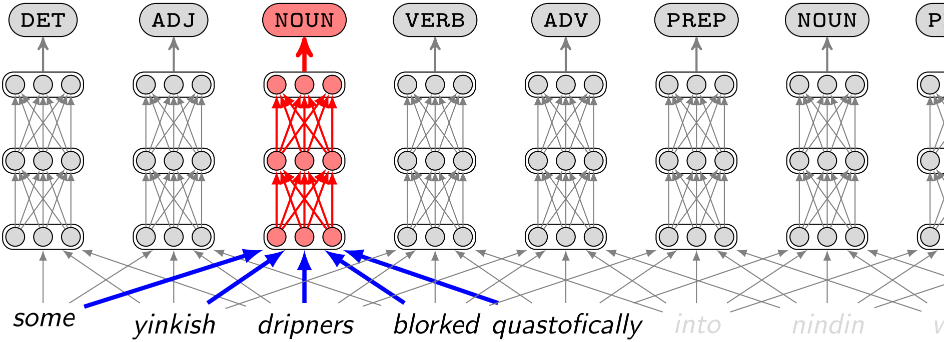
Overview of Natural Language Processing Part II & ACS L390

Gradient Descent and Neural Nets

Weiwei Sun

Department of Computer Science and Technology
University of Cambridge

Michaelmas 2024/25



Gradient Descent and Neural Nets

1. Gradient Descent
2. Feedforward Neural Networks

Gradient Descent

Supervised learning

Assume there is a *good* annotated corpus

$$\mathcal{D} = \left\{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(l)}, y^{(l)}) \right\}$$

How can we get a *good* parameter vector?

Maximum-Likelihood Estimation

$$\hat{\theta} = \arg\max L(\theta)$$

where $L(\theta)$ is the log-likelihood of observing the data $\mathcal{D}$:

$$L(\theta) = \sum_{i=1}^{l} \log p(y^{(i)} | x^{(i)}; \theta)$$

Gradient Descent/Ascent

In general, finding a minimum/maximum is *hard*.

However, a simple idea that often works:

- Initialise θ with some value
- Iteratively improve θ

The derivative tells us whether to increase or decrease (but doesn't tell us how much to increase/decrease by):

$$\theta^{[t+1]} = \theta^{[t]} + \beta \frac{dL}{d\theta}(\theta^{[t]})$$

Gradient Descent for the Log-Linear Model

Assume we have a *parameter vector* θ.

$$p(y|x;\theta) = \frac{\exp(\theta^\top f(x,y))}{\sum_{y' \in \mathcal{Y}} \exp(\theta^\top f(x,y'))}$$

i.e.

$$p(y|x;\theta) \propto \exp(\theta^\top f(x,y))$$

$$\begin{aligned} L(\theta) &= \sum_{i=1}^{l} \log p(y^{(i)}|x^{(i)};\theta) \\ &= \sum_{i=1}^{l} \left(\theta^\top f(x^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp(\theta^\top f(x^{(i)}, y')) \right) \end{aligned}$$

Gradient Descent for the Log-Linear Model

$$L(\theta) = \sum_{i=1}^{l} \left(\theta^\top f(x^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp(\theta^\top f(x^{(i)}, y')) \right)$$

Calculating gradients (chain rule)

$$\begin{aligned}
\frac{dL}{d\theta_k} &= \sum_{i=1}^{l} \left(f_k(x^{(i)}, y^{(i)}) - \frac{\sum_{y' \in \mathcal{Y}} \left(\exp(\theta^\top f(x^{(i)}, y')) f_k(x^{(i)}, y')\right)}{\sum_{y^* \in \mathcal{Y}} \exp(\theta^\top f(x^{(i)}, y^*))} \right) \\
&= \sum_{i=1}^{l} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{l} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \frac{\exp(\theta^\top f(x^{(i)}, y'))}{\sum_{y^* \in \mathcal{Y}} \exp(\theta^\top f(x^{(i)}, y^*))}
\end{aligned}$$

Gradient Descent for the Log-Linear Model

$$L(\theta) = \sum_{i=1}^{l} \left(\theta^\top f(x^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp(\theta^\top f(x^{(i)}, y')) \right)$$

