Introduction to Bias & Variance

Let us consider the problem of fitting a curve through a given set of points.

We can consider two models:

Simple (degree: 1) $y = \hat{f}(x) = w_1 x + w_0$

Complex (degree: 25) $y = \hat{f}(x) = \sum_{i=1}^{25} w_i x^i + w_0$

Note: In both cases we are making an assumption about how y is related to x. We have no idea about the true relation $f(x)$

We are given 500 data points

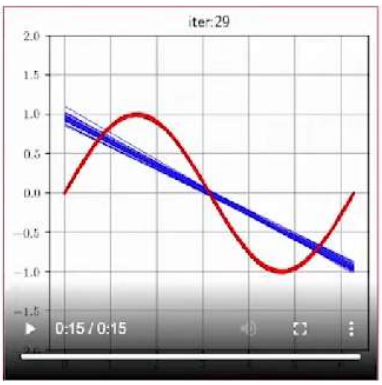
Let's sample 200 data points & train a simple & complex model

We'll repeat the process 'k' times to train multiple models (each model will see different sample of the data)

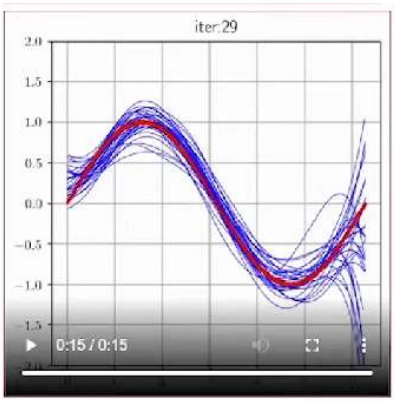
Simple models trained on different samples of data do not differ much from each other.

However they are very far from the true curve (underfitting)

On the other hand, complex models trained on different samples are very different from each other (high variance)

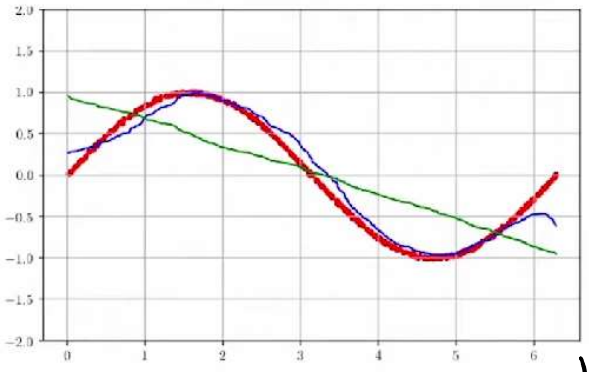


Simple Model



Complex Model

Let $f(x)$ be the true model & $\hat{f}(x)$ be our estimate of the model (simple or complex) then,



Green: Avg values of $\hat{f}(x)$ for simple model

Blue: Avg values of $\hat{f}(x)$ for complex model

Red: True model $f(x)$

Bias $\hat{f}(x) = E[\hat{f}(x)] - f(x)$

(difference b/w avg & true value)

	Simple Model	Complex Model
Bias	Very High	Very Low

Variance $(\hat{f}(x)) = E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)\right]^2$

(How much the different $\hat{f}(x)$'s differ from each other)

	Simple Model	Complex Model
Variance	Very Low	Very High

Training Error vs Testing Error

Consider a new point (x, y) which was not seen before during training.

If we use the model $\hat{f}(x)$ to predict the value of y then the mean square error is given by:

$$E[(y - \hat{f}(x))^2]$$

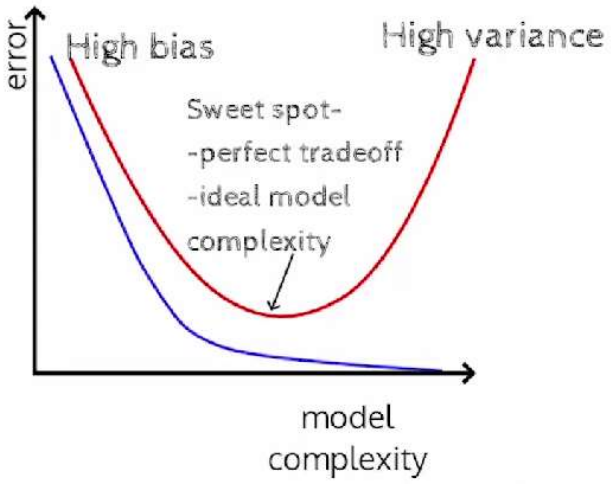
We can show that $E[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \sigma^2 \text{ (irreducible error)}$

The parameters of $\hat{f}(x)$ (all the w_i's) are trained using a training set $\{(x_i, y_i)\}_{i=1}^{n}$

At test time, we are interested in evaluating the model on validation (unseen) set which was not used for training.

