Feed Forward Model

Creating a layer

Function Signature

```
def add_dense(
    neurons: int,
    activation: str = 'sigmoid',
    dropout: float = 0.,
    l1_penalty: float = 0.,
    l2_penalty: float = 0.
) -> None:
```

Parameters

- `neurons` (int): The number of neurons in the dense layer. Required parameter.
- `activation` (str, default='sigmoid'): The activation function to use in the dense layer. Valid options are 'sigmoid', 'relu', 'leaky relu' 'tanh', 'softmax', and 'linear'
- `dropout` (float, default=0.): The dropout rate to use in the dense layer. This parameter should be a float between 0 and 1, where 0 means no dropout and 1 means all neurons are dropped.
- `l1_penalty` (float, default=0.): The L1 regularization penalty to use in the dense layer. This parameter should be a float greater than or equal to 0.

`l2_penalty` (float, default=0.): The L2 regularization penalty to use in the dense layer. This parameter should be a float greater than or equal to 0.

Return Value

This function does not return anything. It modifies the `deeprai.models.FeedForward` instance by adding a dense layer with the specified parameters.

Description

The `add_dense` function adds a dense layer to the `deeprai.models.FeedForward` instance. A dense layer is a layer of fully connected neurons where each neuron is connected to every neuron in the previous layer.

If this is the first layer added to the model, then `add_dense` will automatically treat it as an input layer and ignore any arguments other than `neurons`. This is because an input layer does not have an activation function, dropout, or regularization penalties.

If this is not the first layer added to the model, then `add_dense` will add a dense layer with the specified parameters to the model.

Examples

Here is an example of how to use the `add_dense` function:

```
from deeprai.models import FeedForward


model = FeedForward()
model.add_dense(784)
model.add_dense(128, activation='relu', dropout=0.2)
model.add_dense(64, activation='sigmoid', l2_penalty=0.001)
```

This code creates a `FeedForward` model with an input with 784 neurons, adds a dense layer with 128 neurons, a ReLU activation function, and a 20% dropout rate, and then adds another dense layer with 64 neurons, a sigmoid activation function, and an L2 regularization penalty of 0.001.

Configuring Loss/Optimizers

Function Signature

```
def config(
    optimizer: str = 'gradient descent',
    loss: str = 'mean square error'
) -> None:
```

Parameters

- `optimizer` (str, default='gradient descent'): The optimizer to use during training. Currently, `deeprai` is in beta, so the only valid option for optimizer is 'gradient descent'.
- `loss` (str, default='mean square error'): The loss function to use during training. Valid options are 'mean square error', 'categorical cross entropy', and 'mean absolute error'.

Return Value

This function does not return anything. It modifies the `deeprai.models.FeedForward` instance by setting the optimizer and loss function.

Description

The `config` function sets the optimizer and loss function for the deeprai.models.FeedForward instance. While it is not necessary to call this function, if called, it will use the default values of 'gradient descent' for optimizer and 'mean square error' for loss function.

Currently, deeprai is in beta, so the only valid option for optimizer is 'gradient descent'. The loss parameter sets the loss function to use during training. Valid options are 'mean square error', 'categorical cross entropy', and 'mean absolute error'.

Examples

Here's an example of how to use the `config` function:

```
from deeprai.models import FeedForward


model = FeedForward()
model.add_dense(784)
model.add_dense(128, activation='relu')
model.add_dense(64, activation='relu')
model.add_dense(10, activation='softmax')
model.config(optimizer='gradient descent', loss='categorical cross entropy')
```

This code creates a `FeedForward` model with an input shape of `(784,)`, adds three dense layers with ReLU and softmax activation functions, and sets the optimizer to 'gradient descent' and the loss function to 'categorical cross entropy'.

