

前面经过反向传播，已经计算出了模型的损失函数得分以及梯度，在反向传播完成之后会返回到 `package org.deeplearning4j.optimize.solvers` 包下的 `BaseOptimizer.gradientAndScore()` 方法体重继续执行，该方法体中继续执行。

反向传播计算完的参数还需要经过梯度正则化以及L1，L2参数惩罚

```
1.  @Override
2.  public Pair<Gradient, Double> gradientAndScore() {
3.      oldScore = score;
4.      //包含反向传播，已经算出了模型的损失函数得分以及梯度
5.      model.computeGradientAndScore();
6.
7.      if (iterationListeners != null && iterationListeners.size() > 0) {
8.          for (IterationListener l : iterationListeners) {
9.              if (l instanceof TrainingListener) {
10.                  ((TrainingListener) l).onGradientCalculation(model);
11.              }
12.          }
13.      }
14.
15.      //获取模型中的梯度和损失函数得分
16.      Pair<Gradient, Double> pair = model.gradientAndScore();
17.      //将模型的损失函数得分赋值为优化器的成员变量中
18.      score = pair.getSecond();
19.      //然后根据参数更新梯度
20.      updateGradientAccordingToParams(pair.getFirst(), model,
model.batchSize());
21.      return pair;
22.  }
```

1 updateGradientAccordingToParams

```
1.  @Override
2.  public void updateGradientAccordingToParams(Gradient gradient, Model model, int batchSize) {
3.      //首先判断是ComputationGraph还是MultiLayerNetwork
4.      if (model instanceof ComputationGraph) {
5.          ComputationGraph graph = (ComputationGraph) model;
6.          if (computationGraphUpdater == null) {
7.              computationGraphUpdater = new ComputationGraphUpdater(graph
```

```

8.         }
9.         computationGraphUpdater.update(graph, gradient, getIterationCount(model), batchSize);
10.    } else {
11.
12.        //获取更新器
13.        if (updater == null)
14.            updater = UpdaterCreator.getUpdater(model);
15.
16.        //将model改为Layer类型, 这个时候需要注意, 在多层网络架构的时候
17.        //MultiLayerNetwork 可以认为是输出层
18.        //MultiLayerNetwork is a neural network with multiple layers
19.        //in a stack, and usually an output layer.
20.        Layer layer = (Layer) model;
21.        updater.update(layer, gradient, getIterationCount(model), batchSize);
22.    }

```

1.1 UpdaterCreator.getUpdater(model)

首先需要根据模型设置来获取模型参数的更新器

```

1.    public class UpdaterCreator {
2.
3.        private UpdaterCreator() {}
4.
5.        public static org.deeplearning4j.nn.api.Updater getUpdater(Model layer) {
6.            //判断网络架构
7.            if (layer instanceof MultiLayerNetwork) {
8.                return new MultiLayerUpdater((MultiLayerNetwork) layer);
9.            } else {
10.                return new LayerUpdater();
11.            }
12.        }
13.
14.    }

