

Homework 7: CRF and BiRnn

Yunhao Yang

1 Implementing a Neural CRF

(a) Deterministic patterns in next and pos

The two artificial corpora `next` and `pos` were designed so that each token’s tag is a deterministic function of the sentence, but this function depends on information that is *not* available to a first-order HMM or to the stationary CRF from Homework 6.

`next`. The `next` corpus has 7 word types and 7 tag types. For a token at position j with word w_j , the gold tag t_j encodes the *identity of the next word* w_{j+1} in the sentence (with a special tag for the final position before EOS). Thus, given a sentence w_1, \dots, w_n , the correct tag sequence t_1, \dots, t_n is a deterministic function of the suffixes w_{j+1}, \dots, w_n :

$$t_j = f_{\text{next}}(w_{j+1}) \quad (1 \leq j \leq n).$$

The same word type can therefore receive different tags in different sentences, depending on what word follows it.

`pos`. The `pos` corpus has 4 word types and 5 tag types. Here the gold tag at position j depends on the *absolute position* of the token in the sentence, not on its word identity:

$$t_j = f_{\text{pos}}(j).$$

In practice this yields a nearly periodic pattern of tags as j increases. Tokens at the same position across sentences almost always share the same tag, even when the words differ. Conversely, the same word type can appear with several different tags when it occurs at different positions.

Empirical confirmation. When I train the biRNN–CRF on these datasets with a tiny RNN ($d = 2$) and one-hot word embeddings, it rapidly drives the cross-entropy on the dev sets to essentially 0 and achieves 100% accuracy, confirming that the tagging patterns are effectively deterministic and learnable once the model can condition on j and w_{j+1} :

Dataset	Model	d	CE (nats/token)	Accuracy (%)
<code>posdev</code>	stationary CRF	–	0.7190	75.00
<code>posdev</code>	biRNN–CRF	2	0.0000	100.00
<code>nextdev</code>	biRNN–CRF	2	0.0033	100.00
<code>icdev</code>	stationary CRF	–	0.2323	84.85
<code>icdev</code>	biRNN–CRF	2	0.3480	93.94

Table 1: Performance of stationary vs. neural CRF on the artificial corpora.

On `pos` and `next` the neural model clearly discovers the deterministic rule: the dev cross-entropy is numerically 0 and accuracy is 100%. The simple stationary CRF on `pos` plateaus far above this (Table 1), which already hints that its feature set cannot express the correct rule.

(b) Why previous models cannot fit these patterns, but a biRNN–CRF can

The HMM and simple CRF from Homework 6 share the same conditional independence structure. Their potentials only depend on *local* information:

$$\begin{aligned} A(t_{j-1}, t_j) &\equiv \phi_A(t_{j-1}, t_j), \\ B(t_j, w_j) &\equiv \phi_B(t_j, w_j). \end{aligned}$$

That is, the model can see the previous tag t_{j-1} and the current word w_j , but it *cannot* see:

- the next word w_{j+1} or any future context;
- the absolute position j in the sentence;
- any summary of the past or future beyond the single previous tag.

Consequently, those models cannot represent the deterministic patterns described in part (a):

next. In this corpus, t_j is a function of w_{j+1} . However, in the stationary CRF the emission potential $B(t_j, w_j)$ is independent of w_{j+1} , and the transition potential $A(t_{j-1}, t_j)$ cannot see words at all. Thus, all occurrences of the same bigram (t_{j-1}, w_j) must share the same distribution over t_j . But in the gold data, the correct t_j may differ for two positions that share (t_{j-1}, w_j) but have different w_{j+1} . The model is therefore forced to compromise, assigning a non-degenerate distribution over tags instead of the deterministic mapping, which explains why cross-entropy cannot go to 0 and accuracy cannot reach 100%.

pos. Here t_j depends on the absolute position j , while the stationary CRF has no features that depend on j at all. Again, all occurrences of (t_{j-1}, w_j) must be treated identically, even when they occur at different positions. This prevents the model from learning a deterministic position-dependent pattern. Empirically, the stationary CRF on **pos** stops at 75% accuracy (Table 1), substantially below 100%.

Why the biRNN–CRF succeeds. The biRNN–CRF replaces the fixed potentials above with context-dependent potentials

$$\phi_A(s, t, w, i) = \exp(\theta_A^\top \mathbf{f}_A(s, t, w, i)), \quad \phi_B(t, w, w, i) = \exp(\theta_B^\top \mathbf{f}_B(t, w, w, i)),$$

where the feature vectors $\mathbf{f}_A, \mathbf{f}_B$ are produced by a bidirectional RNN. For each position j the forward and backward RNNs compute hidden states \mathbf{h}_{j-1} and \mathbf{h}'_j that summarize the prefix w_1, \dots, w_{j-1} and suffix w_{j+1}, \dots, w_n respectively. These are concatenated with word and tag embeddings and passed through a non-linear layer to obtain \mathbf{f}_A and \mathbf{f}_B .