Now we have two entities:

$\text{train}_{\text{error}}$ & $\text{test}_{\text{error}}$



Given: n training points,
m testing points.

$$\text{train}_{\text{err}} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

$$\text{test}_{\text{err}} = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{f}(x_i))^2$$

Estimate Errors from Test Data

Let $D = \{x_i, y_i\}_{i=1}^{n}$

We know that some true function exists s.t.

$$y = f(x) + \varepsilon$$

means that y_i is related to x_i by some function f but there is some noise ε in the relation

Let's assume $\varepsilon \sim N(0, \sigma^2)$

We will use $\hat{f}$ to approximate f & estimate the parameters of training data TCD.

$$y = \hat{f}(x)$$

We are interested in knowing

$$E[(\hat{f}(x) - f(x))^2]$$

but we cannot estimate this directly as we do not know f

$$E[(\hat{y} - y)^2] = E[(\hat{f}(x) - f(x) - \varepsilon)^2]$$

$$= E[(\hat{f}(x) - f(x))^2 - 2\varepsilon(\hat{f}(x) - f(x)) + \varepsilon^2]$$

$$= E[(\hat{f}(x) - f(x))^2] - 2E[\varepsilon(\hat{f}(x) - f(x))] + E[\varepsilon^2]$$

$$E[(\hat{f}(x) - f(x))^2] = E[(\hat{y} - y)^2] - E[\varepsilon^2] + 2E[\varepsilon(\hat{f}(x) - f(x))]$$

Case 1: Using Test Data

$$E[(\hat{f}(x) - f(x))^2] = \frac{1}{n+m}\sum^{n+m}(\hat{y}_i - y_i)^2 - \sigma^2 + 2E[\varepsilon(\hat{f}(x) - f(x))]$$

$$E\left[\left(\hat{f}(x) - f(x)\right)^2\right] = \frac{1}{m}\sum_{i=n+1}^{n+m}\left(\hat{y}_i - y_i\right)^2 - \sigma^2 + 2E\left[\varepsilon\left(\hat{f}(x) - f(x)\right)\right]$$

- $E\left[\left(\hat{f}(x) - f(x)\right)^2\right]$: true error
- $\frac{1}{m}\sum_{i=n+1}^{n+m}\left(\hat{y}_i - y_i\right)^2$: empirical estimation of error
- σ^2: small constant
- $2E\left[\varepsilon\left(\hat{f}(x) - f(x)\right)\right]$: covariance $(\varepsilon, (\hat{f}(x) - f(x)))$

$$\text{cov}(X,Y) = E\left[(X-\mu_X)(Y-\mu_Y)\right]$$

$$= E\left[X(Y-\mu_Y)\right] \quad \left[\text{if } \mu_X = E(X) = 0\right]$$

$$= E[XY] - E[X\mu_Y]$$

$$= E[XY] - \mu_Y E[X]$$

$$= E[XY]$$

Now $\varepsilon = y - f(x)$

This $y - f(x)$ is independent of $(\hat{f}(x) - f(x))$ as it is based on test data & the latter is train data.

$\therefore$ y is independent of $\hat{f}(x)$

$\therefore$ ε is independent of $(\hat{f}(x) - f(x))$

$\therefore$ covariance will be 0

$\therefore$ true error = empirical test error + small constant

Case 2: Using Training Data

$$E\left[\left(\hat{f}(x) - f(x)\right)^2\right] = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2 - \sigma^2 + 2E\left[\varepsilon\left(\hat{f}(x) - f(x)\right)\right]$$

- $E\left[\left(\hat{f}(x) - f(x)\right)^2\right]$: true error
- $\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2$: empirical estimation of error
- σ^2: small constant
- $2E\left[\varepsilon\left(\hat{f}(x) - f(x)\right)\right]$: covariance $(\varepsilon, (\hat{f}(x) - f(x)))$

true error

estimation of error

$$E\left[\varepsilon \cdot \left(\hat{f}(x) - f(x)\right)\right] \neq E[\varepsilon] \cdot E\left[\left(\hat{f}(x) - f(x)\right)\right] \neq 0$$

∴ empirical train error is smaller & does not give a true picture

True Error vs Model Complexity

According to Stein's lemma we can show that,

$$\frac{1}{n}\sum_{i=1}^{n}\varepsilon_i\left(\hat{f}(x_i)-f(x_i)\right)=\frac{\sigma^2}{n}\sum_{i=1}^{n}\frac{\partial\hat{f}(x_i)}{\partial y_i}$$

When will $\frac{\partial\hat{f}(x_i)}{\partial y_i}$ be high?

When a small change in the observation causes a large change in the estimation $(\hat{f})$

A complex model will be more sensitive to changes in observations whereas a simple model will be less sensitive

Hence we can say:

$$\text{true error} = \underbrace{\text{emperical train error}}_{} + \underbrace{\text{small constant}}_{} + \Omega \quad \leftarrow \text{model complexity}$$

Hence while training, instead of minimizing $L_{train}(\theta)$ we should minimize:

$$\min_{\text{wrt }\theta} \; L_{train}(\theta) + \Omega(\theta) = L(\theta)$$

$\Omega(\theta)$ acts as an approximate for $\frac{\sigma^2}{n}\sum_{i=1}^{n}\frac{\partial\hat{f}(x_i)}{\partial y_i}$

This is the basis of regularization methods.

This is the basis of regularization methods.

L2 Regularization

Regularization term → Proxy for model complexity.

For L2 regularization we have:

$$\bar{L}(w) = L(w) + \frac{\alpha}{2}\|w\|^2$$

For SGD : $\nabla\bar{L}(w) = \nabla L(w) + \alpha w$

Update rule: $w_{t+1} = w_t - \eta \nabla L(w_t) - \eta\alpha w_t$

Assume w^* is the optimal solution for $L(w)$ (not $\bar{L}(w)$)

i.e. solution in the absence of regularization (w^* optimal → $\nabla L(w^*) = 0$)

Consider $w = w^* + u$

$\therefore u = w - w^*$

$$L(w^* + u) = L(w^*) + u^T \nabla L(w^*) + \frac{1}{2} u^T H u$$

$$L(w) = L(w^*) + (w-w^*)^T \nabla L(w^*) + \frac{1}{2}(w-w^*)^T H (w-w^*)$$

$$= L(w^*) + \frac{1}{2}(w-w^*)^T H (w-w^*) \quad [\nabla L(w^*) = 0]$$

$$\nabla L(w) = \nabla L(w^*) + H(w-w^*) \quad [H \rightarrow \text{Hessian Matrix}]$$

$$= H(w-w^*)$$

Now,

$$\nabla\bar{L}(w) = \nabla L(w) + \alpha w$$

Let $\bar{w}$ be the optimal solution for $\bar{L}(w)$

$\therefore \nabla\bar{L}(\bar{w}) = 0$

Let $\bar{w}$ be the optimal solution for $\bar{L}$

$$\therefore \nabla \bar{L}(\bar{w}) = 0$$

$$H(\bar{w} - w^*) + \alpha \bar{w} = 0$$

$$\therefore (H + \alpha I)\bar{w} = Hw^*$$

(I: identity vector)

$$\therefore \bar{w} = (H + \alpha I)^{-1} H w^*$$

as $\alpha \to 0$, $\bar{w} \to w^*$ [no regularization]

Let us analyze when $\alpha \neq 0$

If H is a symmetric Positive Semi Definite Matrix,

$$H = Q \Lambda Q^T \quad [Q \text{ is orthogonal, } QQ^T = Q^TQ = I]$$

$$\therefore \bar{w} = (H + \alpha I)^{-1} H w^*$$

$$= (Q \Lambda Q^T + \alpha I)^{-1} Q \Lambda Q^T w^*$$

$$= (Q \Lambda Q^T + \alpha Q I Q^T)^{-1} Q \Lambda Q^T w^*$$

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

$$= Q^{T^{-1}} (\Lambda + \alpha I)^{-1} Q^{-1} Q \Lambda Q^T w^*$$

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

when $\alpha = 0$, w^* becomes the optimal sol^n as

$$\therefore \bar{w} = (QDQ^T) w^* \quad \left[D = (\Lambda + \alpha I)^{-1}\Lambda \text{ , a diagonal matrix}\right]$$

w^* gets rotated first by Q^T to give $Q^T w^*$ [$Q \to n \times n$]

w^* gets rotated first by Q^T to give $Q^T w^*$

If $\alpha = 0$, Q rotates $Q^T w^*$ back to give w^*

└ matrix

$Q \to n \times n$
$w^* \to n * 1$
$D \to n \times n$

If $\alpha \neq 0$, $(\Lambda + \alpha I)^{-1} = \begin{bmatrix} \frac{1}{\lambda_1+\alpha} & & \\ & \frac{1}{\lambda_2+\alpha} & \\ & & \ddots \\ & & & \frac{1}{\lambda_n+\alpha} \end{bmatrix}$

Each element i of $Q^T w^*$ gets scaled by $\frac{\lambda_i}{\lambda_i+\alpha}$ before it is rotated back by Q

$$D = (\Lambda + \alpha I)^{-1}\Lambda = \begin{bmatrix} \frac{\lambda_1}{\lambda_1+\alpha} & & \\ & \frac{\lambda_2}{\lambda_2+\alpha} & \\ & & \ddots \\ & & & \frac{\lambda_n}{\lambda_n+\alpha} \end{bmatrix}$$

If $\lambda_i \gg \alpha$ then, $\frac{\lambda_i}{\lambda_i+\alpha} = 1$

If $\lambda_i \ll \alpha$ then, $\frac{\lambda_i}{\lambda_i+\alpha} = 0$

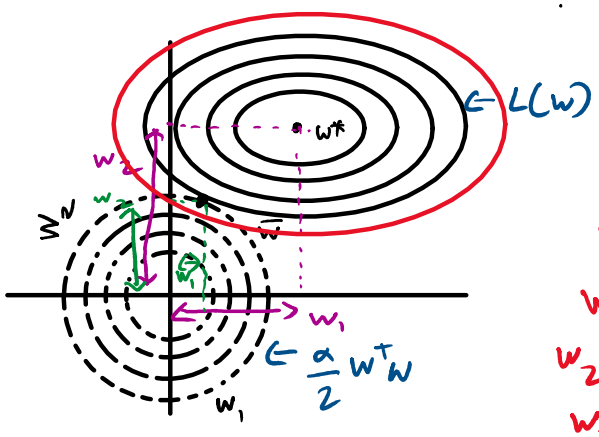
∴ only significant directions will be retained,

$$\text{Effective Parameters} = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i+\alpha} < n$$

& complexity has reduced!

Loss Contours

$\bar{w}$ → optimum of $\tilde{L}(w)$ $= L(\theta) + \Omega(w)$



The weights have shrunk in presence of regularization.

w_1 has decreased more than w_2 coz it is more imp than w_2

$= L(\theta) + \Omega(w)$

w_2 coz it is more imp than w_2

Even the loss we want to be small.

If w_2 shrunk loss would be higher → see red contour

Dataset Augmentation

Given an image of Handwritten digit, we can rotate it by n degrees k times, shift positions, blur it & also change pixels. This will still not change the label.

∴ We exploit the fact that transformations to the image do not change the image & create more training data. This is called data augmentation. Data created using knowledge of the task.

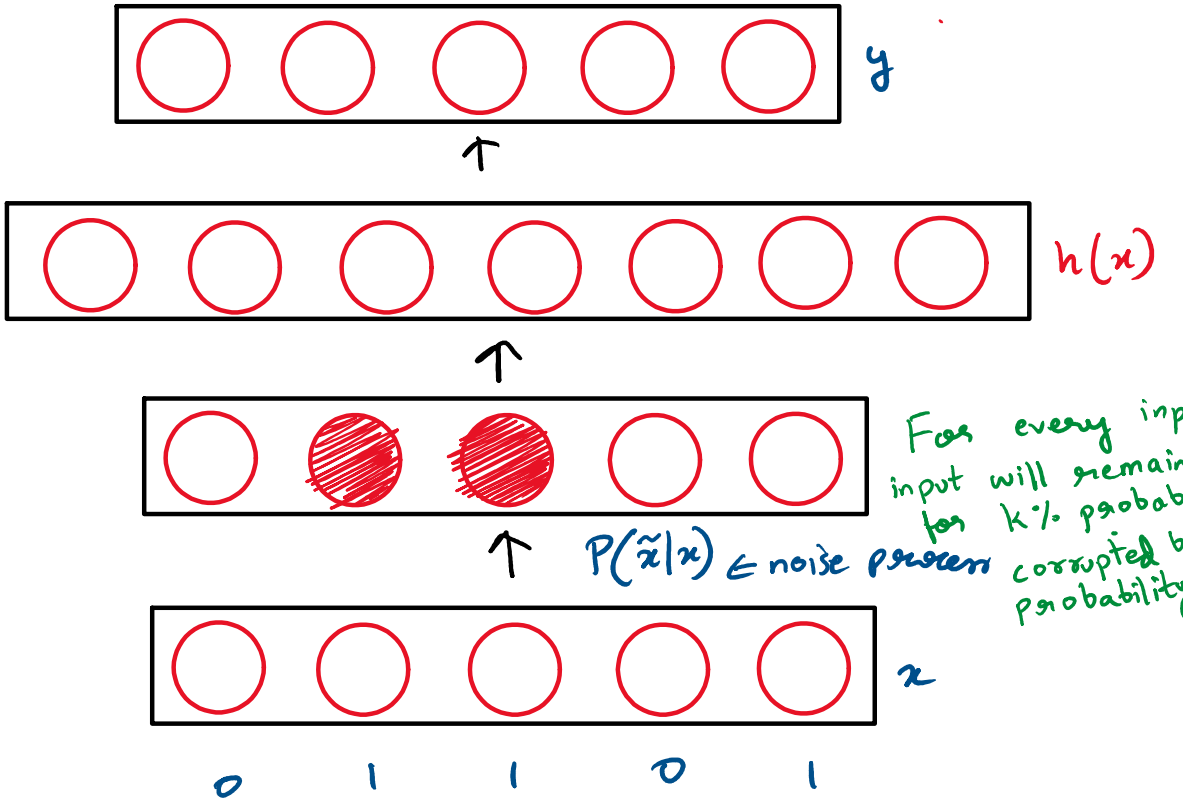
Parameter Sharing

Used in CNNs.

Same filter is applied to different positions of the image.

Or same weight matrix acts on different input neurons.

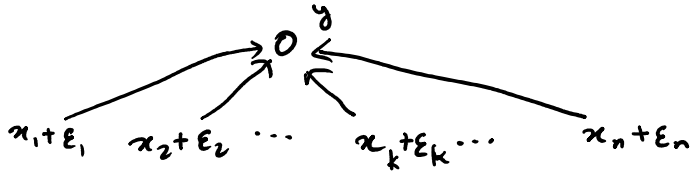
Injecting Noise At Inputs



For every input, the input will remain as it is for k% probability & be corrupted by (1-k)% probability

For every epoch, the corruption happens differently.

$\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_n$



$\varepsilon \sim N(0, \sigma^2)$

$\tilde{x}_i = x_i + \varepsilon_i$

$$\tilde{y} = \sum_{i=1}^{n} w_i \tilde{x}_i$$

$$= \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} w_i \varepsilon_i$$

$$\boxed{\tilde{y} = \hat{y} + \sum_{i=1}^{n} w_i \varepsilon_i}$$

Now, $E\left[(\tilde{y} - y)^2\right] = E\left[\left(\hat{y} + \sum_{i=1}^{n} w_i \varepsilon_i - y\right)^2\right]$

Now, $E\left[(\tilde{y}-y)^2\right] = E\left[\left(\hat{y} + \sum_{i=1}^{n} \tilde{w}_i \varepsilon_i - y\right)^2\right]$

$= E\left[(\hat{y}-y)^2 + \left(\sum_{i=1}^{n} x_i \varepsilon_i\right)^2\right]$

$= E\left[(\hat{y}-y)^2\right] + E\left[\left(\sum_{i=1}^{n} x_i \varepsilon_i\right)^2\right]$

$= E\left[(\hat{y}-y)^2\right] + E\left[2(\hat{y}-y)\sum_{i=1}^{n} \tilde{w}_i \varepsilon_i\right] + E\left[\left(\sum_{i=1}^{n} \tilde{w}_i \varepsilon_i\right)^2\right]$

$= E\left[(\hat{y}-y)^2\right] + 0 + E\left[\sum_{i=1}^{n} \tilde{w}_i^2 \varepsilon_i^2\right]$ [ε_i is independent of ε_j & ε_i is independent of $(\hat{y}-y)$]

$= E\left[(\hat{y}-y)^2\right] + \sigma^2 \sum_{i=1}^{n} \tilde{w}_i^2$ [$E[\varepsilon_i^2] = \sigma^2$]

∴ In a simple output network, w/o any non linearity, adding noise to the inputs is the same as using weight decay! aka L2 norm penalty.

Adding Noise to the Output

For a given classification:

y

0	1	2	3	4	5	6	7	8	9	
0	0	1	0	0	0	0	0	0	0	Hard Targets

If we try to minimize the cross entropy loss:

minimize: $\sum_{i=0}^{9} p_i \log q_i$

p → true label

q → predicted label

true distribution → p: 0 0 1 0 0 0 0 0 0 0

estimated distribution → q

Intuition

Don't trust the label as it is noisy

Instead, add soft targets

Taking a small mass from probability & adding it to all other labels & they'll be my soft targets

$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$1-\varepsilon$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	Soft Targets

ε → small positive constant

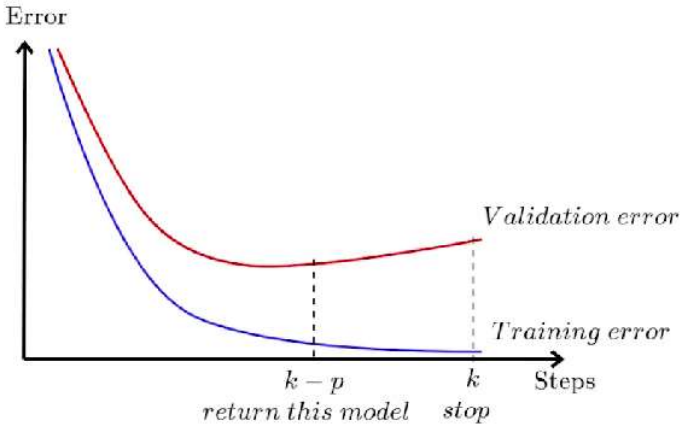
minimize: $\sum_{i=1}^{9} p_i \log q_i$

true distribution + noise: $p = [\frac{\varepsilon}{9} \quad \frac{\varepsilon}{9} \quad 1-\varepsilon \quad \dots \quad \frac{\varepsilon}{9}]$

true distribution + noise: $p = \left\{ \frac{\epsilon}{q}, \frac{\epsilon}{q}, 1-\epsilon, \cdots \frac{\epsilon}{q} \right\}$

estimated distribution: q

Early Stopping



We want to find the sweet spot where training error is low & validation error is also as low as possible.

We start by tracking the validation error.

Have a patience parameter p

After p consecutive steps, if validation error doesn't decrease & continues to increase (remains monotic) then we'll have to stop training.

Suppose we are at step k & for the last p steps, validation error doesn't decrease, we'll throw away the last p updates & retain the model at $(k-p)$ steps!

This is one of the most widely used form of regularization.

Can also be used w/ other regularizers (l_2)

How does this act as a regularizer?

The update rule of SGD: $w_{t+1} = w_t - \eta \nabla w_t$

$$= w_0 - \eta \sum_{i=1}^{t} \nabla w_i$$

Let c be the maximum value of ∇w_i then,

Let z be the maximum value of ∇w_i then,

$$|w_{t+1} - w_0| \leq \eta t |z|$$

t is controlling how far w_t can go from w_0

According to Taylor Series:

$$L(w) = L(w^*) + (w-w^*)^T \nabla L(w^*) + \frac{1}{2}(w-w^*)^T H (w-w^*)$$

$$= L(w^*) + \frac{1}{2}(w-w^*)^T H (w-w^*)$$

[w^* is optimal $\therefore \nabla L(w^*) = 0$]

Taking derivative on both sides

$$\nabla L(w^*) = H(w-w^*)$$

The SGD update rule is:

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$= w_t - \eta H(w_t - w^*)$$

$$w_{t+1} = (I - \eta H) w_t + \eta H w^*$$

Using Eigen Value Decomposition of H as $H = Q \Lambda Q^T$,

$$w_{t+1} = (I - \eta Q \Lambda Q^T) w_t + \eta Q \Lambda Q^T w^*$$

If we start with $w_0 = 0$ then we can show,

$$w_{t+1} = Q\left[I - (I - \varepsilon \Lambda)^t\right] Q^T w^*$$

If we start with $w_0 = 0$ then we ...

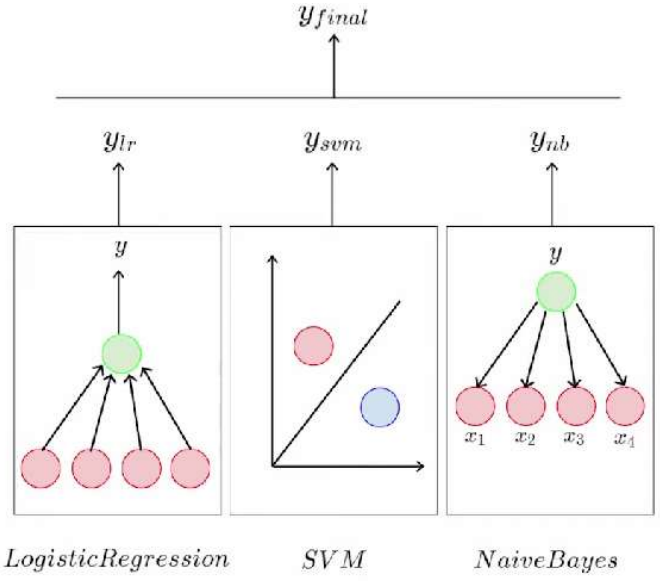
$$w_{t+1} = Q\left[I - (I - \varepsilon\Lambda)^t\right]Q^T w^*$$

We observe that $w_t = \tilde{w}$ if we choose ε, t & α such that,

$$(I - \varepsilon\Lambda)^t = (\Lambda + \alpha I)^{-1}\alpha$$

Ensemble Methods

Idea: Combine the output of different models to reduce generalization error. The models can correspond to different classifiers.



It could also be different instances of the same classifier, trained with:

- different hyperparameters
- different features
- different samples of training data

This is called Bagging.

Bagging: form an ensemble using different forms of the same classifier.

From a given dataset, construct multiple training sets by sampling with replacement $(T_1, T_2, \cdots, T_k)$

Training i^{th} instance of the classifier using training set T_i

When would bagging work?

Consider a set of k Logistic Regression models.

Suppose that each model makes an error ϵ_i on a test example.

example.

Let ε_i be drawn from a zero mean multivariate normal distribution. $\varepsilon_i \sim N(0, \sigma^2)$

Variance $= E[\varepsilon_i^2] = V$

Covariance $= E[\varepsilon_i, \varepsilon_j] = C$

The error made by the average prediction of all the models is $\frac{1}{k}\sum_i \varepsilon_i$ ← from one sample

The expected savvare error: $mse = E\left[\left(\frac{1}{k}\sum_i \varepsilon_i\right)^2\right]$ ↙ from all samples

$$= \frac{1}{k^2} E\left[\sum_i \sum_{i=j} \varepsilon_i \varepsilon_j + \sum_i \sum_{i \neq j} \varepsilon_i \varepsilon_j\right]$$

$$= \frac{1}{k^2} E\left[\sum_i \varepsilon_i^2 + \sum_i \sum_{i \neq j} \varepsilon_i \varepsilon_j\right]$$

$$= \frac{1}{k^2}\left(\sum_i E[\varepsilon_i^2] + \sum_i \sum_{i \neq j} E[\varepsilon_i \varepsilon_j]\right)$$

↑ variance ↑ covariance

$$= \frac{1}{k^2}\left(kV + k(k-1)C\right)$$

$$mse = \frac{1}{k}V + \frac{k-1}{k}C$$

When would bagging work?

If the errors of the model are perfectly correlated, then $V = C$ & $mse = V$ which is the same as the mse

then $V = C$ & $mse = v$ which is the same as the mse of any of the model.

If the errors of the model are independent or uncorrelated then $c = 0$ & the $mse = \frac{1}{k} V$

On average, the ensemble will perform at least as well as it's individual members

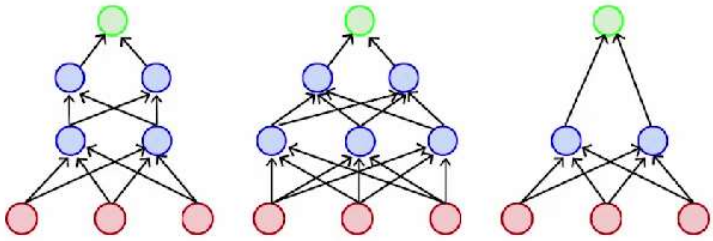
Dropout

Typically, bagging always helps.

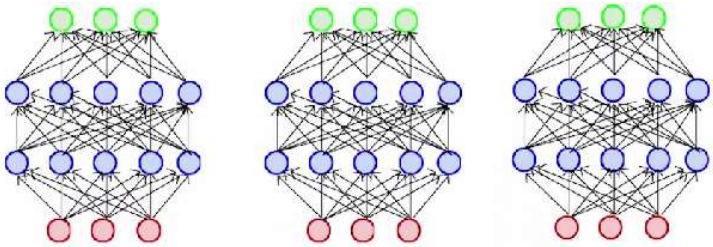
However in the context of deep learning, training several large networks for making an ensemble is expensive.

We have two options:

Option 1: Train several neural networks having different architectures (very expensive)



Option 2: Train multiple instances of the same network, using different training samples (again, expensive)



The solution is: Dropout

It refers to dropping out some units (neurons).

Temporarily remove a node & all its incoming & outgoing connections resulting in a thinned network

connections resulting in a thinned network

Each node is retained with a fixed probability ($p=0.5$) for hidden & $p=0.8$ for visible nodes

For a network of n nodes, each node is retained or dropped resulting in a new architecture.

We can construct 2^n thinned networks from a given neural network

How do we train so many networks?

Trick (1): Share the weights across the networks

Trick (2): Sample a different network for each training instance

Steps

- Initialize all parameters & start training.
- Apply dropout for first training istance (mini batch)
- Compute loss & backpropagate

Which Parameters to update?

- Only the ones who participated !

- Only the ones who participated !

In the second instance (or mini batch), apply dropout again

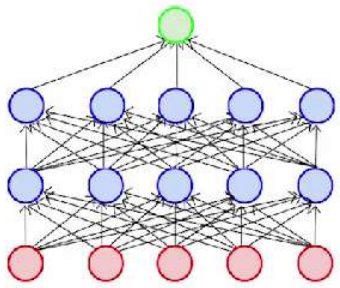
Again compute the loss & backpropagate to the active weights

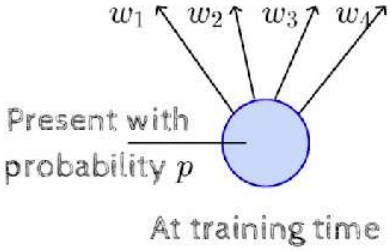
If a particular weight was active for both training instances then it would be updated twice.

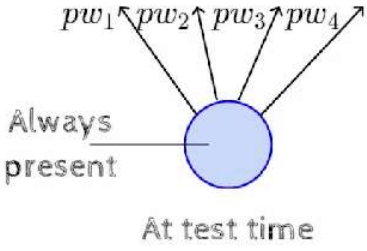
If a particular weight was active for one training instances then it would be updated only once.

What to do during testing ?

If a node was present only with probability p, scale the output by p times.







Dropout, essentially applies a masking noise to the hidden units

It also prevents hidden units from "coadopting"

It also prevents hidden units from coadopting"

↑ relying on other nodes

Suppose h_i learns to detect a face by firing on detecting a nose

Dropping h_i then corresponds to erasing the information that a nose exists

The model should then learn another h_i which redundantly encodes the presence of a nose

Or the model should learn to detect the face using other features

Summary

Grouping Regularization Techniques

$$\mathscr{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} l\big(f(x_i,\theta),y\big)$$

Data x_i

1. Data Augmentation
2. Noise Injection

Architecture choice $f(\cdot)$

1. Dropout
2. Skip connections (CNN)
3. Weight sharing
4. Pooling

Optimizer ∇

1. SGD
2. Early stopping

Penalize cost $\mathscr{L}(\cdot)$

1. L_1
2. L_2

Grouping Regularization Techniques

$$\mathscr{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} l\big(f(x_i,\theta),y\big)$$

Explicit

L_1, L_2

Data Augmentation

Noise Injection

Skip connections (CNN)

Weight sharing

Pooling

Dropout

Implicit

SGD

Large initial learning rate

Small initial learning rate

Early stopping