Creating Training Samples for Multi-Token Outputs

How to structure X and Y when Y has multiple tokens

🎯 The Challenge

When Y has multiple tokens, you can't use simple classification. You need **sequence-to-sequence training**. Here's exactly how to create the training samples:

🔧 Method 1: Autoregressive Training (Most Common)

The Key Insight: Train the model to predict each token one by one, conditioning on all previous tokens.

Example: Question Answering

Raw Data:

```
Question: "What is the capital of France?"
Answer: "The capital of France is Paris."
```

Training Sample Creation:

```
X (Input): "Question: What is the capital of France? Answer:"
Y (Target sequence): ["The", "capital", "of", "France", "is", "Paris", ".", "<END>"]
```

Multiple Training Examples from ONE Sample:

```
Training Example 1:
X: "Question: What is the capital of France? Answer:"
Y: "The"

Training Example 2:
X: "Question: What is the capital of France? Answer: The"
Y: "capital"

Training Example 3:
X: "Question: What is the capital of France? Answer: The capital"
Y: "of"

Training Example 4:
X: "Question: What is the capital of France? Answer: The capital of"
Y: "France"

Training Example 5:
X: "Question: What is the capital of France? Answer: The capital of France"
Y: "is"

Training Example 6:
X: "Question: What is the capital of France? Answer: The capital of France is"
Y: "Paris"

Training Example 7:
X: "Question: What is the capital of France? Answer: The capital of France is Paris"
Y: "."

Training Example 8:
X: "Question: What is the capital of France? Answer: The capital of France is Paris."
Y: "<END>"
```

Mathematical Formulation:

Loss = $\Sigma_i -\log P(y_i \mid x, y_1, y_2, \ldots, y_{i-1})$

Where:
- x = input question
- y_i = i-th token in target answer
- $y_1, y_2, \ldots, y_{i-1}$ = previous tokens in answer

📝 Method 2: Teacher Forcing Training

Concept: During training, use the ground truth previous tokens (not model predictions)

Example: Text Summarization

Raw Data:

```
Article: "Climate change is causing unprecedented changes to Earth's weather patterns. Scientists have observed rising global temperatures, melting ice caps, and more frequent extreme weather events..."

Summary: "Climate change causes rising temperatures and extreme weather."
```

Training Sample:

```
Input Sequence:
"<ARTICLE> Climate change is causing unprecedented changes... <SUMMARY>"

Target Sequence:
["Climate", "change", "causes", "rising", "temperatures", "and", "extreme", "weather", ".", "<END>"]
```

Training Examples Generated:

```
Step 1:
X: "<ARTICLE> Climate change is causing... <SUMMARY>"
Y: "Climate"

Step 2:
X: "<ARTICLE> Climate change is causing... <SUMMARY> Climate"
Y: "change"

Step 3:
X: "<ARTICLE> Climate change is causing... <SUMMARY> Climate change"
Y: "causes"

Step 4:
X: "<ARTICLE> Climate change is causing... <SUMMARY> Climate change causes"
Y: "rising"

... and so on
```

🎪 Method 3: Sequence-to-Sequence with Special Tokens

Example: Translation

Raw Data:

```
English: "Hello, how are you?"
Spanish: "Hola, ¿cómo estás?"
```

Training Sample Format:

```
X: "<TRANSLATE> English: Hello, how are you? Spanish:"
Y: ["Hola", ",", "¿", "cómo", "estás", "?", "<END>"]
```

Autoregressive Training Examples:

```
Example 1:
X: "<TRANSLATE> English: Hello, how are you? Spanish:"
Y: "Hola"

Example 2:
X: "<TRANSLATE> English: Hello, how are you? Spanish: Hola"
Y: ","

Example 3:
X: "<TRANSLATE> English: Hello, how are you? Spanish: Hola,"
Y: "¿"

... continue for each token
```

🔄 Method 4: Batch Training Implementation

How to Create Batches Efficiently

Single Example Expanded:

```python
```

```python
def create_training_samples(input_text, target_sequence):
    samples = []

    # Create one training example for each position in target
    for i in range(len(target_sequence)):
        x = input_text + " " + " ".join(target_sequence[:i])
        y = target_sequence[i]
        samples.append((x, y))

    return samples

# Example usage
input_text = "Question: What is the capital of France? Answer:"
target = ["The", "capital", "of", "France", "is", "Paris", "."]

training_samples = create_training_samples(input_text, target)
# Returns 7 training samples
```

Batch Creation:

```python
# Create a batch from multiple examples
batch_x = []
batch_y = []

for question, answer in dataset:
    samples = create_training_samples(question, answer)
    for x, y in samples:
        batch_x.append(tokenize(x))
        batch_y.append(tokenize(y)[0])  # Single token target

# Now batch_x and batch_y can be fed to the model
```

🧠 Method 5: Modern Approach (GPT-style)

How Modern Models Handle This

Training Data Format:

```
"<INSTRUCTION> Explain photosynthesis <RESPONSE> Photosynthesis is the process by which plants convert sunlight into energy using chlorophyll. <END>"
```

Training Process:

1. **Feed entire sequence** to model
2. **Compute loss** only on the response tokens
3. **Mask out** instruction tokens from loss calculation

