**Background and Method Introduction:**

**Overview of 2-layer NN:**

Neural networks (NNs) have revolutionised various fields, including image classification, by providing powerful tools for learning representations from data. A 2-layer neural network consists of an input layer, one hidden layer, and an output layer. This architecture is a simple yet effective model for various tasks, including image classification.

In image classification, the goal is to classify images into predefined categories or classes. Each image is represented as a vector of pixel values, where each pixel represents a feature. A 2-layer neural network learns to map these input features to the corresponding class labels through a process of forward and backward propagation.

**Architecture of a 2-layer Neural Network for Image Classification:**

**Input Layer:** The input layer consists of neurons corresponding to the input features (pixels) of the image. Each neuron represents a pixel value, and the entire layer forms a vectorized representation of the input image.

**Hidden Layer:** The hidden layer is where the nonlinear transformations of the input data occur. Each neuron in the hidden layer computes a weighted sum of its inputs and applies an activation function (such as the sigmoid or ReLU function) to introduce nonlinearity. This layer learns to extract relevant features from the input data.

**Output Layer:** The output layer produces the final classification decision. It typically consists of neurons equal to the number of classes in the classification task. Each neuron represents the probability of the input belonging to a particular class. The softmax activation function is commonly used in the output layer to convert raw scores into class probabilities.

**Applications in Image Classification:**

The 2-layer neural network architecture is suitable for image classification tasks due to its simplicity and effectiveness. It can learn to extract relevant features from raw pixel data and classify images into multiple categories with high accuracy. While deeper neural network architectures like convolutional neural networks (CNNs) have largely surpassed 2 layer Neural networks in performance for image classification tasks, 2-layer NNs still serve as a fundamental building block and provide insights into the workings of neural networks. Additionally, for simpler datasets or as an educational tool, 2-layer NNs remain relevant and useful.

**Dataset and Tasks Description:**

**About CIFAR-10 Dataset:**

The CIFAR-10 dataset is a widely used benchmark dataset for image classification tasks in machine learning. It consists of 60,000 colour images, evenly distributed across 10 different object classes i.e., aeroplane, automobile (excluding trucks), bird, cat, deer, dog, frog, horse, ship, truck. Each image is 32x32 pixels in size and represented in RGB format. The dataset is further split into 50,000 training images and 10,000 test images. Additionally, there are 10,000 labelled images from the same distribution used for validation.

**Tasks:**

1. PyTorch Model with Inbuilt Functions:

* Objective: Implement image classification on the CIFAR-10 dataset using PyTorch's built-in functions and modules.
* Data Preprocessing: Loaded the CIFAR-10 dataset using torchvision and applied transformations to convert images to tensors and normalise pixel values.
* Model Architecture: Defined a neural network model using PyTorch's nn.Module class, consisting of two fully connected layers. Utilised ReLU activation for the hidden layer and softmax activation for the output layer.
* Training Procedure: Defined the loss function as CrossEntropyLoss and used stochastic gradient descent (SGD) as the optimizer. Trained the model for a specified number of epochs, performing forward and backward passes and updating model parameters.
* Evaluation: Tested the trained model on a separate test set to evaluate its classification accuracy.

