Improve Optimizer Performance based on Scale-up Gradient Implemented for the First Layer

In computer vision, it is natural to pursue a localized data representation. This property is reached by describing each image as a collection of low-dimensional feature vectors (through small image patches or projection into a linear feature space). In a better speech, the objective is producing a compact representation which depends on fewer parameters. Kernel-based classifiers, such as support vector machine (SVM), mostly use Gaussian and polynomial kernels, which suffer from poor determination on probabilistic distribution. According to this limitation, types of kernels are proposed such as Fisher, TOP, Probability Product and Kullback-Leibler. Generally, we have two types of kernels: - Metric kernels - Probability kernels. For a kernel transformation, positive definiteness is an interpretation's central key.

One important property of a kernel is measuring the similarity of two points in the feature space. One point is the pixel, and another is the kernel element. Autocorrelation can take probability distribution in a wide range of values, but due to heavy computation of high dimensionality data, makes kernel matrices ill-conditioned.

Kernel properties say that, If K is a kernel and ,$\alpha > 0$ then αK is also a Kernel; also $K + K$ is a kernel. Correlation kernel methods can be considered as an extension of the linear kernel by this formula:

$$K_L(x_i, x_j) = < x_i, x_j > = x_i^T x_j$$

The Jensen-Shannon (JS) divergence [182], measures two samples from the same source of distribution p and q by this formula:

$$JS\ (p||q) = \frac{1}{2}\ KL\ (p||r) + \frac{1}{2}\ KL\ (q||r)$$

Where $r(x) = \frac{1}{2}\ p(x) + \frac{1}{2}\ q(x)$. This divergence is meant as the distribution average. One popular metric kernel method is the Gaussian/RBF, as a de-facto kernel in machine learning defined by:

$$K\ (x_i, x_j) = \exp(-y||x_i - x_j||^2)$$

A probabilistic kernel is defined as mapping $K: P \times P \rightarrow R$ where P is the space of probability distribution. In the probability product kernel [183], the corresponding kernel is determined by :

$$K(p,q) = \int P(x)^P q(x)^P dx$$

If $P \approx {}^{1}\!/_{2}$, called Bhattacharyya kernel and $P \approx 1$ named expected likelihood kernel or correlation kernel, which measures the correlation of distributions.
A kernel "X" can be presented as Gram Matrix as follows:

$$K \equiv X \circledast X . W$$

Where w: weights and, according to Mercer's theorem, X is a valid Gram Matrix (inner product of matrix). This kernel works based on a geometric information divergence measurement [184]. it's symmetric, positive, semi-definite. Gram is simply rewritten:

$$G_{ij} = X^{(i)^T} X^{(j)}$$

In feature space, the formula is replaced with the feature map parameter:

$$\emptyset : G_{ij} = \emptyset(X^{(i)})^T \emptyset(X^{(j)})$$

kernel function allows applying pattern recognition algorithm to the input with indirect inner-product evaluation or dot-scaled product to produce bilinear maps [185]. The method implements a linear kernel defined by the dot product between two tensors:

$$K(X, X') = X^T X'$$

We implied a probabilistic kernel as a function: $K: X \times X \rightarrow R$, like SVM, to maximize margin hyperplane in a transformed feature space. So, the dimension of input space and feature space is the same. It's helpful for image classification tasks which use high-dimensional data. Linear kernels are suitable for objects represented by fixed lengths for vectors.

In the proposed normalized kernel method to determine the correlation of two distributions, we compute covariance by Gram Matrix (b set of vectors, $S = \{ x_1, \dots, x_t\}$, the Gram matrix (inner product) with a vital role in dual learning is determined as the $G \times G$ (Matrix Multiplication) whose entries are $G_{ij} = < x_i, x_j >$. If we use a kernel to express the Gram matrix in feature space with a feature map $\emptyset$ as : $G_{ij} = < \emptyset(x_i), \emptyset(x_j) > = k(x_i, x_j)$ then is referred to a kernel matrix by displaying the below standard structure:

K	1	2	…	L
1	$k(x_1, x_1)$	$k(x_1, x_2)$	…	$k(x_1, x_L)$
2	$k(x_2, x_1)$	$k(x_2, x_2)$	…	$k(x_2, x_L)$
:	:	:	⋱	:
L	$k(x_L, x_1)$	$k(x_1, x_2)$	…	$k(x_L, x_L)$

And make the transferring operation complete by choosing the mean as ρ (parameter) . The mean parameter as a primal constraint for normalization applied, because of the serious problem in unnormalized kernel methods. The final formula to find effective correlation is defined by :

$$K = \frac{1}{Mean} . (G \times G^T)$$

This algorithm can be seen as power iteration A in :

$$x_{t+1} = Ax_t$$

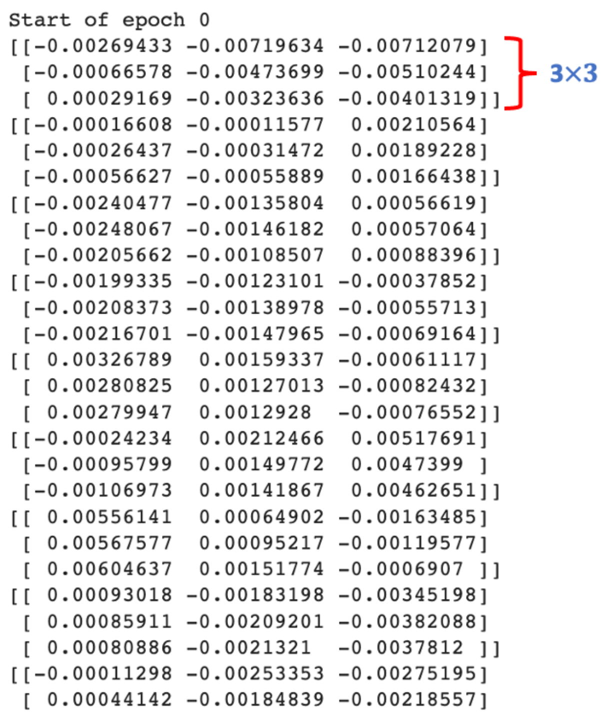
which plays the important role of momentum. Eigendecomposition of A is considered by: $A = \sum_{i=1}^{n} \lambda_i u_i u_i^T$. This method is similar to kernelized feature vector methodology by projecting x data into a new D-dimensional feature space. The feature vector x only enters the model to compare similarity with another feature vector $x^T x'$.

The nature of positive definiteness is automatically obeyed by kernels (often non-negative $K(x,y) \geq 0$, [4]). Evaluating kernel functions on finite sets leads to positive semidefinite matrice by definition, and covariance matrices are positive semidefinite. The theory behind mean multiplication into vectors is limiting spectral density by isolated large eigenvalues [186]. We avoid forcing kernels to be positive definiteness since it decreases kernel expressiveness [184] and damage classification accuracy. We exploit a mercer kernel or positive definite kernel.

The proposed method's algorithm to use two optimizers jointly for training is in the following. Also, we see results of this method implemented for the first layer on two popular image classification datasets respectively: CIFAR10, CIFAR100. The superiority of the proposed method is demonstrated compared to standard optimizers SGD, ADAM, and ADAMAX. We see improvements in the optimizer's accuracy performance when this method is added to the optimization process with a negligible time consumption increase.

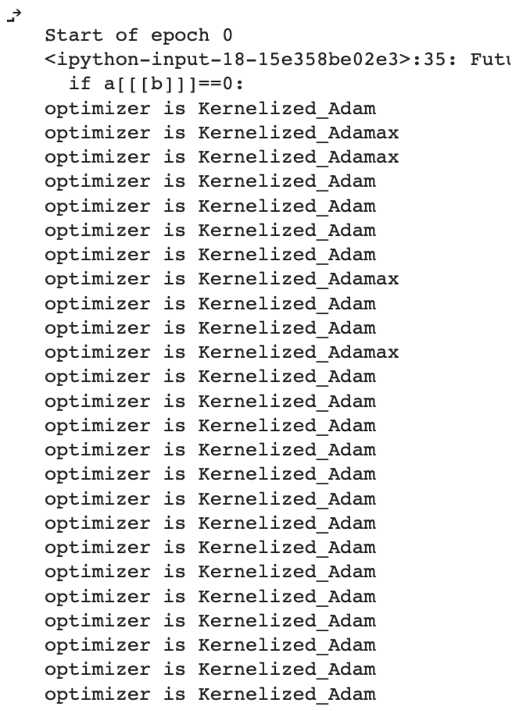
Each Gradient Shape specified to a Layer

Kernel Size	Input Channels	Output Channels



```
Start of epoch 0
[[-0.00269433 -0.00719634 -0.00712079]
 [-0.00066578 -0.00473699 -0.00510244]
 [ 0.00029169 -0.00323636 -0.00401319]]
[[-0.00016608 -0.00011577  0.00210564]
 [-0.00026437 -0.00031472  0.00189228]
 [-0.00056627 -0.00055889  0.00166438]]
[[-0.00240477 -0.00135804  0.00056619]
 [-0.00248067 -0.00146182  0.00057064]
 [-0.00205662 -0.00108507  0.00088396]]
[[-0.00199335 -0.00123101 -0.00037852]
 [-0.00208373 -0.00138978 -0.00055713]
 [-0.00216701 -0.00147965 -0.00069164]]
[[ 0.00326789  0.00159337 -0.00061117]
 [ 0.00280825  0.00127013 -0.00082432]
 [ 0.00279947  0.0012928  -0.00076552]]
[[-0.00024234  0.00212466  0.00517691]
 [-0.00095799  0.00149772  0.0047399 ]
 [-0.00106973  0.00141867  0.00462651]]
[[ 0.00556141  0.00064902 -0.00163485]
 [ 0.00567577  0.00095217 -0.00119577]
 [ 0.00604637  0.00151774 -0.0006907 ]]
[[ 0.00093018 -0.00183198 -0.00345198]
 [ 0.00085911 -0.00209201 -0.00382088]
 [ 0.00080886 -0.0021321  -0.0037812 ]]
[[-0.00011298 -0.00253353 -0.00275195]
 [ 0.00044142 -0.00184839 -0.00218557]
```

4. 1 Gradient Tensor Shape



```
Start of epoch 0
<ipython-input-18-15e358be02e3>:35: Futu
  if a[[[b]]]==0:
optimizer is Kernelized_Adam
optimizer is Kernelized_Adamax
optimizer is Kernelized_Adamax
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adamax
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adamax
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
optimizer is Kernelized_Adam
```

4. 2 Train Model with Two Kernelized Different Optimizers