```

之后构造一个新的更新器类 `MultiLayerUpdater`。

在 `package org.deeplearning4j.nn.updater;` 包下，所调用的更新器的构造函数为：

```
1.  /**
2.   * MultiLayerUpdater: Gradient updater for MultiLayerNetworks.
3.   * Expects backprop gradients for all layers to be in single Gradient
  object,
4.   * keyed by "0_b", "1_w" etc., as per MultiLayerNetwork.backward()
5.   */
6.  public MultiLayerUpdater(MultiLayerNetwork network) {
7.      //获取架构的网络层
8.      Layer[] layers = network.getLayers();
9.      //逐层判断是否为空
10.     for (int i = 0; i < layers.length; i++) {
11.         //守护条件，保证获取到的layer全都不为null
12.         while (layers[i] == null)
13.             layers = network.getLayers();
14.     }
15.     //根据网络层个数构造网络层更新器
16.     layerUpdaters = new Updater[layers.length];
17.     //更新器状态个数
18.     int updaterStateSize = 0;
19.     for (int i = 0; i < layers.length; i++) {
20.         Layer layer = layers[i];
21.         //这里依旧判断当前层是否为空，如果为空则会跑出空指针有慈航
22.         Preconditions.checkNotNull(layer);
23.         //根据当前网络层构建层更新器
24.         layerUpdaters[i] = UpdaterCreator.getUpdater(layer);
25.
26.         //这里的更新器因为使用的是SGD，所以StateSize这里不管传入什么值，返回的均
  为0
27.         updaterStateSize += layerUpdaters[i].stateSizeForLayer(layer);
28.     }
29.     //初始化更新器状态
30.     //Initialize the updater state:
31.     if (updaterStateSize > 0) {
32.         //May be 0 if all SGD updaters, for example
33.         viewArray = Nd4j.createUninitialized(new int[] {1,
  updaterStateSize}, Nd4j.order());
34.     }
35.
36.     //需要跨越多远获取子视图
37.     int soFar = 0;
38.     for (int i = 0; i < layers.length; i++) {
39.         //获取更新器状态
```

```

40.         int thisSize = layerUpdaters[i].stateSizeForLayer(layers[i]);
41.
42.         //如果为0
43.         if (thisSize == 0)
44.             continue;
45.
46.         //如果不为0, 则获取子视图
47.         INDArrary view = viewArray.get(NDArrayIndex.point(0), NDArrayIndex.interval(soFar, soFar + thisSize));
48.
49.         //设置到对应的更新器中
50.         layerUpdaters[i].setStateViewArray(layers[i], view, true);
51.         soFar += thisSize;
52.     }
53. }

```

到这里 `MultiLayerUpdater` 执行完成, 继续返回上层函数

1 updateGradientAccordingToParams

里面继续执行以下语句

```

1.     updater.update(layer, gradient, getIterationCount(model), batchSize);

```

1.2 updater.update()

```

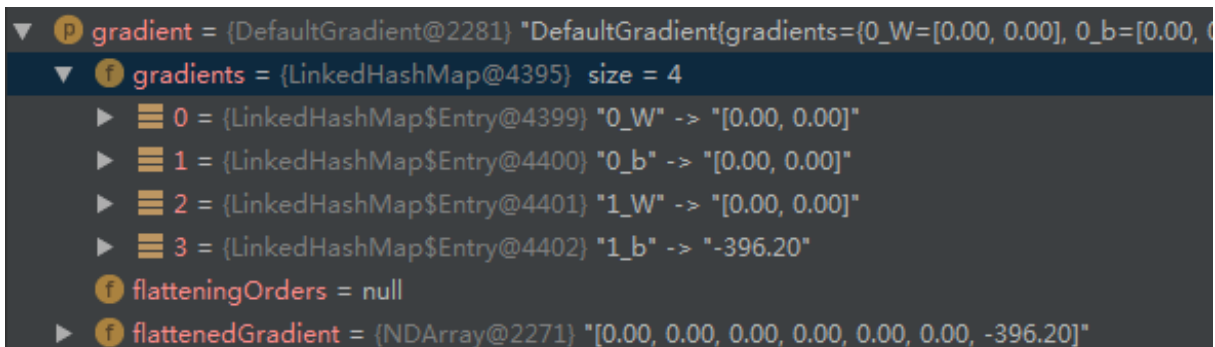
1.     @Override
2.     public void update(Layer layer, Gradient gradient, int iteration, int batchSize) {
3.         MultiLayerNetwork mln = (MultiLayerNetwork) layer;
4.
5.         //根据LayerUpdaters的个数构建 层梯度 的个数
6.         Gradient[] layerGradients = new Gradient[layerUpdaters.length];
7.         //实例化层梯度
8.         for (int i = 0; i < layerGradients.length; i++)
9.             layerGradients[i] = new DefaultGradient();
10.
11.         //然后遍历已经计算好的梯度

```

```

12.         for (Map.Entry<String, INDArarray> gradientPair : gradient.gradientFo
rVariable().entrySet()) {
13.             //获取Key
14.             String key = gradientPair.getKey();
15.             //获取 '_'的位置
16.             int idx = key.indexOf('_');
17.             if (idx == -1)
18.                 throw new IllegalStateException(
19.                     "Invalid key: MultiLayerNetwork Gradient key
does not have layer separator: \"\" + key
20.                                     + "\"");
21.
22.             //截取网络层索引
23.             int layerIdx = Integer.parseInt(key.substring(0, idx));
24.
25.             //截取后面的 w, b部分
26.             String newKey = key.substring(idx + 1);
27.             //根据网络层和w, b设置对应的梯度值
28.             layerGradients[layerIdx].gradientForVariable().put(newKey, grad
ientPair.getValue());
29.         }