Calculating gradients (chain rule)

$$\begin{aligned}
\frac{dL}{d\theta_k} &= \sum_{i=1}^{l} \left(f_k(x^{(i)}, y^{(i)}) - \frac{\sum_{y' \in \mathcal{Y}} \left(\exp(\theta^\top f(x^{(i)}, y')) f_k(x^{(i)}, y') \right)}{\sum_{y^* \in \mathcal{Y}} \exp(\theta^\top f(x^{(i)}, y^*))} \right) \\
&= \sum_{i=1}^{l} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{l} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \frac{\exp(\theta^\top f(x^{(i)}, y'))}{\sum_{y^* \in \mathcal{Y}} \exp(\theta^\top f(x^{(i)}, y^*))} \\
&= \underbrace{\sum_{i=1}^{l} f_k(x^{(i)}, y^{(i)})}_{\textit{empirical counts}} - \underbrace{\sum_{i=1}^{l} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' | x^{(i)}; \theta)}_{\textit{expected counts}}
\end{aligned}$$

Gradient Descent: Algorithm

Maximize $L(\theta)$ where

$$\frac{dL}{d\theta_k} = \underbrace{\sum_{i=1}^{l} f_k(x^{(i)}, y^{(i)})}_{\textit{empirical counts}} - \underbrace{\sum_{i=1}^{l} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' | x^{(i)}; \theta)}_{\textit{expected counts}}$$

Initialize $\theta^{[0]} \leftarrow 0$
for $t = 1, \dots$
    calculate $\Delta = \frac{dL(\theta^{[t-1]})}{d\theta}$
    calculate $\beta_* = \arg\max_\beta L(\theta + \beta \Delta)$ ▷ line search
    update $\theta^{[t]} \leftarrow \theta^{[t-1]} + \beta_* \Delta$

Gradient Descent: Algorithm

Maximize $L(\theta)$ where

$$\frac{dL}{d\theta_k} = \underbrace{\sum_{i=1}^{l} f_k(x^{(i)}, y^{(i)})}_{\textit{empirical counts}} - \underbrace{\sum_{i=1}^{l} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' | x^{(i)}; \theta)}_{\textit{expected counts}}$$

1 **Initialize** $\theta^{[0]} \leftarrow 0$
2 **for** $t = 1, \ldots$
3 **calculate** $\Delta = \frac{dL(\theta^{[t-1]})}{d\theta}$
4 **calculate** $\beta_* = \arg\max_\beta L(\theta + \beta\Delta)$ ▷ line search
5 **update** $\theta^{[t]} \leftarrow \theta^{[t-1]} + \beta_* \Delta$

Challenges

- Go through every training sample to get Δ;
- Line search is another non-trival optimisation problem

Stochastic Gradient Descent

$$L(\theta) \quad = \quad \sum_{i=1}^{l} \log p(y^{(i)}|x^{(i)};\theta)$$

- Randomly use one or several training samples to get a suboptimal gradient Δ';
- Fix β.

Feedforward Neural Networks

Recap: about linear combination

$$p(y|x;\theta) = \frac{\exp(\theta^\top f(x,y))}{\sum_{y' \in \mathcal{Y}} \exp(\theta^\top f(x,y'))}$$

⋯	0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	⋯

f_{1001}: `if` word$_{-2}$=*some* and tag=N

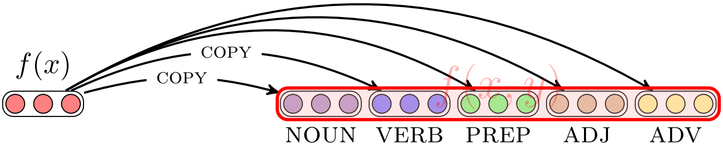
is θ_{1001} positive large?
vote for yes

$\theta^\top f(x,y)$ is a linear combination of θ. That is why a log-linear model is called a linear classifier.

Questions

Can we automate the design of features?

Is linear combination justified?

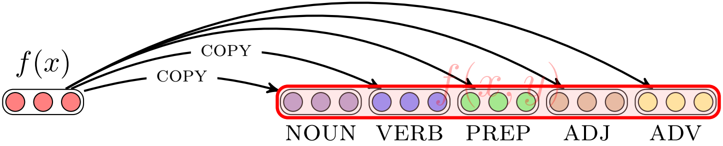


A simple way to define $f(x, y)$ based on $f(x)$ is "copy". We assumed that θ and $f(x, y)$ have DK dimensions:

- D – number of input features
- K – number of output classes

So we can also view θ as comprising K vectors with D dimensions:

$$p(y|x;\theta) \propto \exp(\theta_y^\top f(x))$$

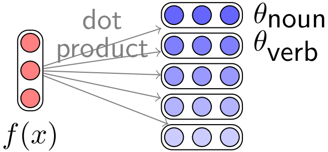


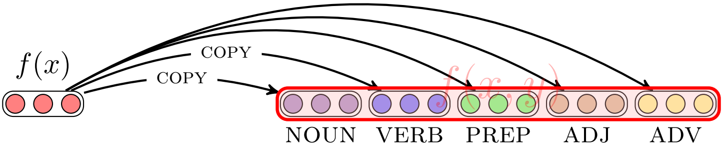
A simple way to define $f(x, y)$ based on $f(x)$ is "copy". We assumed that θ and $f(x, y)$ have DK dimensions:

- D – number of input features
- K – number of output classes

So we can also view θ as comprising K vectors with D dimensions:

$$p(y|x; \theta) \propto \exp(\theta_y^\top f(x))$$



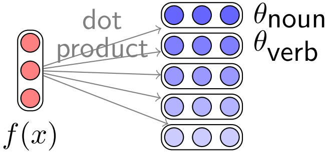


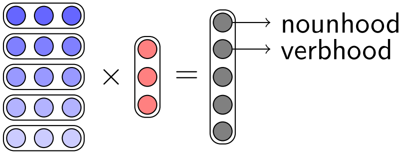
A simple way to define $f(x, y)$ based on $f(x)$ is "copy". We assumed that θ and $f(x, y)$ have DK dimensions:

- D – number of input features
- K – number of output classes

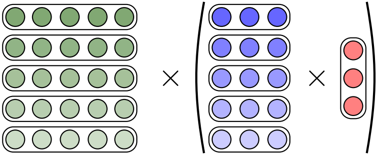
So we can also view θ as comprising K vectors with D dimensions:

$$p(y|x;\theta) \propto \exp(\theta_y^\top f(x))$$

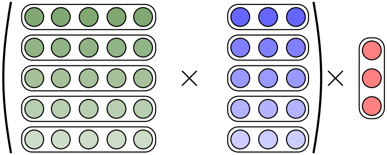




Now consider NER after POS tagging

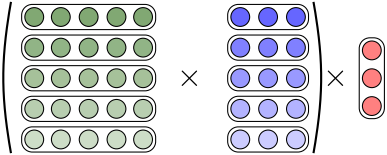


Now consider NER after POS tagging



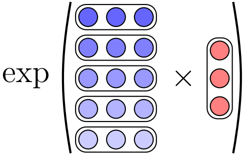
Oops! The combined transformation is linear!

Now consider NER after POS tagging



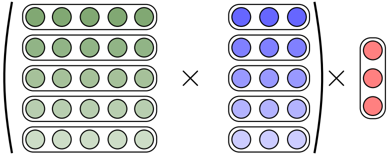
Oops! The combined transformation is linear!

Introduce some non-linearity



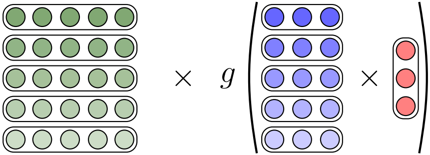
$\exp / g$ is applied component-wise (to each dimension separately)

Now consider NER after POS tagging



Oops! The combined transformation is linear!

