Frameworks

Jordan Boyd-Graber

University of Maryland

Example Implementation: DAN

Deep Unordered Composition Rivals Syntactic Methods for Text Classification

Mohit Iyyer,[1] **Varun Manjunatha,**[1] **Jordan Boyd-Graber,**[2] **Hal Daumé III**[1]
[1]University of Maryland, Department of Computer Science and UMIACS
[2]University of Colorado, Department of Computer Science
{miyyer,varunm,hal}@umiacs.umd.edu, Jordan.Boyd.Graber@colorado.edu

Implementing a non-trivial example . . .

Deep Averaging Network

$$
\begin{aligned}
& w_1, \ldots, w_N \\
& \quad \downarrow \\
z_0 &= \text{CBOW}(w_1, \ldots, w_N) \\
z_1 &= g(W_1 z_0 + b_1) \\
z_2 &= g(W_2 z_1 + b_2) \\
\hat{y} &= \text{softmax}(z_2)
\end{aligned}
$$

- Works about as well as more complicated models
- Strong baseline
- Key idea: Continuous Bag of Words

$$\text{CBOW}(w_1, \ldots, w_N) = \sum_i E[w_i] \quad (1)$$

- Actual non-linearity doesn't matter, we'll use tanh
- Let's implement in PyTorch

Deep Averaging Network

$$w_1, \ldots, w_N$$

$$\downarrow$$

$$z_0 = \text{CBOW}(w_1, \ldots, w_N)$$

$$z_1 = g(z_0)$$

$$z_2 = g(z_1)$$

$$\hat{y} = \text{softmax}(z_2)$$

Initialization

```
def __init__(self, n_classes, vocab_size, emb_dim=300,
             n_hidden_units=300):
    super(DanModel, self).__init__()
    self.n_classes = n_classes
    self.vocab_size = vocab_size
    self.emb_dim = emb_dim
    self.n_hidden_units = n_hidden_units
    self.embeddings = nn.Embedding(self.vocab_size,
                                   self.emb_dim)
    self.classifier = nn.Sequential(
           nn.Linear(self.n_hidden_units,
                     self.n_hidden_units),
           nn.ReLU(),
           nn.Linear(self.n_hidden_units,
                     self.n_classes))
    self._softmax = nn.Softmax()
```

Deep Averaging Network

$$
\begin{aligned}
& w_1, \ldots, w_N \\
& \quad \downarrow \\
z_0 &= \text{CBOW}(w_1, \ldots, w_N) \\
z_1 &= g(z_0) \\
z_2 &= g(z_1) \\
\hat{y} &= \text{softmax}(z_2)
\end{aligned}
$$

Forward

```
def forward(self, batch, probs=False):
    text = batch['text']['tokens']
    length = batch['length']
    text_embed = self._word_embeddings(text)
    # Take the mean embedding. Since padding results
    # in zeros its safe to sum and divide by length
    encoded = text_embed.sum(1)
    encoded /= lengths.view(text_embed.size(0), -1)

    # Compute the network score predictions
    logits = self.classifier(encoded)
    if probs:
        return self._softmax(logits)
    else:
        return logits
```

Deep Averaging Network

$$w_1, \ldots, w_N$$

$$\downarrow$$

$$z_0 = \text{CBOW}(w_1, \ldots, w_N)$$

$$z_1 = g(z_0)$$

$$z_2 = g(z_1)$$

$$\hat{y} = \text{softmax}(z_2)$$

Training

```
def _run_epoch(self, batch_iter, train=True):
    self._model.train()
    for batch in batch_iter:
        model.zero_grad()
        out = model(batches)
        batch_loss = criterion(out,
                               batch['label'])
        batch_loss.backward()
        self.optimizer.step()
```

Summary

- Computation Graph
- Expressions ($\approx$ nodes in the graph)
- Parameters, LookupParameters
- Model (a collection of parameters)
- Optimizers
- Create a graph for each example, compute loss, backdrop, update