2. Self-Coded Neural Network without Regularization:

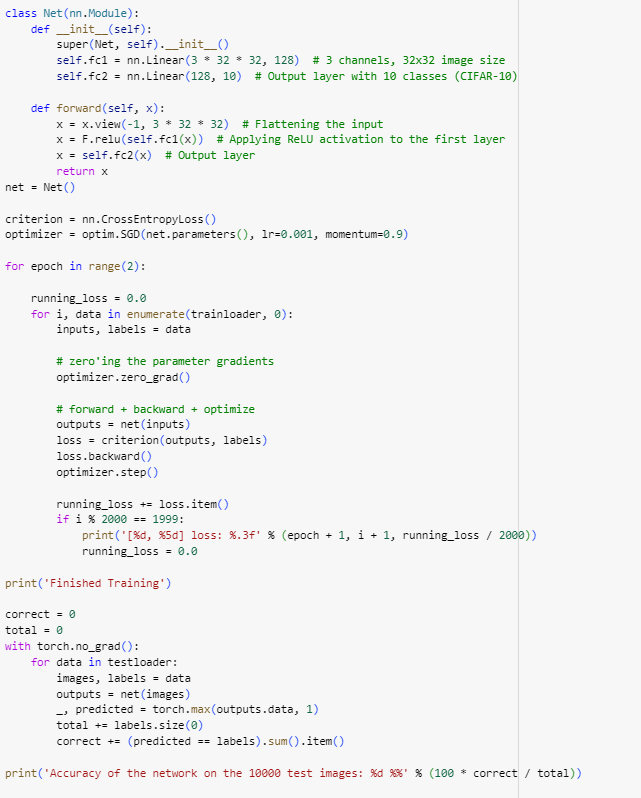
* Objective: Implement image classification on CIFAR-10 using a self-coded 2-layer neural network without regularisation.
* Data Preprocessing: Loaded the CIFAR-10 dataset and transformed it into NumPy arrays. Normalised the pixel values and prepared the data for training.
* Model Architecture: Implemented a 2-layer neural network from scratch using NumPy, consisting of an input layer, a hidden layer, and an output layer. Utilised sigmoid activation for the hidden layer and softmax activation for the output layer.
* Training Procedure: Defined the forward pass, backward pass, and loss computation functions. Trained the model by updating weights and biases through gradient descent for a specified number of epochs.
* Evaluation: Evaluated the trained model's performance by calculating its accuracy on the test set.

3. Self-Coded Neural Network with Regularization and Hyperparameter Tuning:

* Objective: Enhance the self-coded neural network by incorporating L2 regularisation and tuning hyperparameters.
* Model Architecture: Expanded the self-coded neural network to include L2 regularisation to prevent overfitting. Along with the number of hidden units as a hyperparameter.
* Training Procedure: Incorporated L2 regularisation terms into the backward pass to update weights with regularisation, along with different set of hyperparameters.
* Performed hyperparameter tuning using a grid search approach to find optimal values for hidden layer size, regularisation coefficient, learning rate and Number of epochs.
* Evaluation: Evaluated the model's performance with regularisation and tuned hyperparameters on the test set.

**Algorithms Used:**

**2 - Layer NN using pytorch inbuilt functions:**

****

**Fig.1. 2-layer NN using pytorch inbuilt functions**

**Neural Network Architecture:**

* The neural network architecture is defined using the Net class, which inherits from nn.Module.
* It consists of two fully connected layers (fc1 and fc2), where the first layer has 128 neurons and the second layer has 10 neurons (output classes).
* The input to the network is flattened to a vector of size 3 \* 32 \* 32 before passing through the fully connected layers.
* ReLU activation is applied to the output of the first fully connected layer (fc1).

**Loss Function and Optimizer:**

* Cross-entropy loss (nn.CrossEntropyLoss()) is used as the loss function, which is suitable for multi-class classification tasks.
* Stochastic Gradient Descent (SGD) optimizer (optim.SGD) is employed for updating the network parameters.
* The learning rate is set to 0.001, and trained for 2 epochs

**Training Loop:**

* The model is trained for a fixed number of epochs (2 in this case).
* Within each epoch, the training dataset is iterated over in mini-batches using a trainloader.

