Deep Learning for Natural Language Processing

Lecture 3 – Backpropagation and language models

Dr. Ivan Habernal

April 26, 2022

Trustworthy Human Language Technologies
Department of Computer Science
Technical University of Darmstadt

www.trusthlt.org

Outline

Recap: Supervised learning and loss functions

Training as optimiziation

Backpropagation

Language models

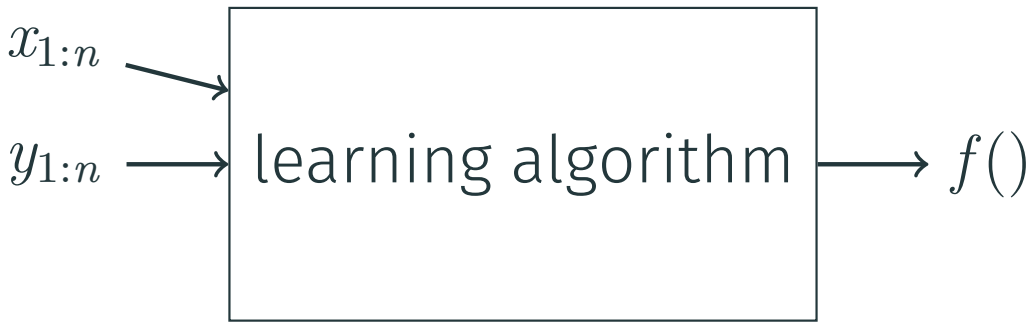
Recap: Supervised learning and loss functions

Recap: Supervised learning

Input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where

- $x_{1:n} = x_1, x_2, ..., x_n$ are input examples
- $y_{a:n} = y_1, y_2, ..., y_n$ are their labels

Find a function $f()$ that maps input examples to their desired labels as accurately as possible



Training as Optimization

Loss function $L(y, \hat{y})$ quantifies the loss suffered when predicting $\hat{y}$ while the true label is y

Find the values of the parameters such that the overall loss $\mathcal{L}$ is minimized

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}, y_i)$$

Logistic loss (Binary cross entropy)

Assumes a set of two target classes labeled 0 and 1, with a correct label $y \in \{0, 1\}$

$$L_{\text{logistic}}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

Model output $\tilde{y}$ transformed using the **sigmoid function**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Binary classification with conditional probability outputs: $\hat{y} = \sigma(\tilde{y}) = \Pr(y = 1|x)$

Inference rule: prediction$= 0$ if $\hat{y} < 0.5$ and prediction $= 1$ if $\hat{y} \geq 0.5$

Categorical cross-entropy loss

True labels y for each example is a vector = probability distribution over C classes

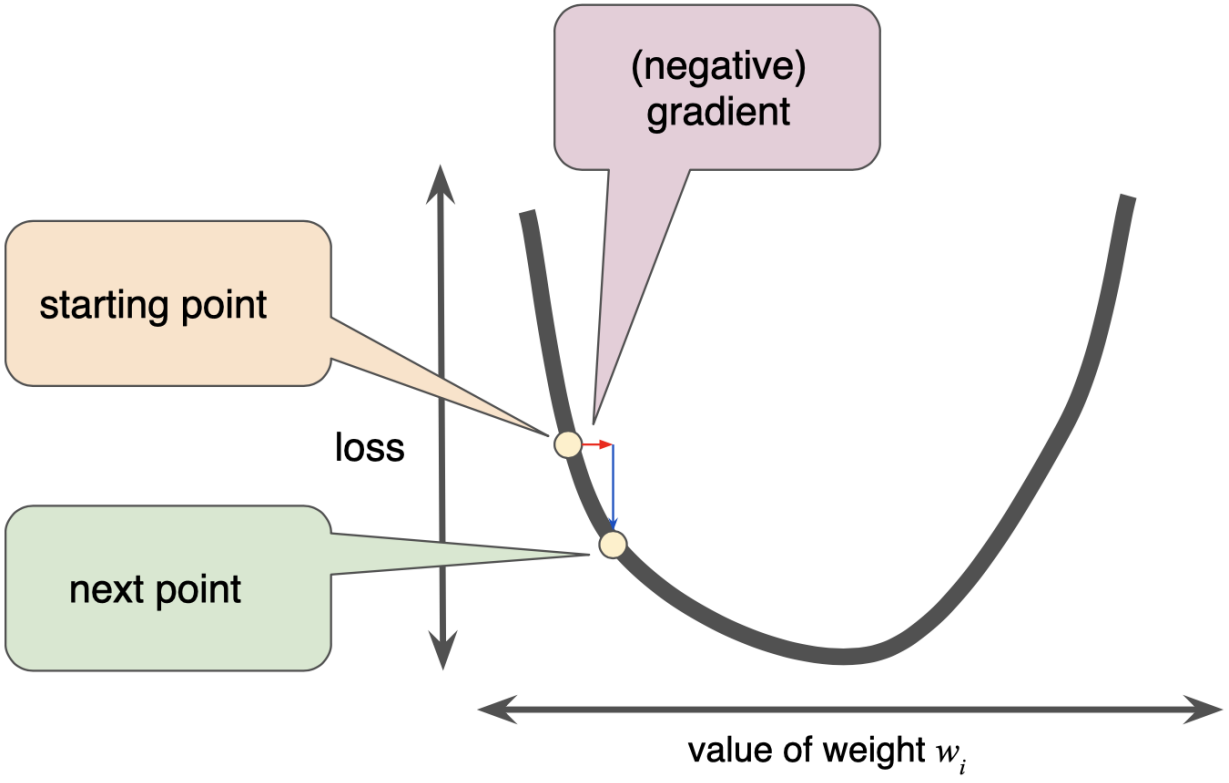
Prediction vector $\hat{y}$ transformed through **softmax**