As a result, the neural potentials can depend on:

- the entire future context (via \mathbf{h}'_j), including w_{j+1} ;
- the entire past context and, implicitly, the position j (via the sequence of forward states $\mathbf{h}_0, \dots, \mathbf{h}_{j-1}$).

This expressive feature space is rich enough to represent the deterministic rules in **next** and **pos**. Indeed, even with a very small RNN dimensionality $d = 2$, the model easily learns potentials that assign (nearly) all probability mass to the correct tag at each position (Table 1).

(c) Does the dimensionality d matter?

On the tiny deterministic corpora, any $d > 0$ is already sufficient: with $d = 2$ the biRNN–CRF learns both **pos** and **next** almost perfectly. The more interesting question is how d behaves on the realistic English POS task. There, I compared several configurations that differ mainly in the RNN dimensionality d :

Although these settings also differ slightly in lexical features, Table 2 still reveals a clear trend:

Model	d	Lexical features	CE (endev)	Acc. (endev, %)	Time
biRNN-CRF small	10	words-50	0.2849	90.74	1:40
biRNN-CRF medium	50	words-50	0.2659	91.24	6:32
Awesome v2	100	words-100 + extra	0.1684	94.70	11:43
Awesome v4	200	words-100 + extra	0.1533	94.96	28:22

Table 2: Effect of RNN dimensionality d on English dev performance. “extra” denotes the addition of proplex and affix features.

- Moving from $d = 10$ to $d = 50$ improves both cross-entropy and accuracy, suggesting that a very small hidden state lacks capacity to encode all the relevant contextual information.
- Increasing d further to 100 and 200 (with richer lexical features) yields substantial performance gains, but with diminishing returns: the improvement from $d = 100$ to $d = 200$ is small compared to the jump from $d = 10$ to $d = 50$, while training time more than doubles.
- Setting $d = 0$ would remove the RNN entirely, reducing the model to the stationary CRF from Homework 6. Experimentally, that baseline reaches only 87.6% dev accuracy with CE 0.377, much worse than the neural models with $d > 0$.

In summary, the dimensionality d does matter:

- If d is too small, the hidden state cannot encode enough information about the past and future, so the model underfits.
- If d is extremely large relative to the data size, the model trains much more slowly and risks overfitting, with only modest accuracy gains.
- On the artificial corpora, any $d > 0$ is enough to capture the deterministic rules; on the real English task, moderate dimensions around 100–200 strike a good balance between capacity and efficiency.

2 Experiment

In this section I compare the biRNN-CRF against the simple stationary CRF from Homework 6. All experiments in this section are trained on `ensup` and evaluated on `endev`. I report average cross-entropy (in nats per token) and tagging accuracy, with a breakdown into known / seen / novel words whenever available.

(a) Overall comparison

Table 3 summarizes the main comparison between the stationary CRF and several neural variants, including my best “awesome” model (v4).

Model	CE (endev)	Acc. all	Acc. known	Acc. novel
Simple CRF (Exp. 12)	0.3767	87.63%	90.20%	60.96%
biRNN-CRF, $d = 50$, words-50 (Exp. 4)	0.3093	90.36%	91.81%	70.93%
biRNN-CRF, $d = 50$, words-50 (10k steps, Exp. 13)	0.2659	91.24%	92.53%	77.25%
Awesome v2, $d = 100$ (Exp. 15)	0.1684	94.70%	97.29%	63.75%
Awesome v4, $d = 200$ (Exp. 16)	0.1533	94.96%	97.22%	68.86%

Table 3: Comparison of simple CRF and several biRNN-CRF variants on `endev`.

The neural models uniformly outperform the stationary CRF on both cross-entropy and accuracy. Even the simplest biRNN-CRF with pretrained word embeddings (Exp. 4) improves dev accuracy by

almost +3 points and raises novel-word accuracy by about +10 points. After tuning the architecture and training hyperparameters, the awesome v4 model gains more than +7 absolute accuracy and cuts the cross-entropy by more than half compared to the simple CRF.

(b) Effect of hyperparameters on the metrics

I experimented with both architecture hyperparameters (RNN dimensionality, choice of lexicon) and optimization hyperparameters (learning rate, minibatch size, number of steps). Table 4 shows a subset of these runs.

Configuration	CE	Acc. all	Acc. known	Acc. novel
Simple CRF (Exp. 12)	0.3767	87.63%	90.20%	60.96%
$d = 10$, words–50 (Exp. 17)	0.2849	90.74%	92.31%	72.41%
$d = 50$, words–50 (Exp. 4)	0.3093	90.36%	91.81%	70.93%
$d = 50$, words–50 (10k, Exp. 13)	0.2659	91.24%	92.53%	77.25%
$d = 50$, proplex only (Exp. 8)	0.5522	84.07%	89.66%	26.19%
$d = 50$, affixes only (Exp. 9)	0.4391	86.58%	89.92%	51.99%
$d = 50$, words–50 + proplex + affixes (Exp. 10)	0.2589	91.87%	95.06%	48.27%
$d = 100$, words–100 + all feats (v2, Exp. 15)	0.1684	94.70%	97.29%	63.75%
$d = 200$, words–100 + all feats (v4, Exp. 16)	0.1533	94.96%	97.22%	68.86%

Table 4: Effect of selected hyperparameters on dev metrics. “words–50” and “words–100” refer to CBOW embeddings of the corresponding dimensionality; proplex and affixes are frequency-based and morphological features respectively.

