TTIC 31110
Speech Technologies

May 21, 2020

Announcements

- Project proposal
  - Please read the assignment carefully and make sure your proposal includes all requested components
  - References can go on an extra page

Outline

Question from last time: Why is the Good-Turing fish example so strange?

- Suppose you are fishing, and have caught 10 carp, 3 cod, 2 tuna, 1 trout, 1 salmon, 1 eel; in addition, the pond also contains flounder and bass
- What is the probability that the next fish you catch is a new species?
- Intuition: It's as probable as the fish you've only seen once, i.e. $p(\text{new species}) = 3/18$!
- Probability that the next fish is a bass: $1/2 \cdot 3/18 = 3/36$
- Probability that the next fish you catch is an eel: $< 1/18$, since we just "stole" some probability mass
- Wait, the probability of an unseen fish is higher than of a previously seen fish???
- Answer: Typically there are many more unseen types than seen types. E.g. 10 unseen fish species $\Longrightarrow$ each has probability $1/10 \cdot 3/18 = 3/180 << 1/18$. Much better.

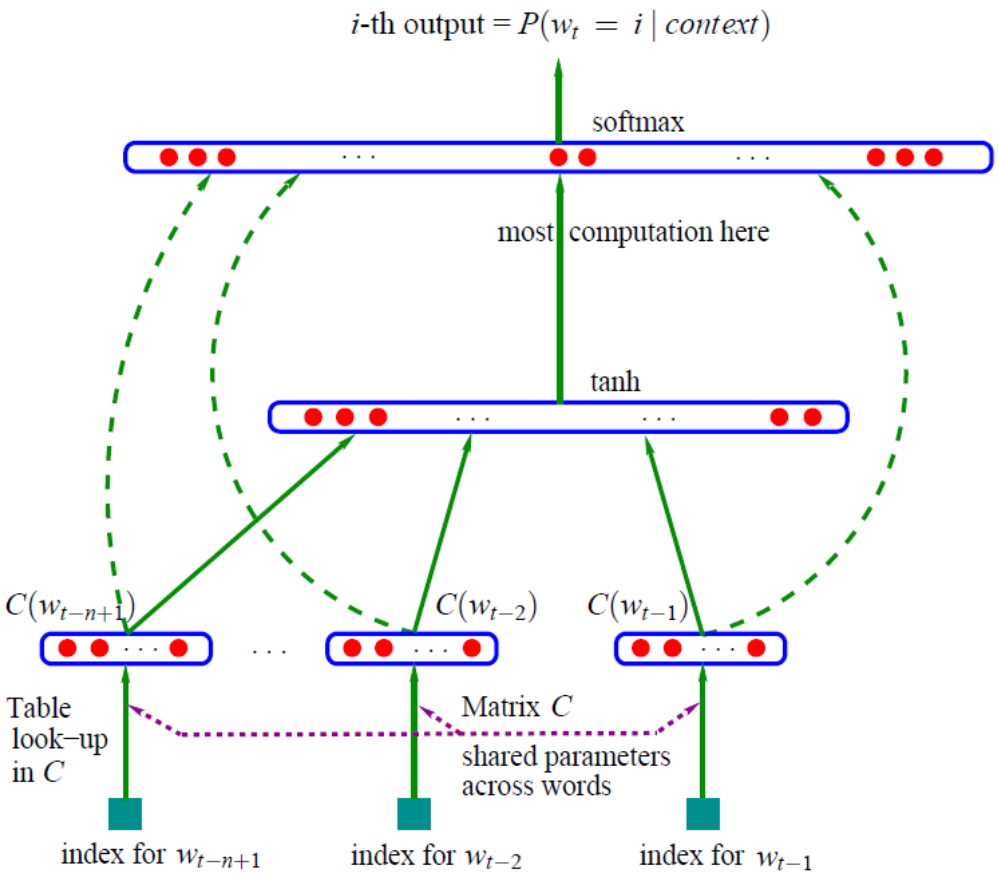
Neural network language models

- Model $p(w_i|h(w_i))$ with a neural network
- Inputs: one-hot vectors for context words
- Outputs: posterior probability of each next word
- (May be hard to do real-time decoding, but more on that later)

Neural network language models: Feedforward

Neural n-gram models

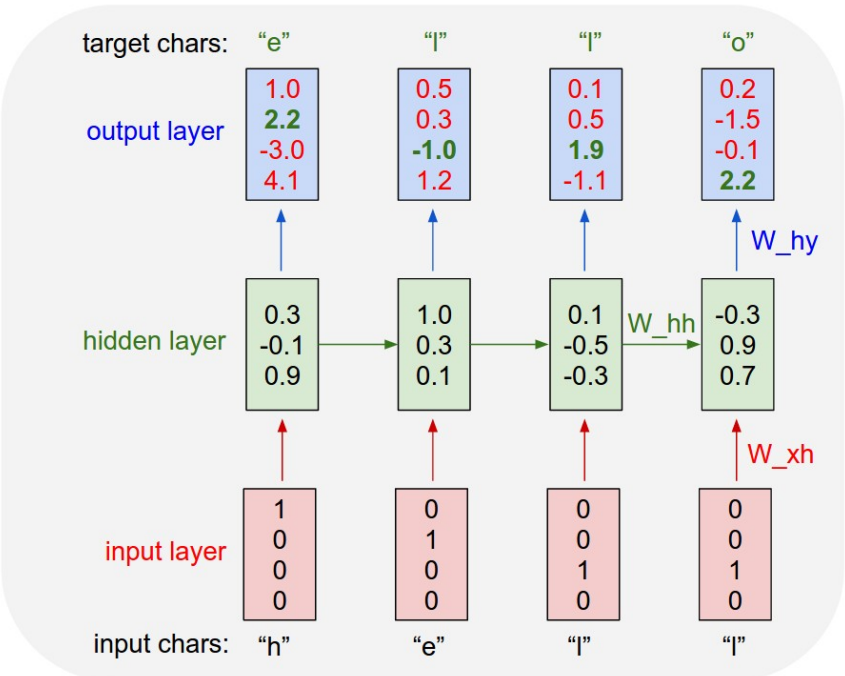
- Introduced by Bengio *et al.* in 2003
- Different way of dealing with rare/unseen n-grams



[Bengio+ 2003]

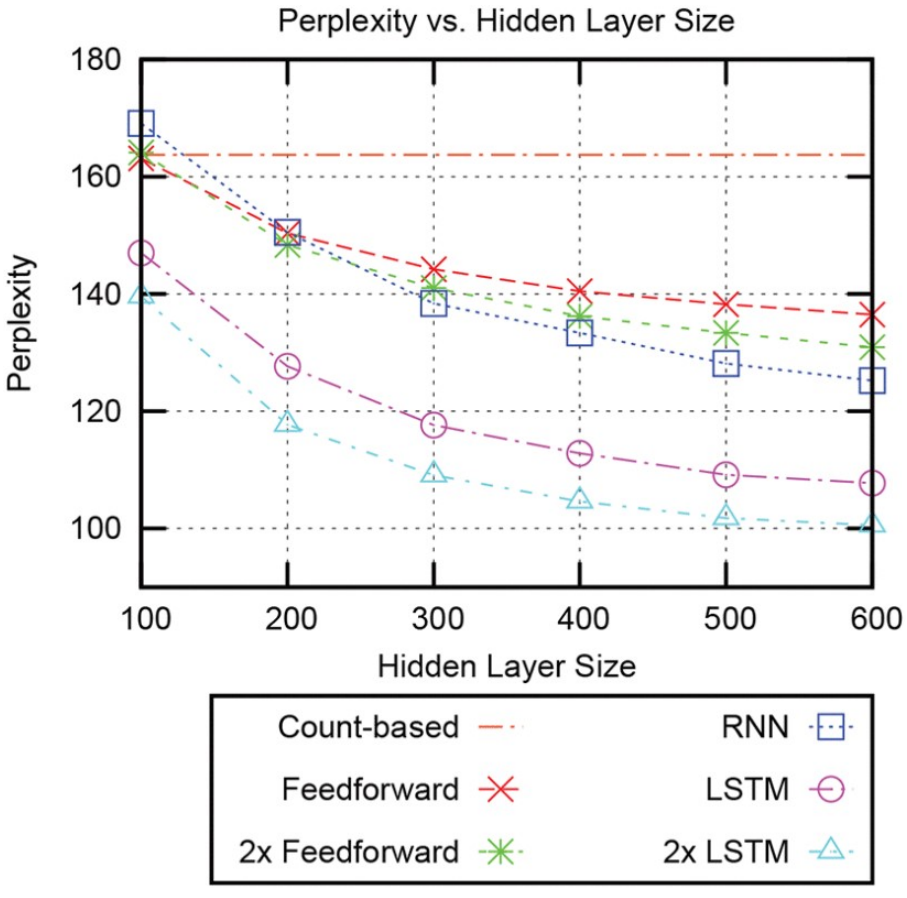
Neural network language models: Recurrent

- Very large (in principle) context needed to predict next word
- Example: The dog who was chasing the squirrel around the tree got tired.
- RNN LMs allow us to model arbitrarily long contexts
- Introduced by Mikolov *et al.* in 2010



[Karpathy]

Neural network language models: A few perplexity results



[Sundermeyer+ 2015]

Decoding with neural language models

- With feedforward LMs, in principle the same as non-neural n-gram LMs
- With recurrent LMs, less obvious what to do...
- **Rescoring** approach for RNN LMs: Decode without the LM, output N best label hypotheses, recompute the score of each hypothesis by combining with the LM score

First-pass decoding with RNN language models

Consider encoder-decoder models

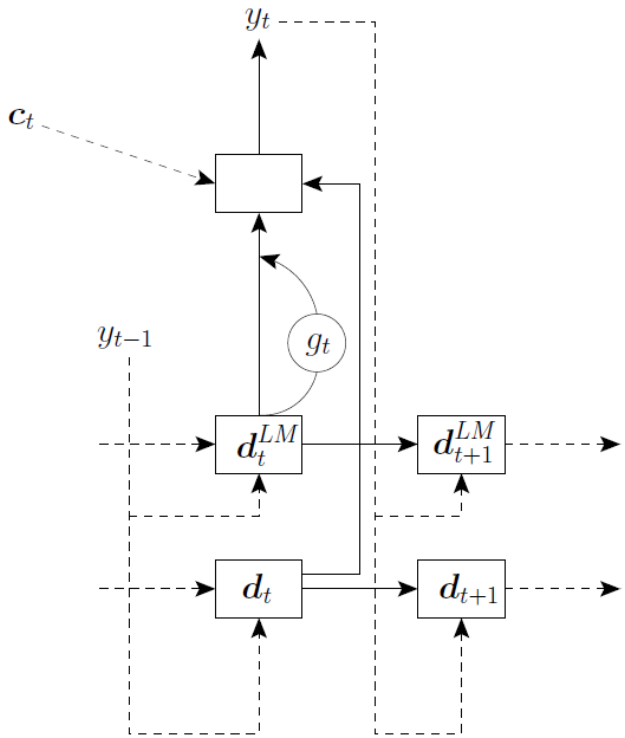
- "Shallow fusion":

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}} \log p(\mathbf{w}|\mathbf{O}) + \lambda \log p(\mathbf{w})$$

  - First term comes from the decoder, second term from the LM
  - Note: first term is the reverse of the usual acoustic model term
  - We are "double-counting" language model probabilities

First-pass decoding with RNN language models (2)

Deep fusion [Gulcehre et al. 2015]: Learn some additional parameters that combine the decoder state and LM state



Cold fusion [Sriram et al. 2017]: Similar to deep fusion, but learn the encoder-decoder parameters from scratch

Other issues: Language modeling for spontaneous speech

LMs trained on written language generalize poorly to transcripts of spoken language

- Utterance $\approx$ spoken correlate of sentence:
  I do uh main- mainly business data processing
- But sometimes not even a full sentence:
  so uh I mean spea- speaking of movies
- No casing, punctuation
- Disfluencies
  - Word fragments (spea-)
  - Filled pauses (uh, um, ...)
- silence is a word
- No clear word boundaries (want to vs. wanna, going to vs. gonna)

Discriminative training

So far, we've considered training with a couple of losses

- For HMMs, mainly maximum likelihood
- For neural models, mainly log loss

What if we train HMMs with log loss? Or any model with some other loss?

Review: training HMM-based speech recognizers

In principle we train to maximize the (log) probability of training data given the model parameters θ:

$$\min_{\theta} \sum_{i=1}^{m} -\log p(x^i, y^i|\theta)$$

x^i : acoustic vectors

y^i : phone/word sequence

θ : model parameters

m : number of training examples (utterances)

Review: training HMM-based speech recognizers

$$\min_{\theta} \sum_{i=1}^{m} -\log p(x^i, y^i|\theta)$$

$$= \min_{\theta_{\text{HMM}},\theta_{\text{LM}}} \sum_{i=1}^{m} -\log p(x^i|y^i, \theta_{\text{HMM}}) p(y|\theta_{\text{LM}})$$

$$= \min_{\theta_{\text{HMM}}} \sum_{i=1}^{m} -\log p(x^i|y^i, \theta_{\text{HMM}}) + \min_{\theta_{\text{LM}}} \sum_{i=1}^{m} -\log p(y^i|\theta_{\text{LM}})$$

$$= \text{HMM likelihood} + \text{LM likelihood}$$

In practice, we train each of these terms separately using different training sets.

Log loss (conditional likelihood, maximum mutual information)

More directly optimizes a quantity we care about:

$$\operatorname*{argmin}_{\theta} \sum_{i=1}^{m} -\log p(y^i|x^i, \theta)$$

For a training sample (x, y),

$$\begin{aligned} & -\log p(y|x) \\ &= -\log \sum_{z \in \mathcal{Z}(y)} p(x, z|y)p(y) + \log \sum_{y'} \sum_{z' \in \mathcal{Z}(y')} p(x, z'|y')p(y') \end{aligned}$$

$\mathcal{Z}(y)$: the set of possible state sequences for y

Computing log loss

$$-\log \sum_{z \in \mathcal{Z}(y)} p(x, z|y)p(y) + \log \sum_{y'} \sum_{z' \in \mathcal{Z}(y')} p(x, z'|y')p(y')$$

- $p(y)$ and $p(y')$ are sometimes dropped.
- $\sum_{z \in \mathcal{Z}(y)}$ and $\sum_{y'} \sum_{z' \in \mathcal{Z}(y')}$ are often approximated with **lattices**, a set of high scoring (y, z) pairs.
- "Numerator" lattices: high scoring paths with same sequence of labels but different segmentations.
- "Denominator" lattices: high scoring paths with different labels and segmentations.

Training with log loss

$$-\log \sum_{z \in \mathcal{Z}(y)} p(x, z|y)p(y) + \log \sum_{y'} \sum_{z' \in \mathcal{Z}(y')} p(x, z'|y')p(y')$$

- Numerator lattices
  - Restrict paths that match the ground truth labels
  - Select high scoring paths
- Denominator lattices
  - Select high scoring paths
- Important property: The gradient of log loss can also be computed via dynamic programming from the lattices

Summary so far: Log loss for HMMs

- It can be better to train with conditional likelihood $p(y|x)$ than with joint likelihood $p(x, y)$.
- Training with log loss is also often referred to as maximum mutual information (MMI)
- We need a working system to generate numerator and denominator lattices (though there is now a "lattice-free MMI" approach).
- The numerator and denominator terms, and gradient of log loss, can be calculated efficiently with dynamic programming.
- Typically does better than maximum likelihood training (results later).

Minimum Classification Error (MCE)

Can we optimize something even more directly related to the error rate?

$$\sum_{i=1}^{m} \frac{1}{2} \left[\operatorname{sign} \left[-\log p(y^i|x^i) + \max_{y' \neq y^i} \log p(y'|x) \right] + 1 \right]$$

We suffer a loss if $\log p(y^i|x^i)$ is not the maximum $\tilde{y}(x^i)$.
$\operatorname{sign}(x)$ is typically approximated with $\text{logistic}(x) = \frac{1}{1+e^{-\eta x}}$ because sign is not differentiable, and logistic looks a lot like sign when $\eta \to \infty$.

Minimum Classification Error (MCE)

- MCE aims to make the score of the ground truth the highest.
- The sign function is typically approximated with a logistic.
- MCE (approximated with logistic) can be optimized with gradient descent.
- Computing the gradient of MCE can be done efficiently with dynamic programming.

Empirical Bayes Risk (EBR)

MCE only considers how well our *top* hypothesis does. What about the others?

$$\frac{1}{m}\sum_{i=1}^{m} E_{y' \sim p(y'|x^i,\theta)}[\text{cost}(y^i, y')]$$

Depending on the cost function, there are a few other names.

- Minimum Phone Error (MPE)
- Minimum Word Error (MWE)
- State-level Minimum Bayes Risk (sMBR)

Cost functions

$$\text{cost}(y, y') = \text{cost}(z(y), z'(y')) = \sum_{s \in z'} \text{cost}(z, s)$$

EBR can be optimized with gradient descent given some conditions on the cost that make dynamic programming work:

- Each y has an associated state sequence $z(y)$.
- The cost decomposes into a sum over states in the sequence.

Cost functions

$$\text{cost}(y, y') = \mathbb{1}_{y \neq y'}$$

Problem: a sequence off by one label should be "less wrong" than a sequence all labels.

Cost functions

Examples

$$\text{cost}(z, z') = \sum_{t=1}^{T} \text{cost}(z_t, z'_t) = \sum_{t=1}^{T} \mathbb{1}_{z_t \neq z'_t}$$

$$\text{cost}(z, z') = \sum_{t=1}^{T} \text{cost}(z_t, z'_t) = \sum_{t=1}^{T} \mathbb{1}_{\text{label}(z_t) \neq \text{label}(z'_t)}$$

Good approximation for edit distance?

Revisiting log loss

$$\begin{aligned} p(y|x) &= E_{y' \sim p(y'|x)}[\mathbb{1}_{y=y'}] \\ &= 1 - E_{y' \sim p(y'|x)}[\mathbb{1}_{y \neq y'}] \end{aligned}$$

$$\begin{aligned} \operatorname*{argmin}_{\theta} -\log p(y|x) &= \operatorname*{argmax}_{\theta} p(y|x) \\ &= \operatorname*{argmax}_{\theta} 1 - E_{y' \sim p(y'|x)}[\mathbb{1}_{y \neq y'}] \\ &= \operatorname*{argmin}_{\theta} E_{y' \sim p(y'|x)}[\mathbb{1}_{y \neq y'}] \end{aligned}$$

MMI is EBR with indicator cost.

Boosted MMI (bMMI)

$$-\log p(y|x) + \log \sum_{y'} \left[\mathsf{cost}(y, y') + p(y'|x)\right]$$

Like log loss, but pay more attention to bad hypotheses.

Hinge loss (max margin, large margin)

$$\max_{y'} \left[\text{cost}(y, y') - \log p(y|x) + \log p(y'|x) \right]$$

- Support Vector Machines (SVMs)
  = linear models optimized with hinge loss

Some results

(Veselý et al., 2013)

System	SWB	CHE	Total
ML GMM	21.2	36.4	28.8
GMM BMMI	18.6	33.0	25.8
DNN CE	14.2	25.7	20.0
DNN MMI	12.9	24.6	18.8
DNN sMBR	12.6	24.1	18.4
DNN MPE	12.9	24.1	18.5
DNN BMMI	12.9	24.5	18.7

DNN = hybrid HMM/NN model with feedforward DNN-based state posteriors

Summary: Discriminative training

- Discriminative training with various non-ML criteria has had huge impact on HMM-based speech recognition
- For end-to-end neural models, log loss is still by far most commone
  - As we've seen, this is not such a bad choice!
  - Some recent work is considering alternative losses for end-to-end neural models as well (e.g., [Prabhavalkar+ 2018])