Catastrophic inerference Method

Monday, June 3, 2024 3:32 PM

https://arxiv.org/pdf/1611.07725
iCaRL: Incremental Classifier and Representation Learning"

Overview:

iCaRL is a method designed to handle class-incremental learning. It allows a neural network to learn new classes over time without forgetting previously learned classes, a problem known as catastrophic forgetting.

Key Components:

1. Nearest-Mean-of-Exemplars Classification:

- Each class is represented by a set of exemplars (sample images).
- For each class, compute a prototype vector, which is the mean of the feature vectors of its exemplars.
- To classify a new image, compute its feature vector and assign it the label of the nearest class prototype.

2. Exemplar Management:

- Exemplar Selection: Choose representative exemplars for new classes.
- Exemplar Reduction: When the memory limit is reached, reduce the number of exemplars by keeping the most representative ones.

3. Representation Learning with Knowledge Distillation:

- Combine new training data with stored exemplars to update the neural network.
- Use a loss function that includes classification loss for new data and distillation loss to preserve old knowledge.

Algorithm Steps:

1. Initialization:
Initialize the neural network and set the memory size K for storing exemplars.

2. Receive New Class Data:
Obtain training examples Xs …. X_t for new classes s to t.

3. Update Representation and Classifiers:

- Combine new training examples with existing exemplar sets to form an augmented training set.
- Store network outputs for previous classes.
- Update the network using a combined loss function (classification and distillation losses).

4. Adjust Exemplars:
Calculate the number of exemplars per class $m = \frac{K}{t}$

5. Reduce Exemplar Sets for Previous Classes:
For each previous class, reduce the exemplar set to size m.

6. Construct New Exemplar Sets:
For each new class, construct an exemplar set of size m.

7. Classify New Instances:
- Compute the feature vector for a new image.
- Calculate the prototype vector for each class.
- Assign the class label with the nearest prototype to the new image's feature vector.

Experiment and Results

1. Datasets:
CIFAR-100: Consists of 100 classes, each with 600 images.
ImageNet ILSVRC 2012: Consists of 1,000 classes with a large number of images per class.

2. Experiment:

Classes in the datasets are split into batches and processed incrementally.
After each batch, the classifier is evaluated on the test set considering only the classes seen so far.

CIFAR-100:
- 32-layer ResNet.
- Max number of exemplars K = 2000.
- Each training step consists of 70 epochs.
- Learning rate starts at 2.0 and is divided by 5 at 49 and 63 epochs.

On CIFAR-100, iCaRL was tested with different batch sizes (2, 5, 10, 20, 50 classes per batch). It consistently outperformed other methods like fine-tuning, fixed representation, and LwF.MC (Learning without Forgetting).

ImageNet ILSVRC 2012:
- 18-layer ResNet.
- Max number of exemplars K = 20000K=20000.
- Each training step consists of 60 epochs.
- Learning rate starts at 2.0 and is divided by 5 at 20, 30, 40, and 50 epochs.

On ImageNet, iCaRL was tested on subsets of 100 and 1,000 classes. It demonstrated higher top-5 accuracy compared to other methods.

Method's Use case
Class-Incremental Learning:
- When new classes of data are introduced sequentially, and the model needs to learn these new classes without forgetting the previous ones.

Limited Memory:

https://arxiv.org/pdf/1903.08671
Gradient Based Sample Selection for Online Continual Learning

Overview:

To overcome catastrophic forgetting in neural networks by using a replay buffer to store previous data for rehearsal. The proposed method optimizes the selection of samples in the replay buffer to maintain performance on previously learned tasks.

Key Components:

1. Formulation as a Constraint Reduction Problem:

The sample selection problem is framed as reducing the number of constraints in a constrained optimization problem. Each constraint ensures that the new model does not forget previously learned information by maintaining diversity in the replay buffer.