Training a network

Function Signature

```
def train_model(
    train_inputs: np.ndarray,
    train_targets: np.ndarray,
    test_inputs: np.ndarray,
    test_targets: np.ndarray,
    batch_size: int = 36,
    epochs: int = 500,
    learning_rate: float = 0.1,
    momentum: float = 0.6,
    early_stop: bool = False,
    verbose: bool = True
) -> None:
```

Parameters

- `train_inputs` (np.ndarray): The input data to use for training. This should be a numpy array of shape `(num_samples, input_shape)`.
- `train_targets` (np.ndarray): The target data to use for training. This should be a numpy array of shape `(num_samples, output_shape)`.
- `test_inputs` (np.ndarray): The input data to use for testing. This should be a numpy array of shape `(num_samples, input_shape)`.
- `test_targets` (np.ndarray): The target data to use for testing. This should be a numpy array of shape `(num_samples, output_shape)`.
- `batch_size` (int, default=36): The batch size to use during training.
- `epochs` (int, default=500): The number of epochs to train the model for.
- `learning_rate` (float, default=0.1): The learning rate to use during training.
- `momentum` (float, default=0.6): The momentum to use during training.
- `early_stop` (bool, default=False): Whether to use early stopping during training. If True, the training will stop when the validation loss stops improving.
- `verbose` (bool, default=True): Whether to print training progress during training.

Return Value

This function does not return anything. It trains the `deeprai.models.FeedForward` instance on the given data and saves the updated weights.

The `train_model` function trains the `deeprai.models.FeedForward` instance on the given training data using the specified hyperparameters. It also evaluates the model on the test data after each epoch and prints the training progress if `verbose=True`.

The `batch_size` parameter specifies the batch size to use during training. The `epochs` parameter specifies the number of epochs to train the model for. The `learning_rate` and `momentum` parameters specify the learning rate and momentum to use during training, respectively.

The `early_stop` parameter specifies whether to use early stopping during training. If `early_stop=True`, the training will stop when the validation loss stops improving.

Examples

Here's an example of how to use the `train_model` function:

```
from deeprai.models import FeedForward

model = FeedForward()
model.add_dense(784)
model.add_dense(128, activation='relu')
model.add_dense(64, activation='relu')
model.add_dense(10, activation='sigmoid')

train_inputs = ...
train_targets = ...
test_inputs = ...
test_targets = ...

model.train_model(
    train_inputs=train_inputs,
    train_targets=train_targets,
    test_inputs=test_inputs,
    test_targets=test_targets,
    batch_size=32,
    epochs=1000,
    learning_rate=0.1,
    momentum=0.6,
    early_stop=True,
    verbose=True
)
```

This code creates a `FeedForward` model with an input shape of `(784,)`, adds three dense layers with ReLU and softmax activation functions, sets

Running data through a network

Function Signature

```
def run(
    self,
    inputs: np.ndarray
) -> np.ndarray:
```

Parameters

- `inputs` (np.ndarray): The input data to run through the network. This should be a numpy array of shape `(input_shape,)`.

Return Value

- `output` (np.ndarray): The output of the network after running the given input through it. This should be a numpy array of shape `(output_shape,)`.

Description

The `run` function takes a single input and runs it through the network, returning the output of the network.

The `inputs` parameter should be a numpy array of shape `(input_shape,)`, where `input_shape` is the shape of the input to the network.

The `output` parameter is a numpy array of shape `(output_shape,)`, where `output_shape` is the shape of the output of the network.

Examples

Here's an example of how to use the `run` function:

```
from deeprai.models import FeedForward


model = FeedForward()
model.add_dense(784)
model.add_dense(128, activation='relu')
model.add_dense(64, activation='relu')
model.add_dense(10, activation='softmax')
model.config(loss='categorical cross entropy')


input_data = np.random.rand(784)
output_data = model.run(input_data)
```

This code creates a `FeedForward` model with a single dense layer of size `784`, followed by two additional dense layers with ReLU activation functions, and a final dense layer with a softmax activation function. The `config` function sets the optimizer to `gradient descent` and the loss function to `categorical cross entropy`.

The `run` function takes a single input data of shape `(784,)` and returns the output of the network as a numpy array of shape `(10,)`.

Viewing network information

Function Signature

```
def specs(self) -> str:
```

Return Value

- `output` (str): A string representation of the network information, including the model type, optimizer, parameters, loss function, and DeeprAI version.

Description

The `specs` function returns a string representation of the network information, including the model type, optimizer, parameters, loss function, and DeeprAI version.