Implementation:

```python
def compute_loss(input_ids, target_ids, loss_mask):
    # input_ids: [batch_size, seq_len]
    # target_ids: [batch_size, seq_len] (shifted by 1)
    # loss_mask: [batch_size, seq_len] (1 for response tokens, 0 for instruction)

    logits = model(input_ids)  # [batch_size, seq_len, vocab_size]

    # Compute loss only where loss_mask = 1
    loss = cross_entropy(logits[:, :-1], target_ids[:, 1:])
    masked_loss = loss * loss_mask[:, 1:]

    return masked_loss.sum() / loss_mask.sum()
```

📊 Concrete Example: Creating Full Dataset

Task: Question Answering with Multi-Token Answers

Raw Dataset:

```json
[
  {
    "question": "What is photosynthesis?",
    "answer": "Photosynthesis is the process plants use to convert sunlight into energy."
  },
  {
    "question": "Who invented the telephone?",
    "answer": "Alexander Graham Bell invented the telephone in 1876."
  },
  {
    "question": "What causes rainbows?",
    "answer": "Rainbows are caused by light refraction through water droplets."
  }
]
```

Generated Training Samples:

```python
training_data = []

for item in raw_dataset:
    question = f"Q: {item['question']} A:"
    answer_tokens = tokenize(item['answer']) + ["<END>"]

    # Create autoregressive samples
    for i in range(len(answer_tokens)):
        x = question + " " + " ".join(answer_tokens[:i])
        y = answer_tokens[i]
        training_data.append((x, y))

# Result:
# Original: 3 examples
# Training samples: ~45 samples (15 tokens average per answer)
```

Final Training Samples:

```
("Q: What is photosynthesis? A:", "Photosynthesis")
("Q: What is photosynthesis? A: Photosynthesis", "is")
("Q: What is photosynthesis? A: Photosynthesis is", "the")
("Q: What is photosynthesis? A: Photosynthesis is the", "process")
...
("Q: Who invented the telephone? A:", "Alexander")
("Q: Who invented the telephone? A: Alexander", "Graham")
...
```

⚡ Training Loop Implementation

Simplified Training Code:

```python
```

```python
def train_multi_token_model(model, training_samples, epochs=3):
    optimizer = Adam(model.parameters(), lr=1e-4)

    for epoch in range(epochs):
        for batch in get_batches(training_samples, batch_size=32):

            # Prepare batch
            input_ids = tokenize_batch([sample[0] for sample in batch])
            target_ids = tokenize_batch([sample[1] for sample in batch])

            # Forward pass
            logits = model(input_ids)
            loss = cross_entropy(logits, target_ids)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if step % 100 == 0:
                print(f"Step {step}, Loss: {loss.item()}")
```

🎯 Key Differences from Single-Token Training

Single Token (GPT-1 style):

```
Training Sample:
X: "This movie was great! <EXTRACT>"
Y: "Positive"  # Single token

Loss: -log P("Positive" | X)
```

Multi-Token (Modern style):

```
Training Samples (multiple from one example):
X₁: "Q: Capital of France? A:"
Y₁: "The"

X₂: "Q: Capital of France? A: The"
Y₂: "capital"

X₃: "Q: Capital of France? A: The capital"
Y₃: "is"

Loss: Σᵢ -log P(Yᵢ | Xᵢ)
```

💡 Why This Approach Works

1. Maximum Learning Signal:

Every token position becomes a learning opportunity

2. Natural Generation:

Model learns to generate sequences token by token

3. Flexible Length:

Can handle answers of any length

4. Context Preservation:

Each prediction has access to full context

🚀 Modern Implementation Tips

1. Attention Masking:

```python
# Ensure model can't see future tokens during training
attention_mask = torch.tril(torch.ones(seq_len, seq_len))
```

2. Loss Masking:

```python
# Only compute loss on answer tokens, not question tokens
loss_mask = create_response_mask(input_ids, response_start_token)
```

3. Efficient Batching:

```python
# Pad sequences to same length for efficient GPU computation
batch = pad_sequences(training_samples, max_length=512)
```

4. Gradient Accumulation:

```python
# Handle large effective batch sizes
for i, batch in enumerate(dataloader):
    loss = compute_loss(batch) / accumulation_steps
    loss.backward()

    if (i + 1) % accumulation_steps == 0:
        optimizer.step()
        optimizer.zero_grad()
```

🎭 The Evolution

GPT-1 (2018):

❌ Couldn't handle multi-token outputs effectively
✅ Proved transfer learning concept

GPT-2 (2019):

✅ Full autoregressive training as shown above
✅ Natural multi-token generation

Modern Models:

✅ Instruction-following format
✅ Conversation-style training
✅ Human feedback integration

🎯 Summary

The key insight: Transform one multi-token example into many single-token predictions by creating training samples for each position in the target sequence.

This approach:

- ✅ Handles any output length

- ✅ Maintains full context
- ✅ Maximizes learning signal
- ✅ Enables natural generation

This is exactly how all modern language models (GPT-2, GPT-3, ChatGPT, etc.) are trained to handle multi-token outputs! 🌟