NLP with TensorFlow: Language Modeling

GRU-based character-level language model dengan advanced decoding strategies (greedy + beam search). Foundation untuk modern GPT architectures. [1]

Language Modeling Concepts dan Dataset Preparation

N-gram Tokenization untuk vocabulary reduction:

```
def create_ngram_vocab(texts, n=3):
    ngrams = []
    for text in texts:
        for i in range(len(text)-n+1):
            ngrams.append(text[i:i+n])
    return list(set(ngrams))

# Alice in Wonderland character-level modeling
text = load_text('alice.txt')
chars = sorted(list(set(text)))
char_to_idx = {c: i for i, c in enumerate(chars)}
idx_to_char = {i: c for i, c in enumerate(chars)}
```

tf.data Pipeline untuk Sequences:

```
def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text

def create_sequences(text, seq_length=100):
    total_chars = len(text)
    dataset = tf.data.Dataset.from_tensor_slices(text)
    sequences = dataset.batch(seq_length+1, drop_remainder=True)
    return sequences.map(split_input_target).shuffle(10000).batch(64)
```

GRU Language Model Implementation

```
def create_language_model(vocab_size, embedding_dim=256, rnn_units=1024):
    inputs = Input(shape=(100,), dtype='int32')

    # Embedding
    x = Embedding(vocab_size, embedding_dim)(inputs)

    # Stacked GRUs
    x = GRU(rnn_units, return_sequences=True, recurrent_initializer='glorot_uniform')(x)
    x = GRU(rnn_units, return_sequences=True)(x)

    # Output projection
    x = Dense(vocab_size)(x)

    model = Model(inputs, x)
    return model

model = create_language_model(len(chars))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
```

Text Generation: Greedy vs Beam Search

Greedy Decoding:

```
def generate_text(model, start_string, num_generate=1000, temperature=1.0):
    input_eval = [char_to_idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []
    model.reset_states()

    for i in range(num_generate):
        predictions = model(input_eval)
        predictions = predictions[:, -1:, :]
        predicted_id = tf.random.categorical(predictions[^0], num_samples=1)[0,0].numpy()

        input_eval = tf.expand_dims([predicted_id], 0)
        text_generated.append(idx_to_char[predicted_id])

    return start_string + ''.join(text_generated)
```

Beam Search Advanced):

```
def beam_search_decode(model, start_string, beam_width=5, max_length=100):
    # Maintain beam hypotheses dengan scores
    beams = [(start_string, 0.0)]  # (text, log_prob)

    for _ in range(max_length):
        new_beams = []
        for text, score in beams:
            input_ids = [char_to_idx[c] for c in text[-100:]]
            input_ids = tf.expand_dims(input_ids, 0)
```

```
            predictions = model(input_ids)[:, -1:, :]
            probs = tf.nn.log_softmax(predictions[^0])

            # Top-k candidates
            top_k = tf.math.top_k(probs, k=beam_width)
            for i in range(beam_width):
                token_id = int(top_k.indices[0, i])
                token_prob = float(top_k.values[0, i])
                new_text = text + idx_to_char[token_id]
                new_beams.append((new_text, score + token_prob))

        # Keep top beam_width hypotheses
        beams = sorted(new_beams, key=lambda x: x[^1], reverse=True)[:beam_width]

    return beams[^0][^0]
```

Model Quality Metrics

Perplexity (exponential of cross-entropy loss):

```
PPL = exp(-1/N * sum log P(w_t | w_1, ..., w_{t-1}))
```

Lower perplexity = better model.

Kesimpulan

Language modeling foundation untuk Transformer-based GPT models dengan beam search sebagai production decoding strategy. [1]