Examples

Here's an example of how to use the `specs` function:

```
from deeprai.models import FeedForward

model = FeedForward()
model.add_dense(784)
model.add_dense(128, activation='relu')
model.add_dense(64, activation='relu')
model.add_dense(10, activation='softmax')
model.config(optimizer='gradient descent', loss='mean square error')

model_specs = model.specs()
print(model_specs)
```

This code creates a `FeedForward` model with a single dense layer of size `784`, followed by two additional dense layers with ReLU activation functions, and a final dense layer with a softmax activation function. The `config` function sets the optimizer to `gradient descent` and the loss function to `mean square error`.

The `specs` function returns a string representation of the network information, including the model type, optimizer, parameters, loss function, and DeeprAI version, which can be printed to the console. The output should look something like this:

```
.---------------.------------------.-----------------.------------------.
|      Key      |       Val        |       Key       |       Val        |
:---------------+------------------+-----------------+------------------:
| Model         | Feed Forward     | Optimizer       | Gradient Descent |
:---------------+------------------+-----------------+------------------:
| Parameters    | 15               | Layer Model     | 2x5x1            |
:---------------+------------------+-----------------+------------------:
| Loss Function | Mean Square Error| DeeprAI Version | 0.0.12 BETA      |
'---------------'------------------'-----------------'------------------'
```

Graphing

Function Signature

```
def graph(self, metric: str = "cost") -> None:
```

Parameters

- `metric` (str, optional): The metric to plot. Default is `"cost"`. Valid metrics are `"cost"`, `"acc"`, or `"accuracy"`, and `"error"`.

Return Value

- None

Description

The `graph` function uses `matplotlib` to plot the change of the specified metric over the epochs. It should be called after training the network.

Examples

Here's an example of how to use the `graph` function:

```
from deeprai.models import FeedForward


model = FeedForward()
model.add_dense(784)
model.add_dense(128, activation='relu')
model.add_dense(64, activation='relu')

model.add_dense(10, activation='softmax')

model.config(optimizer='gradient descent', loss='mean square error')


train_inputs, train_targets, test_inputs, test_targets = # load data


model.train_model(train_inputs, train_targets, test_inputs, test_targets)


model.graph(metric='accuracy')
```

This code creates a `FeedForward` model with a single dense layer of size `784`, followed by two additional dense layers with ReLU activation functions, and a final dense layer with a softmax activation function. The `config` function sets the optimizer to `gradient descent` and the loss function to `mean square error`.

The `train_model` function trains the model on the loaded data. After training, the `graph` function is called with the `"accuracy"` metric to plot the accuracy over the epochs.

The output should be a plot of the specified metric over the epochs.

Saving a Model

Function Signature

`def save(self, file_location: str) -> None:`

Parameters

- `file_location` (str): The location/path where the model's state will be saved.

Return Value

This function does not return anything. It saves the model's state to the specified file location.

Description

The `save` function saves the current state of the `deeprai.models.FeedForward` instance to a file. This allows for easy checkpointing and restoration of trained models. The model's weights, architecture, and configurations are stored at the specified file location. It's recommended to save the model periodically during training to avoid potential data loss.

Examples

Here's an example of how to use the `save` function:

```
from deeprai.models import FeedForward


# ... [Building and training the model] ...


# Saving the model's state to a file
```

```
model.save('path/to/save/model.deepr')
```

This code initializes and trains a `FeedForward` model and then saves its state to the file located at `'path/to/save/model.deepr'`.

Loading a Model

Function Signature

`def load(self, file_location: str) -> None:`

Parameters

- `file_location` (str): The location/path from which the model's state will be loaded.

Return Value

This function does not return anything. It loads the model's state from the specified file location.

Description

The `load` function restores the `deeprai.models.FeedForward` instance from a saved state located at a file. This can be used to continue training from a checkpoint, or to deploy pre-trained models without the need to retrain them. The model's weights, architecture, and configurations are loaded from the specified file location. Before using this function, ensure the model architecture is the same as the one saved in the file.

Examples

Here's an example of how to use the `load` function:

```
from deeprai.models import FeedForward


model = FeedForward()
```

```
# Loading the model's state from a file
model.load('path/to/saved/model.deepr')
```

This code initializes a `FeedForward` model and then loads its state from the file located at `'path/to/saved/model.deepr'`.