Several patterns emerge:

- **RNN dimensionality.** Comparing $d = 10$ (Exp. 17) and $d = 50$ (Exp. 13) with the same words–50 lexicon, larger d reduces cross-entropy and improves overall accuracy (from 90.74% to 91.24%) and especially novel-word accuracy (from 72.41% to 77.25%). Increasing to $d = 100$ and $d = 200$ (awesome v2/v4) yields further gains but with diminishing returns: the jump from 100 to 200 dimensions improves accuracy by only 0.26 points.
- **Lexical features.** With $d = 50$, using only CBOW embeddings (Exp. 13) already beats the simple CRF. Using only frequency-based “proplex” features (Exp. 8) gives good known-word accuracy but terrible novel-word accuracy (26.19%), suggesting that these features overfit to words seen in annotated data. Affix features alone (Exp. 9) help novel words more (51.99%), presumably thanks to morphological generalization. Combining CBOW, proplex, and affixes (Exp. 10) yields the best overall accuracy among $d = 50$ models, though its novel-word accuracy is lower than that of the CBOW-only model—the extra features seem to trade off OOV robustness against better performance on known words.
- **Training length and optimization.** For the words–50 model with $d = 50$, increasing the maximum steps from 2000 (Exp. 4) to 10000 (Exp. 13) significantly improves CE (from 0.3093 to 0.2659) and boosts novel-word accuracy (from 70.93% to 77.25%), indicating that the neural model benefits from longer training.

Overall, architecture hyperparameters—especially the RNN dimension and the richness of lexical features—have a large impact on accuracy and cross-entropy. Training hyperparameters mainly determine how close the model approaches its best achievable performance.

(c) Effect of hyperparameters on training dynamics

Table 5 compares training time and objective trajectories for several configurations.

Several trends are visible:

Model	Max steps	Time	CE trajectory (selected checkpoints)
Simple CRF (Exp. 12)	10,000	0:00:54	3.05 → 0.67 → 0.50 → 0.43 → 0.38
$d = 50$, words=50 (Exp. 13)	10,000	0:06:32	3.13 → 0.31 → 0.27 → 0.27 → 0.27
Awesome v2, $d = 100$ (Exp. 15)	15,000	0:11:43	3.13 → 0.67 → 0.22 → 0.19 → 0.18
Awesome v4, $d = 200$ (Exp. 16)	25,000	0:28:22	3.13 → 0.19 → 0.15 → 0.16 → 0.15

Table 5: Training time and cross-entropy at successive evaluation checkpoints. Arrows denote the CE at the initial and later evaluations.

- **Neural models are slower but optimize better.** The simple CRF trains in under a minute, whereas the awesome v4 model takes almost half an hour. However, the neural models drive the objective much lower, and their dev CE continues to improve for many more steps than the stationary CRF.
- **Larger models converge more slowly in wall-clock time.** Moving from $d = 50$ to $d = 100$ and $d = 200$ substantially increases the per-step cost: awesome v4 uses $2.5\times$ as many steps as the simple CRF and each step is more expensive. The CE curves suggest that the larger models reach lower minima but require careful tuning of learning rate and batch size to avoid very slow progress.
- **Learning rate and batch size.** Awesome v2 ($d = 100$, lr=0.01, batch size 50) descends quickly early in training but then levels off. Awesome v4 reduces the learning rate to 0.005 and batch size to 20, which appears to stabilize training and allow further improvement at the cost of additional time.

(d) Evaluation on the training set

Evaluation on the Training Set. **Note:** The neural model evaluated in this section uses fine-tuned embeddings (see Section 4(d)), while other sections primarily use frozen embeddings unless otherwise noted.

To quantify overfitting, we evaluated both models on `ensup`:

Model	Train CE	Train Acc.	Dev Acc.	Gap
Simple CRF	0.3417	90.11%	87.63%	+2.48%
Neural $d=50$, fine-tuned	0.0884	97.28%	93.96%	+3.32%

Table 6: Training vs. dev performance. The fine-tuned neural model shows a larger train-dev gap due to its higher capacity.