```



```

▼ gradient = {DefaultGradient@2281} "DefaultGradient{gradients={0_W=[0.00, 0.00], 0_b=[0.00, 0.00], 1_W=[0.00, 0.00], 1_b=[0.00, 0.00]}, flatteningOrders=null, flattenedGradient=[0.00, 0.00, 0.00, 0.00, 0.00, 0.00, -396.20]}"
  ▼ gradients = {LinkedHashMap@4395} size = 4
    ▶ 0 = {LinkedHashMap$Entry@4399} "0_W" -> "[0.00, 0.00]"
    ▶ 1 = {LinkedHashMap$Entry@4400} "0_b" -> "[0.00, 0.00]"
    ▶ 2 = {LinkedHashMap$Entry@4401} "1_W" -> "[0.00, 0.00]"
    ▶ 3 = {LinkedHashMap$Entry@4402} "1_b" -> "-396.20"
    f flatteningOrders = null
    ▶ f flattenedGradient = {NDArray@2271} "[0.00, 0.00, 0.00, 0.00, 0.00, 0.00, -396.20]"

```

前面反向传播计算的梯度存储形式。

```

1.         //然后根据对应的值进行模型参数的更新
2.         for (int i = 0; i < layerUpdaters.length; i++) {
3.             layerUpdaters[i].update(mln.getLayer(i), layerGradients[i], ite
ration, batchSize);
4.         }
5.     }

```

1.2.1 layerUpdaters[i].update()

```

1.     @Override

```

```

2.   public void update(Layer layer, Gradient gradient, int iteration, int
miniBatchSize) {
3.       //参数名称
4.       String paramName;
5.       //原始梯度, 更新之后的梯度
6.       INDArrary gradientOrig, gradient2;
7.       //更新器
8.       GradientUpdater updater;
9.
10.      //如果当前层是FrozenLayer, 不更新网络参数
11.      if (layer instanceof FrozenLayer)
12.          return;
13.
14.      preApply(layer, gradient, iteration);

```

1.2.1.1 preApply(layer, gradient, iteration)

根据函数数值是对梯度实现正则化, 根据不同的策略对梯度进行处理。

```

1.   /**
2.    * Apply gradient normalization: scale based on L2, clipping etc.
3.    * RenormalizeL2PerLayer: divide all layer gradients by L2 to rescale
4.    * RenormalizeL2PerParamType: divide each parameter type gradient in
a layer by L2 to rescale
5.    * ClipElementWiseAbsoluteValue: clip gradients per-element
6.    * ClipL2PerLayer: same as RenormalizeL2PerLayer but limited by gradi
ent L2 norm for the layer meeting a threshold
7.    * ClipL2PerParamType: same as RenormalizeL2PerParamType but limited
by gradient L2 norm for each parameter type in a layer meeting a thres
hold
8.    */
9.   public void preApply(Layer layer, Gradient gradient, int iteration) {
10.
11.       GradientNormalization normalization = layer.conf().getLayer().getGr
adientNormalization();
12.       if (normalization == null || normalization == GradientNormalization
.None || layer.conf().isPretrain())
13.           return; //no op
14.
15.       final double threshold = layer.conf().getLayer().getGradientNormali
zationThreshold();
16.
17.       switch (normalization) {
18.           case RenormalizeL2PerLayer:

```

```

19.         double sumSquares = 0.0;
20.         for (INDArray g : gradient.gradientForVariable().values())
21.         {
22.             double l2 = g.norm2Number().doubleValue();
23.             //l2 norm: sqrt(sum_i g_i^2)
24.             sumSquares += l2 * l2;
25.         }
26.         double layerL2 = FastMath.sqrt(sumSquares);
27.         for (INDArray g : gradient.gradientForVariable().values())
28.         {
29.             g.divi(layerL2);
30.         }
31.         break;
32.     case RenormalizeL2PerParamType:
33.         for (INDArray g : gradient.gradientForVariable().values())
34.         {
35.             double l2 = Nd4j.getExecutioner().execAndReturn(new Normalization(g)).getFinalResult().doubleValue();
36.             g.divi(l2);
37.         }
38.         break;
39.     case ClipElementWiseAbsoluteValue:
40.         for (INDArray g : gradient.gradientForVariable().values())
41.         {
42.             BooleanIndexing.replaceWhere(g, threshold, Conditions.greaterThan(threshold));
43.             BooleanIndexing.replaceWhere(g, -threshold, Conditions.lessThan(-threshold));
44.         }
45.         break;
46.     case ClipL2PerLayer:
47.         double sumSquares2 = 0.0;
48.         for (INDArray g : gradient.gradientForVariable().values())
49.         {
50.             double l2 = Nd4j.getExecutioner().execAndReturn(new Normalization(g)).getFinalResult().doubleValue();
51.             //l2 norm: sqrt(sum_i g_i^2)
52.             sumSquares2 += l2 * l2;
53.         }
54.         double layerL22 = FastMath.sqrt(sumSquares2);
55.         if (layerL22 > threshold) {
56.             double scalingFactor = threshold / layerL22; // g = g /
57.             //l2 * threshold ->
58.             for (INDArray g : gradient.gradientForVariable().values()) {

```

```

53.             g.muli (scalingFactor);
54.         }
55.     }
56.     break;
57.     case ClipL2PerParamType:
58.         for (INDArray g : gradient.gradientForVariable().values())
59.         {
60.             double l2 = g.norm2Number().doubleValue();
61.             if (l2 > threshold) {
62.                 double scalingFactor = l2 / threshold;
63.                 g.divi (scalingFactor);
64.             }
65.         }
66.         break;
67.     default:
68.         throw new RuntimeException(
69.             "Unknown (or not implemented) gradient
70. normalization strategy: " + normalization);

```

1.2.1 layerUpdaters[i].update()

在对梯度进行正则化之后

```

1.         //遍历梯度 map
2.         for (Map.Entry<String, INDArray> gradientPair : gradient.gradientFo
3.             rVariable().entrySet()) {
4.             paramName = gradientPair.getKey();
5.             if (!layer.conf().isPretrain() && PretrainParamInitializer.VISI
6.                 BLE_BIAS_KEY.equals(paramName.split("_")[0]))
7.                 continue;
8.             //首先获取原始梯度
9.             gradientOrig = gradientPair.getValue();
10.            //获取学习率衰减策略
11.            LearningRatePolicy decay = layer.conf().getLearningRatePolicy()
12.            ;
13.
14.            //衰减率不为0或者更新器为NESTEROVS则应用衰减策略
15.            if (decay != LearningRatePolicy.None
16.                || layer.conf().getLayer().getUpdater() == org.
17.                deeplearning4j.nn.conf.Updater.NESTEROVS)
18.                applyLrDecayPolicy(decay, layer, iteration, paramName);

```



```

15.
16.         //根据名称和网络层初始化更新器
17.         updater = init(paramName, layer);
18.         //根据原始的提取新梯度
19.         gradient2 = updater.getGradient(gradientOrig, iteration);

```

1.2.1.2 updater.getGradient(gradientOrig, iteration);

使用学习率乘以当前的梯度

```

1.     @Override
2.     public INDArray getGradient(INDArray gradient, int iteration) {
3.         return gradient.muli(learningRate);
4.     }

```

1.2.1 layerUpdaters[i].update()

在获取新地图之后继续执行以下步骤

```

1.         //使用正则化更新梯度以及参数
2.         postApply(layer, gradient2, paramName, miniBatchSize);
3.         //实现正则化之后更新梯度
4.         gradient.setGradientFor(paramName, gradient2);
5.     }

```

1.2.1.2 postApply(layer, gradient2, paramName, miniBatchSize);

实现正则化

```

1.     /**
2.      * Apply the regularization
3.      *
4.      * @param layer
5.      * @param gradient
6.      * @param param
7.      */
8.     public void postApply(Layer layer, INDArray gradient, String param, int
miniBatchSize) {
9.         NeuralNetConfiguration conf = layer.conf();
10.        INDArray params = layer.getParam(param);
11.        if (conf.isUseRegularization() && conf.getL2ByParam(param) > 0)