2. Maximizing Diversity with Gradient Information:

Sample selection is achieved by maximizing the diversity of the samples in the replay buffer using the gradients of the model parameters as features. The idea is that diverse gradient information leads to better generalization and retention of previously learned tasks.

3. Greedy Algorithm for Efficiency:

A greedy algorithm is proposed as an efficient and scalable method for sample selection. It iteratively selects samples that maximize gradient diversity until the buffer is full.

Algorithm Steps:

1. Initialization:
- Initialize the neural network model.
- Set the size of the replay buffer M.

2. Receive New Data:
- Continuously receive new data samples (x,y).

3. Compute Gradients:
- For each new sample (xi,yi), compute the gradient of the loss function ∇ L(xi,yi) with respect to the model parameters.

4. Update Replay Buffer:

Selection Strategy:
- Calculate the cosine similarity between the gradient of the new sample and the gradients of samples already in the buffer.
- Select the sample that maximizes the gradient diversity (i.e., has the lowest cosine similarity with existing samples).

Greedy Algorithm:
- Initialize an empty buffer.
- For each new sample, add it to the buffer if it maximizes the diversity of gradients, up to the buffer size limit.

5. Model Training:
- Train the model using both the new data and the samples stored in the replay buffer. This helps in balancing the learning of new information while retaining previously learned tasks.

Findings of the Paper
- The proposed method (GSS-Greedy) outperforms other sample selection strategies, such as random sampling and reservoir sampling, in maintaining model accuracy on previously learned tasks.
- GSS-Greedy demonstrates significant improvements in continual learning scenarios, especially where task boundaries are not clearly defined.
- The method is efficient and scalable, making it suitable for real-world online learning applications.

Experiment and Results
1. Datasets

- Disjoint MNIST: The MNIST dataset is divided into 5 tasks, with each task containing images of two digits (e.g., Task 1: digits 0-1, Task 2: digits 2-3, etc.).
- Permuted MNIST: The MNIST dataset with 10 different random permutations of the pixel positions, resulting in 10 different tasks.
- Disjoint CIFAR-10: The CIFAR-10 dataset split into 5 tasks based on labels, with each task containing images of two classes (e.g., Task 1: classes 0-1, Task 2: classes 2-3, etc.).

- The datasets are processed sequentially, meaning the model encounters tasks one by one.
- A fixed batch size of 10 samples is used during training.
- Few iterations (1-5) are performed over each batch to simulate a real-time data stream.
- The performance of the model is evaluated by the average test accuracy on all tasks seen so far.

Implementation Details:

Disjoint and Permuted MNIST:
- Model: A two-layer neural network with 100 neurons in each layer.
- Learning Rate: 0.05.
- Buffer Size: 300 samples.

- Result:
  - Disjoint MNIST
  - Buffer Size 300: GSS-Greedy achieved an average test accuracy of 82.6%.
  - Buffer Size 400: GSS-Greedy achieved an average test accuracy of 84.6%.
  - Buffer Size 500: GSS-Greedy achieved an average test accuracy of 84.8%.

Method's Use case

Class-Incremental Learning:

- When new classes of data are introduced sequentially, and the model needs to learn these new classes without forgetting the previous ones.

Limited Memory:

- When there is limited memory to store past data, as iCaRL efficiently manages memory by storing a fixed number of exemplars per class.

Periodic Updates:

- When updates are periodic rather than continuous, such as when new classes of objects or conditions are added in batches.

Need for Exemplar Management:

- When it's beneficial to keep a small, representative subset of previous data to maintain knowledge of old classes.

Limitations of the Method

Memory Constraints:

Although iCaRL manages memory efficiently, the fixed memory size for exemplars can limit the number of classes it can handle effectively.

Computational Overhead:

The need to update exemplars and compute prototype vectors adds computational overhead, which might be significant for very large datasets or real-time applications.

Representation Drift:

Over time, as more classes are added, the feature representation might drift, potentially degrading the performance on older classes despite the use of knowledge distillation.