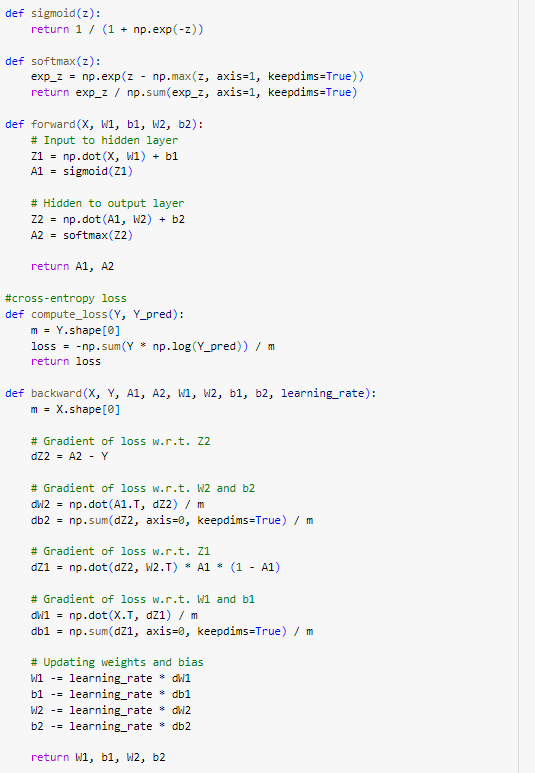
**For each mini-batch:**

* Gradients are zeroed using optimizer.zero\_grad() to clear any accumulated gradients from the previous iteration.
* Forward pass is performed by passing the input data through the network (outputs = net(inputs)).
* Loss is computed using the defined criterion (loss = criterion(outputs, labels)).
* Backpropagation is carried out using loss.backward() to compute gradients.
* Optimizer updates the network parameters based on the computed gradients using optimizer.step().

**Evaluation:**

* After training, the trained model is evaluated on the test dataset to assess its performance.
* The accuracy of the model is calculated by comparing the predicted labels with the ground truth labels (correct += (predicted == labels).sum().item()).

**Self coded version of 2-layer NN:**

****

**Fig. 2. Self coded version of 2-layer NN (without regluarisation)**

**Data Loading and Preprocessing:**

* The CIFAR-10 dataset is loaded using torchvision.
* Data transformation is applied to convert images to PyTorch tensors and normalise them.

**Neural Network Architecture:**

* The neural network consists of an input layer, a hidden layer with sigmoid activation, and an output layer with softmax activation.
* The architecture includes 3072 input neurons (32x32x3), 128 hidden neurons, and 10 output neurons (for the 10 classes in CIFAR-10).
* Weights (W1 and W2) and biases (b1 and b2) are initialised using random values.

**Forward Pass:**

* The forward pass computes the activations of the hidden layer (A1) and the output layer (A2) given the input data.

**Loss Calculation:**

* Cross-entropy loss is computed using the predicted probabilities (A2) and the ground truth labels.

**Backward Pass (Gradient Descent):**

* Gradients of the loss with respect to the parameters (weights and biases) are computed using backpropagation.
* The weights and biases are updated using gradient descent to minimise the loss.

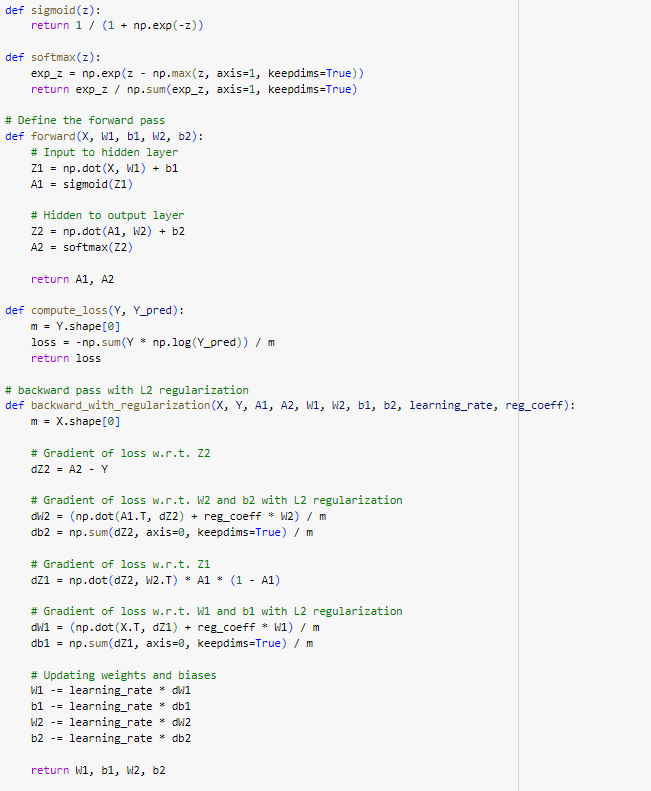
**Training Loop:**

* The training loop iterates over a fixed number of epochs.
* In each epoch, the forward and backward passes are executed on the entire training dataset.

**Testing:**

* After training, the model is evaluated on the test dataset to assess its performance.
* Test accuracy is calculated by comparing the predicted labels with the ground truth labels.

