CIE 555
Neural Networks and Deep Learning

Deep Feedforward Networks

1

Overview

- Review: Maximum likelihood estimation (Section 5.5)
- Review: Regularization (Section 5.2.2)
- Review: Cross-entropy (Section 3.13)
- Neural networks and linear models.
- Deep Feedforward Networks
- Example: Learning XOR (Section 6.1)
- Gradient-Based Learning (Section 6.2)
  - Cost Functions
  - Output Units

2

Maximum Likelihood Estimation

- Let the sample set $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(x)$.
- Let $p_{model}(x; \theta_m)$ a parametric family of probability distributions that maps any configuration *x* to a real number estimating the true probability $p_{data}(x)$.
- The ideal model is an oracle that simply knows the true probability distribution that generates the data.
- If we would like the same dataset to be generated using the model then $p_{model}(\mathbb{X}; \theta)$ needs to be maximized. (that is each point in $\mathbb{X}$ is more likely to be generated from the distribution parameterized by θ)

Goodfellow, Bengio, Courville 2016

3

Maximum Likelihood Estimation

- The maximum likelihood estimator for θ is then defined as:

$$\theta_{ML} = \arg\max_{\theta} p_{model}(\mathbb{X}; \theta)$$

Since all samples in $\mathbb{X}$ are independent

$$\theta_{ML} = \arg\max_{\theta} \prod_{i=1}^{m} p_{model}(x^{(i)}; \theta)$$

To use summation instead of product

$$\theta_{ML} = \arg\max_{\theta} \sum_{i=1}^{m} log\, p_{model}(x^{(i)}; \theta)$$

Goodfellow, Bengio, Courville 2016

4

Point Estimators and Bayesian Statistics

- Point estimators calculates a single value (point) in the parameter space (the best guess).
- In these approaches the true parameter value θ is assumed to be fixed but unknown, while the calculated $\hat{\theta}$ is a function of the dataset (which is seen as random).
- The Bayesian perspective on statistics is quite different. The true parameter θ is unknown or uncertain and thus is represented as a random variable. Two main steps are
  - Before observing the data, we represent our knowledge of θ using the **prior probability distribution**, $p(\theta)$
  - Recover the effect of data on our belief about θ using Bayes' rule

$$p(\theta|x^{(1)}, \ldots, x^{(m)}) = \frac{p(x^{(1)}, \ldots, x^{(m)}|\theta)p(\theta)}{p(x^{(1)}, \ldots, x^{(m)})}$$

Goodfellow, Bengio, Courville 2016

5

Maximum *A Posteriori* (MAP) Estimation

- While the most principled approach is to make predictions using the full Bayesian posterior distribution over the parameter θ , it is still often desirable to have a single point estimate.
- One common reason for desiring a point estimate is that most operations involving the Bayesian posterior for most interesting models are intractable, and a point estimate offers a tractable approximation.
- Rather than simply returning to the maximum likelihood estimate, we can still gain some of the benefit of the Bayesian approach by allowing the prior to influence the choice of the point estimate using the **maximum a posteriori** (MAP) point estimate.
- As with full Bayesian inference, MAP Bayesian inference has the advantage of leveraging information that is brought by the prior and cannot be found in the training data.

Goodfellow, Bengio, Courville 2016

6

MAP Estimation

- The MAP estimate chooses the point of maximal posterior probability

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}}\, p(\theta|x) = \underset{\theta}{\operatorname{argmax}}\, \log p(x|\theta) + \log p(\theta)$$

- Using the prior of θ helps to reduce the variance in the MAP point estimate (in comparison to the ML estimate). However, it does so at the price of *increased bias*.
- Many regularized estimation strategies, such as maximum likelihood learning regularized with weight decay, can be interpreted as making the MAP approximation to Bayesian inference. This view applies when the regularization consists of adding an extra term to the objective function that corresponds to log $p(\theta)$.

Goodfellow, Bengio, Courville 2016

7

Regularization

- Regularization is *any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.*
- The no free lunch theorem has made it clear that there is no best machine learning algorithm, and, in particular, no best form of regularization. Instead we must choose a form of regularization that is well-suited to the particular task we want to solve.
- *Example:*we can modify the training criterion for linear regression to include **weight decay**. To perform linear regression with weight decay, we minimize a sum comprising both the mean squared error on the training and a criterion $J(w)$ that expresses a preference for the weights to have smaller squared L^2 norm. Specifically

$$J(w) = MSE_{train} + \lambda w^T w$$

- Minimizing $J(w)$ results in a choice of weights that make a tradeoff between fitting the training data and being small.

Goodfellow, Bengio, Courville 2016

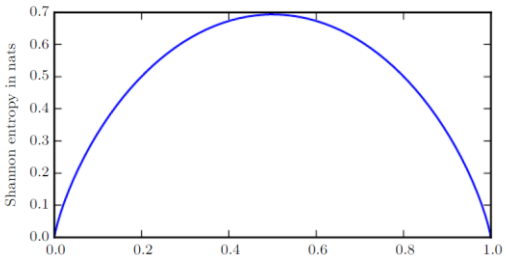
8

Entropy

- We can quantify the amount of **uncertainty** in an entire probability distribution using the **Shannon entropy**

$$H(x) = -\mathbb{E}_{x\sim P}[log_2\, P(x)]$$

- It gives a lower bound on the number of bits (if the logarithm is base 2) needed on average to encode symbols drawn from a distribution *P*.
- Distributions that are nearly deterministic (where the outcome is nearly certain) have low entropy.
- Distributions that are closer to uniform have high entropy.



Goodfellow, Bengio, Courville 2016

9

KL divergence

- If we have two separate probability distributions *P* (x) and *Q* (x) over the same random variable x, we can measure how different these two distributions are using the **Kullback-Leibler (KL) divergence** (also called **relative entropy**):

$$D_{KL}(P \parallel Q) = \mathbb{E}_{x\sim P}\left[log\frac{P(x)}{Q(x)}\right] = \mathbb{E}_{x\sim P}[logP(x) - logQ(x)]$$

- A measure of how one probability distribution is different from a second (non-negative).
- However, it is not a true distance measure because it is not symmetric $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$
- The KL divergence is 0 if and only if *P* and *Q* are the same distribution (for discrete variables)
- It is the **extra** amount of information (measured in bits if we use the base 2 logarithm) needed to send a message containing symbols drawn from probability distribution *P*, when we use a code that was designed to minimize the length of messages drawn from probability distribution *Q*.

Goodfellow, Bengio, Courville 2016

10

Cross-entropy

- A quantity that is closely related to the KL divergence is the **cross-entropy**
- Cross-entropy measures the expectation of the number of bits needed to send a message containing symbols drawn from probability distribution *P*, when we use a code that was designed to minimize the length of messages drawn from probability distribution *Q*.

$$H(P,Q) = H(P) + D_{KL}(P \parallel Q)$$
$$H(P,Q) = -\mathbb{E}_{x\sim P}[logQ(x)]$$

- Minimizing the cross-entropy with respect to *Q* is equivalent to minimizing the KL divergence, because *Q* does not participate in the omitted term.
- When a model is created based on a training set T, and then its cross-entropy is measured on a test set to assess how accurate the model is in predicting the test data.

Goodfellow, Bengio, Courville 2016

11

Neural networks and linear models

- Linear models, such as logistic regression and linear regression, are appealing because they may be fit efficiently and reliably, either in closed form or with convex optimization.
- The obvious defect is that they are limited to linear functions.
- To extend linear models to represent nonlinear functions of *x*, we can apply the linear model not to x itself but to a transformed input$\phi(x)$, where$\phi$is a non-linear transformation. We can also apply the kernel trick.
- How do we choose ϕ?

Goodfellow, Bengio, Courville 2016

12

Choosing the mapping function ϕ

1. Use a very generic ϕ, (Consider for example SVM based on the RBF kernel). Such models have enough capacity to fit any training set, the generalization to the test set, however, remains poor in some advanced problems since these mappings do not encode enough prior information to solve these problems.
2. Manually engineer ϕ (was a dominant approach before deep learning): This approach requires considerable human effort for each separate task, with practitioners specializing in different domains such as speech recognition or computer vision, and with little transfer between domains.
3. Learn ϕ (the deep learning approach): we have a model $y = f(x; \theta, w) = \phi(x; \theta)^T w$.

Goodfellow, Bengio, Courville 2016

13

Learning ϕ

- We have parameters θ that we use to learn ϕ from a broad class of functions, and parameters w that map from $\phi(x)$ to the desired output.
- Can capture the benefit of the first approach by being highly generic—we do so by using a very broad family $\phi(x; \theta)$.
- Can also capture the benefit of the second approach as it allows human practitioners to encode their knowledge by designing families $\phi(x; \theta)$ that they expect will perform well (the advantage here is that they only need to find the right general function rather than finding precisely the right function).
- Disadvantage: gives up on the convexity of the training problem (the benefits outweigh the harms)

Goodfellow, Bengio, Courville 2016

14

Deep Feedforward Networks

- Also known as **feedforward neural networks** and **multilayer perceptrons** (MLPs).
- A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation of a target function $y = f^*(x)$.
- Information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y.
- There are no **feedback** connections in which outputs of the model are fed back into itself.
- When feedforward neural networks are extended to include feedback connections, they are called **recurrent neural networks**
- **CNN** is a special kind of feedforward networks

Goodfellow, Bengio, Courville 2016

15

Deep Feedforward Networks chain representation

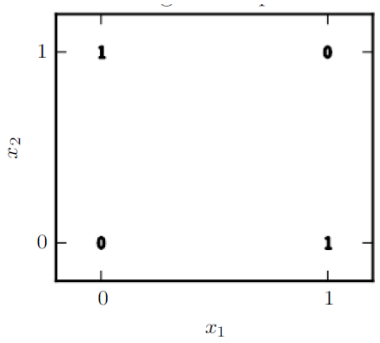
- The deep feedforward network is typically represented via a chain structure that composes different functions.
- For example, a 3-layer network can be represented as $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ where $f^{(1)}$: first **hidden** layer of the network; $f^{(2)}$: second **hidden** layer of the network; $f^{(3)}$: the final layer (**output** layer)
- The length of the chain gives the **depth** of the model.
- The dimensionality of hidden layers determines the **width** of the model

Goodfellow, Bengio, Courville 2016

16

An example: Learning XOR

- The XOR function ("exclusive or") is an operation on two binary values, $x1$ and $x2$.
- When exactly one of these binary values is equal to 1, the XOR function returns 1. Otherwise, it returns 0.
- We want our network to perform correctly on the four points $\mathbb{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$
- To simplify the math, we will use a mean squared error (MSE) loss function (generally not an appropriate cost function for modeling binary data)



$$J(\theta) = \frac{1}{4}\sum_{x\in\mathbb{X}}(f^*(x) - f(x;\theta))^2$$

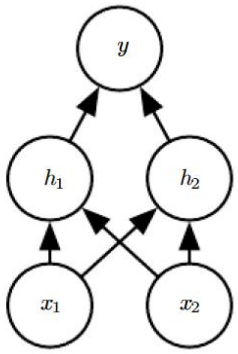
Goodfellow, Bengio, Courville 2016

17

An example: Learning XOR (cont.)

- Suppose that we choose a linear model $f(x; w, b) = x^T w + b$
- Minimizing $J(w, b)$ with respect to *w* and *b* will result in $w = 0$, $b = 0.5$. The linear model simply outputs 0.5 everywhere.
- How the neural network will solve this model?

- This feedforward network has a vector of hidden units *h* that are computed by a function $f^{(1)}(x; W, c)$
- The second layer is the output layer of the network. $y = f^{(2)}(h; w, b)$
- The complete model $f(x; W, c, w, b) = f^{(2)}(f^{(1)}(x))$

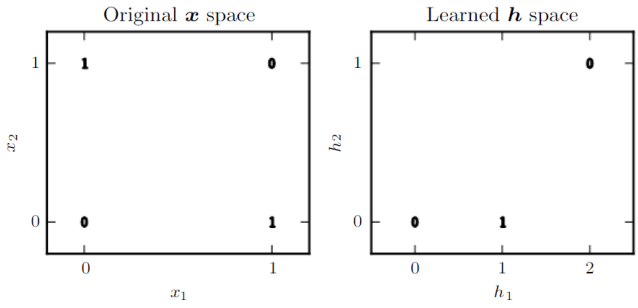


Goodfellow, Bengio, Courville 2016

18

How the neural network will solve this model?

- The network tries to learn a different feature space (h space) in which a linear model is able to represent the solution.
- What function should $f^{(1)}$ compute?
- if $f^{(1)}$ were linear, then the feedforward network as a whole would remain a linear function of its input.
- If $f^{(1)}(x) = W^{\mathsf{T}}x$ and $f^{(2)}(h) = w^{\mathsf{T}}h$ then $f(x) = w^{\mathsf{T}}W^{\mathsf{T}}x = w'^{\mathsf{T}}x$



Goodfellow, Bengio, Courville 2016

19

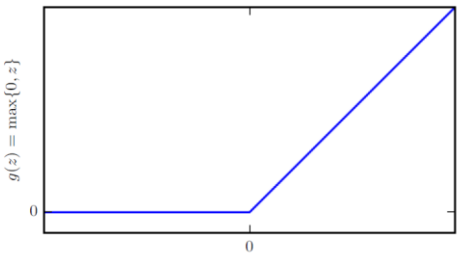
Nonlinearity

- Clearly, we must use a nonlinear function to describe the features.
- Most neural networks do so using an affine transformation (linear mapping) controlled by learned parameters, followed by a fixed, nonlinear function called an activation functio

$$h = f^{(1)}(x) = g(W^{\mathsf{T}}x + c)$$

- In modern neural networks, the default recommendation is to use the **rectified linear unit** or ReLU $g(z) = \max\{0, z\}$

$$f(x; W, c, w, b) = w^{\mathsf{T}} \max\left\{0, W^{\mathsf{T}}x + c\right\} + b$$



Goodfellow, Bengio, Courville 2016

20

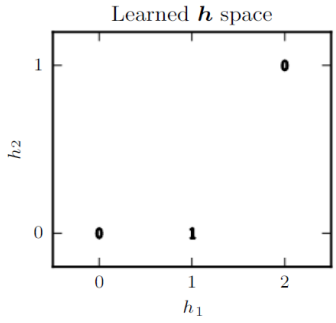
Solving the XOR example

- Using the following parameters, we can demonstrate how neural networks solve the XOR problem

$$x=\begin{bmatrix}0&0\\0&1\\1&0\\1&1\end{bmatrix} W=\begin{bmatrix}1&1\\1&1\end{bmatrix}, c=\begin{bmatrix}0\\-1\end{bmatrix}, w=\begin{bmatrix}1\\-2\end{bmatrix}, b=0$$

$$xW^{\mathsf{T}}=\begin{bmatrix}0&0\\1&1\\1&1\\2&2\end{bmatrix}, xW^{\mathsf{T}}+c=\begin{bmatrix}0&-1\\1&0\\1&0\\2&1\end{bmatrix}, h=g(W^{\mathsf{T}}x+c)=\begin{bmatrix}0&0\\1&0\\1&0\\2&1\end{bmatrix}$$

$y=hw^{\mathsf{T}}=\begin{bmatrix}0\\1\\1\\0\end{bmatrix}$ (the correct answer for every example in the batch.)



Goodfellow, Bengio, Courville 2016

21

Gradient-Based Learning

- Like many machine learning models, training a neural network uses an optimization procedure to minimize a cost function.
- the training algorithm is almost always based on using the gradient to descend the cost function in one way or another. Specific algorithms and refinements will be discussed later.
- Computing the gradient is slightly more complicated for a neural network, but can still be done efficiently and exactly.
- To apply gradient-based learning we must **choose a cost function**, and we must choose **how to represent the output of the model**.

Goodfellow, Bengio, Courville 2016

22

Cost Functions

- The cost functions for neural networks are more or less the same as those for other parametric models, such as linear models
- In most cases, parametric models defines a distribution $p(y|x;\theta)$ and we simply use the principle of maximum likelihood. This means we use the cross-entropy between the training data and the model's predictions as the cost function.
- Sometimes, we take a simpler approach, where rather than predicting a complete probability distribution over y, we merely predict some *statistic (e.g. mean or median) of y conditioned on x*. Specialized loss functions allow us to train a predictor of these estimates.
- The total cost function used to train a neural network will often combine a primary cost function with a regularization term.

Goodfellow, Bengio, Courville 2016

23

Learning Conditional Distributions with Maximum Likelihood

- Most modern neural networks are trained using maximum likelihood. This means that the cost function is simply the negative log-likelihood, equivalently described as the cross-entropy between the training data and the model distribution.
- This cost function is given by

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{data}} log p_{model}(y|x)$$

- One recurring theme throughout neural network design is that the gradient of the cost function must be large and predictable enough to serve as a good guide for the learning algorithm.
- Functions that saturate (become very flat) undermine this objective because they make the gradient become very small. In many cases this happens because the activation functions used to produce the output of the hidden units or the output units saturate.
- The negative log-likelihood helps to avoid this problem for many models. Many **output units involve an exp function that can saturate when its argument is very negative**. The log function in the negative log-likelihood cost function undoes the exp of some output units.

Goodfellow, Bengio, Courville 2016

24

Learning Conditional Statistics

- Instead of learning a full probability distribution $p(y|x;\theta)$ we often want to learn just one conditional statistic of y given x.
- For example, we may have a predictor $p(x;\theta)$ that we wish to predict the mean of y given x (think of the task of regression).
- In this case we think of learning as choosing a function rather than merely estimating a set of parameters.
- A possible cost function in this case is the mean squared error

$$f^* = \arg\min_f \mathbb{E}_{x,y\sim p_{data}} \|y - f(x)\|_2$$

Best predictor of the *mean* value of *y* for each *x*

- Another cost function is the mean absolute error

$$f^* = \arg\min_f \mathbb{E}_{x,y\sim p_{data}} \|y - f(x)\|_1$$

Best predictor of *median* value of *y* for each *x* (different cost functions give different statistics)

Goodfellow, Bengio, Courville 2016

25

Learning conditional statistics (Cont.)

- Unfortunately, mean squared error and mean absolute error often lead to poor results when used with gradient-based optimization.
- Some output units that saturate produce very small gradients when combined with these cost functions.
- This is one reason that the cross-entropy cost function is more popular than mean squared error or mean absolute error, even when it is not necessary to estimate an entier distribution $p(y \mid x)$.

Goodfellow, Bengio, Courville 2016

26

Output Units

- The choice of cost function is tightly coupled with the choice of output unit.
- Most of the time, we simply use the cross-entropy between the data distribution and the model distribution. The choice of how to represent the output then determines the form of the cross-entropy function.
- We will discuss:
  - Linear Units
  - Sigmoid Units
  - SoftMax Units
  - Other Output Types

Linear Units (for Gaussian Output Distributions)

- No nonlinearity at the output layer.
- Given the hidden features *h*, a layer of linear output units produces a vector

$$\hat{y} = W^{\top} h + b$$

- Linear output layers are often used to produce the mean of a conditional Gaussian distribution $p(y|x) = \mathcal{N}(y; \hat{y}, I)$ [think of a regression task: you are required to estimate the average value of y given x].
- Maximizing the log-likelihood is then equivalent to minimizing the mean squared error.
- Because linear units do not saturate, it is not difficult to use them for gradient based optimization algorithms.

Sigmoid Units (for Bernoulli Output Distributions)

- Many tasks require predicting the value of a binary variable *y* .
- Classification problems with two classes can be cast in this form.
- The maximum-likelihood approach is to define a Bernoulli distribution over *y* conditioned on *x*.
- A Bernoulli distribution is defined by just a single number. The neural net needs to predict only $P\ (y\ =\ 1 \mid x)$. For this number to be a valid probability, it must lie in the interval $[0, 1]$.
- If we use a linear model, for example:

$$P\ (y\ =\ 1 \mid x) = max\left\{0, min\left\{1, W^{\top} h + b\right\}\right\}$$

- we will not be able to train it very effectively with gradient descent). Any time that $W^{\top} h + b$ strayed outside the unit interval, the gradient of the output of the model with respect to its parameters would be 0 (the learning algorithm no longer has a guide for how to improve the corresponding parameters).

Goodfellow, Bengio, Courville 2016

29

Sigmoid Units (cont.)

- A sigmoid output unit is defined by

$$\hat{y}\ =\ \sigma\left(W^{\top} h + b\right)$$

- We can think of the sigmoid output unit as having two components. First, it uses a linear layer to compute $z = W^{\top} h + b$. Next, it uses the sigmoid activation function to convert *z* into a probability.
- This approach to predicting the probabilities in log-space is natural to use with maximum likelihood learning. Because the cost function used with maximum likelihood is $-\log P(y \mid x)$,
- The log in the cost function undoes the exp of the sigmoid. Without this effect, the saturation of the sigmoid could prevent gradient based learning from making good progress.

Goodfellow, Bengio, Courville 2016

30

Softmax Units (for the probability distribution over n classes)

- Any time we wish to represent a probability distribution over a discrete variable with *n* possible values, we may use the softmax function.
- Softmax function can be seen as a generalization of the sigmoid function which was used to represent a probability distribution over a binary variable.
- In the case of binary variables, we wished to produce a single number
$$\hat{y} = P(y = 1 \mid x)$$
- To generalize to the case of a discrete variable with *n* values, we now need to produce a vector $\hat{y}$, with $\hat{y}_i = P(y = i \mid x)$
- We require not only that each element of $\hat{y}_i$ be between 0 and 1, but also that the entire vector sums to 1 so that it represents a valid probability distribution.

Goodfellow, Bengio, Courville 2016

31

Softmax Units (cont.)

- First, a linear layer predicts unnormalized log probabilities:
$$z = W^{\top} h + b$$
- where $z_i = \log \tilde{P}(y = i \mid x)$. The softmax function can then exponentiate and normalize z to obtain the desired $\hat{y}$
$$softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$
- As with the logistic sigmoid, the use of the exp function works very well when training the softmax to output a target value y using maximum log-likelihood. In this case, we wish to maximize $\log P(y = i; z) = \log softmax(z)_i$. Defining thsoftmax in terms of exp is natural because the log in the log-likelihood can undo the exp of the softmax:
$$\log softmax(z)_i = z_i - \log \sum_j \exp(z_j)$$

Goodfellow, Bengio, Courville 2016

32

Softmax Units (cont.)

$$\log softmax(z)_i = z_i - \log \sum_j \exp(z_j)$$

- The first term of the equation shows that the input z_i always has a direct contribution to the cost function. Because this term cannot saturate, we know that learning can proceed, even if the contribution of z_i to the second term of becomes very small.
- When maximizing the log-likelihood, the first term encourages z_i to be pushed up, while the second term encourages all of *z* to be pushed down.

Goodfellow, Bengio, Courville 2016

33

Softmax Units (cont.)

$$\log softmax(z)_i = z_i - \log \sum_j \exp(z_j)$$

- To gain some intuition for the second term, $\log \sum_j \exp(z_j)$, observe that this term can be roughly approximated by $\max_j z_j$.
- This approximation is based on the idea that $\exp(z_k)$ is insignificant for any z_k that is noticeably less than $\max_j z_j$.
- The intuition we can gain from this approximation is that the negative log-likelihood cost function always strongly penalizes the most active incorrect prediction (incorrect if i is the correct class and z_i is small).
- If the correct answer already has the largest input to the softmax, then the $-z_i$ term and the $\log \sum_j \exp(z_j) \approx \max_j z_j \approx z_i$ will roughly cancel.
- This example will then contribute little to the overall training cost, which will be dominated by other examples that are not yet correctly classified.

Goodfellow, Bengio, Courville 2016

34