$$\hat{y}[i] = \frac{\exp\left(z_{[i]}\right)}{\sum_{j=1}^{|C|} \exp\left(z_{[j]}\right)}$$

$$L_{\text{cross-entropy}}(\hat{y}, y) = -\sum_{i=1}^{|C|} y_{[i]} \log\left(\hat{y}_{[i]}\right)$$

Training as optimiziation

Gradient-based Optimization



Gradient-based Optimization

Compute the loss over the training set

Compute the gradient of the loss with respect to the parameters

Update parameter values in the opposite directions of the gradient

(Online) Stochastic Gradient Descent

Algorithm 2.1 Online stochastic gradient descent training.

Input:

- Function $f(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function L.

1: **while** stopping criteria not met **do**
2: Sample a training example $\boldsymbol{x_i}, \boldsymbol{y_i}$
3: Compute the loss $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$
4: $\hat{\boldsymbol{g}} \leftarrow$ gradients of $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
5: $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
6: **return** Θ

(Taken from: Neural Network Methods for Natural Language Processing, Yoav Goldberg)

(Minibatch) Stochastic Gradient Descent

Algorithm 2.2 Minibatch stochastic gradient descent training.

Input:
- Function $f(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function L.

---

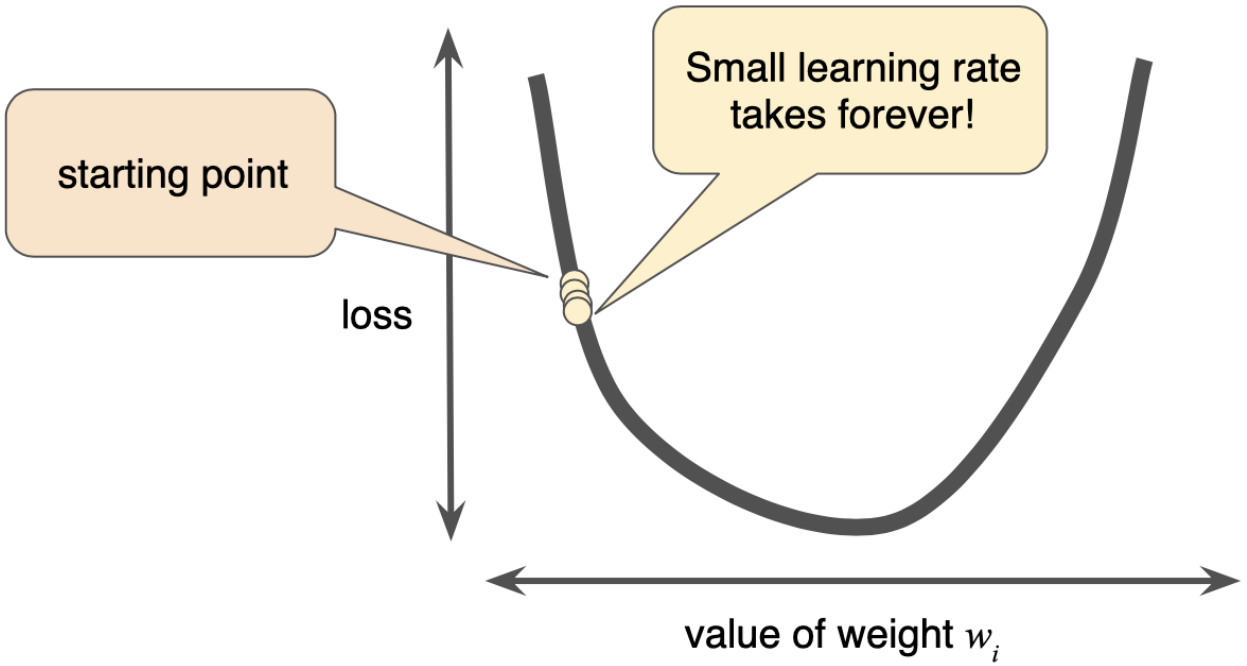
while stopping criteria not met **do**
  Sample a minibatch of m examples $\{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_m}, \boldsymbol{y_m})\}$
  $\hat{\boldsymbol{g}} \leftarrow 0$
  for $i = 1$ to m **do**
    Compute the loss $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$
    $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} +$ gradients of $\frac{1}{m} L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
  $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
return Θ