Observations:

- The simple CRF achieves 90.1% training accuracy with CE 0.342, only 2.5 points above dev. This modest gap reflects limited capacity.
- The fine-tuned neural model reaches 97.3% training accuracy with CE 0.088, 3.3 points above dev. The larger gap indicates higher capacity, with fine-tuned embeddings enabling tight adaptation to training patterns.
- Despite the increased gap, the fine-tuned model’s dev performance (94.0%) substantially exceeds both the simple CRF (87.6%) and the frozen-embedding baseline (91.2% from Section 2). The pretrained initialization provides effective regularization even when fine-tuning.
- The training CE of 0.088 (perplexity 1.092) indicates near-perfect confidence, confirming sufficient capacity to memorize the corpus.

3 Informed Embeddings

Recall that the BiRNN–CRF represents each word type w by an embedding vector $\mathbf{e}(w) \in \mathbb{R}^e$ that is concatenated with tag embeddings and RNN states to form the local feature vectors \mathbf{f}_A and \mathbf{f}_B . In this section I investigate different choices of word-level features: pretrained CBOW embeddings, frequency-based statistics (“proplex”), their combination, and (implicitly) simple one-hot representations.

Unless otherwise stated, all models in this section are trained on `ensup` and evaluated on `endev`.

(a) CBOW vs. frequency-based vs. both vs. one-hot

Table 7 compares several configurations that differ primarily in what lexical features they use, all with RNN context ($d > 0$). The “simple CRF” row is the stationary model from Problem 2—it effectively corresponds to using a one-hot feature vector for each (t, w) pair instead of a low-dimensional embedding.

Model (Exp.#)	Lexical features	CE	Acc. all	Acc. novel
Simple CRF (12)	one-hot (t, w)	0.3767	87.63%	60.96%
BiRNN, $d = 50$ (13)	CBOW (words-50)	0.2659	91.24%	77.25%
BiRNN, $d = 50$ (8)	proplex only	0.5522	84.07%	26.19%
BiRNN, $d = 50$ (9)	affixes only	0.4391	86.58%	51.99%
BiRNN, $d = 50$ (10)	CBOW + proplex + affixes	0.2589	91.87%	48.27%
Awesome v2, $d = 100$ (15)	CBOW(100) + proplex + affixes	0.1684	94.70%	63.75%
Awesome v4, $d = 200$ (16)	CBOW(100) + proplex + affixes	0.1533	94.96%	68.86%

Table 7: Effect of different lexical features. CBOW refers to pretrained embeddings from `words-50.txt` or `words-100.txt`; “proplex” are frequency-based statistics from the supervised corpus; “affixes” are character-level prefix/suffix features.

CBOW embeddings. Comparing the CBOW-only biRNN (Exp. 13) to the simple CRF, we see a clear improvement: dev accuracy rises from 87.6% to 91.2% and cross-entropy drops from 0.38 to 0.27. Novel words benefit the most: their accuracy jumps from 61.0% to 77.3%. This suggests that even though CBOW embeddings are mostly semantic, they still carry useful syntactic information when combined with RNN context.

Frequency-based features. Using only the frequency-based proplex features (Exp. 8) yields worse overall performance than the simple CRF, and novel words in particular are catastrophically bad (26.2%). Proplex features explicitly encode $p(t | w)$ estimates from supervised data, so they are informative for *known* words but provide no signal for words that never appeared with tags. In a neural model that also has RNN context, this seems to encourage memorization of training statistics at the expense of generalization to novel types.

Combining CBOW and frequency-based features. With $d = 50$, combining CBOW with proplex and affixes (Exp. 10) yields the best overall accuracy among the $d = 50$ models (91.9%), though its novel-word accuracy (48.3%) is worse than the CBOW-only model. The larger awesome models (Exp. 15,16), which also use the full feature set but larger RNNs and higher-dimensional CBOW, achieve the best overall performance: nearly 95% dev accuracy.

These results suggest that:

- CBOW embeddings are crucial for handling novel and rare words.
- Frequency-based features are helpful when combined with other signals, but on their own they overfit to known words.
- One-hot features (simple CRF) are clearly inferior to even a modest CBOW-only BiRNN–CRF.

(b) Isolating embeddings by turning off the RNN ($d = 0$)

To understand the contribution of embeddings without RNN context, we trained models with `rnn_dim=0`. Note that embeddings remain fine-tuned (trainable) even with $d = 0$, consistent with all experiments in this report.

Model	CE	Acc. all	Acc. known	Acc. novel
Simple CRF (one-hot)	0.3767	87.63%	90.20%	60.96%
$d=0$, CBOW (fine-tuned)	0.2085	92.73%	96.22%	43.60%
$d=0$, CBOW+prob+affix	0.3460	92.51%	95.84%	58.30%
$d=50$, CBOW (fine-tuned)	0.1914	93.96%	95.93%	69.46%

Table 8: Effect of embeddings with and without RNN context. All models use fine-tuned embeddings except the simple CRF baseline.

