adam\_projected\_t\_impl  
lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >, 18

admm\_optimizer\_t\_impl  
lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >, 20

backward  
lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >, 24

compute  
lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >, 26

derivative  
lipnet::activation\_t< T, TYPE >, 12

evaluate  
lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >, 21  
lipnet::cross\_entropy\_t< T >, 32  
lipnet::squared\_error\_t< T >, 119

fast\_gradient\_descent\_t\_impl  
lipnet::fast\_gradient\_descent\_t\_impl< T, P, VAR, GRAD >, 41

forward  
lipnet::activation\_t< T, TYPE >, 12  
lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >, 26

gradient\_descent\_projected\_t\_impl  
lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD >, 56

image\_t, 59

inner  
lipnet::prod\_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >, 113  
lipnet::prod\_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > >, 115

layer\_t  
lipnet::layer\_t< T, I, O >, 63

lipnet::activation\_t< T, TYPE >, 11  
derivative, 12  
forward, 12

lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled >, 13  
adam\_barrier\_t\_impl, 14  
run, 14

lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled >::parameter\_t, 108

lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled >::statistics\_t, 121

lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD >, 15  
adam\_momentum\_t\_impl, 16  
run, 16

lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD >::parameter\_t, 109

lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD >::statistics\_t, 123

lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >, 17  
adam\_projected\_t\_impl, 18  
project, 18  
run, 18

lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >::parameter\_t, 110

lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >::statistics\_t, 123

lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >, 19  
admm\_optimizer\_t\_impl, 20  
evaluate, 21  
optimize1, 21  
optimize2, 21  
residual, 22  
run, 22

lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >::parameter\_t, 109

lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >::statistics\_t, 121

lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >, 23  
backward, 24  
compute, 26  
forward, 26  
run, 26

lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t, 69

lipnet::barrierfunction\_t< T, N >, 27

- lipnet::barrierfunction\_wot\_t< T, N >, 28
- lipnet::calculate\_lipschitz\_t< T, N >, 28
- lipnet::cholesky\_diagentry< T, N, NARGS >, 29
- lipnet::cholesky\_diagentry\_impl< T, N >, 30
- lipnet::cholesky\_diagentry\_impl< T, N, NS >, 30
- lipnet::cholesky\_subentry< T, NI, NO, RE, NARGS >, 30
- lipnet::cholesky\_subentry\_impl< T, NI, NO >, 31
- lipnet::cholesky\_subentry\_impl< T, NI, NO, NS >, 30
- lipnet::cholesky\_topology< T, N >, 31
- lipnet::cholesky\_topology< T, N >::type, 126
- lipnet::cross\_entropy\_t< T >, 31
  - evaluate, 32
- lipnet::data\_container\_t< T >, 32
- lipnet::data\_container\_t< T >::data\_t< saveing >, 34
- lipnet::data\_container\_t< T >::tuple\_t< saveing >, 126
- lipnet::data\_container\_t< T >::view\_t< saveing >, 127
- lipnet::decompos\_diagentry< T, N, NARGS >, 35
- lipnet::decompos\_diagentry\_impl< T, N >, 35
- lipnet::decompos\_diagentry\_impl< T, N, NS >, 35
- lipnet::decompos\_subentry< T, NI, NO, RE, NARGS >, 35
- lipnet::decompos\_subentry\_impl< T, NI, NO >, 36
- lipnet::decompos\_subentry\_impl< T, NI, NO, NS >, 36
- lipnet::equation\_system\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >, 37
  - solve, 38
- lipnet::equation\_system\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticVector< T, N3, blaze::columnVector > >, 38
  - solve, 39
- lipnet::equation\_system\_t< V1, V2 >, 37
- lipnet::fast\_gradient\_descent\_t\_impl< T, P, VAR, GRAD >, 40
  - fast\_gradient\_descent\_t\_impl, 41
  - run, 41
- lipnet::fast\_gradient\_descent\_t\_impl< T, P, VAR, GRAD >::parameter\_t, 109
- lipnet::fast\_gradient\_descent\_t\_impl< T, P, VAR, GRAD >::statistics\_t, 122
- lipnet::feasibility\_t< IMPL, T, DIR >, 44
  - operator<<, 44
  - operator(), 44
- lipnet::feasibilitycheck\_t< T, N >, 45
- lipnet::feasibilitycheck\_wot\_t< T, N >, 46
- lipnet::function\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >, 47
- lipnet::function\_t< blaze::StaticVector< T, N, blaze::columnVector > >, 47
- lipnet::function\_t< layer\_t< T, I, O > >, 48
- lipnet::function\_t< liptrainweights\_t< T, N... > >, 48
- lipnet::function\_t< std::tuple< ARGS... > >, 49
- lipnet::function\_t< V >, 46
- lipnet::generate\_batch\_data< T, B, N >, 49
- lipnet::generate\_batch\_data< T, B, N, NS >, 49
- lipnet::generate\_batch\_data\_remove\_first< T, B, N, NS >, 50
- lipnet::generate\_data< T, N >, 51
- lipnet::generate\_data< T, N, NS >, 50
- lipnet::generate\_data\_remove\_first< T, N, NS >, 51
- lipnet::generator\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >, 52
- lipnet::generator\_t< blaze::StaticVector< T, N, blaze::columnVector > >, 53
- lipnet::generator\_t< layer\_t< T, I, O > >, 54
- lipnet::generator\_t< liptrainweights\_t< T, N... > >, 54
- lipnet::generator\_t< parameter\_decompo\_t< T, N... > >, 54
- lipnet::generator\_t< std::tuple< ARGS... > >, 55
- lipnet::generator\_t< V >, 52
- lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD >, 55
  - gradient\_descent\_projected\_t\_impl, 56
  - project, 57
  - run, 57
- lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD >::parameter\_t, 110
- lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD >::statistics\_t, 120
- lipnet::helper\_function\_t< V >, 58
  - v1, 58
  - v2, 58
- lipnet::helper\_inner\_t< T, V1, V2 >, 58
  - value, 58
- lipnet::helper\_norm\_t< T, V >, 59
  - value, 59
- lipnet::inverse\_diagentry< T, N, NARGS >, 60
- lipnet::inverse\_diagentry\_impl< T, N >, 60
- lipnet::inverse\_diagentry\_impl< T, N, NS >, 60
- lipnet::inverse\_subentry< T, NI, NO, RE, NARGS >, 61
- lipnet::inverse\_subentry\_impl< T, NI, NO >, 61
- lipnet::inverse\_subentry\_impl< T, NI, NO, NS >, 61
- lipnet::inverse\_topology< T, N >, 61
- lipnet::inverse\_topology< T, N >::type, 126
- lipnet::join\_tuples< std::tuple< NEW... >, std::tuple< NEXT... > >, 62
- lipnet::join\_tuples< typename, typename >, 62
- lipnet::layer\_t< T, I, O >, 63
  - layer\_t, 63
- lipnet::linsearch\_t< IMPL, T, DIRECTION >, 64
  - operator<<, 65
  - operator(), 65
- lipnet::lipcalc\_parameter\_t< T, N >, 66
- lipnet::liptrainweights\_t< T, N >, 66
- lipnet::loader\_t< T >, 66
  - load, 67
- lipnet::metainfo\_t< IMPL >, 71
- lipnet::mosek\_projection\_wot\_t< T, N >, 71
  - projection, 72
- lipnet::network\_data\_t< T, IN, OUT >, 73
- lipnet::network\_libcalc\_t< T, N >, 73
  - solve, 74
- lipnet::network\_libtrain\_enforcing\_t< T, N >, 74
  - train, 75



- lipnet::network\_problem\_batch\_admm\_t< T, ATYPE, LOSS, BATCH, N >, [76](#)
  - operator(), [77](#)
  - weights\_bar, [78](#)
- lipnet::network\_problem\_batch\_admm\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t, [68](#)
- lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N >, [78](#)
  - network\_problem\_batch\_l2\_t, [79](#)
  - operator(), [80](#)
- lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t, [69](#)
- lipnet::network\_problem\_batch\_t< T, ATYPE, LOSS, BATCH, N >, [80](#)
  - operator(), [81](#)
- lipnet::network\_problem\_batch\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t, [68](#)
- lipnet::network\_problem\_liptrain\_enforcing\_adam\_t< T, ATYPE, LOSS, BATCH, N >, [82](#)
  - loss, [83](#)
  - network\_problem\_liptrain\_enforcing\_adam\_t, [83](#)
  - optimize1, [84](#)
  - optimize2, [84](#)
  - residual, [85](#)
- lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >, [86](#)
  - network\_problem\_log\_barrier\_t, [87](#)
  - operator(), [87](#), [88](#)
  - run, [89](#)
- lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >::feasibility\_t, [42](#)
- lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t, [69](#)
- lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >, [90](#)
  - network\_problem\_log\_barrier\_wot\_t, [91](#)
  - operator(), [92](#), [93](#)
  - run, [93](#)
- lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >::feasibility\_t, [43](#)
- lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t, [70](#)
- lipnet::network\_problem\_projection\_t< T, ATYPE, LOSS, BATCH, N >, [94](#)
  - network\_problem\_projection\_t, [96](#)
  - operator(), [96](#)
  - projection, [96](#)
- lipnet::network\_problem\_projection\_t< T, ATYPE, LOSS, BATCH, N >::metainfo\_t, [70](#)
- lipnet::network\_t< T, ATYPE, N >, [97](#)
  - query, [98](#)
- lipnet::network\_t< T, ATYPE, N >::data\_serialization\_t< saving >, [33](#)
- lipnet::network\_t< T, ATYPE, N >::topology\_serialization\_t, [125](#)
- lipnet::network\_topology< T, NI, NO, NARGS >, [99](#)
- lipnet::network\_topology\_impl< T, NI, NO >, [100](#)
- lipnet::network\_topology\_impl< T, NI, NO, NS >, [99](#)
- lipnet::norm\_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >, [101](#)
  - norm, [101](#)
- lipnet::norm\_t< T, blaze::StaticVector< T, N, blaze::columnVector > >, [102](#)
  - norm, [103](#)
- lipnet::norm\_t< T, layer\_t< T, I, O > >, [103](#)
- lipnet::norm\_t< T, lipcalc\_parameter\_t< T, N... > >, [103](#)
- lipnet::norm\_t< T, liptrainweights\_t< T, N... > >, [104](#)
- lipnet::norm\_t< T, std::tuple< ARGS... > >, [104](#)
- lipnet::norm\_t< T, V >, [100](#)
- lipnet::optimizer\_t< T, P, IMPL, VARS >, [104](#)
  - operator(), [105](#), [106](#)
  - run, [107](#)
- lipnet::optimizer\_t< T, P, IMPL, VARS >::main\_statistics\_t, [67](#)
- lipnet::optimizer\_t< T, P, IMPL, VARS >::stats\_type\_exists< TT, typename void\_t< typename TT::statistics\_t >::type >, [125](#)
- lipnet::optimizer\_t< T, P, IMPL, VARS >::stats\_type\_exists< TT, U >, [124](#)
- lipnet::optimizer\_t< T, P, IMPL, VARS >::void\_t< TT >, [127](#)
- lipnet::parameter\_decompo\_t< T, N >, [107](#)
- lipnet::parameter\_tparam< T, N, NARGS >, [111](#)
- lipnet::parameter\_tparam\_impl< T, N, NS >, [111](#)
- lipnet::parameter\_tparam\_impl< T, N, R >, [111](#)
- lipnet::problem\_t< T, TYPE, IMPL, ARGS >, [111](#)
- lipnet::prod\_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >, [112](#)
  - inner, [113](#)
  - outer, [114](#)
- lipnet::prod\_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > >, [114](#)
  - inner, [115](#)
  - outer, [115](#)
- lipnet::prod\_t< T, layer\_t< T, I1, O1 >, layer\_t< T, I2, O2 > >, [116](#)
- lipnet::prod\_t< T, lipcalc\_parameter\_t< T, N... >, lipcalc\_parameter\_t< T, N... > >, [116](#)
- lipnet::prod\_t< T, liptrainweights\_t< T, N... >, liptrainweights\_t< T, N... > >, [116](#)
- lipnet::prod\_t< T, parameter\_decompo\_t< T, N... >, parameter\_decompo\_t< T, N... > >, [117](#)
- lipnet::prod\_t< T, std::tuple< ARGS1... >, std::tuple< ARGS2... > >, [117](#)
- lipnet::prod\_t< T, V1, V2 >, [112](#)
- lipnet::series\_t< T >, [117](#)
- lipnet::solve\_function\_helper< P, VAR >, [118](#)
- lipnet::squared\_error\_t< T >, [118](#)
  - evaluate, [119](#)
- lipnet::statistics\_helper, [119](#)
- lipnet::statistics\_helper::stats\_type\_exists< TT, typename void\_t< typename TT::statistics\_t >::type >, [125](#)

- lipnet::statistics\_helper::stats\_type\_exists< TT, U >, outer  
124
- lipnet::statistics\_helper::void\_t< TT >, 128
- load
  - lipnet::loader\_t< T >, 67
- loss
  - lipnet::network\_problem\_liptrain\_enforcing\_adam\_t< T, ATYPE, LOSS, BATCH, N >, 83
- network\_problem\_batch\_l2\_t
  - lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N >, 79
- network\_problem\_liptrain\_enforcing\_adam\_t
  - lipnet::network\_problem\_liptrain\_enforcing\_adam\_t< projection  
T, ATYPE, LOSS, BATCH, N >, 83
- network\_problem\_log\_barrier\_t
  - lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >, 87
- network\_problem\_log\_barrier\_wot\_t
  - lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >, 91
- network\_problem\_projection\_t
  - lipnet::network\_problem\_projection\_t< T, ATYPE, LOSS, BATCH, N >, 96
- norm
  - lipnet::norm\_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor > >, 101
  - lipnet::norm\_t< T, blaze::StaticVector< T, N, blaze::columnVector > >, 103
- operator<<
  - lipnet::feasibility\_t< IMPL, T, DIR >, 44
  - lipnet::linesearch\_t< IMPL, T, DIRECTION >, 65
- operator()
  - lipnet::feasibility\_t< IMPL, T, DIR >, 44
  - lipnet::linesearch\_t< IMPL, T, DIRECTION >, 65
  - lipnet::network\_problem\_batch\_admm\_t< T, ATYPE, LOSS, BATCH, N >, 77
  - lipnet::network\_problem\_batch\_l2\_t< T, ATYPE, LOSS, BATCH, N >, 80
  - lipnet::network\_problem\_batch\_t< T, ATYPE, LOSS, BATCH, N >, 81
  - lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >, 87, 88
  - lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >, 92, 93
  - lipnet::network\_problem\_projection\_t< T, ATYPE, LOSS, BATCH, N >, 96
  - lipnet::optimizer\_t< T, P, IMPL, VARS >, 105, 106
- optimize1
  - lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >, 21
  - lipnet::network\_problem\_liptrain\_enforcing\_adam\_t< T, ATYPE, LOSS, BATCH, N >, 84
- optimize2
  - lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >, 21
  - lipnet::network\_problem\_liptrain\_enforcing\_adam\_t< std::detail::detector< Default, std::void\_t< Op< Args... >, ATYPE, LOSS, BATCH, N >, 84
- lipnet::prod\_t< T, blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >, 114
- lipnet::prod\_t< T, blaze::StaticVector< T, N1, blaze::columnVector >, blaze::StaticVector< T, N2, blaze::columnVector > >, 115
- project
  - lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >, 18
  - lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD >, 57
  - lipnet::mosek\_projection\_wot\_t< T, N >, 72
  - lipnet::network\_problem\_projection\_t< T, ATYPE, LOSS, BATCH, N >, 96
- query
  - lipnet::network\_t< T, ATYPE, N >, 98
- residual
  - lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >, 22
  - lipnet::network\_problem\_liptrain\_enforcing\_adam\_t< T, ATYPE, LOSS, BATCH, N >, 85
- run
  - lipnet::adam\_barrier\_t\_impl< T, P, VAR, GRAD, feasibility\_enabled >, 14
  - lipnet::adam\_momentum\_t\_impl< T, P, VAR, GRAD >, 16
  - lipnet::adam\_projected\_t\_impl< T, P, VAR, GRAD >, 18
  - lipnet::admm\_optimizer\_t\_impl< T, P, X, Z, DUAL >, 22
  - lipnet::backpropagation\_batch\_t< T, ATYPE, LOSS, BATCH, N >, 26
  - lipnet::fast\_gradient\_descent\_t\_impl< T, P, VAR, GRAD >, 41
  - lipnet::gradient\_descent\_projected\_t\_impl< T, P, VAR, GRAD >, 57
  - lipnet::network\_problem\_log\_barrier\_t< T, ATYPE, LOSS, BATCH, N >, 89
  - lipnet::network\_problem\_log\_barrier\_wot\_t< T, ATYPE, LOSS, BATCH, N >, 93
  - lipnet::optimizer\_t< T, P, IMPL, VARS >, 107
- solve
  - lipnet::equation\_system\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticMatrix< T, N3, N4, blaze::rowMajor > >, 38
  - lipnet::equation\_system\_t< blaze::StaticMatrix< T, N1, N2, blaze::rowMajor >, blaze::StaticVector< T, N3, blaze::columnVector > >, 39
  - lipnet::network\_libcalc\_t< T, N >, 74
  - std::detail::detector< Default, AlwaysVoid, Op, Args >, 36
  - std::detail::detector< Default, std::void\_t< Op< Args... > >, Op, Args... >, 36

std::nonesuch, [100](#)

std::void\_type, [128](#)

train

lipnet::network\_libtrain\_enforcing\_t< T, N >, [75](#)

v1

lipnet::helper\_function\_t< V >, [58](#)

v2

lipnet::helper\_function\_t< V >, [58](#)

value

lipnet::helper\_inner\_t< T, V1, V2 >, [58](#)

lipnet::helper\_norm\_t< T, V >, [59](#)

weights\_bar

lipnet::network\_problem\_batch\_admm\_t< T,  
ATYPE, LOSS, BATCH, N >, [78](#)