Introduce some non-linearity



$\exp / g$ is applied component-wise (to each dimension separately)

Feedforward Neural Networks

Think about multi-class classificaition:

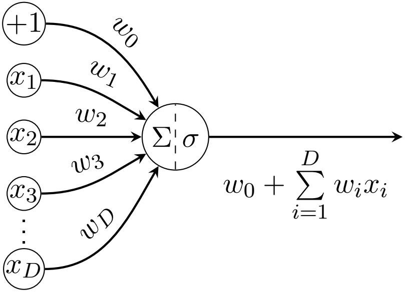
- D – number of features (input)
- K – number of classes (output)
- $\mathbf{x}$ – the input feature vector

Think about a particular class, say y_k. We describe the "friendship" between $\mathbf{x}$ and y_k in the following way:

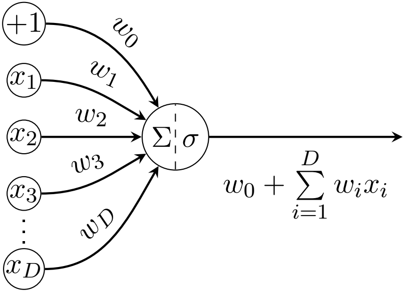
$$\text{score\_function}(x, y_k) = w_0 + \sum_{i=1}^{D} w_i x_i$$

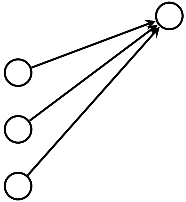
where $\mathbf{w}$ measures how much each feature w_i contributes to y_k.

Feedforward Neural Networks



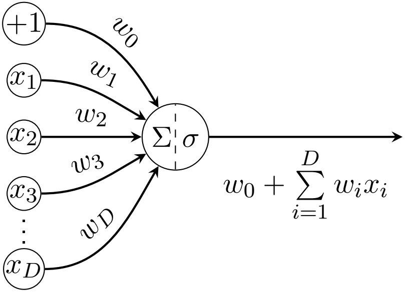
Feedforward Neural Networks

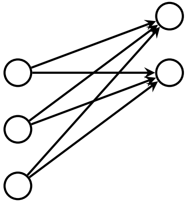




For each class y_k, we do the same thing.

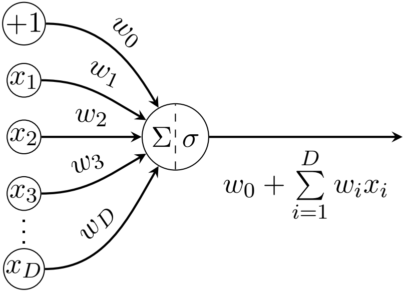
Feedforward Neural Networks

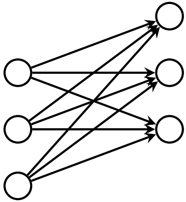




For each class y_k, we do the same thing. Again

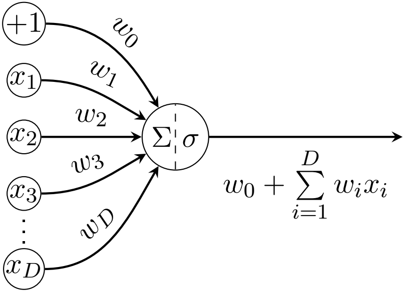
Feedforward Neural Networks

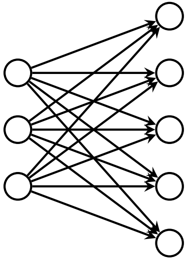




For each class y_k, we do the same thing. Again and again

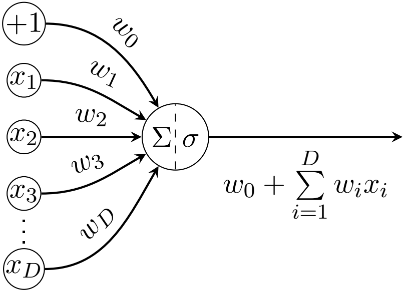
Feedforward Neural Networks

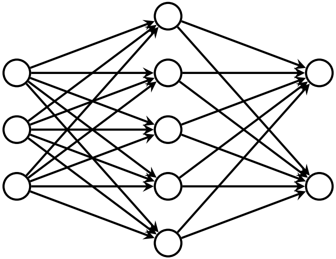




For each class y_k, we do the same thing. Again and again and again. This is called perceptron, which was invented by Frank Rosenblatt in 1958.