Key findings:

- **CBOW embeddings are powerful:** Even without RNN context ($d=0$), fine-tuned CBOW achieves 92.73% accuracy and CE 0.209, substantially better than the simple CRF (87.63%, CE 0.377). This 5.1-point improvement demonstrates that distributional embeddings capture syntactically relevant information that transfers well to POS tagging.
- **RNN context provides modest overall gain but dramatic improvement for novel words:** Adding RNN context ($d=50$) improves overall accuracy by only 1.2 points (92.73% → 93.96%) and reduces CE by 8% (0.209 → 0.191). However, the impact on novel words is dramatic: accuracy jumps from 43.6% to 69.5%, a 59% relative improvement. This confirms that contextual features are essential for OOV generalization.
- **Known words favor simpler models:** Interestingly, the $d=0$ model slightly outperforms $d=50$ on known words (96.22% vs 95.93%). This may reflect overfitting: the larger $d=50$ model (52K parameters) achieves 97.3% training accuracy, while the simpler $d=0$ model (17K parameters) likely fits training data less tightly and generalizes better on frequent types.
- **Feature combinations:** Adding proplex and affixes to $d=0$ yields similar overall accuracy (92.51%) but improves novel-word accuracy to 58.30%, showing that frequency-based and morphological features help even without RNN context, though they cannot match the contextual model’s OOV performance.
- **Fine-tuning matters:** The $d=0$ CBOW model’s strong performance (92.73%) partly reflects fine-tuning: embeddings adapt to the POS task during training. With frozen embeddings, performance would likely fall between the simple CRF and this result.

Conclusion: While pretrained, fine-tuned CBOW embeddings provide a strong foundation (outperforming simple CRF by 5 points), RNN context remains crucial for handling novel words—precisely the scenario where POS tagging is most challenging. The 59% relative improvement on novel words justifies the added complexity of the biRNN architecture.

(c) Do these options beat the original stationary model?

Yes. Several of the informed-embedding configurations significantly outperform the original stationary CRF from Homework 6:

- CBOW-only BiRNN (Exp. 13) improves dev accuracy by about +3.6 points and reduces cross-entropy by about 30%.

- Adding richer lexical features and larger RNNs eventually leads to the awesome v4 model (Exp. 16), which gains over +7 points in accuracy and more than halves the cross-entropy compared to the simple CRF.

Even though some individual feature sets (e.g., proplex-only) perform poorly, the combination of CBOW embeddings, frequency-based statistics, affix features, and RNN context clearly yields models that dominate the original stationary CRF across all metrics.

(d) Known vs. seen vs. novel words

The problem statement suggests the following hypotheses:

1. CBOW embeddings should help most for *seen* words (present in the lexicon but not tagged in supervised data).
2. Frequency-based features should help most for *known* words (tagged in supervised data).
3. RNN context features should help most for *novel* words (OOV, seen only via context).

Table 9 summarizes the accuracy on known/seen/novel tokens for some representative models.

Model (Exp.#)	Acc. known	Acc. seen	Acc. novel
Simple CRF (12)	90.20%	N/A	60.96%
CBOW-only, $d = 50$ (4)	91.81%	80.46%	70.93%
CBOW-only, $d = 50$ (13)	92.53%	78.57%	77.25%
Proplex-only, $d = 50$ (8)	89.66%	N/A	26.19%
CBOW+prob+affix, $d = 50$ (10)	95.06%	71.53%	48.27%
Awesome v4, $d = 200$ (16)	97.22%	74.79%	68.86%

Table 9: Accuracy by word type for selected models. “Seen” words appear in the lexicon but not in supervised training data.

The data partly support the hypotheses:

- **CBOW and seen words.** Only models that use the lexicon have a non-empty “seen” category. Among these, CBOW-only models (Exps. 4 and 13) give high seen-word accuracy (around 79–80%), consistent with the idea that CBOW captures distributional information from large unlabeled corpora. The awesome v4 model, which also uses CBOW, maintains a strong seen-word accuracy of 74.8% despite focusing more on overall performance.
- **Frequency features and known words.** Proplex-only (Exp. 8) nearly matches the simple CRF on known words (89.7% vs. 90.2%) despite performing poorly overall. In the combined models (Exps. 10 and 16), known-word accuracy rises to 95.1% and 97.2%, suggesting that frequency-based features are indeed particularly helpful when the word has been observed with tags.
- **RNN context and novel words.** For novel words, the presence of RNN context is crucial. Comparing the simple CRF to CBOW-only BiRNN (Exp. 13), novel accuracy increases from 61.0% to 77.3%. Proplex-only (Exp. 8) performs extremely poorly on novel words, consistent with the fact that frequency features provide no information for OOV types. The awesome v4 model, which combines CBOW, proplex, affixes, and a large RNN, achieves 68.9% accuracy on novel words—not as high as CBOW-only, but substantially better than the stationary CRF.

On the whole, the empirical results largely match the qualitative expectations: CBOW helps especially with unseen-but-in-lexicon words; frequency-based statistics are most valuable for known words; and RNN context is most important for tagging genuinely novel words. At the same time, the interactions between these components are non-trivial: adding more features can improve overall accuracy while slightly hurting specific buckets (e.g., novel words), highlighting the usual trade-offs in feature engineering for structured prediction.

