Title: Chatbot Neural Network Training Documentation

---

1. Overview This document explains the workflow of a PyTorch-based chatbot model, including data preprocessing, model architecture, training with K-Fold Cross-Validation, and evaluation. The model takes tokenized user input sentences and predicts the appropriate intent tag.

---

2. Dataset Loading and Preprocessing • Directory Path: `D:\projectllp\logathon\college\NewIntents` • JSON files containing intents are loaded. • Each JSON file structure: - `intents`: List of intent objects - Each intent object: - `tag`: Intent label - `patterns`: List of example sentences

Processing Steps: 1. Tokenization: `tokenize(pattern)` splits sentences into words. 2. Stemming: `stem(word)` reduces words to their root form. 3. Ignore punctuation: `['?', '!', '.', ',']`. 4. Bag-of-Words: `bag_of_words(sentence, words)` creates a vector for each sentence indicating presence (1) or absence (0) of known words. 5. Labels: `tags.index(tag)` encodes intent labels numerically.

Resulting Arrays: - `X`: Bag-of-words vectors (input features) - `y`: Encoded intent tags (output labels)

---

3. Dataset Class

```
class ChatDataset(Dataset):
    def __init__(self, x_data, y_data):
        self.n_samples = len(x_data)
        self.x_data = x_data
        self.y_data = y_data

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.n_samples
```

- Implements PyTorch Dataset for easy batching. - `__getitem__`: Returns a single input-output pair. - `__len__`: Returns total number of samples.

---

4. Neural Network Model Architecture

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
```

```
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.dropout = nn.Dropout(0.5)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = torch.relu(self.l1(x))
        x = self.dropout(x)
        x = torch.relu(self.l2(x))
        x = self.dropout(x)
        x = self.l3(x)
        return x
```

Layer Details: • **Linear Layers:** Map input to hidden features and then to output. • **ReLU Activation:** Introduces non-linearity to model complex patterns. • **Dropout:** Randomly zeroes 50% of activations during training to prevent overfitting. • **Output Layer:** Produces raw logits for each intent class.

---

5. Training Configuration • `num_epochs = 3000` • `batch_size = 16` • `learning_rate = 0.0001` • `hidden_size = 256` • Optimizer: `Adam` • Loss: `CrossEntropyLoss` for multi-class classification • Device: GPU if available, else CPU

K-Fold Cross-Validation: - `k_folds = 5` - Splits dataset into 5 folds, trains on 4, tests on 1, repeating 5 times. - Tracks fold-wise training accuracy, testing accuracy, and loss.

---

6. Training Loop Steps per fold: 1. Reset model, optimizer, and loss. 2. Loop over epochs: - Forward pass: `outputs = model(words)` - Loss computation: `loss = criterion(outputs, labels)` - Backward pass: `loss.backward()` - Update weights: `optimizer.step()` 3. Record last epoch loss. 4. Compute training accuracy for the fold. 5. Compute test accuracy for the fold.

Visualization: - Bar plots for training accuracy, testing accuracy, and loss per fold. - Combined accuracy plot shows train vs test trends.

---

7. Inputs and Outputs - **Input:** Bag-of-words vector of tokenized sentence. - `input_size = len(X[0])` - **Output:** Logits per intent. - `output_size = len(tags)` - Use `torch.max(outputs, dim=1)` to get predicted intent. - **Dynamic Adjustments:** - To change input features, modify `all_words` and bag-of-words logic. - To change outputs, modify `tags` or number of intent classes.

---

8. Preprocessing Concept - Converts raw text into numeric vectors. - Tokenization splits sentences. - Stemming reduces words to root forms for consistency. - Bag-of-words encodes presence/absence of known words. - This ensures the model can process variable input sentences dynamically.

---

9. Model Concept - Feedforward neural network with two hidden layers and dropout. - **Forward pass:** Input → Linear → ReLU → Dropout → Linear → ReLU → Dropout → Output - No activation on output layer since `CrossEntropyLoss` expects raw logits. - Network learns to map input patterns to intent labels.

---

10. Activation and Layer Rationale - **ReLU:** Efficient, avoids vanishing gradient, introduces non-linearity. - **Dropout:** Reduces overfitting, improves generalization. - **Linear layers:** Transform input feature space to higher-dimensional representation. - **CrossEntropyLoss:** Suitable for multi-class classification.

---

11. Cross-Validation Purpose - Measures model stability across different data splits. - Provides insight into variance in performance. - Helps detect overfitting or underfitting.

---

12. Output Interpretation - Raw logits per class from output layer. - `torch.max` selects the class with highest score. - Can convert logits to probabilities using `softmax` if needed for confidence scores. - Provides predicted intent for chatbot response generation.

---

13. Summary - Dynamic preprocessing allows adding/removing patterns and tags. - Model architecture supports variable input size based on vocabulary. - Cross-validation ensures robustness. - Outputs are flexible; changing `tags` automatically changes the output layer dimension. - Dropout and ReLU improve generalization and convergence.

End of Document