CS60010: Deep Learning

Sudeshna Sarkar

Spring 2019

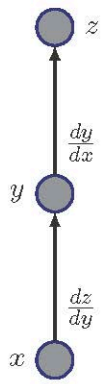
24 Jan 2019

Part1: Backpropagation Cont

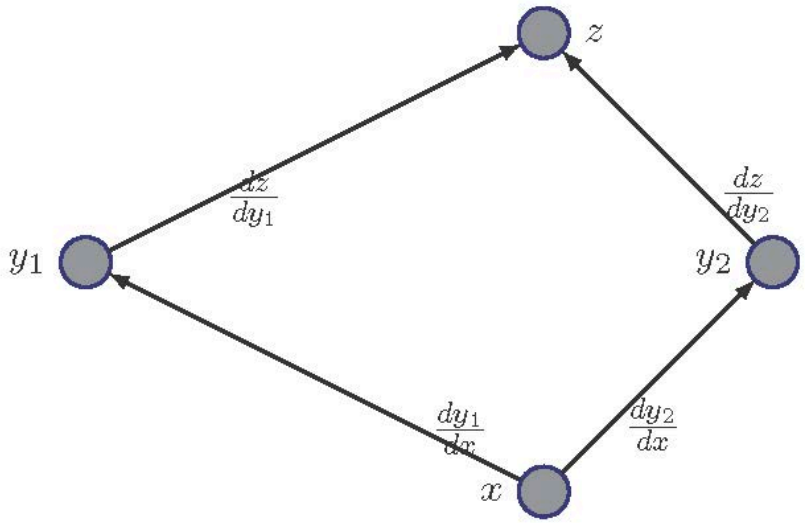
Backprop: Chain Rule

- Backpropagation computes the chain rule, in a manner that is highly efficient
- Let $f, g : \mathbb{R} \to \mathbb{R}$
- Suppose $y = g(x)$ and $z = f(y) = f(g(x))$
- Chain rule:

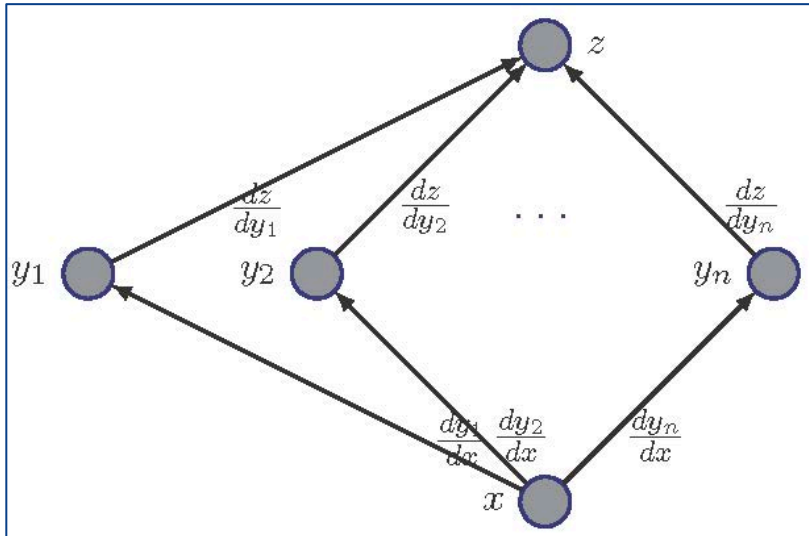
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



Chain rule: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$



Multiple Paths: $\frac{dz}{dx} = \frac{dz}{dy_1}\frac{dy_1}{dx} + \frac{dz}{dy_2}\frac{dy_2}{dx}$



Multiple Paths: $\frac{dz}{dx} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx}$

Chain Rule

- Consider $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$
- Let $g : \mathbb{R}^m \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$
- Suppose $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

- In vector notation:

$$\begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{pmatrix} = \begin{pmatrix} \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_1} \\ \vdots \\ \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_m} \end{pmatrix} = \nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^T \nabla_{\mathbf{y}} z$$

Chain Rule

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z$$

- $\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)$ is the $n \times m$ Jacobian matrix of g
- **Gradient** of $\mathbf{x}$ is a multiplication of a Jacobian matrix $\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)$ with a vector i.e. the gradient $\nabla_{\mathbf{y}} z$
- Backpropagation consists of applying such Jacobian-gradient products to each operation in the computational graph
- In general this need not only apply to vectors, but can apply to tensors w.l.o.g

Chain Rule

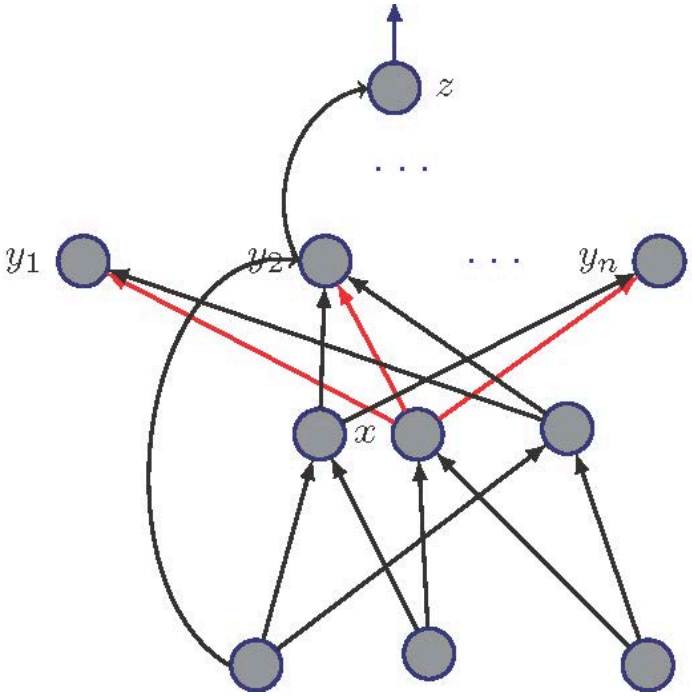
- We can ofcourse also write this in terms of tensors
- Let the gradient of z with respect to a tensor $\mathbf{X}$ be $\nabla_{\mathbf{X}} z$
- If $\mathbf{Y} = g(\mathbf{X})$ and $z = f(\mathbf{Y})$, then:

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}$$

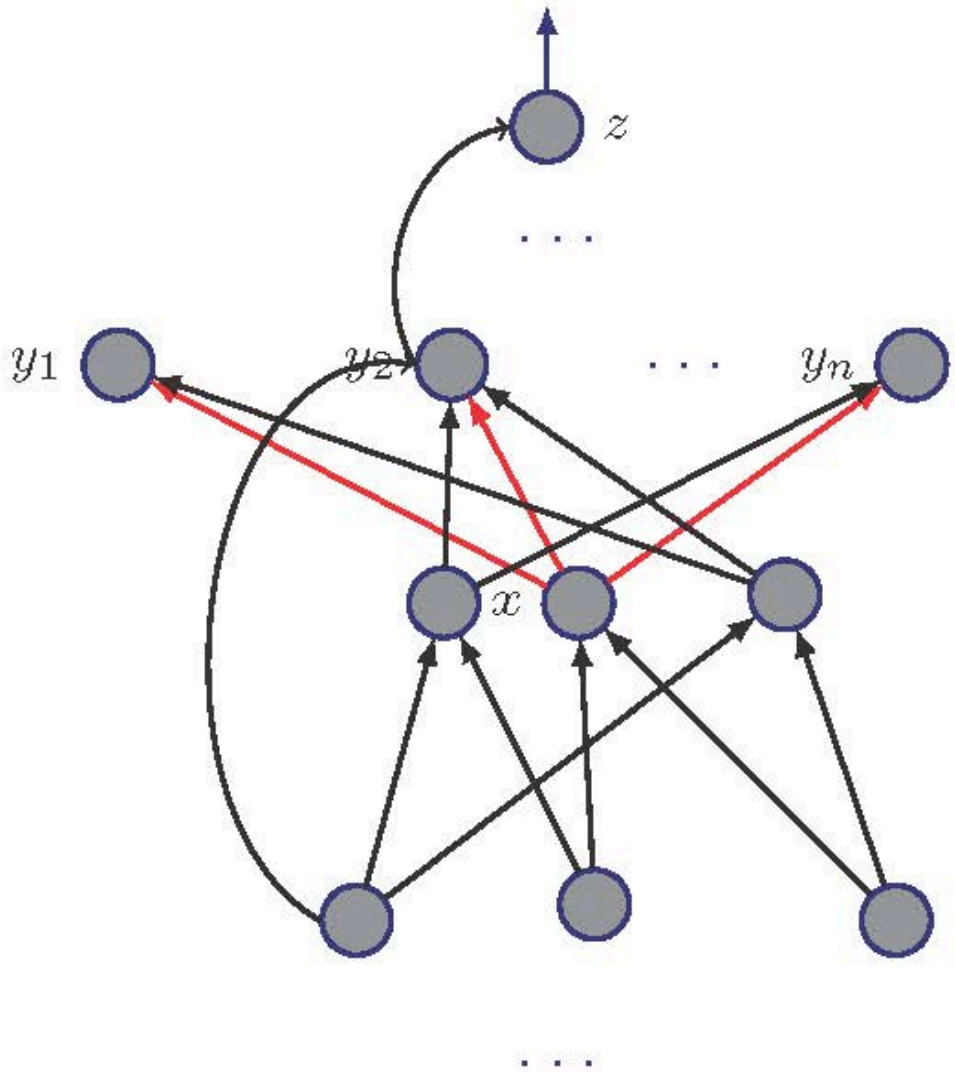
Recursive application in a computation graph

- Writing an algebraic expression for the gradient of a scalar with respect to *any* node in the computational graph that *produced* that scalar is straightforward using the chain-rule
- Let for some node x the successors be: $\{y_1, y_2, \ldots y_n\}$
- Node: Computation result
- Edge: Computation dependency

$$\frac{dz}{dx} = \sum_{i=1}^{n} \frac{dz}{dy_i} \frac{dy_i}{dx}$$



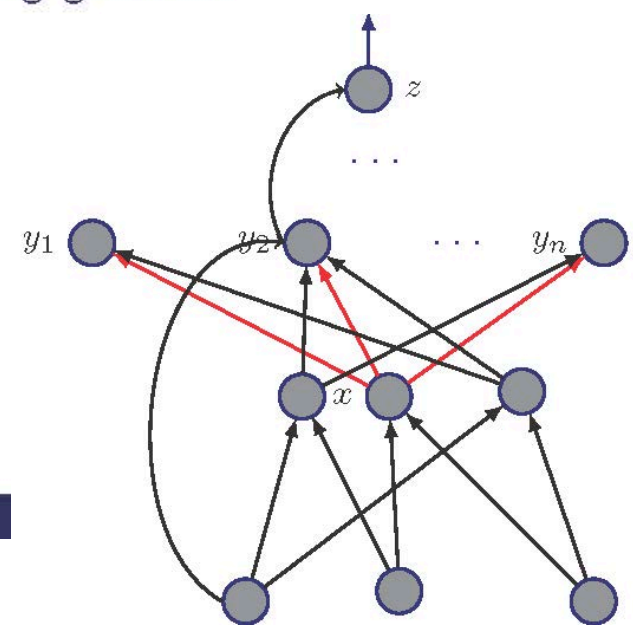
Flow Graph (for previous slide)



Recursive Application in a Computational Graph

- Fpropagation: Visit nodes in the order after a topological sort
- Compute the value of each node given its ancestors
- Bpropagation: Output gradient = 1
- Now visit nods in reverse order
- Compute gradient with respect to each node using gradient with respect to successors
- Successors of x in previous slide $\{y_1, y_2, \ldots y_n\}$:

$$\frac{dz}{dx} = \sum_{i=1}^{n} \frac{dz}{dy_i} \frac{dy_i}{dx}$$



Automatic Differentiation

- Computation of the gradient can be automatically inferred from the symbolic expression of fprop
- Every node type needs to know:
  - How to compute its output
  - How to compute its gradients with respect to its inputs given the gradient w.r.t its outputs
- Makes for rapid prototyping

Computational Graph for a MLP

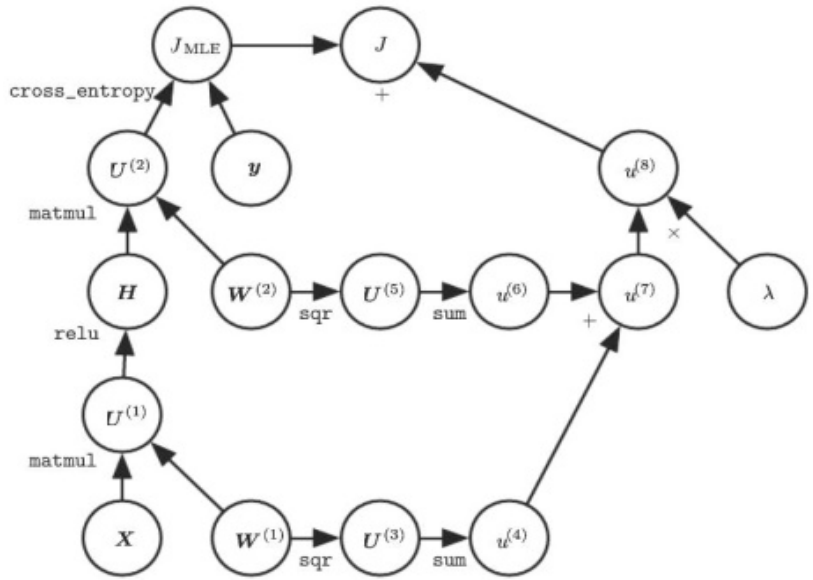


Figure: Goodfellow *et al.*

- To train we want to compute $\nabla_{W^{(1)}} J$ and $\nabla_{W^{(2)}} J$

Two paths lead backwards from J to weights: Through cross entropy and through regularization cost
Weight decay cost is relatively simple: Will always contribute 2W(i) to gradient on W(i)

CS60010: Deep Learning

Sudeshna Sarkar

Spring 2018

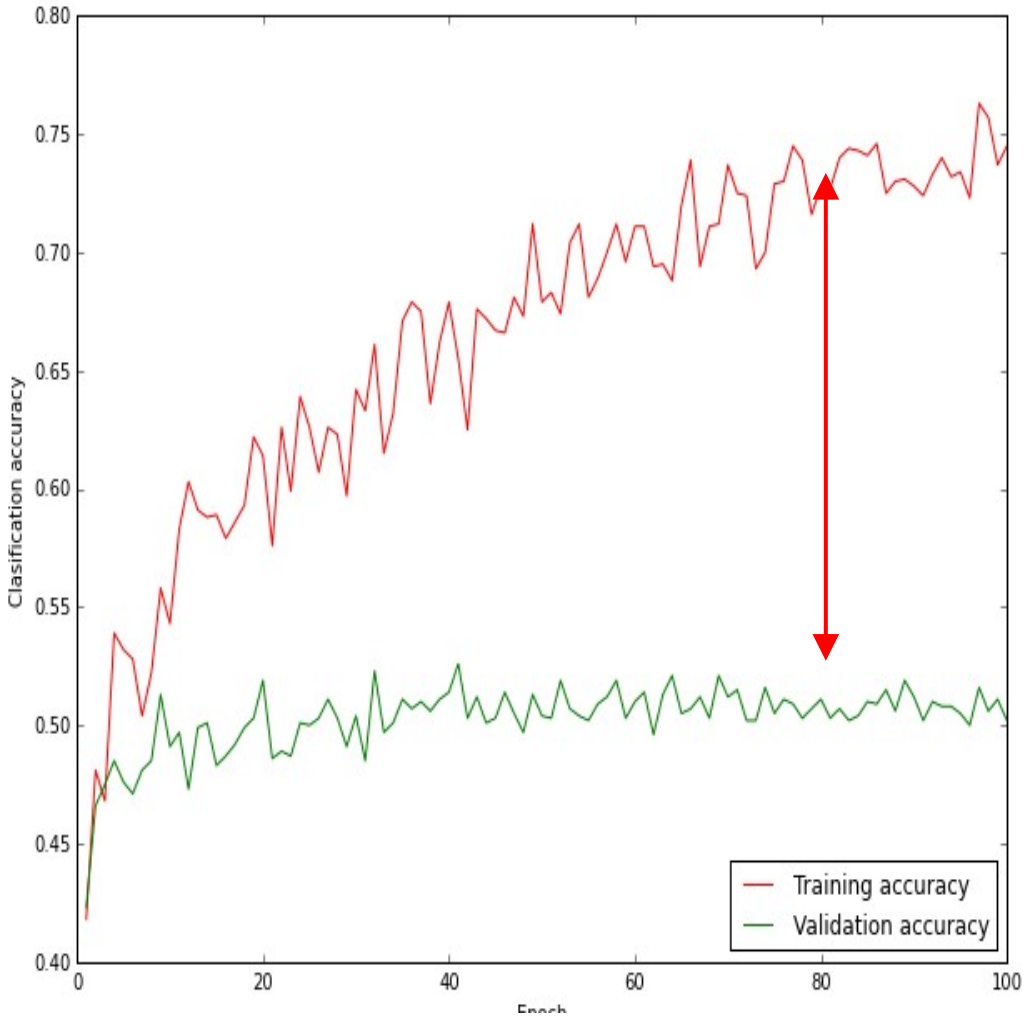
24 Jan 2019

Part 2

REGULARIZATION

Hyperparameters

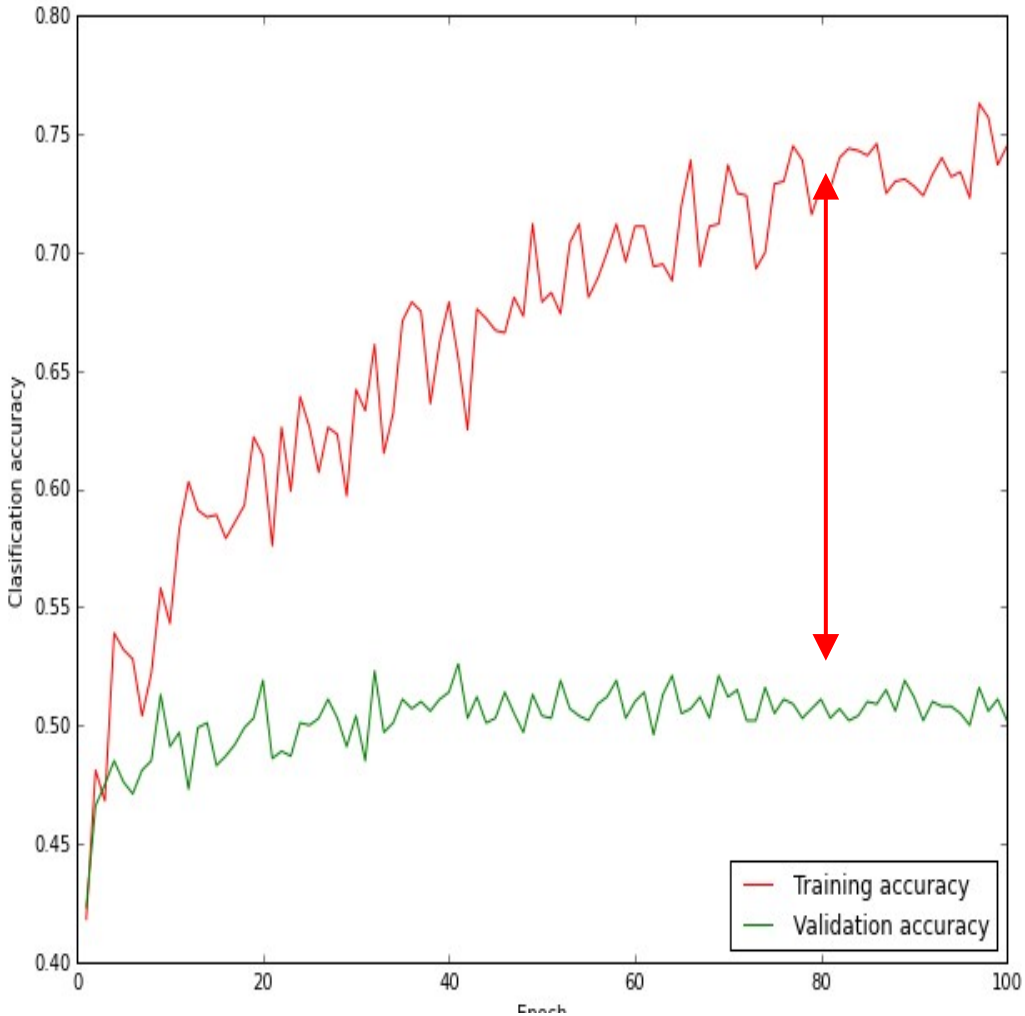
Visualize the accuracy



big gap

no gap

Monitor and visualize the accuracy:



big gap = overfitting

⇒ increase regularization strength?

no gap

=> increase model capacity?