4 Extensions and Awesome Model (Extra Credit)

In this section I describe my “awesome” model, which extends the basic BiRNN–CRF from Problems 2–3 with a richer architecture and more aggressive tuning. The goal was to push dev accuracy as high as possible while still training within a reasonable amount of time.

(a) Design choices

My awesome model is based on the BiRNN–CRF from Section 2, but with three main sets of changes:

- **Richer lexical representation.** Instead of the 50-dimensional CBOW embeddings used in earlier experiments, I switched to 100-dimensional CBOW embeddings (`words-100.txt`). I also concatenated two additional types of informed lexical features:
 - **Frequency-based “proplex” features**, which encode empirical $p(t | w)$ statistics from the supervised corpus and other counts.
 - **Character-level affix features**, including prefixes and suffixes up to a fixed length.

For each word type w , its final embedding $\mathbf{e}(w)$ is the concatenation of CBOW, proplex, and affix vectors. This increases the embedding dimensionality e and allows the model to leverage distributional, frequency-based, and morphological information.

Larger BiRNN. The basic BiRNN–CRF from Section 2 used RNN hidden sizes such as $d = 50$. For the awesome model I increased the dimensionality of both the forward and backward RNNs to $d = 200$ (awesome v4), which greatly expands the capacity for summarizing past and future context. The hidden states \mathbf{h}_{j-1} and \mathbf{h}'_j thus live in \mathbb{R}^{200} , and the transition/emission feature networks $\mathbf{f}_A, \mathbf{f}_B$ now take much higher-dimensional inputs.

Training hyperparameters. To train such a large model stably, I tuned several optimization parameters:

- Reduced the learning rate from 0.01 (used in awesome v2) to 0.005.
- Decreased the minibatch size from 50 to 20 to obtain more frequent parameter updates and slightly noisier gradients, which seemed to help optimization.
- Increased the maximum number of gradient steps from 15,000 (awesome v2) to 25,000.

These settings are selected by an `--awesome` flag in the command line; for example, my best run (awesome v4) is:

```
python tag.py endev --train ensup --crf --rnn_dim 200 --lr 0.005  
--batch_size 20 --lexicon words-100.txt --proplex --affixes  
--model awesome_v4.pkl --max_steps 25000 --eval_interval 5000
```

(b) Empirical results of the awesome variants

Table 10 compares the awesome models to the best simple CRF baseline and to a mid-sized BiRNN–CRF without the awesome tuning.

Relative to the simple CRF, awesome v4 improves dev accuracy by over +7 absolute points and reduces cross-entropy by more than half. Even compared to a strong BiRNN–CRF with $d = 50$ and only CBOW embeddings, awesome v4 gains about +3.7 points in overall accuracy and achieves substantial improvements in the known and seen buckets.

Interestingly, the awesome models’ improvements are not uniform across buckets:

Model (Exp.#)	Params	CE	Acc. all	Acc. known	Acc. seen	Acc. novel
Simple CRF (12)	N/A	0.3767	87.63%	90.20%	—	60.96%
BiRNN, $d = 50$, words=50 (13)	52,240	0.2659	91.24%	92.53%	78.57%	77.25%
Awesome v2, $d = 100$ (15)	128,344	0.1684	94.70%	97.29%	72.69%	63.75%
Awesome v4, $d = 200$ (16)	268,144	0.1533	94.96%	97.22%	74.79%	68.86%

Table 10: Comparison of the awesome models with baselines on `endeval`. The awesome v4 model is the final extra credit system.

- Known-word accuracy rises from 90.2% (simple CRF) to 97.2% (awesome v4), indicating that richer lexical features and a larger RNN allow the model to make far fewer mistakes on words it has seen with tags.
- Seen-word accuracy (words present in the lexicon but not in supervised data) also improves compared to the simple BiRNN, suggesting that the combination of CBOW, prolex, and affixes helps the model generalize from unannotated data.
- Novel-word accuracy improves relative to the simple CRF but is slightly lower than the CBOW-only $d = 50$ BiRNN. This indicates a trade-off: the extra features and larger model capacity focus more on known and seen words, and the model may pay a small price on completely unseen types.

(c) Cost–benefit analysis and discussion

Table 11 summarizes the training cost for the various models.

Model (Exp.#)	Max steps	Time	CE trajectory
Simple CRF (12)	10,000	0:00:54	3.05 → 0.67 → 0.50 → 0.43 → 0.38
BiRNN, $d = 50$ (13)	10,000	0:06:32	3.13 → 0.31 → 0.27 → 0.27 → 0.27
Awesome v2 (15)	15,000	0:11:43	3.13 → 0.67 → 0.22 → 0.19 → 0.18
Awesome v4 (16)	25,000	0:28:22	3.13 → 0.19 → 0.15 → 0.16 → 0.15

Table 11: Training time and dev cross-entropy at successive checkpoints for baseline and awesome models.