Why Minibatch SGD?

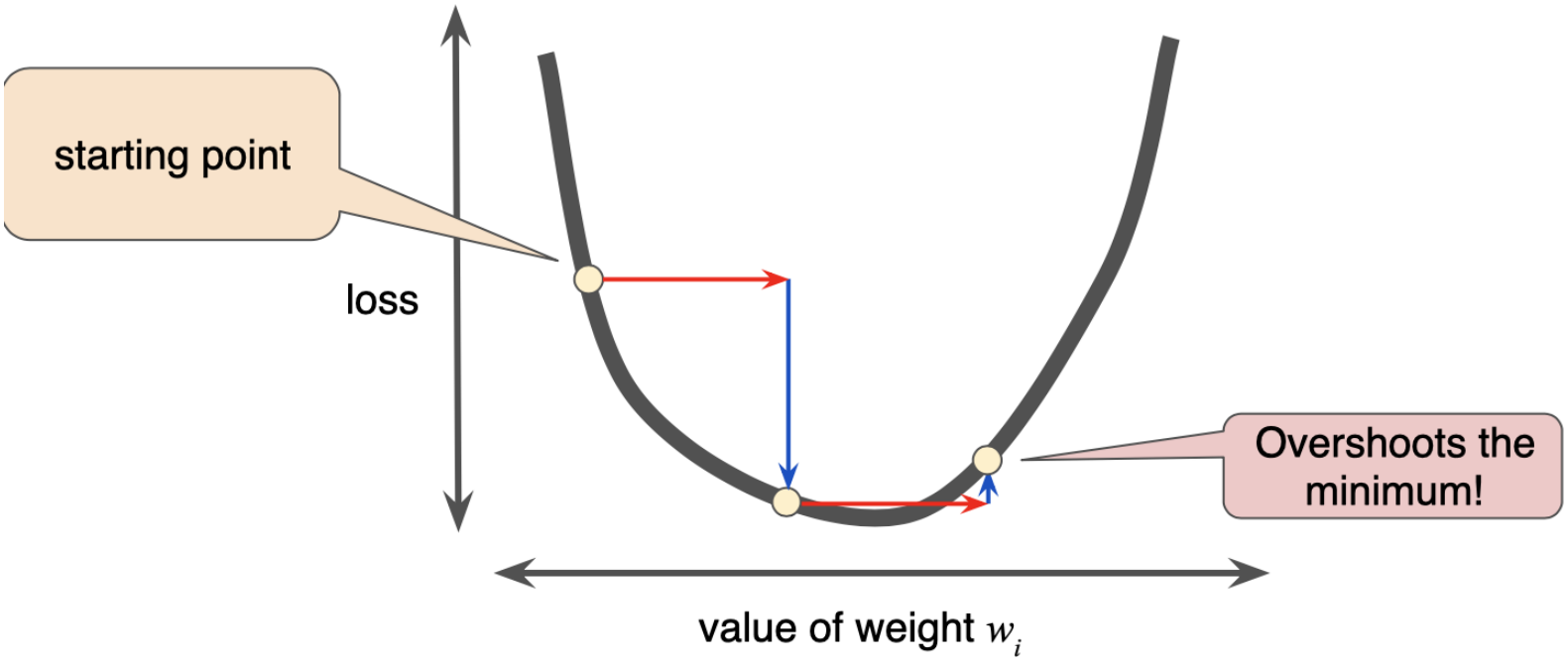
Less expensive than computing gradient on all training examples and then update