```

```

12.         gradient.addi(params.mul(conf.getL2ByParam(param))); //dC/dw =
           dC0/dw + lambda/n * w where C0 is pre-l2 cost function
13.         if (conf.isUseRegularization() && conf.getL1ByParam(param) > 0)
14.
           gradient.addi(Transforms.sign(params).mul(conf.getL1ByParam(param)));
15.         if (conf.isMiniBatch())
16.             gradient.divi(miniBatchSize);
17.
18.     }

```

LayerUpdater

```

1.  package org.deeplearning4j.nn.updater;
2.
3.  /**
4.   * @author Adam Gibson
5.   */
6.  public class LayerUpdater implements Updater {
7.      protected Map<String, GradientUpdater> updaterForVariable = new Lin
kedHashMap<>();
8.      protected INDArray viewArray;
9.
10.     @Override
11.     public void setStateViewArray(Layer layer, INDArray viewArray, bool
ean initialize) {
12.         //Need to split this up into each parameter type...
13.
14.         Map<String, INDArray> params = layer.paramTable();
15.         int count = 0;
16.         for (Map.Entry<String, INDArray> entry : params.entrySet()) {
17.             INDArray paramsArray = entry.getValue();
18.             GradientUpdater gu = init(entry.getKey(), layer);
19.             int thisSize = gu.stateSizeForInputSize(entry.getValue().le
ngth());
20.             if (thisSize == 0)
21.                 continue;
22.             INDArray subset = viewArray.get(NDArrayIndex.point(0), NDAr
rayIndex.interval(count, count + thisSize));
23.             gu.setStateViewArray(subset, paramsArray.shape(), paramsArr
ay.ordering(), initialize);
24.             count += thisSize;
25.         }

```

```

26.     }
27.
28.     public Map<String, GradientUpdater> getUpdaterForVariable() {
29.         return updaterForVariable;
30.     }
31.
32.     @Override
33.     public INDArray getStateViewArray() {
34.         return viewArray;
35.     }
36.
37.     @Override
38.     public int stateSizeForLayer(Layer layer) {
39.         Preconditions.checkNotNull(layer);
40.         Map<String, INDArray> params = layer.paramTable();
41.         int count = 0;
42.         for (Map.Entry<String, INDArray> entry : params.entrySet()) {
43.             GradientUpdater gu = init(entry.getKey(), layer);
44.             count += gu.stateSizeForInputSize(entry.getValue().length()
45. );
46.         }
47.         return count;
48.     }
49.
50.     @Override
51.     public void update(Layer layer, Gradient gradient, int iteration, i
52. nt miniBatchSize) {
53.         String paramName;
54.         INDArray gradientOrig, gradient2;
55.         GradientUpdater updater;
56.
57.         if (layer instanceof FrozenLayer)
58.             return;
59.
60.         preApply(layer, gradient, iteration);
61.         for (Map.Entry<String, INDArray> gradientPair : gradient.gradie
62. ntForVariable().entrySet()) {
63.             paramName = gradientPair.getKey();
64.             if (!layer.conf().isPretrain() && PretrainParamInitializer.
65.     VISIBLE_BIAS_KEY.equals(paramName.split("_")[0]))
66.                 continue;
67.             gradientOrig = gradientPair.getValue();
68.             LearningRatePolicy decay =
69. layer.conf().getLearningRatePolicy();
70.             if (decay != LearningRatePolicy.None

```

```

66.         || layer.conf().getLayer().getUpdater() ==
org.deeplearning4j.nn.conf.Updater.NESTEROVS)
67.         applyLrDecayPolicy(decay, layer, iteration, paramName);
68.         updater = init(paramName, layer);
69.         gradient2 = updater.getGradient(gradientOrig, iteration);
70.         postApply(layer, gradient2, paramName, miniBatchSize);
71.         gradient.setGradientFor(paramName, gradient2);
72.     }
73. }
74.
75. /**
76.  * Apply the regularization
77.  *
78.  * @param layer
79.  * @param gradient
80.  * @param param
81.  */
82. public void postApply(Layer layer, INDArray gradient, String param,
int miniBatchSize) {
83.     NeuralNetConfiguration conf = layer.conf();
84.     INDArray params = layer.getParam(param);
85.     if (conf.isUseRegularization() && conf.getL2ByParam(param) > 0)
86.         gradient.addi(params.mul(conf.getL2ByParam(param))); //dC/d
w = dC0/dw + lambda/n * w where C0 is pre-l2 cost function
87.     if (conf.isUseRegularization() && conf.getL1ByParam(param) > 0)
88.         gradient.addi(Transforms.sign(params).muli(conf.getL1ByPara
m(param)));
89.     if (conf.isMiniBatch())
90.         gradient.divi(miniBatchSize);
91. }
92.
93.
94. /**
95.  * Update momentum if schedule exist
96.  */
97. public void applyMomentumDecayPolicy(Layer layer, int iteration, St
ring variable) {
98.     NeuralNetConfiguration conf = layer.conf();
99.     if (conf.getLayer().getMomentumSchedule().containsKey(iteration
)) {
100.
conf.getLayer().setMomentum(conf.getLayer().getMomentumSchedule().get(
iteration));
101.         if (updaterForVariable.get(variable) != null) {
102.