Dependence on Exemplars:

The quality of the exemplars significantly affects the performance. Poor selection of exemplars might lead to suboptimal representation and classification.

https://openaccess.thecvf.com/content_CVPR_2019/papers/Hou_Learning_a_Unified_Classifier_Incrementally_via_Rebalancing_CVPR_2019_paper.pdf

LUCIR: Learning a unified classifier incrementally via rebalancing

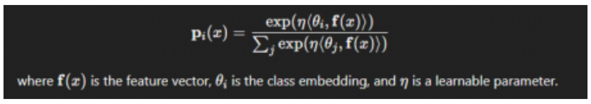
Overview:

The proposed framework for incremental learning consists of three major components to handle the imbalance between old and new classes effectively.

Key Components:

1. Cosine Normalization:

- To ensure the weight vectors for all classes have similar magnitudes, mitigating the bias towards new classes.
- Implementation:
  - Normalize the class embeddings and features using L2-normalization.
  - Compute the predicted probability using the cosine similarity between the normalized vectors.
  - Introduce a learnable scalar η to control the sharpness of the softmax distribution.

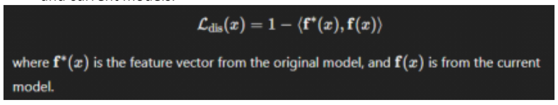


$$\mathbf{p}_i(x) = \frac{\exp(\eta\langle\theta_i, \mathbf{f}(x)\rangle)}{\sum_j \exp(\eta\langle\theta_j, \mathbf{f}(x)\rangle)}$$

where $\mathbf{f}(x)$ is the feature vector, θ_i is the class embedding, and η is a learnable parameter.

2. Less-Forget Constraint:

- To maintain the geometric configuration of old class embeddings and features.
- Implementation:
  - Introduce a distillation loss that compares the current model's feature orientations to those of the original model.
  - Use l2-normalization for the features and class embeddings.
  - Compute the distillation loss as the cosine similarity between the normalized features from the original and current models.



$$\mathcal{L}_{\text{dis}}(x) = 1 - \langle \mathbf{f}^*(x), \mathbf{f}(x)\rangle$$

where $\mathbf{f}^*(x)$ is the feature vector from the original model, and $\mathbf{f}(x)$ is from the current model.

3. Inter-Class Separation:

- Objective: To ensure a clear margin between old and new class embeddings, reducing ambiguities.

- Result:
  - Disjoint MNIST
  - Buffer Size 300: GSS-Greedy achieved an average test accuracy of 82.6%.
  - Buffer Size 400: GSS-Greedy achieved an average test accuracy of 84.6%.
  - Buffer Size 500: GSS-Greedy achieved an average test accuracy of 84.8%.

  - Permuted MNIST:
  - GSS-Greedy performed better than other methods, achieving an average accuracy of 77.3%.

Disjoint CIFAR-10:

- Model: ResNet18.
- Learning Rate: 0.01.
- Buffer Size: 1,000 samples.

- Result:
  - GSS-Greedy significantly outperformed random sampling, achieving an average accuracy of 33.56%.

Method's Use case

Streaming Data Learning:

- When data arrives continuously, and the model must update incrementally without access to the entire dataset.

Maintaining Performance on Previous Tasks:

- When it's crucial to avoid catastrophic forgetting and maintain performance on previously learned tasks while learning from new data.

Continuous Real-Time Updates:

- When the system needs to handle real-time data and updates continuously, selecting the most informative samples dynamically.

High-Impact Sample Selection:

- When it's important to choose samples that will have the most significant impact on improving the ~~model's~~ performance.

Limitations of the Method

Computational Overhead:

The need to compute gradients for each sample and perform the selection process adds computational complexity, which can be significant for large datasets or real-time applications.

Scalability:

While the greedy algorithm improves efficiency, the method might still face challenges when scaling to extremely large datasets or when operating under strict real-time constraints.