Converges empirically faster to a good solution than full-batch learning

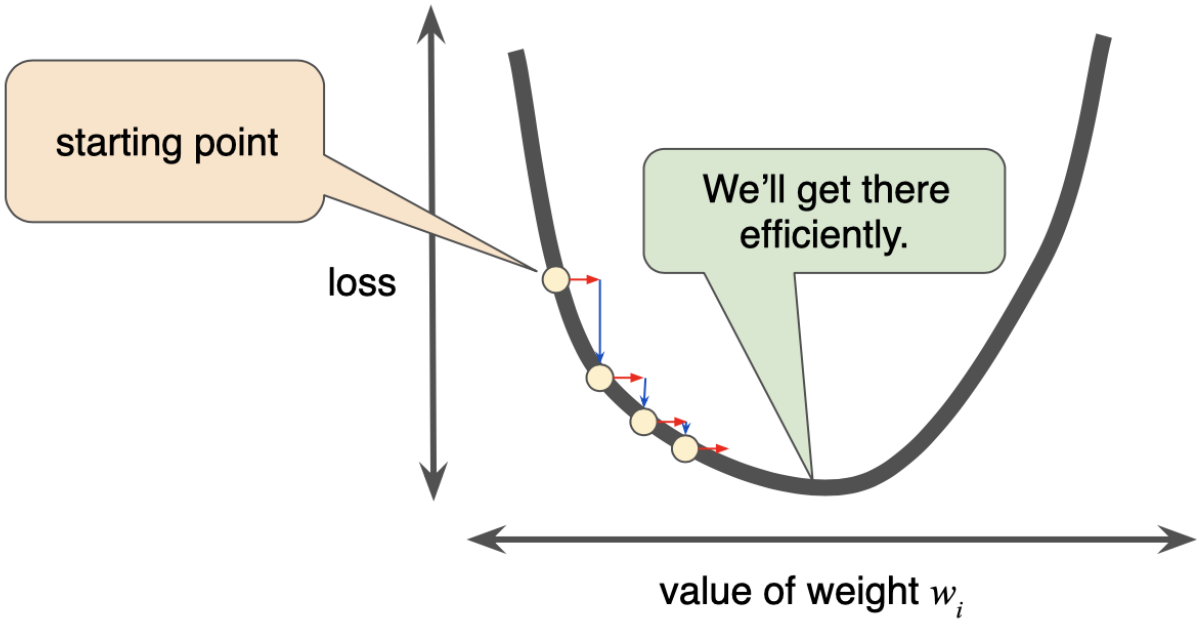
Small Learning Rate



Large Learning Rate



Adaptive Learning Rate



- Adam (Kingma and Ba, 2015)
- AdaGrad [Duchi et al., 2011]
- AdaDelta [Zeiler, 2012]
- RMSProp [Tieleman and Hinton, 2012]

These methods train usually faster than SGD

Backpropagation

Backpropagation

- Recursive algorithm that computes the derivatives of a nested functions using the chain rule, while caching intermediary derivatives
- Chain rule: Let $L = f(g(w))$

$$\frac{\partial L}{\partial w} = \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial w}$$

Chain rule example

Consider $y = e^{\sin(x^2)}$

$$y = f(u) = e^u$$
$$u = g(v) = \sin v = \sin\left(x^2\right)$$
$$v = h(x) = x^2$$

$$\frac{dy}{du} = f'(u) = e^u = e^{\sin(x^2)}$$
$$\frac{du}{dv} = g'(v) = \cos v = \cos\left(x^2\right)$$
$$\frac{dv}{dx} = h'(x) = 2x$$

Derivative at the point $x = a$ is (in Leibniz notation)