Regularization

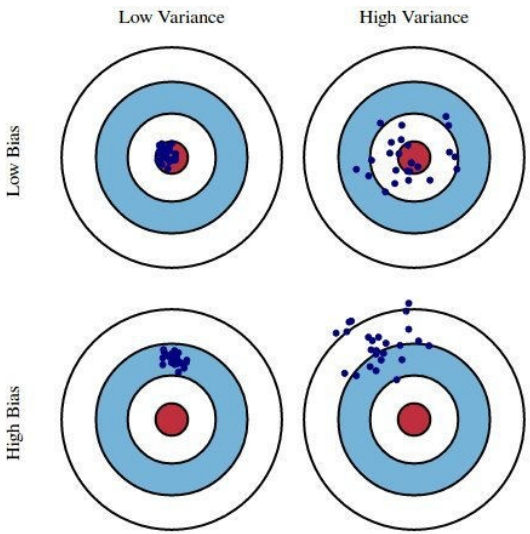
- In general: any method to prevent overfitting or help the optimization
- Overfitting: Empirical loss and expected loss are different
  - When does it happen?
    - Smaller data set
    - Larger the hypothesis class
- Regularization strategies:
  1. Extra constraints on ML model, eg adding restrictions on the par values.
  2. Extra terms in the objective function - soft constraint on parameter values.
  3. Ensemble method

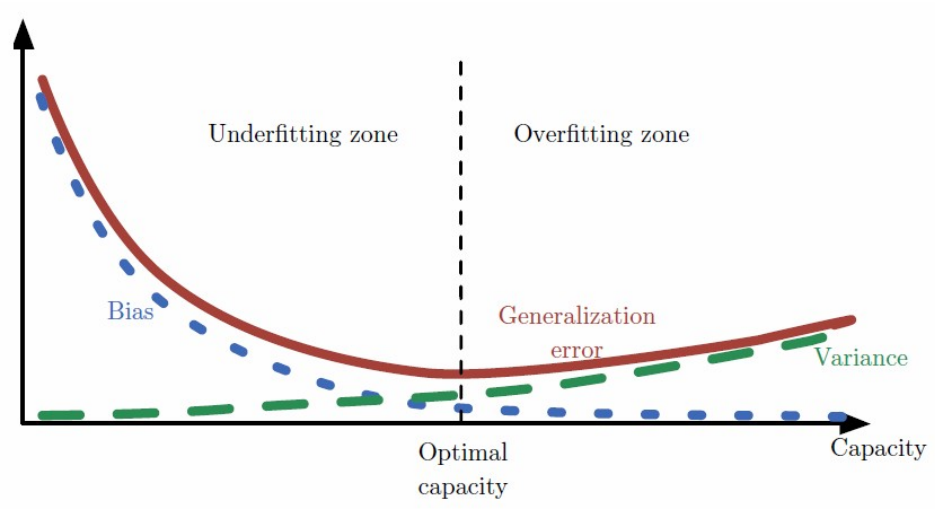
  Sometimes penalties and constraints are necessary to make an under-determined problem determined.

  - Regularization of an estimator works by trading increased bias for reduced variance.

Bias variance trade off

- Regularization of an estimator works by trading increased bias for reduced variance





Source : http://www.kdnuggets.com/2016/08/bias-variance-tradeoff-overview.html

Norms (Definition)

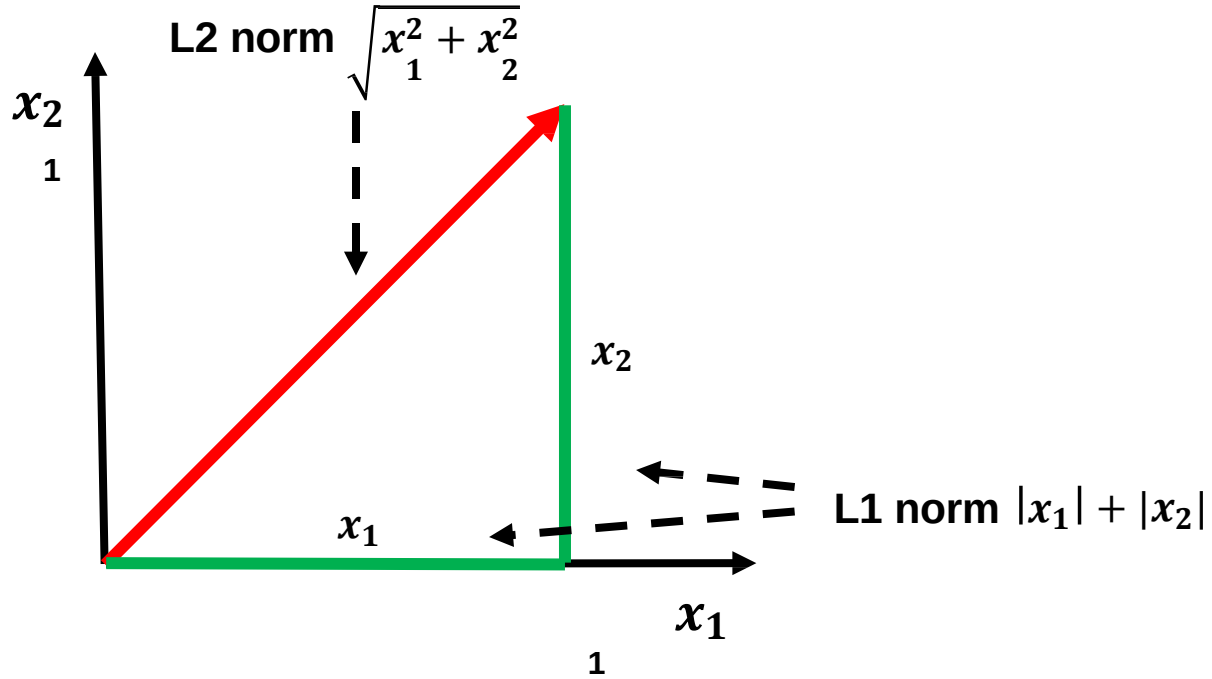
- **A way to measure the size of vector**

$$\|x\|_p = \left(\sum_i {x_i}^p\right)^{\frac{1}{p}}$$

- **Thus, L1 norm and L2 norm are?**

Norms (Example)

- $\boldsymbol{x}^T = [\mathbf{1}, \mathbf{1}]$



Squared L2 Norm

- **Squared L2 norm is used instead of original L2 norm for regularization in machine learning task**

- **All of the derivatives of the L2 norm depend on the entire vector**

- **The derivatives of the squared L2 norm with respect to each element of x each depend only on the corresponding element of x**

Squared L2 Norm

- $\boldsymbol{x^T = [x_1, x_2]}$
- **L2 norm**

$$\boldsymbol{f(x) = \sqrt{x_1^2 + x_2^2}}$$

$$\boldsymbol{f'(x) = x_1(x_1^2 + x_2^2)^{?\,\frac{1}{2}}}$$

- **Squared L2 norm**

$$\boldsymbol{f(x) = x_1^2 + x_2^2}$$

$$\boldsymbol{f'(x) = 2x_1}$$

Norms (Matrix norm)

- **L1 norm and L2 norm are defined for the way to measure size of vector**

- **Sometimes machine learning task would require a size of matrix as well**

- **Frobenius norm is :**
  - **A way to measure the size of matrix**
  - **Known as 'Matrix norm'**

$$||A||_F = \sqrt{\sum_{i,j} A_{i,j}^2},$$

Norm penalties

- **Limiting the capacity of models by adding norm penalty $\Omega(\Theta)$ to the objective function J**

Regularized objective function Penalty term

$$\boxed{\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})} = \boxed{J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})} + \boxed{\alpha\Omega(\boldsymbol{\theta})}$$

Original objective function

- **Not modifying the model in inference phase, but adding penalties to the objective function in learning phase**

- **Also known as weight decay**

L2 norm Regularization

- **Substituting squared L2 norm to the $\Omega(\Theta)$**

$||\theta||_2^2$

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

Calculating gradient

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

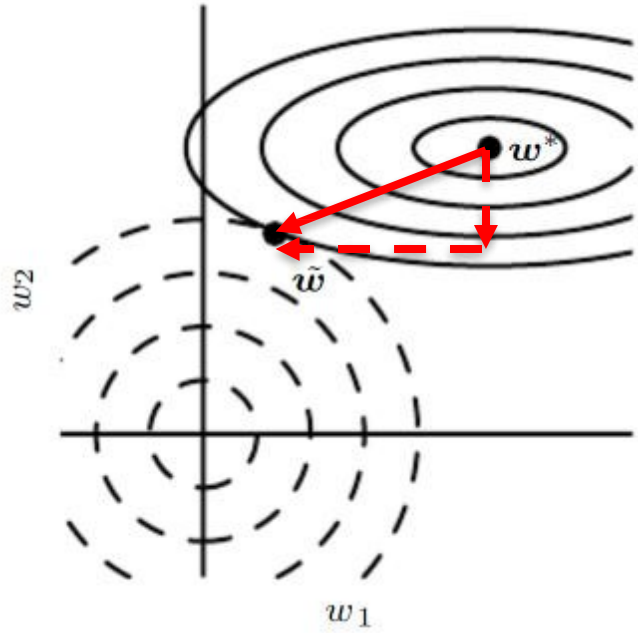
- **Applying weight update**

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right).$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

Effect of L2 norm Regularization

- **Only directions along which the parameters contribute significantly to reducing the objective function are preserved relatively intact**



L1 norm Regularization

- **Substituting L1 norm to the Ω(Θ)**

$||\theta||_1$

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

- **Calculating gradient**

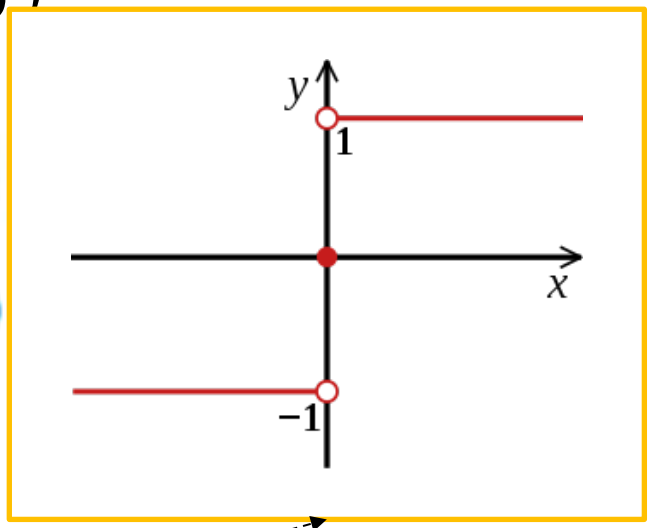
$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\mathrm{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

L1 norm Regularization

- **Substituting L1 norm to the $\Omega(\Theta)$**

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$



- **Calculating gradient**

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \mathrm{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

L1 regularization may cause the parameters to become zero for large enough α

Norm Regularization without bias

- **Usually, bias of each weight is excluded in penalty terms**

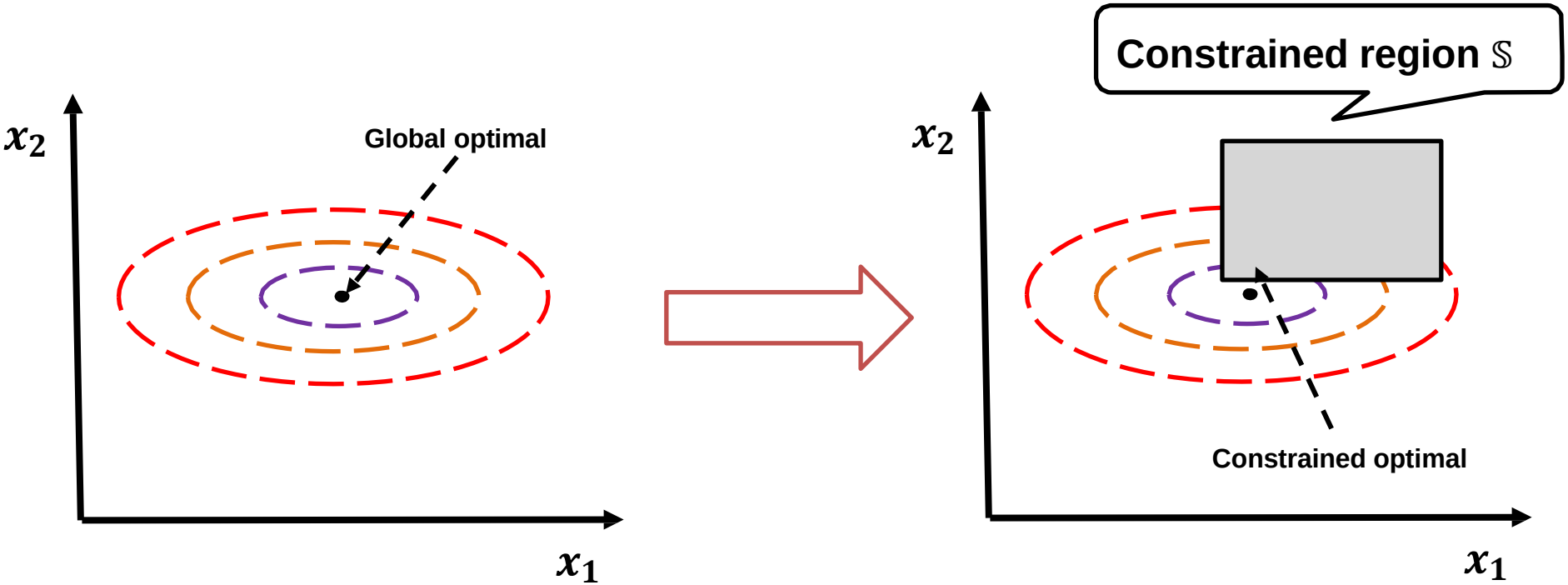
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{w})$$

- **The reason is :**
  - **The biases typically require less data to fit than the weights**
  - **Each weight specifies how two variables interact while biases specify interaction of one variables**
  - **Regularizing the bias parameters can cause underfitting**

Norm Penalties as Constrained Optimization

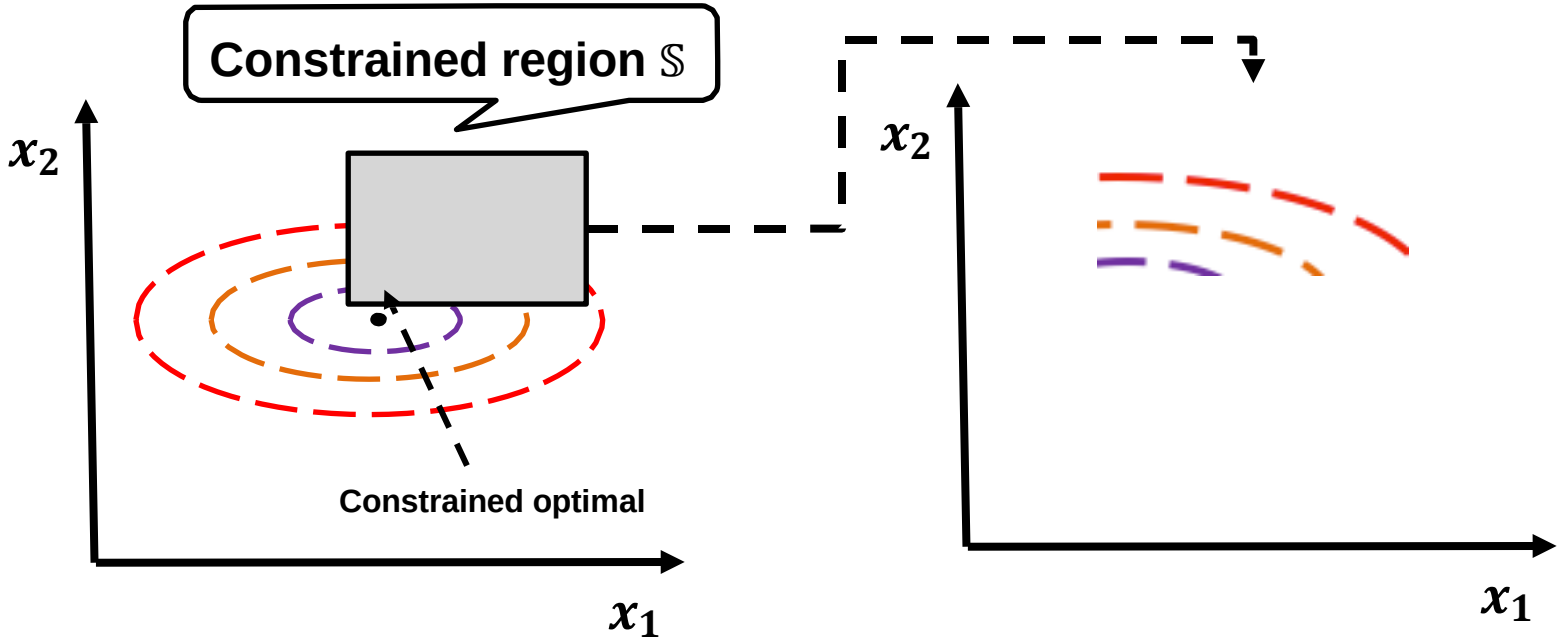
Constrained optimization

- **Sometimes one may wish to find the maximal or minimal value of $f(x)$ for value of x in some set $\mathbb{S}$**



Expression of constrained function

- **To express function with constrained condition is difficult**



Generalized Lagrange function

- A possible approach is to design a different, unconstrained optimization problem whose solution can be converted into a solution to the original constrained problem

  The unconstrained optimization function is called “Generalized Lagrange function”

Generalized Lagrange function

- **Generalized Lagrange function is defined as:**

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}).$$

- **Where the constrained region is:**

$$\mathbb{S} = \{\boldsymbol{x} | \forall \boldsymbol{i}, \boldsymbol{g}^{(\boldsymbol{i})}(\boldsymbol{x}) = \boldsymbol{0} \ \boldsymbol{and} \ \forall \boldsymbol{j}, \boldsymbol{h}^{(\boldsymbol{j})} \leq \boldsymbol{0}\}$$

- **We can find optimal $\boldsymbol{x}$ in region $\mathbb{S}$ by solving:**

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}).$$

Norm Penalties with respect to Constrained Optimization

- **Cost function regularized by a parameter norm penalty**

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \underbrace{J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})}_{\text{Original cost function}} + \underbrace{\alpha\Omega(\boldsymbol{\theta})}_{\text{norm penalty, constrained term}}.$$

- **If we wanted to constrain $\Omega(\boldsymbol{\theta})$ to be less than some constant k, we could construct a generalized Lagrange function**

$$\mathcal{L}(\boldsymbol{\theta}, \alpha; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k).$$

Norm Penalties with respect to Constrained Optimization

To gain some insight into the effect of the constraint, we can fix α^* and view the problem as just a function of $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^* = \boldsymbol{argmin}_{\boldsymbol{\theta}}\, \mathcal{L}\,(\boldsymbol{\theta}, \boldsymbol{\alpha}^*) = \boldsymbol{argmin}_{\boldsymbol{\theta}}\, [\, \boldsymbol{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \boldsymbol{\alpha}^* \boldsymbol{\Omega}\,(\boldsymbol{\theta})\ ]$$

- **This is exactly the same as the regularized training problem of minimizing $\boldsymbol{J}$**

Data Augmentation and Noise Robustness

Introduction to Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data

- Dataset augmentation is a technique that creating fake data and adding it to the training set

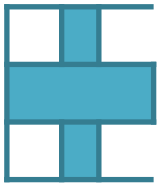
Original data

Artificial data

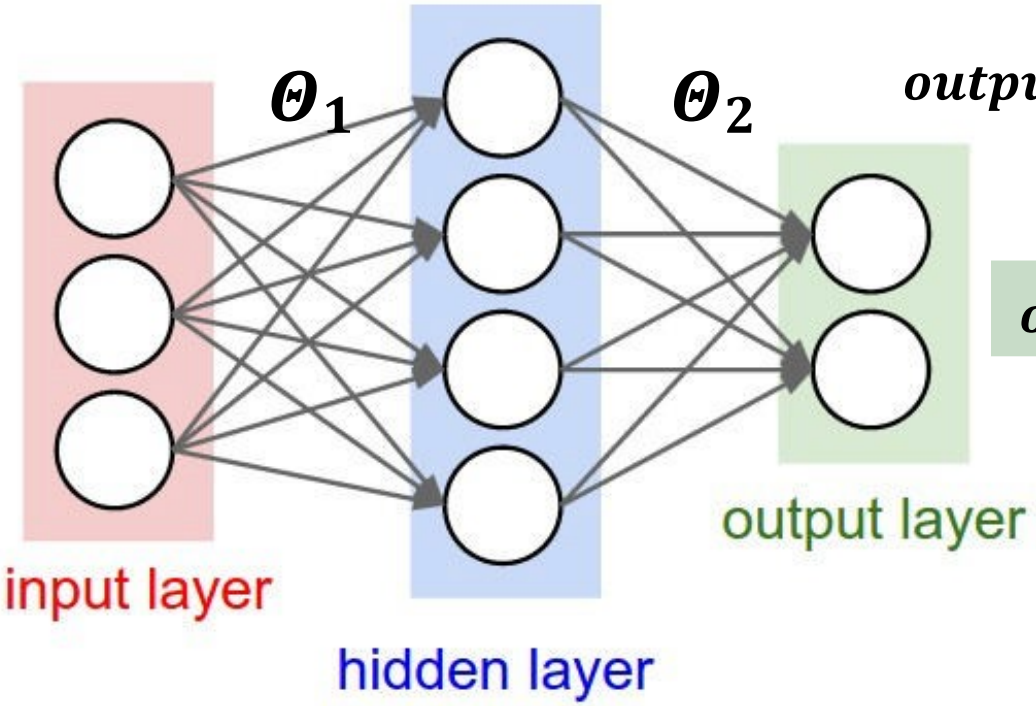
Image source : https://www.pexels.com/photo

Injecting noise (Training data)

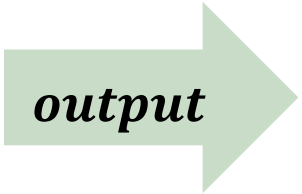
- **Injecting random noise into input data to improve robustness**



Random Noise Filter

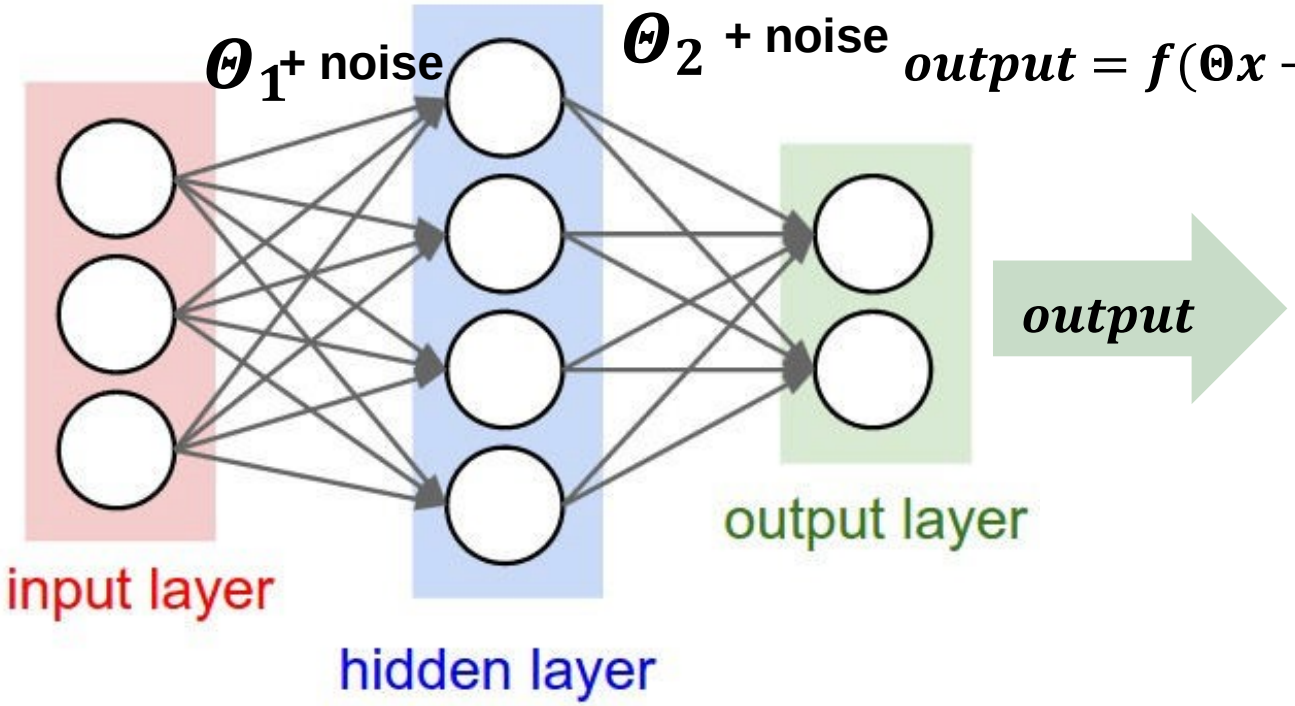


$$output = f(\Theta x + b)$$



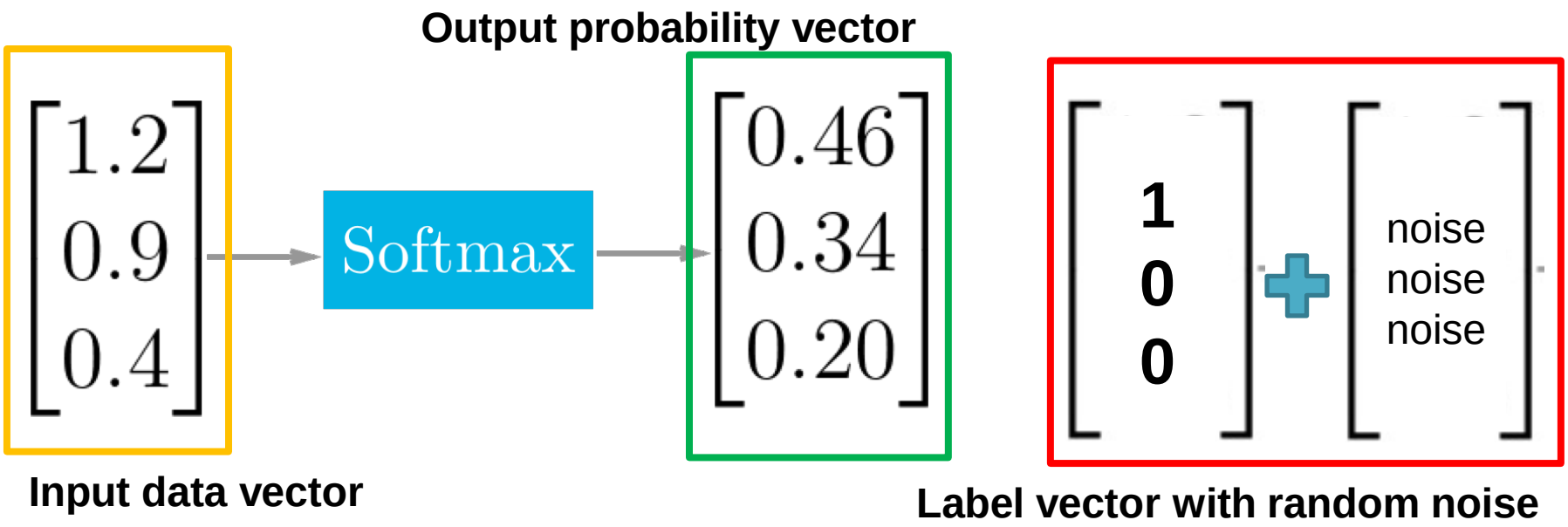
Injecting noise (Weight)

- **Injecting random noise into weight to improve robustness**
- **This makes the model relatively insensitive to small variations in the weights**



Injecting noise (Label)

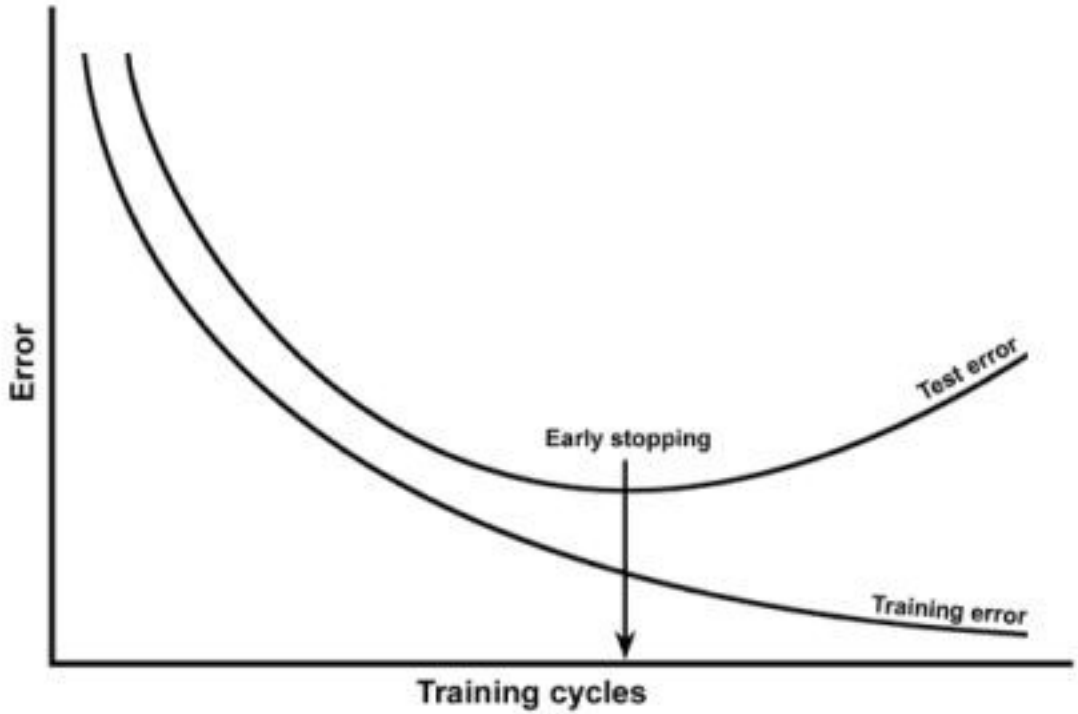
- **Most datasets have some amount of mistakes in the y labels**