Dependency on Gradient Information:

The effectiveness of the method relies on the quality of gradient information. If the gradients do not capture relevant features for diversity, the performance may degrade.

Memory Constraints:

The fixed size of the replay buffer limits the number of samples that can be stored, potentially affecting the model's ability to remember older tasks as more tasks are learned. This constraint may lead to suboptimal performance if not managed properly.

- Implementation:
    - Use reserved samples of old classes as anchors.
    - Identify hard negative samples from new classes that have high responses to the old class samples.
    - Introduce a margin ranking loss to separate the old class embeddings from the hard negatives.

$$\mathcal{L}_{\mathrm{mr}}(x) = \sum_{k=1}^{K} \max(m - \langle \theta(x), \mathbf{f}(x) \rangle + \langle \theta_k, \mathbf{f}(x) \rangle, 0)$$

where $\theta(x)$ is the ground-truth class embedding, θ_k are the hard negative class embeddings, and m is the margin.

4. Inter-Class Separation:

- Combine all the losses to form the final loss function:

$$\mathcal{L} = \frac{1}{|N|} \sum_{x \in N} (\mathcal{L}_{\mathrm{ce}}(x) + \lambda \mathcal{L}_{\mathrm{dis}}(x)) + \frac{1}{|N_o|} \sum_{x \in N_o} \mathcal{L}_{\mathrm{mr}}(x)$$

- N is the training batch, N_o are the reserved old samples, and λ is an adaptive weight.

Experiment and Results

1. Datasets:

- CIFAR-100: Contains 60,000 images across 100 classes, with 500 images per class for training and 100 images per class for evaluation.
- ImageNet: Contains 1.2 million training images and 50,000 validation images across 1,000 classes.

2. Experiment:

Models are pre-trained on half of the classes, with the remaining classes introduced incrementally.

CIFAR-100:

- With 10 incremental phases, the method shows a performance improvement of more than 6% compared to iCaRL.

ImageNet ILSVRC 2012:

- With 10 incremental phases on the full dataset, the method shows a performance improvement of more than 13% compared to iCaRL.

Method's Use case

The method is applicable in scenarios where models need to be updated continuously with new data, such as:

- Online services with streaming data.
- Product categorization systems that need to recognize new products over time.
- Face recognition systems that need to identify new individuals.

Limitations of the Method

Dependence on Reserved Samples:

The method requires a small subset of old class samples to be stored, which may not always be feasible.

Complexity:

Balancing the training process and managing the distillation and margin ranking losses introduce additional complexity.

Scalability:

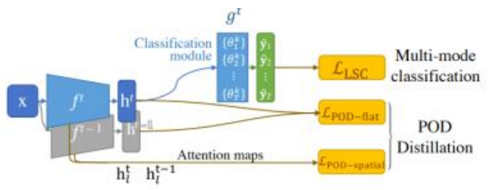
Performance may degrade with a very large number of incremental phases or when the number of new classes is very large.

https://arxiv.org/pdf/2004.13513

PODNet (Pooled Output Distillation Networks)

Overview:

PODNet is a framework designed to address the problem of catastrophic forgetting in incremental learning by using a new distillation loss named Pooled Outputs Distillation (POD) and a novel classification layer called Local Similarity Classifier (LSC).



Key Components:

1. POD (Pooled Outputs Distillation) Loss:

- To constrain the model's evolution and prevent forgetting of old classes while allowing learning of new classes.
- Different pooling strategies to balance the flexibility and retention of old knowledge.

$$\mathcal{L}_{\text{POD-final}}(\mathbf{x}) = \frac{\lambda_c}{L-1} \sum_{\ell=1}^{L-1} \mathcal{L}_{\text{POD-spatial}}\left(f_\ell^{t-1}(\mathbf{x}), f_\ell^t(\mathbf{x})\right) + \lambda_f \mathcal{L}_{\text{POD-flat}}\left(f^{t-1}(\mathbf{x}), f^t(\mathbf{x})\right) .$$