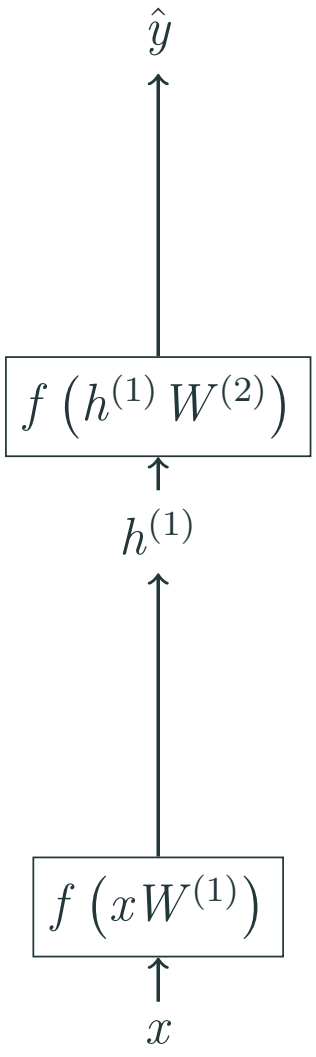
$$\frac{dy}{dx} = \left.\frac{dy}{du}\right|_{u=g(h(a))} \cdot \left.\frac{du}{dv}\right|_{v=h(a)} \cdot \left.\frac{dv}{dx}\right|_{x=a}$$