**Self coded version of 2-Layer NN with Regularisation and hyper-parameter tuning:**



**Fig. 3. Self coded version of 2 layer NN with L2 regularisation**

**Neural Network Architecture:**

* The neural network consists of two fully connected layers: an input layer and an output layer.
* Each neuron in the input layer represents a pixel in the input image, and there are 3072 neurons in total (32x32x3 for CIFAR-10).
* The output layer has 10 neurons, corresponding to the 10 classes in the CIFAR-10 dataset.

**Forward Pass:**

* During the forward pass, the input data is propagated through the network to compute the activations of the hidden layer (A1) and the output layer (A2).
* The activation function used for the hidden layer is the sigmoid function, which introduces non-linearity into the model.
* The output layer uses the softmax activation function to convert the raw output scores into probabilities representing the likelihood of each class.

**Loss Calculation with L2 Regularization:**

* The cross-entropy loss is computed using the predicted probabilities (A2) and the ground truth labels.
* To prevent overfitting, L2 regularisation is applied to the loss, which penalises large weights in the network.
* The regularisation term is added to the loss function, controlled by a regularisation coefficient.

**Backward Pass with L2 Regularization:**

* Backpropagation is used to compute the gradients of the loss with respect to the parameters (weights and biases) of the network.
* L2 regularisation is incorporated into the gradients to adjust the parameter updates and shrink the weights towards zero.
* This helps to prevent the model from becoming too complex and overfitting the training data.

**Training Loop:**

* The training loop iterates over a grid of hyperparameters, including hidden layer sizes, regularisation coefficients, learning rates, and epochs.
* For each combination of hyperparameters, the model is trained on the training dataset for a specified number of epochs.
* During training, the loss is monitored to track the convergence of the model.

**Validation:**

* After training, the model's performance is evaluated on the test dataset to assess its generalisation ability.
* Test accuracy is calculated by comparing the model's predictions with the ground truth labels.
* This step ensures that the model has learned meaningful patterns from the training data and can accurately classify unseen examples.

**Grid Search for Hyperparameter Tuning:**

* A grid search strategy is employed to systematically explore the hyperparameter space and identify the combination of hyperparameters that maximises the test accuracy.
* Different combinations of hyperparameters are tried, and the model's performance is evaluated for each combination.

**Best Parameters and Test Accuracy:**

* The grid search identifies the best combination of hyperparameters that yield the highest test accuracy.
* The best parameters, including the hidden layer size, regularisation coefficient, learning rate, and number of epochs, are reported along with the corresponding test accuracy.

**Classification Results:**

|  | **Using Pytorch functions**  **(2 epochs, SGD optimiser)** | **Self coded 2-Layer NN**  **(30 epochs)** | **2-Layer NN with L2 regularisation and hyperparameter tuning** |
| --- | --- | --- | --- |
| **Test Accuracy** | 44% | 9.8% | 34.85% |

From the above classification results, 2-layer NN implemented using pytorch function (with SGD optimiser) produced the best result with 44% accuracy, even though it is trained for only 2 epochs. Whereas my self coded 2 layer Neural Network performed very poorly with an accuracy of 9.8%. After L2 regularisation and tuning of the hyperparameters, my self coded version was able to achieve an accuracy of 34.85%. This clearly shows the importance of usage of regularisation and hyperparameter tuning. And using of pytorch functions which provide inbuilt optimisers is more efficient in terms of accuracy, speed and overall performance, as evident from above results.

**Methods of Improvement:**

**Hyperparameter Tuning:**

* Hyperparameters such as learning rate, regularisation coefficient, hidden layer size, and number of epochs are systematically varied and optimised using grid search.
* This helps to find the optimal combination of hyperparameters that maximises the model's performance on the test dataset.

**L2 Regularization:**

* L2 regularisation is applied to the loss function during training to prevent overfitting.
* It penalises large weights in the network, encouraging the model to learn simpler representations that generalise better to unseen data.

**Activation Functions:**

* Sigmoid activation function is used in the hidden layer to introduce non-linearity into the model.
* Softmax activation function is applied to the output layer to convert raw scores into probabilities, making the model suitable for multi-class classification.

**Data Preprocessing:**

* The input images are normalised to have zero mean and unit standard deviation across each channel.
* This standardisation helps in speeding up the training process and improving the convergence of the optimization algorithm.

**Grid Search:**

* A grid search strategy is employed to explore a range of hyperparameters and find the best combination that optimises the model's performance.The grid search identifies the settings that yield the highest test accuracy.