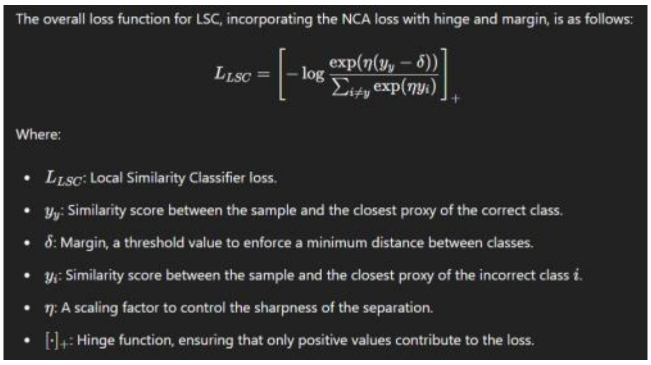
Summary of Pooling Effects

- **Pixels Pooling**: High rigidity, exact pixel-wise matching, less flexibility for learning new tasks.
- **Channel Pooling**: Moderate flexibility, allows reorganization across channels, balances feature retention and adaptation.
- **Spatial Pooling**: Preserves overall spatial distribution, moderate rigidity, suitable for spatially consistent tasks.
- **Width and Height Pooling**: Intermediate flexibility, preserves specific directional patterns, balances between

spatial and channel pooling.

1. Local Similarity Classifier (LSC):

- Uses multiple proxies per class to better capture the variations within each class.
- Improves model's robustness to shifts in feature distributions.



The overall loss function for LSC, incorporating the NCA loss with hinge and margin, is as follows:

$$L_{LSC} = \left[-\log \frac{\exp(\eta(y_y - \delta))}{\sum_{i \neq y} \exp(\eta y_i)} \right]_+$$

Where:

- L_{LSC}: Local Similarity Classifier loss.
- y_y: Similarity score between the sample and the closest proxy of the correct class.
- δ: Margin, a threshold value to enforce a minimum distance between classes.
- y_i: Similarity score between the sample and the closest proxy of the incorrect class i.
- η: A scaling factor to control the sharpness of the separation.
- $[\cdot]_+$: Hinge function, ensuring that only positive values contribute to the loss.

The similarity score yyy between a sample x and a proxy p is typically calculated as the negative Euclidean distance or cosine similarity.

Experiment and Results

1. Datasets:

- **CIFAR100**: 60,000 images of 100 classes.
- **ImageNet100**: 100-class of ImageNet1000.
- **ImageNet1000**: 1.2 million images of 1,000 classes.

2. Experiment:

Models are pre-trained on half of the classes, with the remaining classes introduced incrementally.

CIFAR-100:

- Number of tasks varies, e.g., 5, 10, 25, and 50 steps
- PODNet with Nearest-Mean-Exemplars (NME) and CNN-based classifiers outperforms previous methods, with relative improvements corelated with the number of steps.
- Example: 50 steps of 1 class each, PODNet surpasses existing models by 12.1 percentage points.

ImageNet100 and ImageNet1000:

- Similar trends with significant improvements over existing methods.
- Example: On ImageNet100, 5 steps of 10 classes each, PODNet improves accuracy by 6.51 percentage points.

Method's Use case

Suitable for applications where models need continuous updates with new data, such as:

- Online services with evolving data streams.
- Systems requiring periodic updates without complete retraining (e.g., recommendation systems, evolving classification tasks).

Limitations of the Method

- **Dependence on Memory**:
  - Requires a fixed memory budget for storing samples of old classes, which might not be feasible in all scenarios.
  - The balance between old and new class retention introduces additional complexity in model training.
- **Complexity in Training**:
  - The need for multiple loss components (POD and LSC) increases training complexity and computational requirements.
  - Performance might degrade with a very large number of incremental phases or classes if not managed properly.