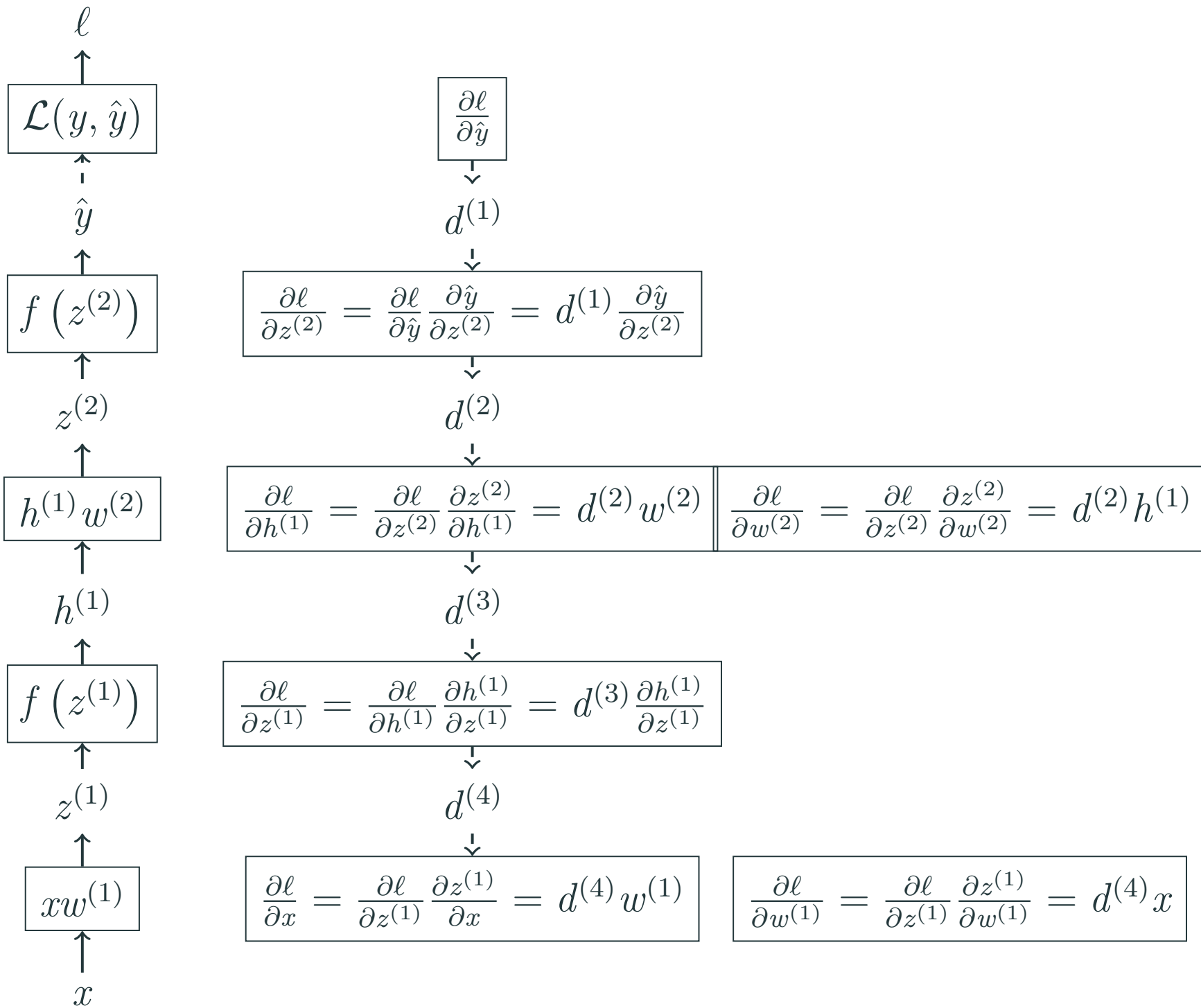
Backpropagation

Consists of two steps

- Forward pass $\rightarrow$ use current parameter values to compute the loss value
- Backward pass $\rightarrow$ use the gradient of the loss to update the parameter values

Model: Multilayer Perceptron (MLP)




ℓ
$\mathcal{L}(y, \hat{y})$
$\hat{y}$
$f\left(z^{(2)}\right)$
$z^{(2)}$
$h^{(1)}\, w^{(2)}$
$h^{(1)}$
$f\left(z^{(1)}\right)$
$z^{(1)}$
$xw^{(1)}$
x
$\frac{\partial \ell}{\partial \hat{y}}$
$d^{(1)}$
$\frac{\partial \ell}{\partial z^{(2)}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} = d^{(1)} \frac{\partial \hat{y}}{\partial z^{(2)}}$
$d^{(2)}$
$\frac{\partial \ell}{\partial h^{(1)}} = \frac{\partial \ell}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(1)}} = d^{(2)}\, w^{(2)}$
$\frac{\partial \ell}{\partial w^{(2)}} = \frac{\partial \ell}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}} = d^{(2)}\, h^{(1)}$
$d^{(3)}$
$\frac{\partial \ell}{\partial z^{(1)}} = \frac{\partial \ell}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial z^{(1)}} = d^{(3)} \frac{\partial h^{(1)}}{\partial z^{(1)}}$
$d^{(4)}$
$\frac{\partial \ell}{\partial x} = \frac{\partial \ell}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial x} = d^{(4)}\, w^{(1)}$
$\frac{\partial \ell}{\partial w^{(1)}} = \frac{\partial \ell}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}} = d^{(4)}\, x$

Backprop + SGD

Output of backprop is gradient wrt. parameters of the neural network

Once we have the gradient we can use SGD rule to update the parameter values

Language models

Language Models (LMs)

Language modeling = assigning probability to a sentence in a language

Example: $\Pr(\text{`The cat sat on the mat .'})$

Ideal performance at language modeling: predict the next token in a sequence with a number of guesses that is the identical to or lower than the number of guesses required by a human

Crucial component in real-world NLP applications: conversational AI, machine-translation, text summarization, etc.

Language Models (LMs)

Assume a sequence of words $w_{1:n} = w_1 w_2 \ldots w_{n-1} w_n$

$$\begin{aligned}\Pr(w_1, w_2, \ldots, w_{n-1}, w_n) = &\Pr(w_1) \cdot \\ &\Pr(w_2|w_1) \cdot \\ &\Pr(w_3|w_1, w_2) \cdot \\ &\Pr(w_4|w_1, w_2, w_3) \cdot \\ &\ldots \cdot \Pr(w_n|w_{1:n-1})\end{aligned}$$

Each word is predicted conditioned on the preceding words

Probability of the last token conditioned on $n-1$ preceding words

Markov assumption

kth order Markov-assumption: next word in a sequence depends only on the last k words

$$\Pr(w_i|w_{1:i-1}) \approx \Pr\left(w_i|w_{(i-1)-k:i-1}\right)$$

Probability of a sequence of tokens $w_{1:n}$

$$\Pr(w_{1:n}) \approx \prod_{i=1}^{n} \Pr(w_i|w_{i-k:i-1})$$

Computationally-friendly version (What is the problem with the above formula?)

Markov assumption

kth order Markov-assumption: next word in a sequence depends only on the last k words

$$\Pr(w_i|w_{1:i-1}) \approx \Pr\left(w_i|w_{(i-1)-k:i-1}\right)$$

Probability of a sequence of tokens $w_{1:n}$

$$\Pr(w_{1:n}) \approx \prod_{i=1}^{n} \Pr(w_i|w_{i-k:i-1})$$

Computationally-friendly version (What is the problem with the above formula?)

$$\log_2 \Pr(w_{1:n}) \approx \sum_{i=1}^{n} \log_2 \left(\Pr(w_i|w_{i-k:i-1})\right)$$

Evaluating language models

Perplexity: How well a LM predicts likelihood of unseen sentence

$$\text{Perp}_{w_{1:n}}(\text{LM}) = 2^{-\frac{1}{n}\sum_{i=1}^{n}\log_2 \text{LM}(w_i|w_{1:i-1})}$$

Low perplexity values $\rightarrow$ better language model (assigns high probabilities to the unseen sentences)

We can compare several language models with one another

Perplexities of two language models are only comparable with respect to the same evaluation dataset

Estimating Probabilities

Count-based: The estimates can be derived from corpus counts

Let $\#(w_{i:j})$ be the count of the sequence of words $w_{i:j}$ in a corpus

The maximum likelihood estimate (MLE) of $\Pr(w_i|w_{i-1-k:i-1})$

$$\Pr(w_i|w_{i-1-k:i-1}) = \frac{\#(w_{i-1-k:i})}{\#(w_{i-1-k:i-1})}$$

Example: $w_1 w_2 w_3 =$ the cat sat

$$\Pr(w_3|w_{1:2}) = \frac{\#(\text{the cat sat})}{\#(\text{the cat})}$$

Estimating Probabilities

What if $\#(w_{i:j}) = 0$?

Estimating Probabilities

What if $\#(w_{i:j}) = 0$?

$\rightarrow$ infinite perplexity

One way of avoiding zero-probability N-grams is to use smoothing techniques

Additive smoothing: assume $|V|$ is the vocabulary size and $0 < \alpha \leq 1$

$$\Pr(w_i | w_{i-1-k:i-1}) = \frac{\#(w_{i-1-k:i}) + \alpha}{\#(w_{i-1-k:i-1}) + \alpha |V|}$$

Pro and Cons of Discussed LMs

Easy to train, scale to large corpora, and work well in practice

Scaling to larger N-grams is a problem for MLE-based language models.

Large number of words in the vocabulary means that statistics for larger N-grams will be sparse

MLE-based language models suffer from lack of generalization across contexts

Having observed “black car” and “blue car” does not influence our estimates of the sequence “red car” if we haven’t seen it before

Neural Language Models

Use neural networks to estimate probabilities of a LM

We can overcome the shortcomings of the MLE-based LMs because neural networks

- enable conditioning on increasingly large context sizes with only a linear increase in the number of parameters
- support generalization across different contexts

We focus on the neural LM that was introduced by (Bengio et al., 2003)

Neural Language Models

Let $w_{1:k}$ be the given context

We want to estimate $\Pr(w_{k+1}|w_{1:k})$

Design an MLP neural net, which takes $w_{1:k}$ as input and returns $\Pr(w_{k+1})$ over all words in vocabulary V as output

$$x = [v(w_1), v(w_2), ..., v(w_k)]$$
$$h^{(1)} = g(xW^{(1)} + b^{(1)})$$
$$\Pr(w_{k+1}) = \text{softmax}(h^{(1)}W^{(2)} + b^{(2)})$$

Training examples: word k-grams from the training set, where the identities of the first $k-1$ words are used as features. Last word: target label for the classification

Loss function: cross-entropy loss

(Neural) LMs for language generation

Assume that we are given $w_{1:k}$ as context

Predict the next word w_{k+1} from vocabulary V

Feed $w_{1:k}$ to our trained MLP-based LM

Our LM returns $\Pr(w_{k+1})$ of each word in V

Pick the word with the maximum probability to generate the next word

Add the the predicted word to the context and repeat the above procedure

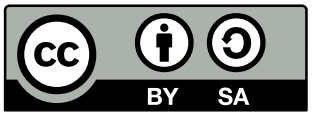
References

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin (2003). “A Neural Probabilistic Language Model”. In: *Journal of Machine Learning Research* 3, pp. 1137–1155. URL: https://research.jmlr.org/papers/v3/bengio03a.html.

References ii

Kingma, D. P. and J. L. Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015*. Ed. by Y. Bengio and Y. LeCun. San Diego, CA, USA, pp. 1–15. URL: https://arxiv.org/abs/1412.6980.

License and credits



Credits

Ivan Habernal, Mohsen Mesgar, Steffen Eger

Content from ACL Anthology papers licensed under CC-BY
`https://www.aclweb.org/anthology`

Pictures courtesy of `https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent`