

# Chapter 7: Regularization for Deep Learning

Lei Liu

September 16, 2018

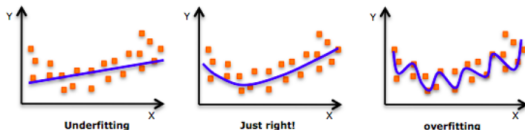
# Outline

- Background
- Parameter Norm Penalties
- Norm Penalties as Constrained Optimization
- Regularization and Under-Constrained Problems
- Dataset Augmentation
- Noise Robustness
- Semi-Supervised Learning
- Multitask Learning

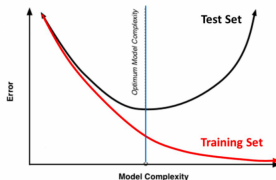
# Outline

- Early Stopping
- Parameter Tying and Parameter Sharing
- Sparse Representations
- Bagging and Other Ensemble Methods
- Dropout
- Adversarial Training
- Tangent Distance, Tangent Prop and Manifold Tangent Classifier

# Background



Training Vs. Test Set Error



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

# Background

## Why regularization?

- In practice, an overly complex model family does not necessarily include the target function or the true data-generating process.
- What this means is that controlling the complexity of the model is not a simple matter of finding the model of the right size, with the right number of parameters.
- Instead, we find that the best fitting model (in the sense of minimizing generalization error) is **a large model** that has been **regularized appropriately**.

# Parameter Norm Penalties

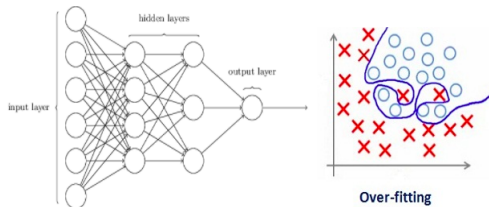
We denote the regularized objective function by:

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

where  $\alpha \in [0, \infty)$  is a hyperparameter that weights the relative contribution of the penalty term,  $\Omega$ , relative to the standard objective function  $J$ .

# Background

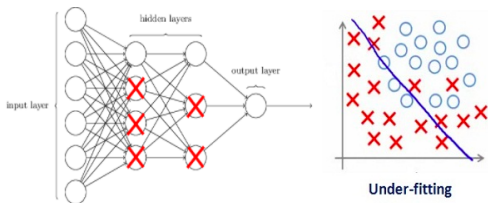
How does Regularization help reduce Overfitting?



**Figure:** The regularization coefficient is zero (no regularization): over-fitting

# Background

How does Regularization help reduce Overfitting?

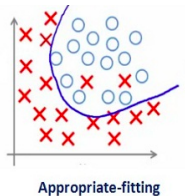


**Figure:** The regularization coefficient is so high that some of the weight matrices are nearly equal to zero, which results in a much simpler linear network and slight underfitting of the training data.



# Background

How does Regularization help reduce Overfitting?



**Figure:** A good regularization coefficient can obtain a well-fitted model.

# Parameter Norm Penalties

## $L^2$ Parameter Regularization

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \frac{1}{2}(\mathbf{H}\mathbf{w} - \mathbf{y})^T(\mathbf{H}\mathbf{w} - \mathbf{y}) + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$

which is also known as weight decay in neural networks.

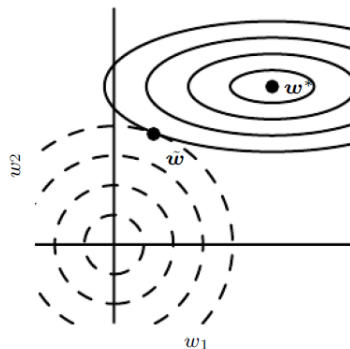
- The optimal solution is

$$\arg \min_{\mathbf{w}} \tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = (\mathbf{H}^T\mathbf{H} + \alpha\mathbf{I})^{-1}\mathbf{H}^T\mathbf{y}.$$

- Consider the cases that  $\alpha = 0, \infty$
- Positive  $\alpha$  will cause the magnitude of the weights to be smaller than in the usual linear solution
- A different view of regularization: we want to optimize the error while keeping the  $L^2$  norm of the weights,  $\mathbf{w}^T\mathbf{w}$ , bounded.

# Parameter Norm Penalties

## $L^2$ Parameter Regularization



**Figure:** An illustration of the effect of  $L^2$  (or weight decay) regularization on the value of the optimal  $\mathbf{w}$ .

# Parameter Norm Penalties

## $L^1$ Regularization

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \frac{1}{2}(\mathbf{H}\mathbf{w} - \mathbf{y})^T(\mathbf{H}\mathbf{w} - \mathbf{y}) + \alpha\|\mathbf{w}\|_1$$

which is also known as weight decay in neural networks.

- The optimal solution of  $\arg \min_{\mathbf{w}} \tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y})$  is

$$w_i = \text{sign}(w_i^*) \max\{|w_i^*| - \alpha/h_{ii}, 0\}$$

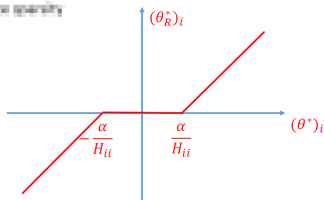
where  $\mathbf{w}^*$  is the optimal solution of  $J(\cdot)$  (without regularization).

- In comparison to  $L^2$  regularization,  $L_1$  regularization results in a solution that is more sparse.

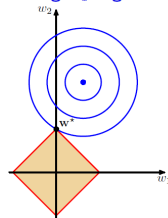
# Parameter Norm Penalties

## $L^1$ Parameter Regularization

ridge penalty



Visualizing  $L_1$  regularization



- If  $\alpha$  is big enough, the circle is very likely to intersect the diamond at one of the corners
- This makes  $L^1$  regularization much more likely to make some weights exactly 0

# Norm Penalties as Constrained Optimization

Consider a constrained Penalty:

$$\begin{aligned} \arg \min_{\theta} \quad & J(\theta; \mathbf{X}, \mathbf{y}) \\ \text{s.t.} \quad & \Omega(\theta) \leq \eta \end{aligned}$$

The Lagrangian is:

$$\mathcal{L}(\theta, \alpha) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha(\Omega(\theta) - \eta)$$

The solution to the constrained problem is given by

$$\arg \min_{\theta} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\theta, \alpha)$$

# Norm Penalties as Constrained Optimization

we can fix  $\alpha$  and view the problem as just a function of  $\theta$ :

$$\theta^* = \arg \min_{\theta} \mathcal{L}(\theta, \alpha^*) = \arg \min_{\theta} J(\theta; \mathbf{X}, \mathbf{y}) + \alpha^* \Omega(\theta).$$

This is the same as the regularized training problem of minimizing  $\tilde{J}$ .

- In principle, one can solve for  $\eta$ , but the relationship between  $\eta$  and  $\alpha^*$  depends on the form of  $J$ .
- While we do not know the exact size of the constraint region, we can control it roughly by increasing or decreasing  $\alpha$  in order to grow or shrink the constraint region.

# Regularization and Under-Constrained Problems

- Many linear models in machine learning depend on inverting the matrix  $\mathbf{X}^T \mathbf{X}$ . This is not possible when  $\mathbf{X}^T \mathbf{X}$  (fewer rows) is singular.
- Many forms of regularization correspond to inverting  $\mathbf{X}^T \mathbf{X} + \alpha \mathbf{I}$  instead. This regularized matrix is guaranteed to be invertible.
- Recall that one definition of the pseudoinverse  $\mathbf{X}^\dagger$  of a matrix  $\mathbf{X}$  is

$$\mathbf{X}^\dagger = \lim_{\alpha \rightarrow 0} (\mathbf{X}^T \mathbf{X} + \alpha \mathbf{I})^{-1} \mathbf{X}^T.$$

Pseudoinverse is the limit of equation as the regularization coefficient shrinks to zero.

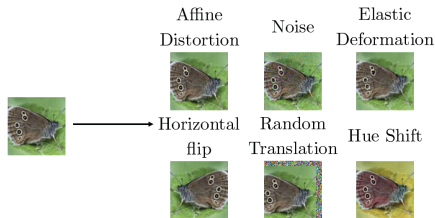


# Dataset Augmentation

The amount of data is limited  $\Rightarrow$  create fake data and add it to the training set

- Transformations
- Injecting noise

## Dataset Augmentation



Digits dataset



Note that noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.

# Regularization and Under-Constrained Problems

- For some models the addition of noise with infinitesimal variance at the **input** of the model is equivalent to imposing a penalty on the norm of the weights.
- Add noise to the **weights** (Bayesian inference over the weights.)

Noise applied to the **weights** can also be interpreted as equivalent to a more traditional **form of regularization**, encouraging **stability** of the function to be learned.

$$\begin{aligned} J &= \mathbb{E}_{p(\mathbf{x}, y)} [(\hat{y}(\mathbf{x}) - y)^2] \\ \Rightarrow \tilde{J} &= \mathbb{E}_{p(\mathbf{x}, y, \epsilon_{\mathbf{w}})} [(\hat{y}_{\epsilon_{\mathbf{w}}}(\mathbf{x}) - y)^2] \\ &= \mathbb{E}_{p(\mathbf{x}, y, \epsilon_{\mathbf{w}})} [\hat{y}_{\epsilon_{\mathbf{w}}}(\mathbf{x})^2 - 2y\hat{y}_{\epsilon_{\mathbf{w}}}(\mathbf{x}) - y^2] \\ &= \mathbb{E}_{p(\mathbf{x}, y)} [(\hat{y}(\mathbf{x}) - y)^2] + \epsilon \mathbb{E}_{p(\mathbf{x}, y)} [\|\nabla_{\mathbf{w}} \hat{y}(\mathbf{x})\|^2] \end{aligned}$$

with  $\hat{y}_{\epsilon_{\mathbf{w}}}(\mathbf{x}) = \hat{y}(\mathbf{x}) + \nabla_{\mathbf{w}} \hat{y}(\mathbf{x}) \epsilon_{\mathbf{w}}$ ,  $\epsilon_{\mathbf{w}} \sim \mathcal{CN}(0, \epsilon \mathbf{I})$ .

# Regularization and Under-Constrained Problems

## Injecting Noise at the Output Targets

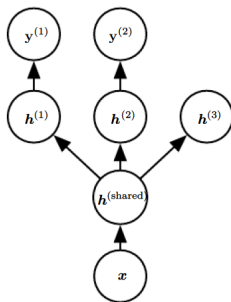
- Most datasets have some number of mistakes in the  $y$  labels. It can be harmful to maximize  $\log p(y|\mathbf{x})$  when  $y$  is a mistake. One way to prevent this is to explicitly model the noise on the labels.
- For example, we can assume that for some small constant  $\epsilon$ , the training set label  $y$  is correct with probability  $1 - \epsilon$ , and otherwise any of the other possible labels might be correct.
- Soft information output?

# Semi-Supervised Learning

- Both unlabeled examples from  $P(\mathbf{x})$  and labeled examples from  $P(\mathbf{x}, \mathbf{y})$  are used to estimate  $P(\mathbf{y}|\mathbf{x})$  or predict  $\mathbf{y}$  from  $\mathbf{x}$ .
- Instead of having separate unsupervised and supervised components in the model, one can construct models in which a generative model of either  $P(\mathbf{x})$  or  $P(\mathbf{x}, \mathbf{y})$  shares parameters with a discriminative model of  $P(\mathbf{y}|\mathbf{x})$ .
- By controlling how much of the generative criterion is included in the total criterion, one can find a better trade-off than with a purely generative or a purely discriminative training criterion.
- More constraints? So, regularization?

# Multitask Learning

- Multitask learning is a way to improve generalization by pooling the examples (which can be seen as soft constraints imposed on the parameters) arising out of several tasks.
- Additional training examples put more pressure on the parameters (when part of a model is **shared across tasks**, it is **more constrained** toward good values), often yielding better generalization.



# Early Stopping

**Early stopping** is a kind of cross-validation strategy where we keep one part of the training set as the **validation set**. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model.



# Early Stopping

---

Let  $n$  be the number of steps between evaluations.

Let  $p$  be the “patience,” the number of times to observe worsening validation set error before giving up.

Let  $\theta_o$  be the initial parameters.

$\theta \leftarrow \theta_o$

$i \leftarrow 0$

$j \leftarrow 0$

$v \leftarrow \infty$

$\theta^* \leftarrow \theta$

$i^* \leftarrow i$

while  $j < p$  do

    Update  $\theta$  by running the training algorithm for  $n$  steps.

$i \leftarrow i + n$

$v' \leftarrow \text{ValidationSetError}(\theta)$

    if  $v' < v$  then

$j \leftarrow 0$

$\theta^* \leftarrow \theta$

$i^* \leftarrow i$

$v \leftarrow v'$

    else

$j \leftarrow j + 1$

    end if

end while

Best parameters are  $\theta^*$ , best number of training steps is  $i^*$ .

# Early Stopping

---

**Algorithm 7.2** A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

---

Let  $\mathbf{X}^{(\text{train})}$  and  $\mathbf{y}^{(\text{train})}$  be the training set.  
Split  $\mathbf{X}^{(\text{train})}$  and  $\mathbf{y}^{(\text{train})}$  into  $(\mathbf{X}^{(\text{subtrain})}, \mathbf{X}^{(\text{valid})})$  and  $(\mathbf{y}^{(\text{subtrain})}, \mathbf{y}^{(\text{valid})})$  respectively.  
Run early stopping (algorithm 7.1) starting from random  $\theta$  using  $\mathbf{X}^{(\text{subtrain})}$  and  $\mathbf{y}^{(\text{subtrain})}$  for training data and  $\mathbf{X}^{(\text{valid})}$  and  $\mathbf{y}^{(\text{valid})}$  for validation data. This returns  $i^*$ , the optimal number of steps.  
Set  $\theta$  to random values again.  
Train on  $\mathbf{X}^{(\text{train})}$  and  $\mathbf{y}^{(\text{train})}$  for  $i^*$  steps.

---

---

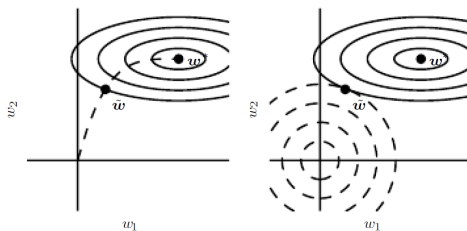
**Algorithm 7.3** Meta-algorithm using early stopping to determine at what objective value we start to overfit, then continue training until that value is reached.

---

Let  $\mathbf{X}^{(\text{train})}$  and  $\mathbf{y}^{(\text{train})}$  be the training set.  
Split  $\mathbf{X}^{(\text{train})}$  and  $\mathbf{y}^{(\text{train})}$  into  $(\mathbf{X}^{(\text{subtrain})}, \mathbf{X}^{(\text{valid})})$  and  $(\mathbf{y}^{(\text{subtrain})}, \mathbf{y}^{(\text{valid})})$  respectively.  
Run early stopping (algorithm 7.1) starting from random  $\theta$  using  $\mathbf{X}^{(\text{subtrain})}$  and  $\mathbf{y}^{(\text{subtrain})}$  for training data and  $\mathbf{X}^{(\text{valid})}$  and  $\mathbf{y}^{(\text{valid})}$  for validation data. This updates  $\theta$ .  
 $\epsilon \leftarrow J(\theta, \mathbf{X}^{(\text{subtrain})}, \mathbf{y}^{(\text{subtrain})})$   
while  $J(\theta, \mathbf{X}^{(\text{valid})}, \mathbf{y}^{(\text{valid})}) > \epsilon$  do  
    Train on  $\mathbf{X}^{(\text{train})}$  and  $\mathbf{y}^{(\text{train})}$  for  $n$  steps.  
end while



# Early Stopping



**Figure:** An illustration of the effect of early stopping.

# Early Stopping

Consider:

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*), \quad \nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

where  $\mathbf{H}$  is the Hessian matrix of  $J$  w.r.t.  $\mathbf{w}$  evaluated at  $\mathbf{w}^*$ . Then,

$$\begin{aligned}\mathbf{w}^\tau &= \mathbf{w}^{\tau-1} - \epsilon \nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = \mathbf{w}^{\tau-1} - \epsilon \mathbf{H}(\mathbf{w}^{\tau-1} - \mathbf{w}^*) \\ \mathbf{Q}^T(\mathbf{w}^\tau - \mathbf{w}^*) &= (\mathbf{I} - \epsilon \mathbf{\Lambda}) \mathbf{Q}^T(\mathbf{w}^{\tau-1} - \mathbf{w}^*), \quad \mathbf{H} = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^T \\ \mathbf{Q}^T \mathbf{w}^\tau &= [\mathbf{I} - (\mathbf{I} - \epsilon \mathbf{\Lambda})^\tau] \mathbf{Q}^T \mathbf{w}^*\end{aligned}$$

Comparison with  $L^2$  regularization:

$$\mathbf{Q}^T \tilde{\mathbf{w}} = [\mathbf{I} - (\alpha \mathbf{I} + \mathbf{\Lambda})^{-1} \alpha] \mathbf{Q}^T \mathbf{w}^*.$$

Thus, early stopping is equivalent to  $L^2$  regularization if  $\tau$  and  $\alpha$  are properly chosen such as

$$(\mathbf{I} - \epsilon \mathbf{\Lambda})^\tau = (\alpha \mathbf{I} + \mathbf{\Lambda})^{-1} \alpha.$$

# Parameter Tying and Parameter Sharing

- Sometimes, there should be some dependencies between the model parameters.
- E.g. model A:  $\hat{y}^{(A)} = f(\mathbf{w}^{(A)}, \mathbf{x})$ ; model B:  $\hat{y}^{(B)} = f(\mathbf{w}^{(B)}, \mathbf{x})$ .

If we know  $\mathbf{w}^{(A)}$  is close to  $\mathbf{w}^{(B)}$ , we can use a parameter norm penalty of the form  $\Omega(\mathbf{w}^{(A)}, \mathbf{w}^{(B)}) = \|\mathbf{w}^{(A)} - \mathbf{w}^{(B)}\|^2$ .

- Parameter sharing

# Sparse Representations

- Sometimes, the activations of the units in a neural network are sparse.

$$\begin{matrix} \begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} \\ y \in \mathbb{R}^m \end{matrix} = \begin{matrix} \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \\ B \in \mathbb{R}^{m \times n} \end{matrix} \begin{matrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix} \\ h \in \mathbb{R}^n \end{matrix}$$

Figure: Sparse Parametrization and Sparse Representations,  $\mathbf{h} = f(\mathbf{x})$  ( $\mathbf{h}$  not related to  $\theta$ )

- In this case we can add an  $L^1$  norm penalty on the representation.

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha \|\mathbf{h}\|_1$$

- Orthogonal matching pursuit (OMP- $k$ ) can be used to solve

$$\arg \min_{\|\mathbf{h}\|_0 \leq k} \|\mathbf{x} - \mathbf{W}\mathbf{h}\|.$$

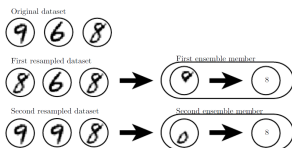
Thus, obtain the sparse representation.

# Bagging and Other Ensemble Methods

- **Bagging** is a technique for reducing generalization error by combining several models. Train different models separately, then have all the models vote on the output for test examples. Called **Model averaging** and **Ensemble methods**.
- Suppose each model makes an error  $\epsilon_i$  on each example with  $E[\epsilon_i^2] = v$  and  $E[\epsilon_i \epsilon_j] = c$ . Then the average error satisfies

$$E[(\frac{1}{k} \sum_i \epsilon_i)^2] = \frac{1}{k} E[\sum_i (\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j)] = \frac{1}{k} v + \frac{k-1}{k} c. \quad (1)$$

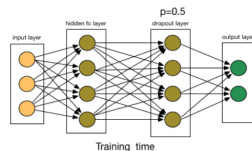
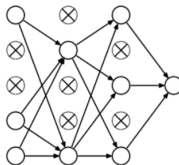
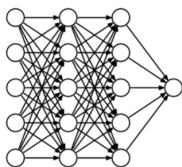
If  $c \rightarrow 0$ , MSE is  $\frac{1}{k} v$ .



- Neural networks reach many solution points that they can often benefit from model averaging even if all the models are trained on the same dataset.

# Dropout




- This is the one of the most interesting types of regularization techniques. It also produces very good results and is consequently the most frequently used regularization technique in the field of deep learning.
- **Dropout:** at **every iteration**, it **randomly selects some nodes and removes** them along with all of their incoming and outgoing connections as shown below.



- So each iteration has a different set of nodes and this results in a different set of outputs. It can also be thought of as an **ensemble technique** in machine learning.

# Adversarial Training

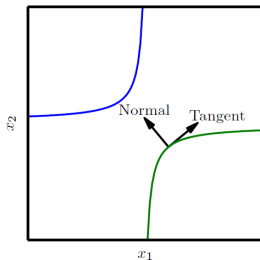
- $x'$  can be so similar to  $x$  that a human observer cannot tell the difference between the original example and the adversarial example, but the network can make highly different predictions.

|   |                 |   |     |   |
|---|-----------------|---|-----|---|
|  | $+ .007 \times$ |  | $=$ |  |
| $x$   |                 | $\text{sign}(\nabla_x J(\theta, x, y))$   |     | $x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$                              |
| $y = \text{"panda"}$  |                 | $\text{"nematode"}$   |     | $\text{"gibbon"}$   |
| $w / 57.7\%$  |                 | $w / 8.2\%$   |     | $w / 99.3\%$  |
| confidence  |                 | confidence  |     | confidence  |

- Reduce the error rate via **adversarial training** - training on **adversarially perturbed examples** from the training set.
- **linear functions** change rapidly with numerous inputs. If each input changes by  $\epsilon$ , then a linear function with weights  $\mathbf{w}$  changes by  $\epsilon \|\mathbf{w}\|_1$ , which is very large if  $\mathbf{w}$  is highdimensional.
- **Neural networks** have the flexibility to capture linear trends in the training data while still learning to resist local perturbation.

# Tangent Propagation

- Many learning algorithms overcome the curse of dimensionality by assuming that the data lies near a low-dimensional manifold.
- Tangent distance:** uses as nearest neighbor distance between points  $\mathbf{x}_1$  and  $\mathbf{x}_2$  the distance between the manifolds  $M_1$  and  $M_2$  to which they respectively belong.



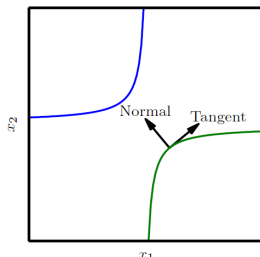


# Tangent Propagation

- **Tangent prop algorithm** trains a neural net classifier with an extra penalty to make each output  $f(\mathbf{x})$  of the neural net locally invariant to known factors of variation.
- Local invariance is achieved by requiring  $\nabla_{\mathbf{x}} f(\mathbf{x})$  to be orthogonal to the known manifold tangent vectors  $\mathbf{v}^{(i)}$  at  $\mathbf{x}$ , or adding a regularization penalty:

$$\Omega(f) = \sum_i (\nabla_{\mathbf{x}}^T f(\mathbf{x}) \mathbf{v}^{(i)})$$

- $\mathbf{v}^{(i)}$  can be transformations, such as translation, rotation, and scaling in images.
- Relations to dataset augmentation and adversarial training.



# Tangent Propagation

- Relations to dataset augmentation and adversarial training.  
Different to the dataset augmentation, the network is **explicitly trained to correctly classify distinct inputs that were created by applying more than an infinitesimal amount of these transformations**. Tangent propagation does **not require explicitly visiting a new input point**. Instead, it analytically regularizes the model to **resist perturbation in the directions of the specified transformation**.
- Drawbacks:
  - First, it only regularizes the model to resist infinitesimal perturbation. Explicit dataset augmentation confers resistance to larger perturbations.
  - Second, the infinitesimal approach poses difficulties for models based on rectified linear units. (sigmoid or tanh units ok)

# Conclusions

- In this chapter, we have described most of the general strategies used to regularize neural networks.
- Regularization is a central theme of machine learning and as such will be revisited periodically in most of the remaining chapters.

# *Thanks!*

Q&A?

September 16, 2018