```

```

103.         updaterForVariable.get(variable).update(conf.getLearningRateByParam(va
            riable),
104.             conf.getLayer().getMomentumSchedule().ge
105.             t(iteration));
106.         }
107.         } else if (updaterForVariable.get(variable) != null) {
108.
109.         updaterForVariable.get(variable).update(conf.getLearningRateByParam(va
            riable),
110.             conf.getLayer().getMomentum());
111.         }
112.     }
113.
114.     /**
115.      * Update learning rate based on policy
116.      */
117.     public void applyLrDecayPolicy(LearningRatePolicy decay, Layer laye
118.     r, int iteration, String variable) {
119.         NeuralNetConfiguration conf = layer.conf();
120.         double decayRate = layer.conf().getLrPolicyDecayRate();
121.         double lr = conf.getLearningRateByParam(variable);
122.         switch (decay) {
123.             case Exponential:
124.                 conf.setLearningRateByParam(variable, lr * Math.pow(dec
125.                 ayRate, iteration));
126.                 break;
127.             case Inverse:
128.                 conf.setLearningRateByParam(variable,
129.                     lr / Math.pow((1 + decayRate * iteration
130.                     ), conf.getLrPolicyPower()));
131.                 break;
132.             case Step:
133.                 conf.setLearningRateByParam(variable,
134.                     lr * Math.pow(decayRate, Math.floor(iter
135.                     ation / conf.getLrPolicySteps())));
136.                 break;
137.             case TorchStep:
138.                 if (iteration > 1 && conf.getLrPolicySteps() %
139.                 iteration == 0)
140.                     conf.setLearningRateByParam(variable, lr *
141.                     decayRate);
142.                 break;
143.             case Poly:
144.                 conf.setLearningRateByParam(variable, lr * Math
145.                     .pow((1 - ((double) iteration) / conf.ge

```

```

tNumIterations()), conf.getLrPolicyPower()));
137.         break;
138.         case Sigmoid:
139.             conf.setLearningRateByParam(variable,
140.                 lr / (1 + Math.exp(-decayRate * (iteration - conf.getLrPolicySteps()))));
141.             break;
142.         case Schedule:
143.             if (conf.getLayer().getLearningRateSchedule().containsKey(iteration))
144.                 conf.setLearningRateByParam(variable, conf.getLayer().getLearningRateSchedule().get(iteration));
145.             break;
146.     }
147.     if (layer.conf().getLayer().getUpdater() == org.deeplearning4j.nn.conf.Updater.NESTEROVS) {
148.         applyMomentumDecayPolicy(layer, iteration, variable);
149.     } else if (updaterForVariable.get(variable) != null) {
150.
151.         updaterForVariable.get(variable).update(conf.getLearningRateByParam(variable));
152.     }
153.
154.     /**
155.      * Apply gradient normalization: scale based on L2, clipping etc.
156.      * RenormalizeL2PerLayer: divide all layer gradients by L2 to rescale
157.      * RenormalizeL2PerParamType: divide each parameter type gradient in a layer by L2 to rescale
158.      * ClipElementWiseAbsoluteValue: clip gradients per-element
159.      * ClipL2PerLayer: same as RenormalizeL2PerLayer but limited by gradient L2 norm for the layer meeting a threshold
160.      * ClipL2PerParamType: same as RenormalizeL2PerParamType but limited by gradient L2 norm for each parameter type in a layer meeting a threshold
161.      */
162.     public void preApply(Layer layer, Gradient gradient, int iteration)
163.     {
164.         GradientNormalization normalization = layer.conf().getLayer().getGradientNormalization();
165.         if (normalization == null || normalization == GradientNormalization.None || layer.conf().isPretrain())
166.             return; //no op