Early Stopping

Necessity of early stopping

- **Learning too much iterations causes overfitting**



Feature of early stopping

- **Early stopping regards iteration number as hyper parameter and find optimal value of it**

- **Computation cost to find optimal iteration number is negligible**
  - **This can be parallelized while training process on separate machine**

- **Early stopping doesn’t affect the formula of cost function**

PARAMETER TYING AND PARAMETER SHARING

Parameter Tying

- **Parameter dependency**
  - Other ways to express prior knowledge of parameters
  - We may know from domain and model architecture that there should be some dependencies between model parameters

The goal of parameter tying

- We want to express that certain parameters should be close to one another

A scenario of parameter tying

- Two models performing the same classification task (with same set of classes) but with somewhat different input distributions
  - Model $\boldsymbol{A}$ with parameters w(A)
  - Model $\boldsymbol{B}$ with parameters w(B)
- The two models will map the input to two different, but related output
- If the tasks are similar enough (perhaps with similar input and output distributions) then we believe that the model parameters should be close to each other:
- We can leverage this information via regularization Use a parameter norm penalty

$$\Omega(wA, wB) = \|wA - wB\|^2$$

penalty for parameter tying

Regularized objective function

Penalty term

$$\boxed{\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})} = \boxed{J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})} + \boxed{\alpha\Omega(\boldsymbol{\theta})}$$

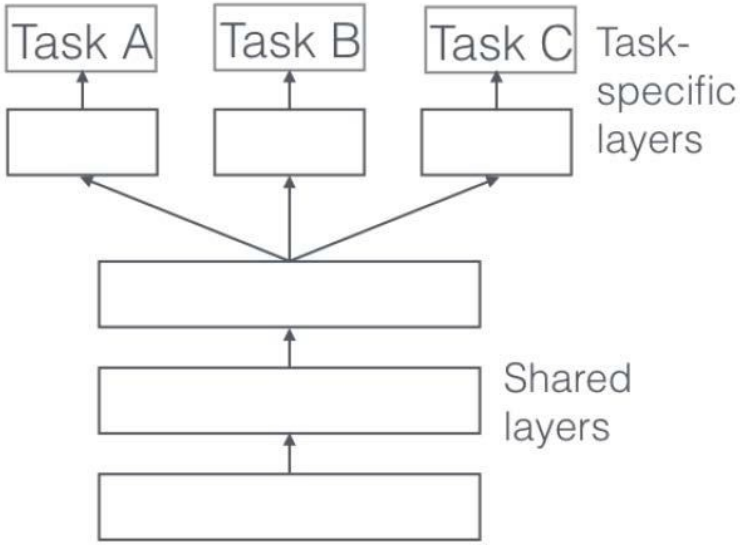
Original objective function

Multi-Task Learning (MTL)

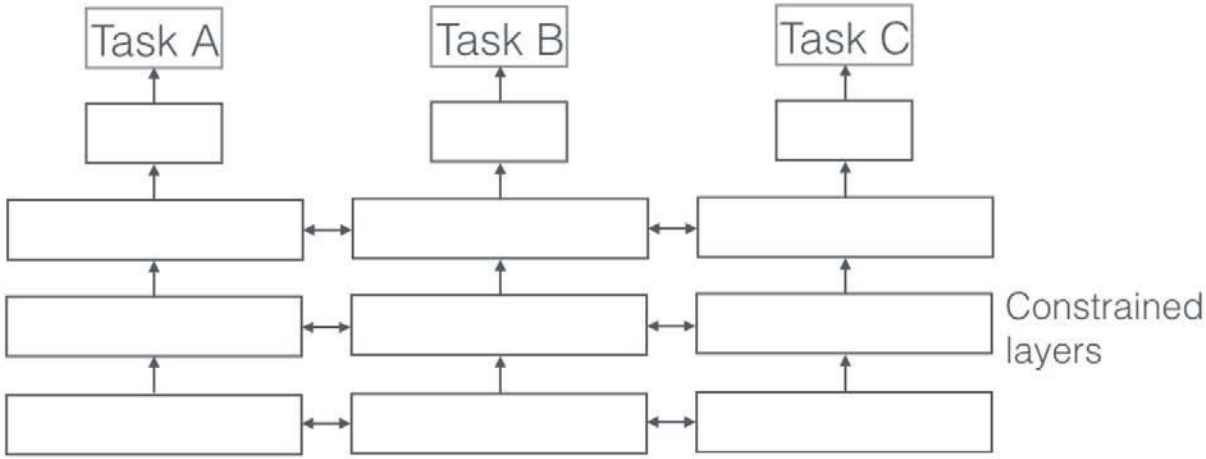
- **Sharing the representation between related tasks**
  - **We can enable out model to generalize better on our original task**
  - **Another approach of bagging (with different cost functions)**
- **MTL is also known as:**
  - **Joint learning, Learning to learn, learning with auxiliary tasks**
- **Optimizing more than one loss function**
- **Improves generalization by leveraging the domain-specific information contained in the training data**
  - **Even if the problem optimizing one loss functions, there might be the chances to improve performance by adding an auxiliary task upon the major task**
- **Motivated by human learning process**

Two MTL Methods

- **Hard parameter sharing**
  - **Greatly reduce the risk of overfitting**
  - **Similar concept of bagging**



- **Soft parameter sharing**
  - **Take a role of regularization**



Learning task relationship with Regularization

Notation

- **Task T, for each task t, we have a model m_t with parameters a_t of dimensionality d**
- **The parameter vector a_t and parameter matrix A is:**

$$a_t = [a_{1,t}, \cdots, a_{d,t}] \qquad A = \begin{bmatrix} \vdots & \vdots & \vdots \\ a_{\cdot 1} & \cdots & a_{\cdot T} \\ \vdots & \vdots & \vdots \end{bmatrix}$$

i-th features of the model for every task

Parameter a_j corresponding to the j-th model

Learning task relationship

$$\Omega = \|\bar{a}\|^2 + \frac{\lambda}{T}\sum_{t=1}^{T}\|a_{\cdot,t} - \bar{a}\|^2 \qquad \textit{where,} \quad \bar{a} = \left(\sum_{t=1}^{T} a_{\cdot,t}\right)/T$$

Regularized objective function

Penalty term

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

Original objective function