References

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
- Lectures: http://www.deeplearningbook.org/lecture_slides.html
- https://www.tensorflow.org/tutorials/word2vec

Contents

- The goal of deep learning
- Why nonlinear? How?
- General framework of feedforward nets
- Learning tasks
  - Regression
  - Classification
- Gradient Descent, mini batch, SGD
  - Backpropagation
- Regularization
  - Early stopping
  - Dropout

Representations Matter

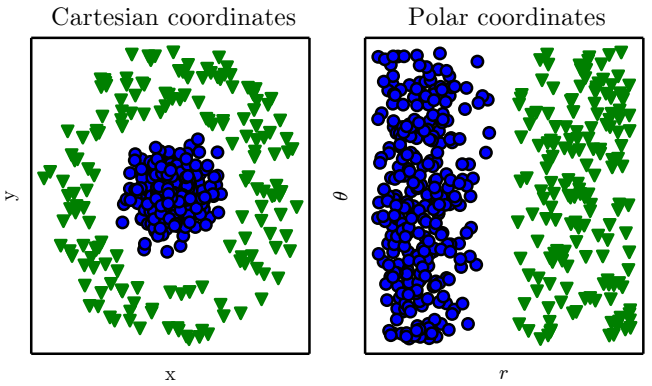


Figure 1.1

Learning Multiple Components

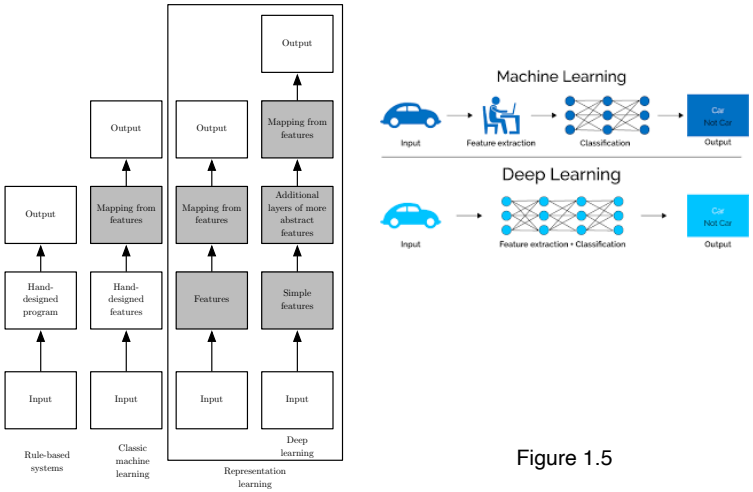


Figure 1.5

The goal of Deep Learning

Look for the (nonlinear) function

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n, f(\boldsymbol{x}, \theta) = f_\theta(\boldsymbol{x})$$

With the greatest **consistency** with the data

$$D = \{(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \rightsquigarrow \boldsymbol{y}\}$$

What is meaning of **consistency** in this context?

- Ease estimations $\boldsymbol{y} \approx \Psi(f(\boldsymbol{x}, \theta))$
- Make minimum a loss function (*inconsistency with the data)*

The goal of Deep Learning

Finally, the goal of Deep Learning is to solve an optimization equation like this

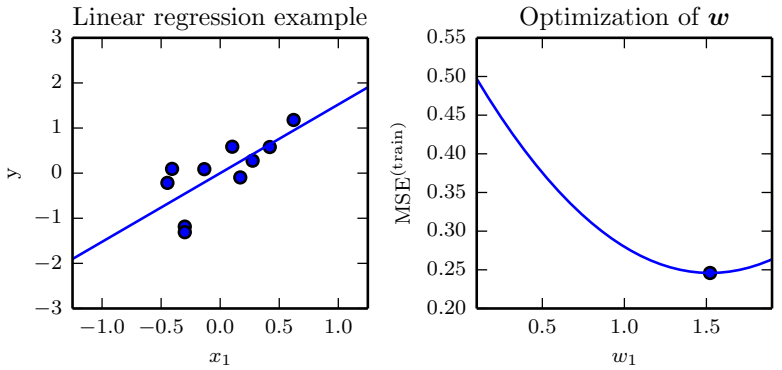
$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \frac{1}{|D|} \sum_{(x,y)\in D} \operatorname{loss}(y, f_\theta(\boldsymbol{x}))$$

Example: linear regression (n = 1) with a quadratic loss

$$f_w(\boldsymbol{x}) = \boldsymbol{w}\ \boldsymbol{x}$$

$$\operatorname{loss}(\boldsymbol{y}, f_w(\boldsymbol{x})) = \frac{1}{2}(\boldsymbol{y} - f_w(\boldsymbol{x}))^2$$

Linear Regression



$$y \approx w_1 \boldsymbol{x}_1$$

Figure 5.1

Objetivo del Deep Learning

Given

$$D = \{(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \rightsquigarrow \boldsymbol{y}\}$$ Given

Design and find a meta parameter $\boldsymbol{\theta}$

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n, f(\boldsymbol{x}, \theta) = f_\theta(\boldsymbol{x})$$ Design

Such that minimizes the mean of the loss function

$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \frac{1}{|D|} \sum_{(x,y)\in D} \operatorname{loss}(y, f_\theta(\boldsymbol{x}))$$ Program

that yields to

$$\boldsymbol{y} \approx \Psi(f(\boldsymbol{x}, \theta))$$ Design

Why nonlinear?

When the function f is nonlinear, the optimization problem is not convex

The SGD (all versions) only guarantee the convergence (find a solution) whenever the optimization problem is convex

Why nonlinear? **Because it makes up for the result**

Representations Matter

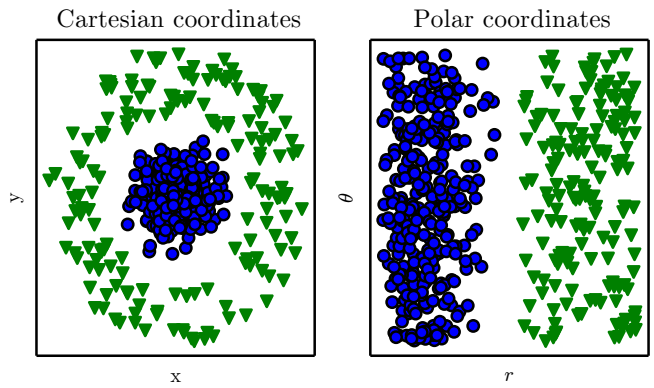
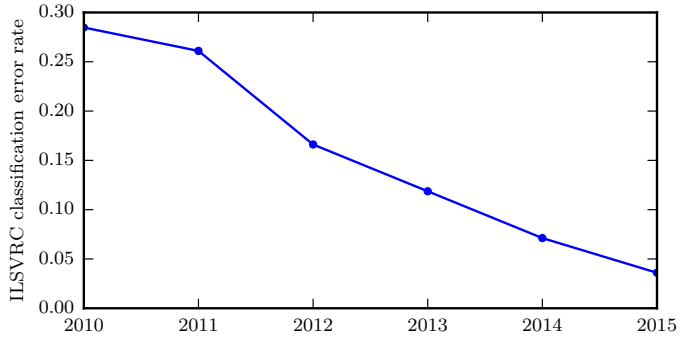


Figure 1.1

Solving Object Recognition

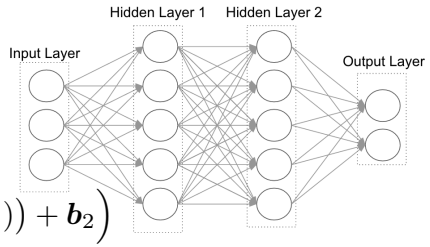
Figure 1.12



The **ImageNet** project is a large visual database designed for use in visual object recognition software research. As of 2016, over ten million URLs of images have been hand-annotated by ImageNet to indicate what objects are pictured. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes.

How to achieve nonlinearity?

$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n, f_\theta(\boldsymbol{x})$ $\quad \theta \leftarrow \underset{\theta}{\operatorname{argmin}} \frac{1}{|D|} \sum_{(x,y)\in D} \operatorname{loss}(y, f_\theta(\boldsymbol{x}))$



$$\boldsymbol{h}_1 = g(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$$
$$\boldsymbol{h}_2 = g(\boldsymbol{W}_2\boldsymbol{h}_1 + \boldsymbol{b}_2)$$
$$f(\boldsymbol{x}) = g\Big(\boldsymbol{W}_2\big(g(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)\big) + \boldsymbol{b}_2\Big)$$

$g(\boldsymbol{z}) = \max\{\mathbf{0}, \boldsymbol{z}\}$ Rectifier Linear Unit (Function) (ReLU)

How to achieve nonlinearity?

$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n, f_\theta(\boldsymbol{x})$ $\quad \theta \leftarrow \underset{\theta}{\operatorname{argmin}} \frac{1}{|D|} \sum_{(x,y)\in D} \operatorname{loss}(y, f_\theta(\boldsymbol{x}))$

Optimization requires derivatives

Gradient Descent

To solve

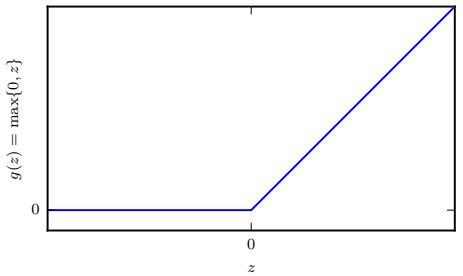
$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}}\ \mathbf{J}(\theta)$$

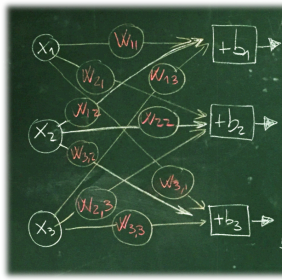
GD is the algorithm

$\theta \longleftarrow \theta_0$
repeat
$\quad \theta \longleftarrow \theta - \gamma * \frac{\partial J}{\partial \theta}$
until no more improvement can be reached
return θ

Rectifier Linear Activation (ReLU)

$$\boldsymbol{h}_1 = g(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$$





In practice one can safely disregard the non-differentiability of the hidden unit activation functions. It is very unlikely that the underlaying value was 0

Old fashion activation functions

- Instead of ReLU, functions such as sigmoid or hyperbolic tangent (similar) were used.
- Nowadays, are not used because they have almost zero slopes for most of the values. They saturate and are not useful.

Universal Approximator Theorem

- One hidden layer is enough to *represent* (not *learn*) an approximation of any function to an arbitrary degree of accuracy
- So why deeper?
  - Shallow net may need (exponentially) more width
  - Shallow net may overfit more

General Framework

General Framework

$$D = \{(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \rightsquigarrow \boldsymbol{y}\}$$

We learn a distribution

$$\Pr(\boldsymbol{y}|\boldsymbol{x}, \theta)$$

That involves the function

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n, f_\theta(\boldsymbol{x})$$

This requires maximum likelihood

$$\underset{\theta}{\operatorname{argmax}} \prod_{D} \Pr(\boldsymbol{y}|\boldsymbol{x}, \theta)$$

General Framework

This means to maximize -log

$$\underset{\theta}{\operatorname{argmin}} \; -\log\left(\prod_{D} \Pr(\boldsymbol{y}|\boldsymbol{x},\theta)\right) = \underset{\theta}{\operatorname{argmin}} \; -\sum_{D} \log \Pr(\boldsymbol{y}|\boldsymbol{x},\theta)$$

In other words, using as loss functions

Cross entropy $\quad -\log \Pr(\boldsymbol{y}|\boldsymbol{x},\theta)$

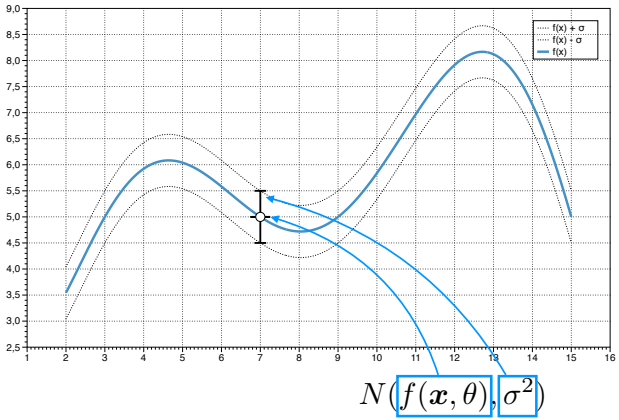
Learning Tasks

- **Feedforward** (feedforward networks, feedforward neural networks, or multilayer perceptrons (MLPs)) fully connected
- **Convolutional** networks (convolutional neural networks, CNNs), are a specialized kind of neural network for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels
- **Recurrent** neural networks (RNN) are a family of neural networks for processing sequential data. They extend feedforward neural networks to include feedback connections.

Learning Tasks

- General setting for some learning tasks:
  - (Multi)-regression
  - classification
- Regularization

Regression



σ is constant and it is estimated using the sample deviation

Regression

$$y \in \mathbb{R};\ D = \{(\boldsymbol{x}, y), ...\}$$

$$\boxed{\Pr(y|\boldsymbol{x}, \theta) = N(y; f(\boldsymbol{x}, \theta), \sigma^2)}$$

Probability distribution involving f

$$N(y; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} exp\Big(-\frac{1}{2\sigma^2}(y-\mu)^2 \Big)$$

Regression

$$y \in \mathbb{R};\ D = \{(\boldsymbol{x}, y), ...\}$$

$$\boxed{\Pr(y|\boldsymbol{x}, \theta) = N(y; f(\boldsymbol{x}, \theta), \sigma^2)}$$

Probability distribution involving f

$$N(y; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} exp\Big(-\frac{1}{2\sigma^2}(y-\mu)^2 \Big)$$

$$-\log \Pr(y|\boldsymbol{x}, \theta) = \log\sigma + \frac{1}{2}\log(2\pi) + \frac{1}{2\sigma^2}(y - f(\boldsymbol{x}, \theta))^2$$

constants

$$\arg\min_\theta -\log \prod_D \Pr(y|\boldsymbol{x}, \theta) = \arg\min_\theta \sum_D (y - f(\boldsymbol{x}, \theta))^2$$

Loss function

Binary classification

$$y \in \{0, 1\};\ D = \{(\boldsymbol{x}, y), ...\}$$

We search for a probability distribution proportional to

$$\Pr(y|\boldsymbol{x}, \theta) \propto exp(y f(\boldsymbol{x}, \theta))$$

This is NOT a probability distribution

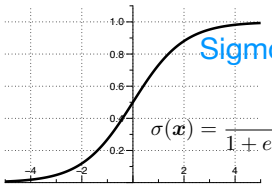
Thus,

$$\Pr(y|\boldsymbol{x}, \theta) = \frac{exp(y f(\boldsymbol{x}, \theta))}{\sum_{y'} exp(y' f(\boldsymbol{x}, \theta))} = \frac{exp(y f(\boldsymbol{x}, \theta))}{1 + exp(f(\boldsymbol{x}, \theta))}$$

Softmax

Binary classification

$$y \in \{0, 1\}$$
$$D = \{(\boldsymbol{x}, y), ...\}$$

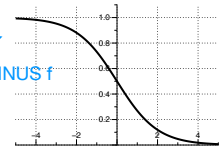


The probability increase with f value

$$\Pr(y=1|\boldsymbol{x}, \theta) = \frac{exp(f(\boldsymbol{x}, \theta))}{1 + exp(f(\boldsymbol{x}, \theta))} = \frac{1}{\frac{1}{exp(f(\boldsymbol{x}, \theta))} + 1} = \sigma(f(\boldsymbol{x}, \theta))$$

$$\Pr(y=0|\boldsymbol{x}, \theta) = \frac{1}{1 + exp(f(\boldsymbol{x}, \theta))} = \sigma(-f(\boldsymbol{x}, \theta))$$

The probability increase with MINUS f value



$$\boxed{\Pr(y|\boldsymbol{x}, \theta) = \sigma\big((2y-1) f(\boldsymbol{x}, \theta)\big)}$$

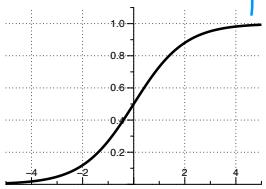
Binary classification

$$y \in \{0,1\};\ D = \{(\boldsymbol{x}, y), ...\}$$

$$\Pr(y|\boldsymbol{x},\theta) = \sigma\big((2y-1)f(\boldsymbol{x},\theta)\big)$$

This probability > 0.5 when are equal the signs of (2y-1) and f: predictions are right whenever

Right
- y=1 (=> (2y-1)=+1) and f are +
- y=0 (=> (2y-1)= -1) and f are -



Erroneous whenever

Error $(2y-1)f(\boldsymbol{x},\theta) < 0$

Binary classification

$$y \in \{0,1\};\ D = \{(\boldsymbol{x}, y), ...\}$$

$$\Pr(y|\boldsymbol{x},\theta) = \sigma\big((2y-1)f(\boldsymbol{x},\theta)\big)$$

The loss function is

$$-\log \Pr(y|\boldsymbol{x},\theta) = -\log \sigma((2y-1)f(\boldsymbol{x},\theta))$$

Given that

$$-\log(\sigma(z)) = -\log\frac{1}{1+e^{-z}} = -\log(1) + \log(1+e^{-z}) = \text{softplus}(-z)$$

0

Softplus

Thus, we have

$$-\log \Pr(y|\boldsymbol{x},\theta) = \text{softplus}((1-2y)f(\boldsymbol{x},\theta))$$

Loss function

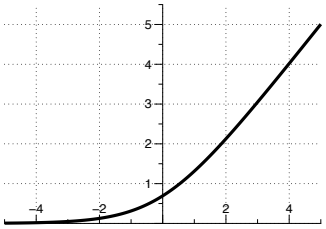
Binary classification

$$y \in \{0,1\};\ D = \{(\boldsymbol{x}, y), ...\}$$

$$\Pr(y|\boldsymbol{x},\theta) = \sigma\big((2y-1)f(\boldsymbol{x},\theta)\big)$$

The loss function is

$$\text{softplus}((1-2y)f(\boldsymbol{x},\theta)))$$



The classification error appears whenever the argument of softplus is positive

Error $(1-2y)f(\boldsymbol{x},\theta) > 0$

Error -> a lot of slope

NO error -> slope almost zero (saturates)

Multiclass classification

$$y \in \{1,2,...,n\};\ D = \{(\boldsymbol{x}, y), ...\}$$

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n, f(\boldsymbol{x},\theta) = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n)$$

$$\Pr(y = i|\boldsymbol{x},\theta) \propto exp(\boldsymbol{z}_i)$$

"Distribution" involving f

$$\Pr(y = i|\boldsymbol{x},\theta) = \frac{exp(\boldsymbol{z}_i)}{\sum_j exp(\boldsymbol{z}_j)} = \text{softmax}(\boldsymbol{z})_i$$

Softmax

The goal is to minimize

$$-\log \text{softmax}(\boldsymbol{z})_i = \log \sum_j exp(\boldsymbol{z}_j) - \boldsymbol{z}_i$$

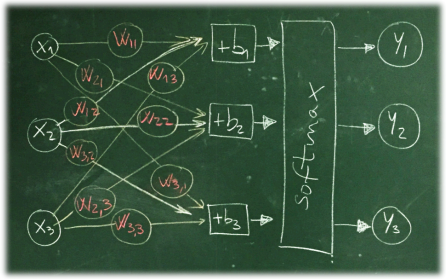
Loss function

Multiclass classification

$$y \in \{1, 2, ..., n\};\ D = \{(\boldsymbol{x}, y), ...\}$$

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n, f(\boldsymbol{x}, \theta) = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n)$$

$$\begin{aligned}\Pr(y = i|\boldsymbol{x}, \theta) \\ &= \frac{exp(\boldsymbol{z}_i)}{\sum_j exp(\boldsymbol{z}_j)} \\ &= \text{softmax}(\boldsymbol{z})_i\end{aligned}$$



Gradient

Gradient

The **gradient** of a differentiable function

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}$$

of several variables $\mathbf{x} = (x_1, \ldots, x_n)$

is the vector whose value at a point **a** is the vector whose components are the partial derivatives of f at **a**

$$\nabla_x f(a) = \left(\frac{\partial f}{\partial x_1}(a), \ldots, \frac{\partial f}{\partial x_n}(a)\right)$$

Gradient Descent

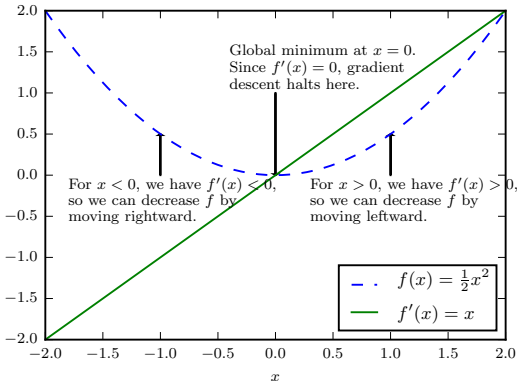


Figure 4.1

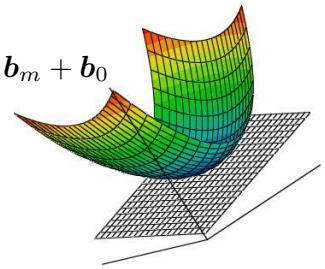
Gradient Descent

The Taylor polynomial of order 1 states that for values **f(x)** when **x** is close to **a** can be approximated using the derivative as follows

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + (\boldsymbol{x} - \boldsymbol{a})^T \nabla_x f(\boldsymbol{a})$$

That is to say,

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + \boldsymbol{x}_1 \boldsymbol{b}_1 + ... + \boldsymbol{x}_m \boldsymbol{b}_m + \boldsymbol{b}_0$$
$$(\boldsymbol{b}_1, ..., \boldsymbol{b}_m) = \nabla_x f(\boldsymbol{a})$$
$$\boldsymbol{b}_0 = -\boldsymbol{a}^T \nabla_x f(\boldsymbol{a})$$



Gradient Descent

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + (\boldsymbol{x} - \boldsymbol{a})^T \nabla_x f(\boldsymbol{a})$$

If we state that **x** is **a** modified by a small increase (γ) in the address of a unit vector **u** (|| **u** || = 1)

$$\boldsymbol{x} = \boldsymbol{a} + \gamma \boldsymbol{u}$$

We have that

$$f(\boldsymbol{a} + \gamma \boldsymbol{u}) = f(\boldsymbol{a}) + \gamma \boldsymbol{u}^T \nabla_x f(\boldsymbol{a})$$

Gradient Descent

To find a vale **x** such that **f(x) < f(a)**, we must find a direction (**u**) that

$$\begin{aligned} &\underset{u}{\operatorname{argmin}}\ \boldsymbol{u}^T \nabla_x f(\boldsymbol{a}) \\ &\quad = \underset{u}{\operatorname{argmin}}\ \|\boldsymbol{u}\| \|\nabla_x f(\boldsymbol{a})\| \cos(\boldsymbol{u}, \nabla_x f(\boldsymbol{a})) \\ &\quad = \underset{u}{\operatorname{argmin}}\ \cos(\boldsymbol{u}, \nabla_x f(\boldsymbol{a})) \end{aligned}$$

Therefore **u** must be parallel to the oposite of the gradient (**cos = -1**)

$$\boldsymbol{u} \parallel -\nabla_x f(\boldsymbol{a})$$

Gradient Descent

Therefore (if γ is *small*)

$$\begin{aligned} f(\boldsymbol{a} - \gamma \nabla_x f(\boldsymbol{a})) &= f(\boldsymbol{a}) - \gamma \nabla_x f(\boldsymbol{a})^T \nabla_x f(\boldsymbol{a}) \\ &= f(\boldsymbol{a}) - \gamma \|\nabla_x f(\boldsymbol{a})\|^2 \\ &< f(\boldsymbol{a}) \end{aligned}$$

After the assignment

$$\boldsymbol{a} \leftarrow \boldsymbol{a} - \gamma \nabla_x f(\boldsymbol{a})$$

We have a point with lower f-value than f(a), whenever γ is small

Gradient Descent

Tying up loose ends. The goal was to solve

$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \frac{1}{|D|} \sum_{(x,y)\in D} \operatorname{loss}(y, f_\theta(\boldsymbol{x}))$$

We look for the θ value that minimizes the mean of the loss function over the dataset D

$$\begin{aligned}\mathbf{J}(\theta) &= \frac{1}{|D|} \sum_{(x,y)\in D} \operatorname{loss}(\boldsymbol{y}, f_\theta(\boldsymbol{x})) \\ &= \mathbb{E}_{(x,y)\sim p(D)} \operatorname{loss}(\boldsymbol{y}, f_\theta(\boldsymbol{x}))\end{aligned}$$

Gradient Descent

To solve

$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}}\ \mathbf{J}(\theta)$$

GD is the algorithm

$\theta \longleftarrow \theta_0$
repeat
  $\theta \longleftarrow \theta - \gamma * \frac{\partial J}{\partial \theta}$
until no more improvement can be reached
return θ

Mini Batch and SGD

GD has the problem to compute

$$\frac{\partial \mathbf{J}(\theta)}{\partial \theta} = \frac{1}{|D|} \sum_{(x,y)\in D} \frac{\partial}{\partial \theta} \operatorname{loss}(\boldsymbol{y}, f_\theta(\boldsymbol{x}))$$

Thus, to make GD useful, we estimate this average with the mean of a subset D' of D

$$\frac{\partial \mathbf{J}(\theta)}{\partial \theta} \cong \frac{1}{|D'|} \sum_{(x,y)\in D'} \frac{\partial}{\partial \theta} \operatorname{loss}(\boldsymbol{y}, f_\theta(\boldsymbol{x}))$$

This is called the Stochastic Gradient Descent with mini batch. Whenever D' has only one element, we have SGD

Final remarks

Back-Propagation

- Back-propagation is “just the chain rule” of calculus

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \qquad (6.44)$$

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z, \qquad (6.46)$$

- But it’s a particular implementation of the chain rule
  - Uses dynamic programming (table filling)
  - Avoids recomputing repeated subexpressions
  - Speed vs memory tradeoff

Regularization

“Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

Regularization

- Parameter norm penalization
- Early stopping
- Dropout

Norm Penalties

Squared L2: Encourages small weights, equivalent to MAP Bayesian estimation with Gaussian prior

$$\mathbf{J}(\theta) + \frac{\nu}{2}\|\theta\|^2 = \frac{1}{|D|}\sum_{(x,y)\in D} \text{loss}(\boldsymbol{y}, f(\boldsymbol{x}, \theta)) + \frac{\nu}{2}\|\theta\|^2$$

Also known as ridge regression or Tikhonov regularization

Regression and classification

Norm Penalties

L1: Encourages sparsity, equivalent to MAP Bayesian estimation with Laplace prior. It has been used extensively as a feature selection mechanism. LASSO.

$$\mathbf{J}(\theta) + \nu\|\theta\|_1 = \frac{1}{|D|}\sum_{(x,y)\in D} \text{loss}(\boldsymbol{y}, f(\boldsymbol{x},\theta)) + \nu\|\theta\|_1$$

(sub-gradient)

$$\nabla_\theta \mathbf{J}(\theta) + \nu \text{ sign}(\theta)$$

Early stopping

Early stopping: terminate when validation set performance is better

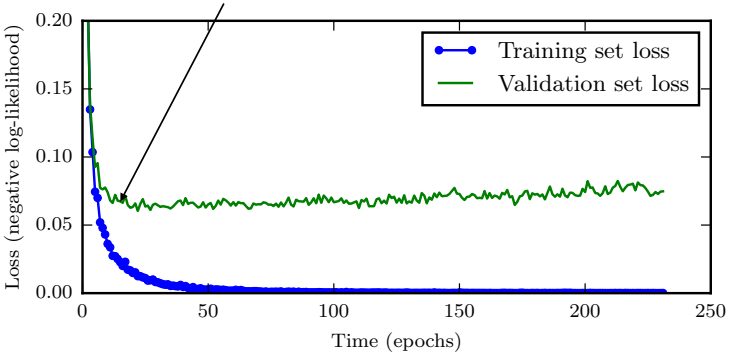


Figure 7.3

Early stopping

- It is probably the most commonly used form of regularization in deep learning.
- Requires a validation set
- After initial training with early stoping has completed, a second training with all data (train + validation) is performed with new starting values

Dropout

- Dropout train an *ensemble: all sub-nets* which can be formed by removing nodes that are not output
- There is an exponential number of subnets. Then dropout uses a sampling
- Typically, inputs are included with probability 0.8 and hidden nodes with 0.5

Dropout

Figure 7.6