```

```

167.
168.         final double threshold = layer.conf().getLayer().getGradientNor
malizationThreshold();
169.
170.         switch (normalization) {
171.             case RenormalizeL2PerLayer:
172.                 double sumSquares = 0.0;
173.                 for (INDArray g : gradient.gradientForVariable().values
()) {
174.                     double l2 = g.norm2Number().doubleValue();
175.                     //l2 norm: sqrt(sum_i g_i^2)
176.                     sumSquares += l2 * l2;
177.                 }
178.                 double layerL2 = FastMath.sqrt(sumSquares);
179.                 for (INDArray g : gradient.gradientForVariable().values
()) {
180.                     g.divi(layerL2);
181.                 }
182.                 break;
183.             case RenormalizeL2PerParamType:
184.                 for (INDArray g : gradient.gradientForVariable().values
()) {
185.                     double l2 = Nd4j.getExecutioner().execAndReturn(new
Norm2(g)).getFinalResult().doubleValue();
186.                     g.divi(l2);
187.                 }
188.                 break;
189.             case ClipElementWiseAbsoluteValue:
190.                 for (INDArray g : gradient.gradientForVariable().values
()) {
191.                     BooleanIndexing.replaceWhere(g, threshold,
Conditions.greaterThan(threshold));
192.                     BooleanIndexing.replaceWhere(g, -threshold,
Conditions.lessThan(-threshold));
193.                 }
194.                 break;
195.             case ClipL2PerLayer:
196.                 double sumSquares2 = 0.0;
197.                 for (INDArray g : gradient.gradientForVariable().values
()) {
198.                     double l2 = Nd4j.getExecutioner().execAndReturn(new
Norm2(g)).getFinalResult().doubleValue();
199.                     //l2 norm: sqrt(sum_i g_i^2)
200.                     sumSquares2 += l2 * l2;
201.                 }

```

```

202.         double layerL22 = FastMath.sqrt(sumSquares2);
203.         if (layerL22 > threshold) {
204.             double scalingFactor = threshold / layerL22; // g =
g / l2 * threshold ->
205.             for (INDArray g : gradient.gradientForVariable().va
lues()) {
206.                 g.muli(scalingFactor);
207.             }
208.         }
209.         break;
210.     case ClipL2PerParamType:
211.         for (INDArray g : gradient.gradientForVariable().values
()) {
212.             double l2 = g.norm2Number().doubleValue();
213.             if (l2 > threshold) {
214.                 double scalingFactor = l2 / threshold;
215.                 g.divi(scalingFactor);
216.             }
217.         }
218.         break;
219.     default:
220.         throw new RuntimeException(
221.             "Unknown (or not implemented) gradient
normalization strategy: " + normalization);
222.     }
223. }
224.
225.
226. public void init() {
227.     //No op
228. }
229.
230. public GradientUpdater init(String variable, Layer layer) {
231.     GradientUpdater updater = updaterForVariable.get(variable);
232.     if (updater == null) {
233.         org.deeplearning4j.nn.conf.Updater u = layer.conf().getLaye
r().getUpdaterByParam(variable);
234.         switch (u) {
235.             case SGD:
236.                 updater = new
org.nd4j.linalg.learning.Sgd(layer.conf().getLearningRateByParam(variable));
237.                 break;
238.             case ADAM:
239.                 updater = new

```



```

240.         Adam(layer.conf().getLearningRateByParam(variable),
                layer.conf().getLayer().getAdamMeanDe
ecay(),
241.                layer.conf().getLayer().getAdamVarDe
cay(), layer.conf().getLayer().getEpsilon());
242.         break;
243.         case ADADELTA:
244.             updater = new
AdaDelta(layer.conf().getLayer().getRho(),
layer.conf().getLayer().getEpsilon());
245.             break;
246.         case NESTEROVS:
247.             updater = new Nesterovs(layer.conf().getLayer().get
Momentum(),
248.                layer.conf().getLearningRateByParam(
variable));
249.             break;
250.         case ADAGRAD:
251.             updater = new
AdaGrad(layer.conf().getLearningRateByParam(variable),
252.                layer.conf().getLayer().getEpsilon()
);
253.             break;
254.         case RMSPROP:
255.             updater = new
org.nd4j.linalg.learning.RmsProp(layer.conf().getLearningRateByParam(v
ariable),
256.                layer.conf().getLayer().getRmsDecay(
), layer.conf().getLayer().getEpsilon());
257.             break;
258.         case NONE:
259.             updater = new NoOpUpdater();
260.             break;
261.         case CUSTOM:
262.             throw new UnsupportedOperationException("Custom upd
aters: not yet implemented");
263.         default:
264.             throw new IllegalArgumentException("Unknown
updater: " + u);
265.     }
266.     updaterForVariable.put(variable, updater);
267. }
268. return updater;
269. }
270.

```

```

271.         @Override
272.         public boolean equals(Object other) {
273.             if (!(other instanceof LayerUpdater))
274.                 return false;
275.             return updaterForVariable.equals(((LayerUpdater) other).updater
ForVariable);
276.         }
277.
278.         @Override
279.         public int hashCode() {
280.             int result = 19;
281.             result = 31 * result + (updaterForVariable == null ? 0 : update
rForVariable.hashCode());
282.             return result;
283.         }
284.
285.         @Override
286.         public Updater clone() {
287.             Map<String, GradientUpdater> newMap = new HashMap<>();
288.             for (Map.Entry<String, GradientUpdater> entry :
updaterForVariable.entrySet()) {
289.                 newMap.put(entry.getKey(), entry.getValue().getAggregator(t
rue).getUpdater());
290.             }
291.
292.             LayerUpdater updater;
293.             try {
294.                 updater = this.getClass().getConstructor().newInstance();
295.             } catch (Exception e) {
296.                 throw new RuntimeException(e);
297.             }
298.             updater.updaterForVariable = newMap;
299.             return updater;
300.         }
301.     }

```

applyLrDecayPolicy

```

1.  /**
2.      *   Update learning rate based on policy
3.      */
4.      public void applyLrDecayPolicy(LearningRatePolicy decay, Layer laye

```

```

r, int iteration, String variable) {
5.     NeuralNetConfiguration conf = layer.conf();
6.     double decayRate = layer.conf().getLrPolicyDecayRate();
7.     double lr = conf.getLearningRateByParam(variable);
8.     switch (decay) {
9.         case Exponential:
10.            conf.setLearningRateByParam(variable, lr * Math.pow(decayRate, iteration));
11.            break;
12.        case Inverse:
13.            conf.setLearningRateByParam(variable,
14.                lr / Math.pow((1 + decayRate * iteration), conf.getLrPolicyPower()));
15.            break;
16.        case Step:
17.            conf.setLearningRateByParam(variable,
18.                lr * Math.pow(decayRate, Math.floor(iteration / conf.getLrPolicySteps())));
19.            break;
20.        case TorchStep:
21.            if (iteration > 1 && conf.getLrPolicySteps() % iteration == 0)
22.                conf.setLearningRateByParam(variable, lr * decayRate);
23.            break;
24.        case Poly:
25.            conf.setLearningRateByParam(variable, lr * Math
26.                .pow((1 - ((double) iteration) / conf.getNumIterations()), conf.getLrPolicyPower()));
27.            break;
28.        case Sigmoid:
29.            conf.setLearningRateByParam(variable,
30.                lr / (1 + Math.exp(-decayRate * (iteration - conf.getLrPolicySteps()))));
31.            break;
32.        case Schedule:
33.            if (conf.getLayer().getLearningRateSchedule().containsKey(iteration))
34.                conf.setLearningRateByParam(variable, conf.getLayer().getLearningRateSchedule().get(iteration));
35.            break;
36.    }
37.    if (layer.conf().getLayer().getUpdater() == org.deeplearning4j.nn.conf.Updater.NESTEROVS) {
38.        applyMomentumDecayPolicy(layer, iteration, variable);
39.    } else if (updaterForVariable.get(variable) != null) {

```

```
40.     updaterForVariable.get(variable).update(conf.getLearningRateByParam(va
41.         riable));
42.     }
```