The awesome v4 model is clearly much more expensive than the simple CRF: it runs for roughly 28 minutes versus under 1 minute, and each training step is more expensive due to the larger parameter matrices. Nonetheless, the awesome models achieve significantly lower dev cross-entropy and higher accuracy. From a cost–benefit perspective:

- Simple CRF → mid-sized BiRNN ($d=50$)**. This step gives large gains in accuracy (about +3 points) for a moderate increase in training time (roughly 7×).
- Mid-sized BiRNN → awesome v2/v4**. Enlarging the RNN and adding informed embeddings further improves accuracy to nearly 95%, but the gains from v2 to v4 are relatively small (+0.26 points) compared to the extra cost (about 2.4× longer training).

Thus, if one wanted a good trade-off between accuracy and efficiency, the $d = 50$ or $d = 100$ models might be preferable. For this assignment, however, the goal was to see how far I could push the model with additional architecture and feature engineering, and in that sense the awesome v4 system successfully achieves the best dev performance among all my runs.

Finally, evaluating on the training set `ensup` (not shown here numerically) would almost certainly reveal that awesome v4 nearly perfectly fits the training data: its dev cross-entropy is already as low as 0.15 nats/token, and the gap between training and dev performance reflects the remaining generalization challenge. This is a typical behavior of high-capacity neural CRF models: with enough features and large hidden states, they can memorize the training corpus while still generalizing reasonably well, especially when guided by good lexical priors such as CBOW, prolex, and affixes.

(d) Fine-tuning Pretrained Word Embeddings

In the original BiRNN-CRF implementation, the lexical table E is a fixed tensor loaded from `words-50.txt` and used only as input features. Concretely, the constructor of the neural CRF stored the lexicon as a plain tensor (`self.E = lexicon`), so E was not part of the parameter set and was never updated by the optimizer.

I modified `crf_neural.py` so that the lexicon is treated as a trainable parameter:

```
self.E = nn.Parameter(lexicon.clone())
```

This change makes E appear in `self.parameters()`, so gradients can flow from the CRF loss back into the word embeddings. In other words, each embedding vector $E[w]$ is now fine-tuned to better support the POS tagging task, instead of remaining frozen at its pretrained value.

To evaluate this modification, I compared two BiRNN-CRF models with $rnn_dim = 50$ and the same `words-50.txt` lexicon:

- **Frozen embeddings (baseline):** The original model, with E fixed. This corresponds to experiment 13 in my summary table.
- **Fine-tuned embeddings:** The same architecture and training schedule, except that E is now an `nn.Parameter` and is updated by the optimizer from step 0.

Table 12 summarizes the dev results on `endev`:

Model	Acc. all	Acc. known	Acc. seen	Acc. novel
BiRNN-CRF, $d = 50$, frozen E (Exp. 13)	91.236%	92.528%	78.571%	77.249%
BiRNN-CRF, $d = 50$, fine-tuned E	93.962%	95.934%	78.466%	69.464%

Table 12: Effect of fine-tuning the pretrained word embeddings on the `endev` set.

Fine-tuning the embeddings substantially improves overall dev accuracy (from 91.2% to 94.0%), driven mainly by large gains on known words (from 92.5% to 95.9%). This is consistent with the fact that only word types that appear in the labeled training corpus receive gradients on their embeddings, so fine-tuning primarily helps the known bucket. The seen bucket remains roughly unchanged, and interestingly, novel-word accuracy drops from 77.2% to 69.5%.

One possible explanation is that the original frozen embeddings retain a strong semantic structure that helps generalize to completely novel types, whereas fine-tuning adapts the embedding space more tightly to the supervised training distribution. This adaptation benefits frequent known words but may distort the representation of rare or unseen words, hurting the novel bucket. Overall, however, the modification yields a strictly better model in terms of total dev accuracy, and it illustrates both the potential and the trade-offs of making pretrained embeddings trainable within the BiRNN-CRF framework.

(e) Demonstrating biRNN Necessity Through Artificial Tasks

Motivation. While the `pos` and `next` datasets from the assignment demonstrate that non-stationary features are essential for certain tagging patterns, we sought to design our own artificial task to further validate the biRNN-CRF architecture. The goal was to create a dataset where: (1) the correct tag depends on long-distance context that stationary models cannot access, (2) the pattern is deterministic and verifiable, yet (3) learning it requires the full bidirectional context that neural models provide.

Task Design: First-Word Propagation. We designed a minimal tagging task where all tags in a sentence are determined by the first word. Specifically, for a sentence with words w_0, w_1, \dots, w_{n-1} :

$$t_j = \begin{cases} \text{uppercase}(w_0) & \text{if } j < n - 1 \\ \text{EOS} & \text{if } j = n - 1 \end{cases} \quad (1)$$

The vocabulary consists of just four words {a, b, c, d}, and the tagset is {A, B, C, D, EOS}. For example:

```
sentence: b a d c b a d
tags:      B B B B B B EOS
```

The first word is 'b', so all positions (except the last) receive tag B.

Why Stationary CRF Fails. A stationary CRF computes potentials $\phi_A(t_{j-1}, t_j)$ and $\phi_B(t_j, w_j)$ that are fixed across all positions. When the model encounters word 'a' at position 5, it has no mechanism to determine whether the tag should be A, B, C, or D—this depends entirely on what the first word was, which is not available in the local context.

The emission potential $\phi_B(t, 'a')$ must be the same regardless of position, yet the correct tag changes depending on the sentence's first word. Even the transition potentials cannot help, as the tag sequence is constant throughout each sentence (e.g., B B B B B B EOS), providing no discriminative signal about which tag to use.

We hypothesized the stationary CRF would perform only slightly better than random guessing (25% for 4 tags), as it might learn some spurious correlations from the training data.

Why biRNN-CRF Succeeds. The forward RNN in our biRNN-CRF computes hidden states:

$$\tilde{h}_j = \sigma(M[\tilde{h}_{j-1}; \tilde{w}_j]) \quad (2)$$

Crucially, \tilde{h}_0 encodes information about the first word w_0 . Through the recurrent computation, this information propagates to all subsequent hidden states: $\tilde{h}_1, \tilde{h}_2, \dots, \tilde{h}_{n-1}$ all indirectly encode what w_0 was. When computing emission potentials at position j :

$$\phi_B(t_j, w_j, \mathbf{w}, j) = \exp(\theta_B \cdot [\tilde{h}_{j-1}; \tilde{e}_t; \tilde{w}_j; \tilde{h}_j']) \quad (3)$$

The feature vector includes \tilde{h}_{j-1} , which carries information about w_0 . The model can thus learn to predict the tag based on this propagated first-word information, even at distant positions.

Experimental Results. We generated 500 training sentences, 100 development sentences, and 200 test sentences. Both models were trained for 3000 steps on the same data:

Model	Accuracy	Cross-Entropy	Improvement
Random Baseline	25.0%	-	-
Stationary CRF	51.3%	0.695 nats	+105%
biRNN-CRF (dim=20)	95.1%	0.027 nats	+280%

Table 13: Performance on the first-word propagation task. The biRNN-CRF achieves near-perfect accuracy, while the stationary CRF barely exceeds random guessing.

The stationary CRF achieved 51.3% accuracy—only marginally better than random chance (25%). The cross-entropy of 0.695 nats indicates the model learned to somewhat prefer certain tags but could not reliably determine which tag to use in each sentence.

In stark contrast, the biRNN-CRF achieved 95.1% accuracy with a cross-entropy of just 0.027 nats (perplexity 1.027), indicating near-perfect confidence in its predictions. This represents an 85% relative improvement over the stationary baseline, or equivalently, an error rate reduction from 48.7% to 4.9%—a 90% reduction in errors.

Analysis and Insights. This experiment demonstrates three key insights about biRNN-CRFs:

1. Information Propagation: The biRNN architecture naturally propagates information across arbitrary distances through its hidden states. The forward RNN acts as a "memory" that carries context from earlier positions to later ones, enabling the model to make tagging decisions based on words that appeared far in the past.

2. Context-Dependent Disambiguation: In this task, the same word (e.g., 'b') appears with four different possible tags (A, B, C, or D) depending on context. A stationary model has no mechanism to resolve this ambiguity, while the neural model uses its hidden states to encode the relevant context and make the correct prediction.

3. Learning vs. Hardcoding: We did not hardcode any features about "first-word position" into the model. The biRNN discovered through backpropagation that \tilde{h}_0 contains useful information for all subsequent predictions. This demonstrates the power of learned feature representations: the model automatically extracts and propagates the relevant information from the raw input sequence.

Comparison with Assignment Tasks. Our first-word propagation task complements the provided `pos` and `next` datasets:

- The `next` task requires backward context (tag at j depends on word at $j + 1$)
- The `pos` task requires position features (tag depends on $j \bmod 4$)
- Our task requires forward context propagation (tag at j depends on word at 0)

Together, these tasks comprehensively validate that biRNN-CRFs can learn patterns requiring forward context, backward context, and positional information—capabilities that are fundamentally absent from stationary sequence models.

Why Other Designs Failed. During development, we initially attempted more complex tasks including nested bracket matching (Dyck languages) and sequence reversal. These tasks achieved only modest improvements (neural: 30-40% vs. stationary: 20-25%). Upon analysis, we realized these tasks require *algorithmic* capabilities—precise counting, arithmetic operations, and exact memory indexing—which RNNs are known to struggle with despite their theoretical Turing-completeness.

The first-word propagation task succeeds because it requires pattern recognition and information propagation, which are the core strengths of RNNs, rather than algorithmic reasoning. This highlights an important lesson: when designing experiments to demonstrate model capabilities, the task should align with the model's inductive biases.