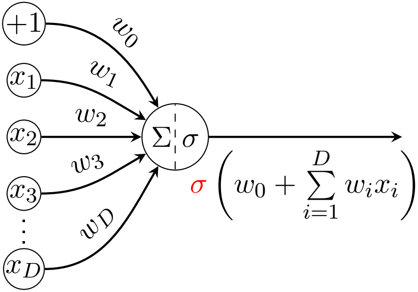
Feedforward Neural Networks

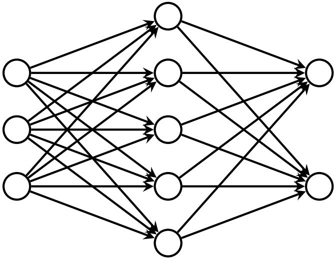




For each class y_k, we do the same thing. Again and again and again. This is called perceptron, which was invented by Frank Rosenblatt in 1958.
Things will be much more fun if we have a stack of perceptrons.

Feedforward Neural Networks



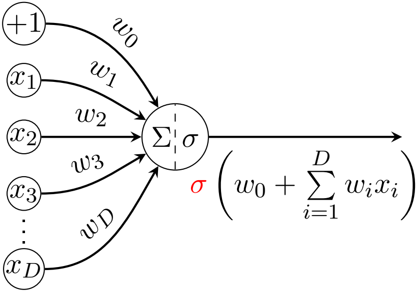


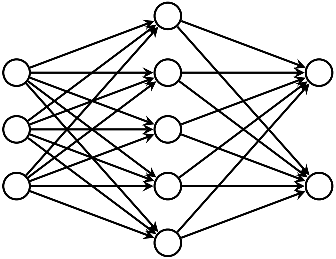
For each class y_k, we do the same thing. Again and again and again. This is called perceptron, which was invented by Frank Rosenblatt in 1958. Things will be much more fun if we have a stack of perceptrons. Oops, must add something...

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

Otherwise, simple matrix multiplication.

Feedforward Neural Networks





For each class y_k, we do the same thing. Again and again and again. This is called perceptron, which was invented by Frank Rosenblatt in 1958. Things will be much more fun if we have a stack of perceptrons. Oops, must add something...

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

Otherwise, simple matrix multiplication. Now you can do non-linear classification.

Feedforward Neural Networks

A feedforward neural network is a composition of simple functions:

- 1 layer: $\exp(W_1 x)$ ▷ log-linear model
- 2 layers: $\exp(W_2 g(W_1 x))$
- 3 layers: $\exp(W_3 g(W_2 g(W_1 x)))$
- 4 layers: $\exp(W_4 g(W_3 g(W_2 g(W_1 x))))$
- . . .

Both $\exp$ and g are applied component-wise (to each dimension separately)

Feedforward Neural Networks

A feedforward neural network is a composition of simple functions:

- 1 layer: $\exp(W_1 x)$ ▷ log-linear model
- 2 layers: $\exp(W_2 g(W_1 x))$
- 3 layers: $\exp(W_3 g(W_2 g(W_1 x)))$
- 4 layers: $\exp(W_4 g(W_3 g(W_2 g(W_1 x))))$
- $\ldots$

Both $\exp$ and g are applied component-wise (to each dimension separately)

The activation function g should be non-linear, e.g.:

- Rectified linear unit: $\text{ReLU}(z) = \max(0, z)$
- Hyperbolic tangent: $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$
- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$

Gradient Descent for Neural Nets

Assume there is a good annotated corpus:

$$\mathcal{D} = \left\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(l)}, y^{(l)})\right\}$$

Aim to maximise the log-likelihood (also called "cross entropy"):

$$L(\theta) = \sum_{i=1}^{l} \log p(y^{(i)} | x^{(i)}; \theta)$$

Gradient Descent for Neural Nets

Assume there is a good annotated corpus:

$$\mathcal{D} = \left\{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(l)}, y^{(l)}) \right\}$$

Aim to maximise the log-likelihood (also called "cross entropy"):

$$L(\theta) = \sum_{i=1}^{l} \log p(y^{(i)} | x^{(i)}; \theta)$$

θ now contains parameters from many layers,
but we can still use gradient descent:

- Backpropagation: efficient application of chain rule
- Iterate many times over training set

Reading

- D Jurafsky and J Martin. *Speech and Language Processing* Chapter 6. `web.stanford.edu/~jurafsky/slp3/6.pdf`

* Essence of linear algebra `www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab`