11-695: AI Engineering
ML Reviews II

LTI/SCS

Spring 2020

Outline

Linear Regression

- Linear Regression model:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \epsilon$$

where the noise

$$\epsilon \sim \mathbf{N}(\mathbf{0}, \sigma^2)$$

- MLE estimator

$$\hat{\mathbf{w}}_{MLE} = \underset{\mathbf{w}}{\text{argmin}} \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y^{(i)} - \mathbf{w}^T x^{(i)})^2$$

- Closed-form solution for MLE (*a.k.a normal equations*):

$$\hat{\mathbf{w}}_{MLE} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Linear Regression

- With prior $\mathbf{w} \sim \mathbf{N}(\mathbf{0}, \lambda^{-1}\mathbf{I}) = \frac{1}{(2\pi)^{D/2}} \exp(-\frac{\lambda}{2}\mathbf{w}^T\mathbf{w})$ then MAP estimator

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y^{(i)} - \mathbf{w}^T x^{(i)})^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Closed-form solution for MAP:

$$\hat{\mathbf{w}}_{MAP} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

- Problems?

Linear Regression

- With prior $\mathbf{w} \sim \mathbf{N}(\mathbf{0}, \lambda^{-1}\mathbf{I}) = \frac{1}{(2\pi)^{D/2}} \exp(-\frac{\lambda}{2}\mathbf{w}^T\mathbf{w})$ then MAP estimator

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y^{(i)} - \mathbf{w}^T x^{(i)})^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Closed-form solution for MAP:

$$\hat{\mathbf{w}}_{MAP} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

- Problems? (Moore-Penrose) pseudo-inverse $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}$ takes $O(n^3)$

Logistic Regression

- MLE estimator

$$\hat{\mathbf{w}}_{MLE} = \underset{\mathbf{w}}{\text{argmin}} \sum_{i=1}^{n} \log\left(1 + \exp(-y^{(i)}\mathbf{w}^T x^{(i)}\right)$$

- With the same prior for $\mathbf{w}$, MAP estimator:

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\text{argmin}} \sum_{i=1}^{n} \log\left(1 + \exp(-y^{(i)}\mathbf{w}^T x^{(i)}\right) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Closed-form solution:

Logistic Regression

- MLE estimator

$$\hat{\mathbf{w}}_{MLE} = \underset{\mathbf{w}}{\text{argmin}} \sum_{i=1}^{n} \log\left(1 + \exp(-y^{(i)}\mathbf{w}^T x^{(i)}\right)$$

- With the same prior for $\mathbf{w}$, MAP estimator:

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\text{argmin}} \sum_{i=1}^{n} \log\left(1 + \exp(-y^{(i)}\mathbf{w}^T x^{(i)}\right) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Closed-form solution: non-existent

What have we got here

- In most real-world problems, data is big

What have we got here

- In most real-world problems, data is big
- In which case, finding the exact solution is *intractable*

What have we got here

- In most real-world problems, data is big
- In which case, finding the exact solution is *intractable*
- Workaround: approximate solutions

Outline

Gradient Descent (Cauchy) [1]

Carnegie Mellon

- Unconstrained optimization problem with smooth function f:

$$\min_x f(x)$$

[1] https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf

Gradient Descent (Cauchy) [1]

Carnegie Mellon

- Unconstrained optimization problem with smooth function f:

$$\min_x f(x)$$

- Idea: at some point x_t, approximate $f(x_t)$ with a parabola:

$$G_t(x, x_t) = f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2\eta}(x - x_t)^T (x - x_t)$$

$$x_{t+1} = \operatorname*{argmin}_x G_t(x) = x_t - \eta \nabla f(x_t) \tag{1}$$

[1] https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf

Gradient Descent (Cauchy) [1]

Carnegie Mellon

- Unconstrained optimization problem with smooth function f:

$$\min_x f(x)$$

- Idea: at some point x_t, approximate $f(x_t)$ with a parabola:

$$G_t(x, x_t) = f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2\eta}(x - x_t)^T (x - x_t)$$

$$x_{t+1} = \operatorname*{argmin}_x G_t(x) = x_t - \eta \nabla f(x_t) \tag{1}$$

- Algorithm: initially guess x_0 and repeat (1) and

[1] https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf

Gradient Descent (Cauchy) [1]

Carnegie Mellon

- Unconstrained optimization problem with smooth function f:

$$\min_x f(x)$$

- Idea: at some point x_t, approximate $f(x_t)$ with a parabola:

$$G_t(x, x_t) = f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2\eta}(x - x_t)^T (x - x_t)$$

$$x_{t+1} = \operatorname*{argmin}_x G_t(x) = x_t - \eta \nabla f(x_t) \tag{1}$$

- Algorithm: initially guess x_0 and repeat (1) and stop *somewhere*.

[1] https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf

Gradient Descent (Cauchy)

- Iterative algorithm
- Is sensitive to the chosen step size

Gradient Descent (Cauchy)

- Iterative algorithm
- Is sensitive to the chosen step size
- Pros: simple, cheap, fast for strongly convex functions
- Cons: only work for smooth functions, slow convergence rate

Newton's Method

- GD is the first-order method, and slow to converge in most cases
- Idea: use a better parabola for approximation

$$G_t(x, x_t) = f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2}(x - x_t)^T \mathrm{H}_t (x - x_t)$$

$$x_{t+1} = x_t - \mathrm{H}_t^{-1} \nabla f(x_t) \tag{2}$$

where $\mathrm{H}_t = \nabla^2 f(x_t)$ is the Hessian matrix.

Newton's Method

Carnegie Mellon

- GD is the first-order method, and slow to converge in most cases
- Idea: use a better parabola for approximation

$$G_t(x, x_t) = f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2}(x - x_t)^T \mathrm{H}_t (x - x_t)$$

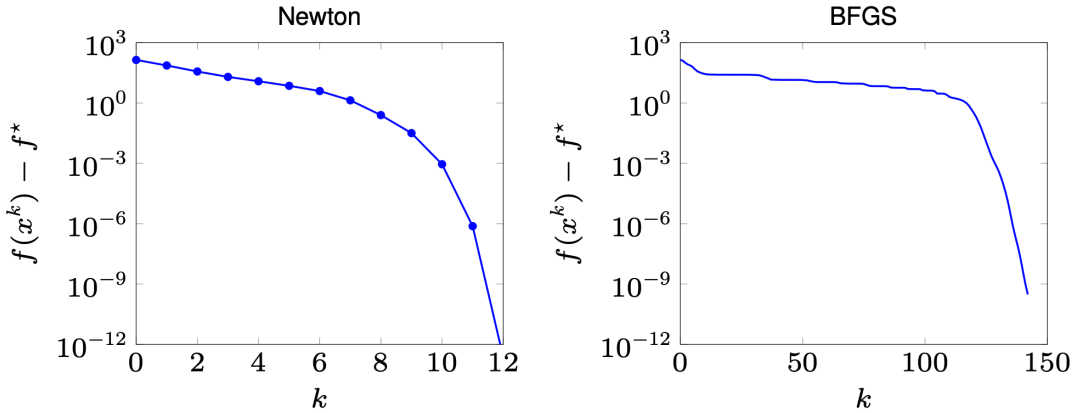
$$x_{t+1} = x_t - \mathrm{H}_t^{-1} \nabla f(x_t) \tag{2}$$

  where $\mathrm{H}_t = \nabla^2 f(x_t)$ is the Hessian matrix.

- Hence, Newton's is the second-order method.
- Variance with a step size: $x_{t+1} = x_t - \eta \mathrm{H}_t^{-1} \nabla f(x_t)$

Quasi-Newton's Method

Carnegie Mellon

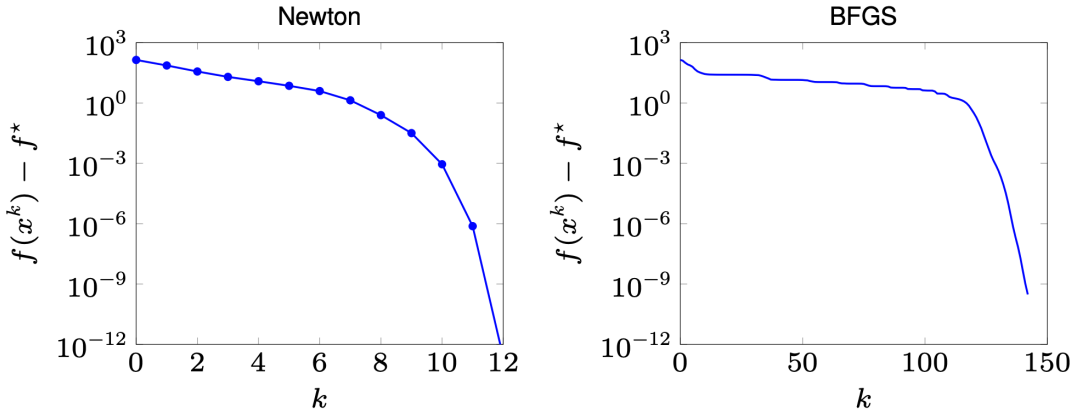


- Newton's method
  - Pros: has quadratic convergence rate vs. linear in GD

Image credit: Ryan Tibshirani

Quasi-Newton's Method

Carnegie Mellon

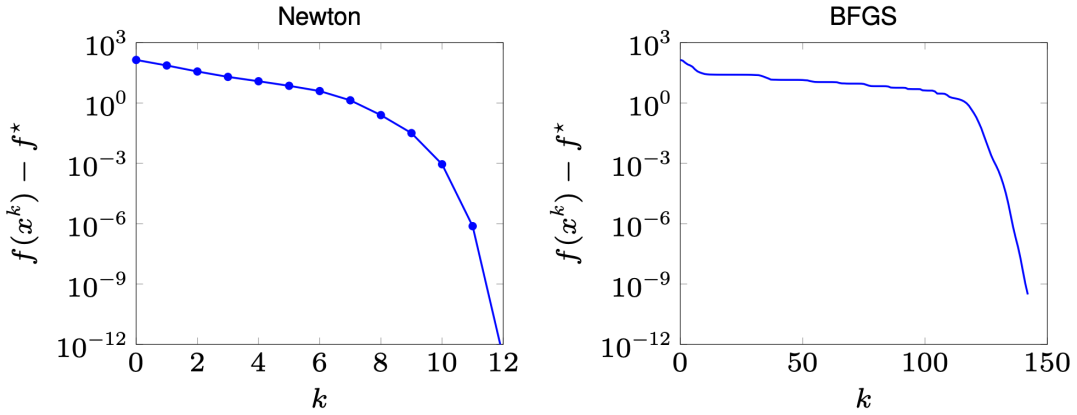


- Newton's method
  - Pros: has quadratic convergence rate vs. linear in GD
  - Cons: very expensive for Hessian calculation and its inverse: $O(n^3)$

Image credit: Ryan Tibshirani

Quasi-Newton's Method

Carnegie Mellon

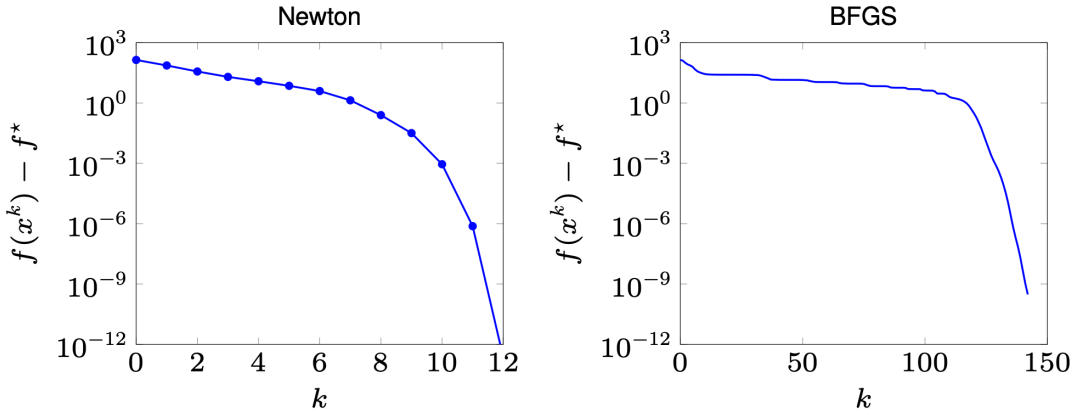


- Newton's method
  - Pros: has quadratic convergence rate vs. linear in GD
  - Cons: very expensive for Hessian calculation and its inverse: $O(n^3)$
- Idea of Quasi-Newton's (sometimes called *secant*) method: approximate Hessian H with $\tilde{\text{H}}$ and thus gain $O(n^2)$

Image credit: Ryan Tibshirani

Quasi-Newton's Method

Carnegie Mellon

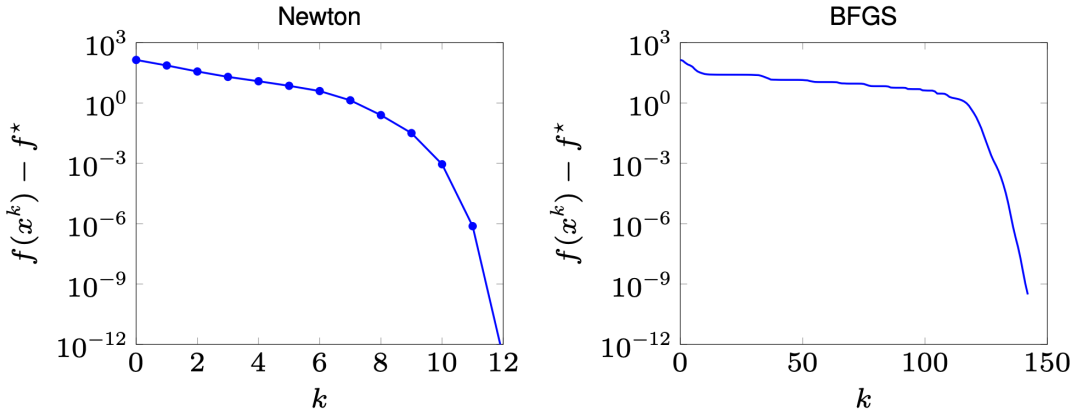


- Newton's method
  - Pros: has quadratic convergence rate vs. linear in GD
  - Cons: very expensive for Hessian calculation and its inverse: $O(n^3)$
- Idea of Quasi-Newton's (sometimes called *secant*) method: approximate Hessian H with $\tilde{\text{H}}$ and thus gain $O(n^2)$
- Skip the details, but it has super linear convergence rate

Image credit: Ryan Tibshirani

Quasi-Newton's Method

Carnegie Mellon



- Newton's method
  - Pros: has quadratic convergence rate vs. linear in GD
  - Cons: very expensive for Hessian calculation and its inverse: $O(n^3)$
- Idea of Quasi-Newton's (sometimes called *secant*) method: approximate Hessian H with $\tilde{\text{H}}$ and thus gain $O(n^2)$
- Skip the details, but it has super linear convergence rate
- Although cheaper than Newton's, it is still complicated and not efficient as GD

Image credit: Ryan Tibshirani

Gradient Descent: How to apply

Carnegie Mellon

- Data: $\mathbb{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$
- Each step, we calculate the gradient of the loss function:

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \sum_{i=1}^{n} l(\mathbf{f}(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)}) = \sum_{i=1}^{n} \nabla_\theta l(\mathbf{f}(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})$$

Gradient Descent: How to apply

Carnegie Mellon

- Data: $\mathbb{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$
- Each step, we calculate the gradient of the loss function:

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \sum_{i=1}^{n} l(\mathbf{f}(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)}) = \sum_{i=1}^{n} \nabla_\theta l(\mathbf{f}(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})$$

- Problem?

Gradient Descent: How to apply

Carnegie Mellon

- Data: $\mathbb{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$
- Each step, we calculate the gradient of the loss function:

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \sum_{i=1}^{n} l(\mathbf{f}(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)}) = \sum_{i=1}^{n} \nabla_\theta \, l(\mathbf{f}(\mathbf{x}^{(i)}, \theta), \mathbf{y}^{(i)})$$

- Problem?
  - ImageNet: $n = 1,200,000$
  - English-German translation: $n = 4,500,000$
  - Google 1-billion-words data: $n = 1,000,000,000$
  - Human Genes: $n =$???

Stochastic Gradient Descent (SGD)

Carnegie Mellon

- Each step, randomly draw a sample $\mathbf{x}^{(k)} \in \mathbf{X}$ and approximate

$$\nabla_\theta \mathcal{L}(\theta) \approx \nabla_\theta l(\mathbf{f}(\mathbf{x}^{(k)}, \theta), \mathbf{y}^{(k)})$$

- Why?

Stochastic Gradient Descent (SGD)

Carnegie Mellon

- Each step, randomly draw a sample $\mathbf{x}^{(k)} \in \mathbf{X}$ and approximate

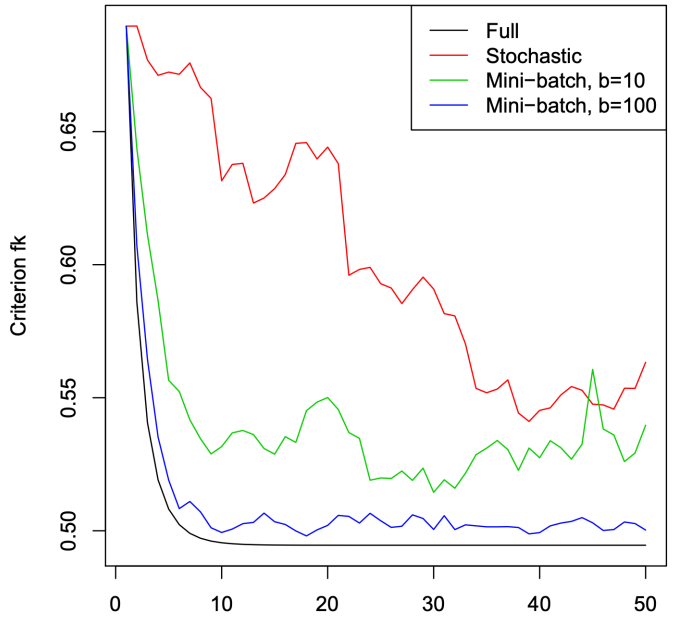
$$\nabla_\theta \mathcal{L}(\theta) \approx \nabla_\theta l(\mathbf{f}(\mathbf{x}^{(k)}, \theta), \mathbf{y}^{(k)})$$

- Why? Unbiased estimate of full gradient:

$$\mathbb{E}[\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(k)}, \theta), \mathbf{y}^{(k)})] = \nabla_\theta \mathcal{L}(\theta),$$

and it's much doable for large-scale datasets.

Stochastic Gradient Descent (SGD)

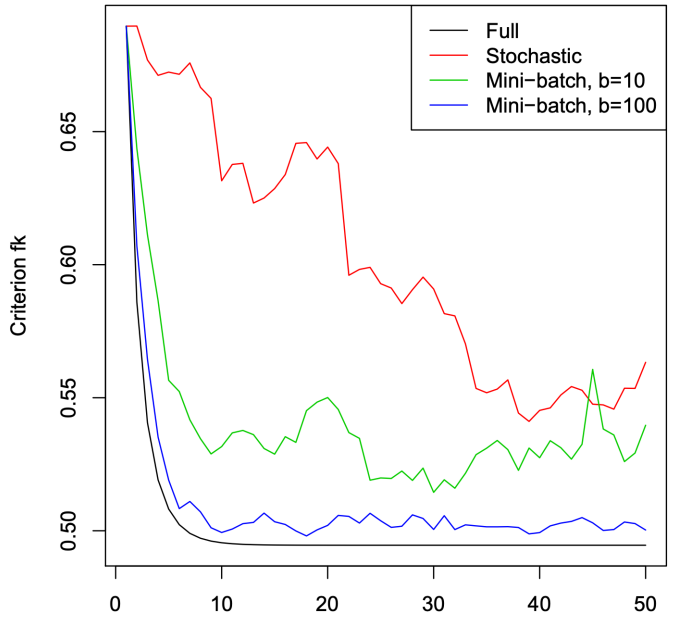


- In practice, we often use a *mini-batch* version of SGD, in which we choose a subset of $b << n$ samples. Why?

Image credit: Ryan Tibshirani

Stochastic Gradient Descent (SGD)

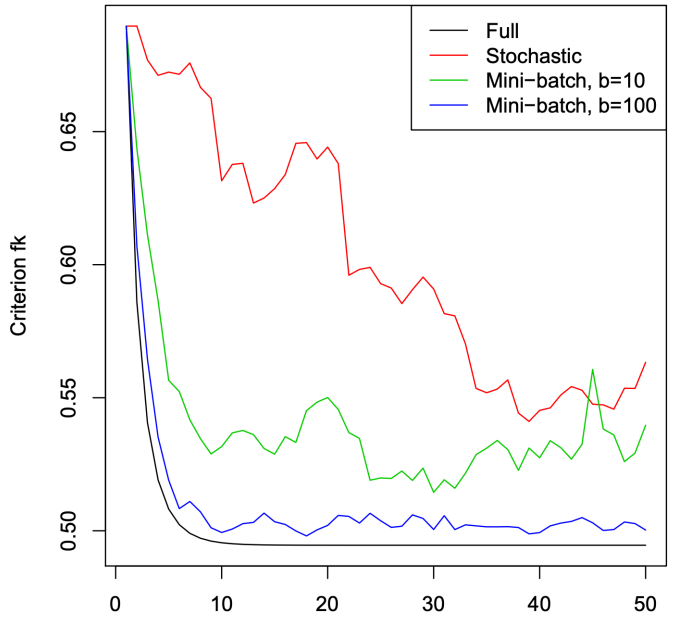
Carnegie Mellon



- In practice, we often use a *mini-batch* version of SGD, in which we choose a subset of $b << n$ samples. Why?
- The most important method for neural networks and large-scale data

Image credit: Ryan Tibshirani

Stochastic Gradient Descent (SGD)



- In practice, we often use a *mini-batch* version of SGD, in which we choose a subset of $b << n$ samples. Why?
- The most important method for neural networks and large-scale data
- Many variances of SGD, which come later in the course.

Image credit: Ryan Tibshirani

Choice for Large-Scale Datasets

Carnegie Mellon

Table 2. Asymptotic equivalents for various optimization algorithms: gradient descent (GD, eq. 2), second order gradient descent (2GD, eq. 3), stochastic gradient descent (SGD, eq. 4), and second order stochastic gradient descent (2SGD, eq. 5). Although they are the worst optimization algorithms, SGD and 2SGD achieve the fastest convergence speed on the expected risk. They differ only by constant factors not shown in this table, such as condition numbers and weight vector dimension.

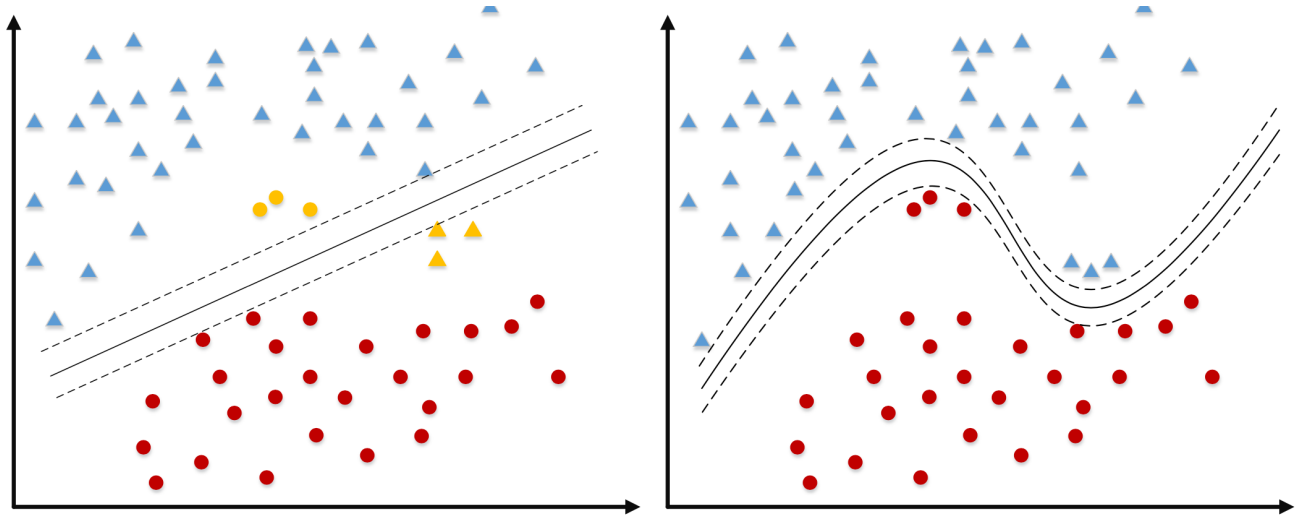
	GD	**2GD**	**SGD**	**2SGD**
Time per iteration :	n	n	1	1
Iterations to accuracy ρ :	$\log \frac{1}{\rho}$	$\log\log \frac{1}{\rho}$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
Time to accuracy ρ :	$n \log \frac{1}{\rho}$	$n \log\log \frac{1}{\rho}$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
Time to excess error ε :	$\frac{1}{\varepsilon^{1/\alpha}} \log^2 \frac{1}{\varepsilon}$	$\frac{1}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon} \log\log \frac{1}{\varepsilon}$	$\frac{1}{\varepsilon}$	$\frac{1}{\varepsilon}$

- Stochastic algorithms are faster
- First-order methods are clearly cheaper

Image credit: Léon Bottou

Outline

How a Good Model should be?

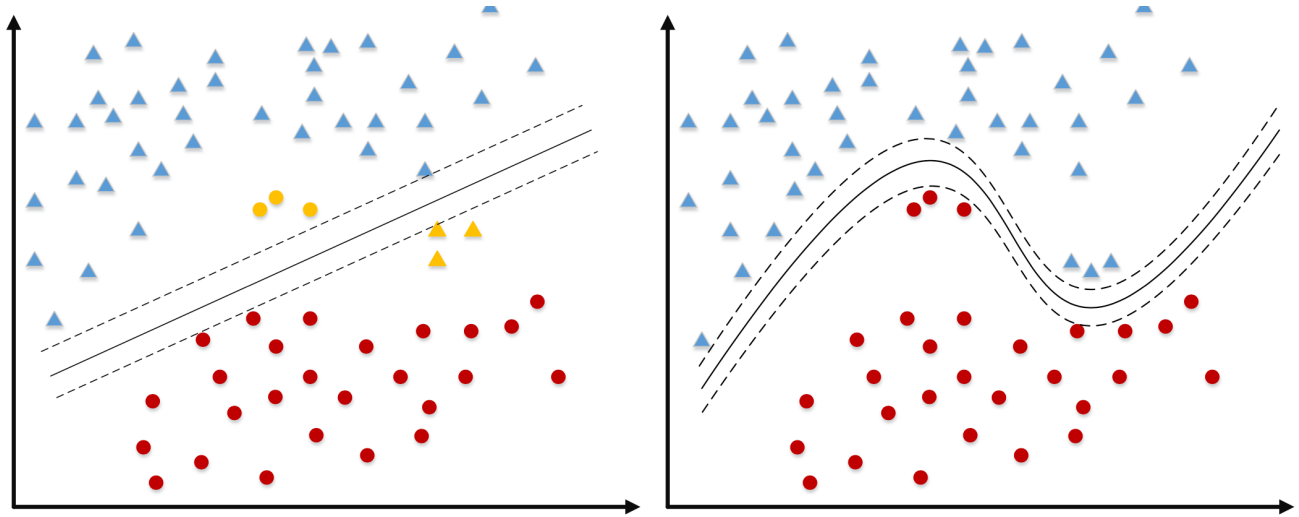


- Fit well with current data (train, validation, test).

Image credit: Kamran Kowsari *et al.*

How a Good Model should be?

Carnegie Mellon

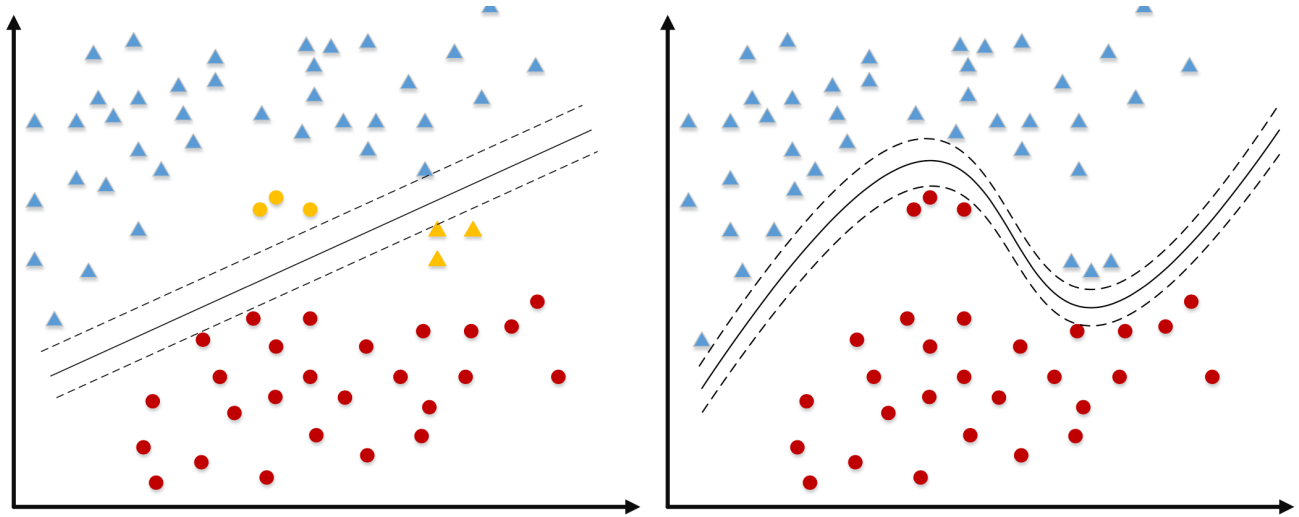


- Fit well with current data (train, validation, test).
  - Be able to learn well the relationship between **X** and **y**

Image credit: Kamran Kowsari *et al.*

How a Good Model should be?

Carnegie Mellon

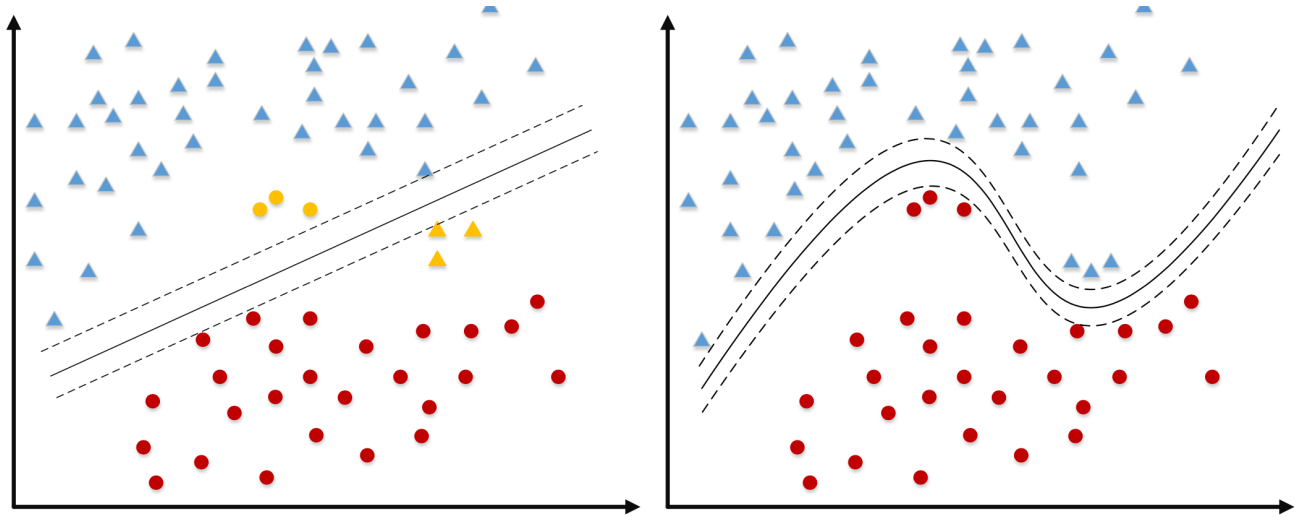


- Fit well with current data (train, validation, test).
  - Be able to learn well the relationship between $\mathbf{X}$ and $\mathbf{y}$
  - Linear or Nonlinear?

Image credit: Kamran Kowsari *et al.*

How a Good Model should be?

Carnegie Mellon

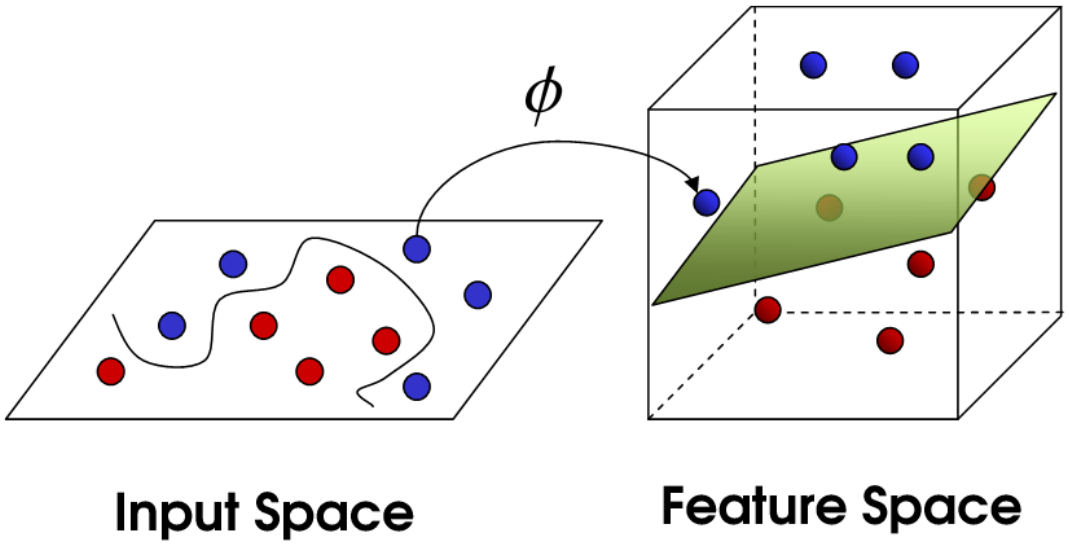


- Fit well with current data (train, validation, test).
  - Be able to learn well the relationship between $\mathbf{X}$ and $\mathbf{y}$
  - Linear or Nonlinear?
- *Generalize* well with data in the *similar* domain

Image credit: Kamran Kowsari *et al.*

Nonlinear choice: Basis Functions

Carnegie Mellon



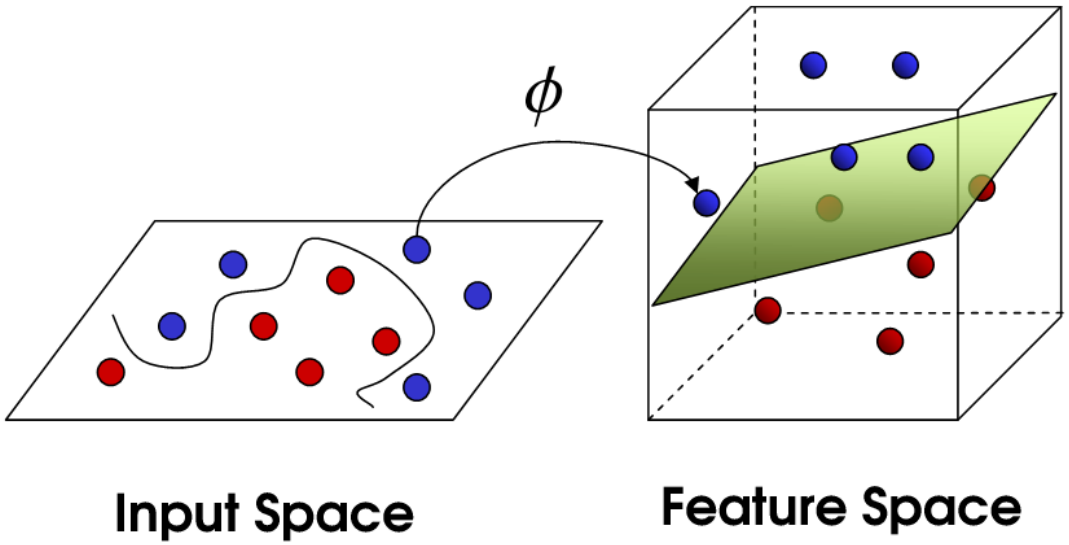
- Apply a *feature mapping* on input data with a basis function:

$$\mathbf{x} \Rightarrow \Phi(\mathbf{x})$$

  - Non linear of input, but (still) linear of params
  - Model is unchanged

Image credit: Catarina Moreira

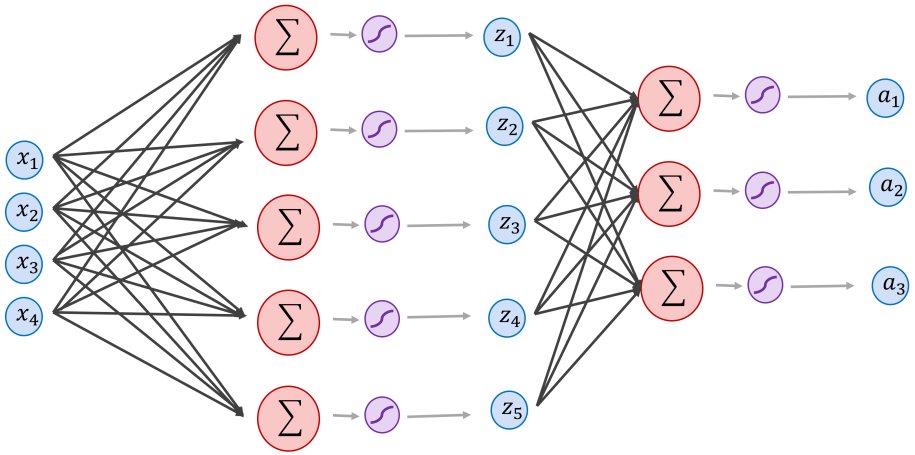
Nonlinear choice: Basis Functions



- Cons
  - Handcrafted features: expert knowledge
  - Curse of Dimensionality

Image credit: Catarina Moreira

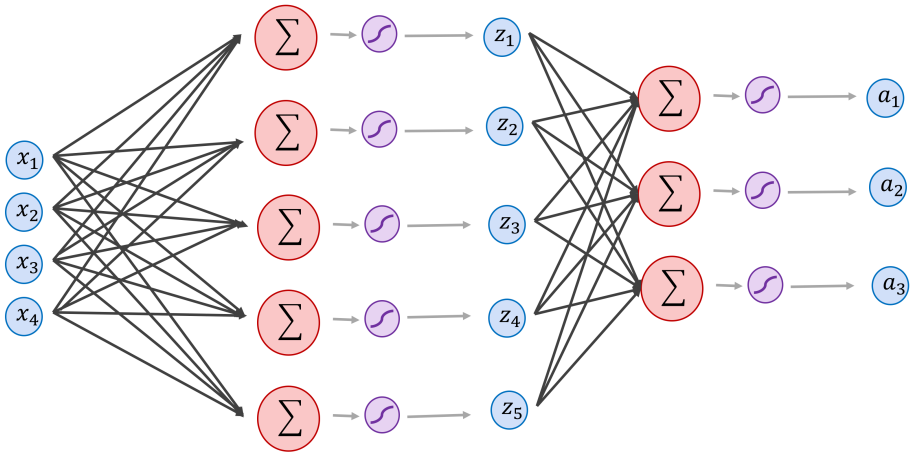
Nonlinear choice: *Adaptive* Basis Func



- A nonlinear function that is
  - Agnostic to input dimension
  - Able to learn an efficient feature mapping space

Image credit: Vicente Ordóñez Román

Nonlinear choice: *Adaptive* Basis Func

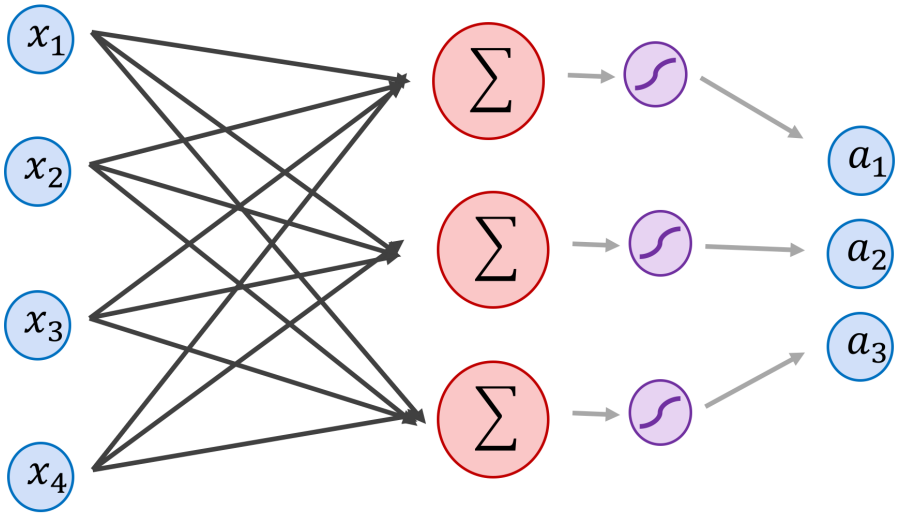


- A nonlinear function that is
  - Agnostic to input dimension
  - Able to learn an efficient feature mapping space
- Such design is found in neural networks: sigmoid, tanh, ReLU, ...

Image credit: Vicente Ordóñez Román

Outline

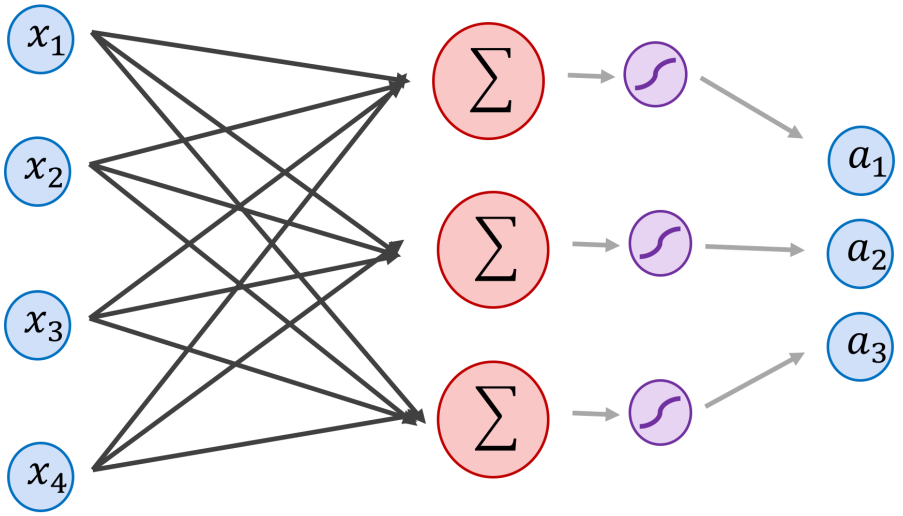
Construction of NNs



- Supervised learning input $(\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{n \times d} \times \mathbb{R}^n$
- Two basic operations
  - Linear: $o_i = W_i^T a_{i-1} + b_i$
  - Nonlinear (by activation functions) : $a_i = \phi(o_i)$

Image credit: Vicente Ordóñez Román

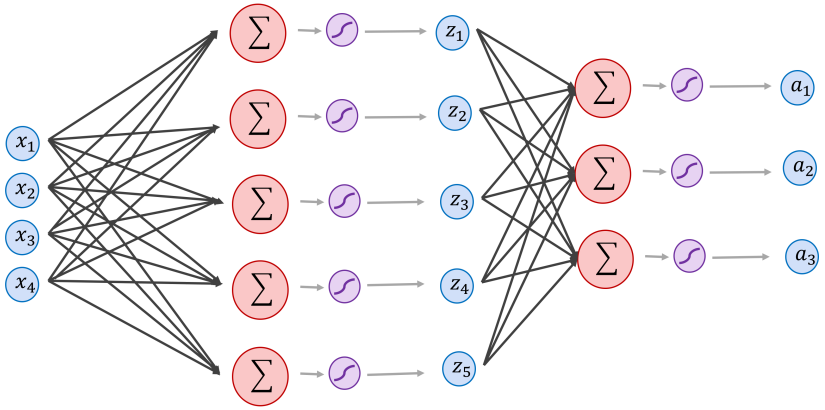
Construction of NNs



- Supervised learning input $(\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{n \times d} \times \mathbb{R}^n$
- Two basic operations
  - Linear: $o_i = W_i^T a_{i-1} + b_i$
  - Nonlinear (by activation functions) : $a_i = \phi(o_i)$
- Usually comprise of a sequence of such pair of basic operations
  - To improve capacity,
  - Obviously, with a cost

Image credit: Vicente Ordóñez Román

Construction of NNs



- Mathematically (note the dimensions):

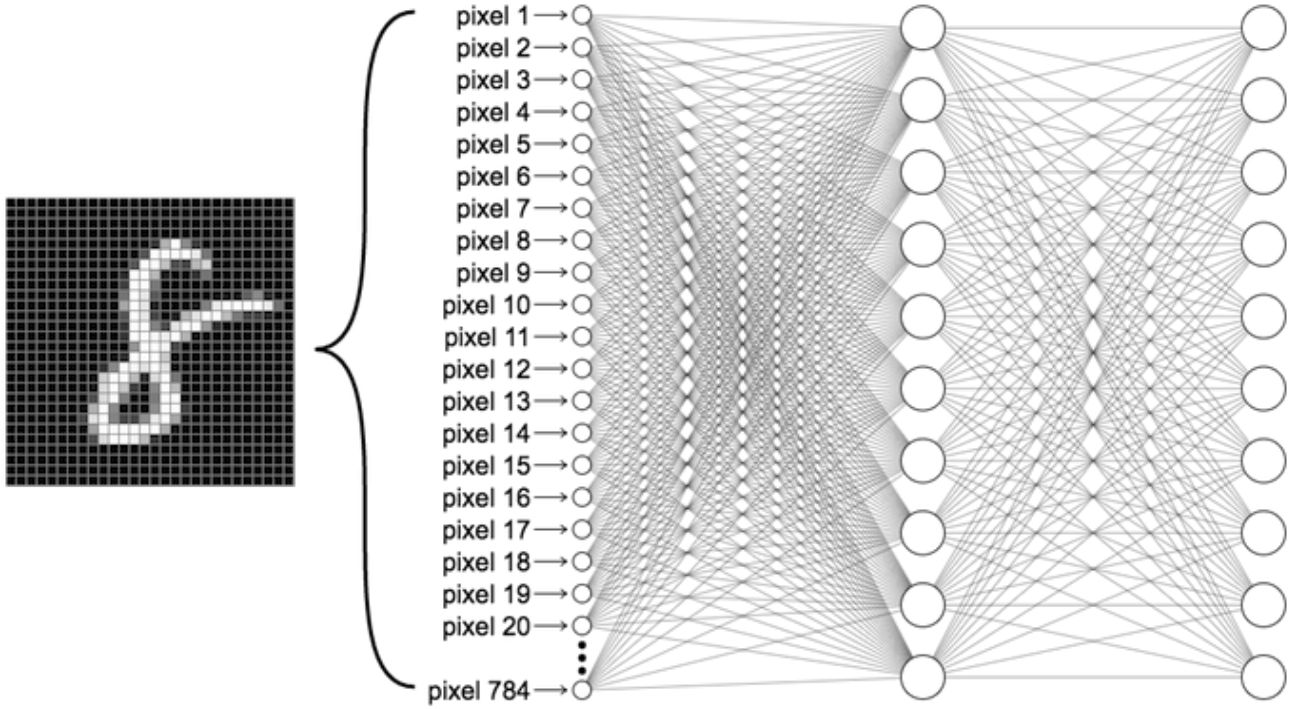
$$\hat{\mathbf{y}} = (\phi_n \circ \mathbf{f}_{W_n} \circ \phi_{n-1} \circ \mathbf{f}_{W_{n-1}} \ldots \phi_1 \circ \mathbf{f}_{W_1})(\mathbf{X})$$

$$\hat{\mathbf{W}} = \{W_n, W_{n-1}, \ldots, W_1\} = \underset{\mathbf{W}}{\operatorname{argmin}}\, \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$$

- Visually: A sequence of hidden layers
  - Each has the two basic operations above,
  - Except?

Image credit: Vicente Ordóñez Román

Example: MNIST



- Demo 1: `https://ml4a.github.io/demos/f_mnist_weights/`
- Demo 2: `http://playground.tensorflow.org/`

Image credit: Gene Kogan