# CS7015 (Deep Learning) : Lecture 8

## Regularization: Bias Variance Tradeoff, l2 regularization, Early stopping, Dataset augmentation, Parameter sharing and tying, Injecting noise at input, Ensemble methods, Dropout

Mitesh M. Khapra

Department of Computer Science and Engineering
Indian Institute of Technology Madras

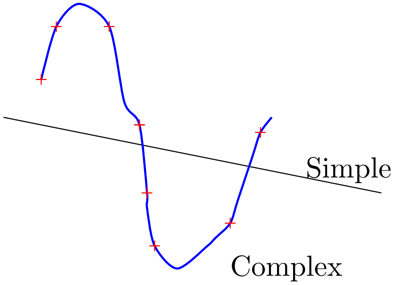
## Acknowledgements

- Chapter 7, Deep Learning book
- Ali Ghodsi's Video Lectures on Regularization[a]
- Dropout: A Simple Way to Prevent Neural Networks from Overfitting[b]

[a] Lecture 2.1 and Lecture 2.2
[b] Dropout

# Module 8.1 : Bias and Variance

We will begin with a quick overview of bias, variance and the trade-off between them.



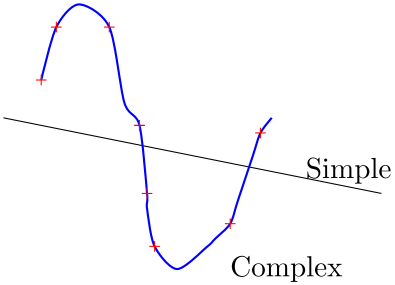

The points were drawn from a sinusoidal function (the true $f(x)$)

- Let us consider the problem of fitting a curve through a given set of points
- We consider two models :

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

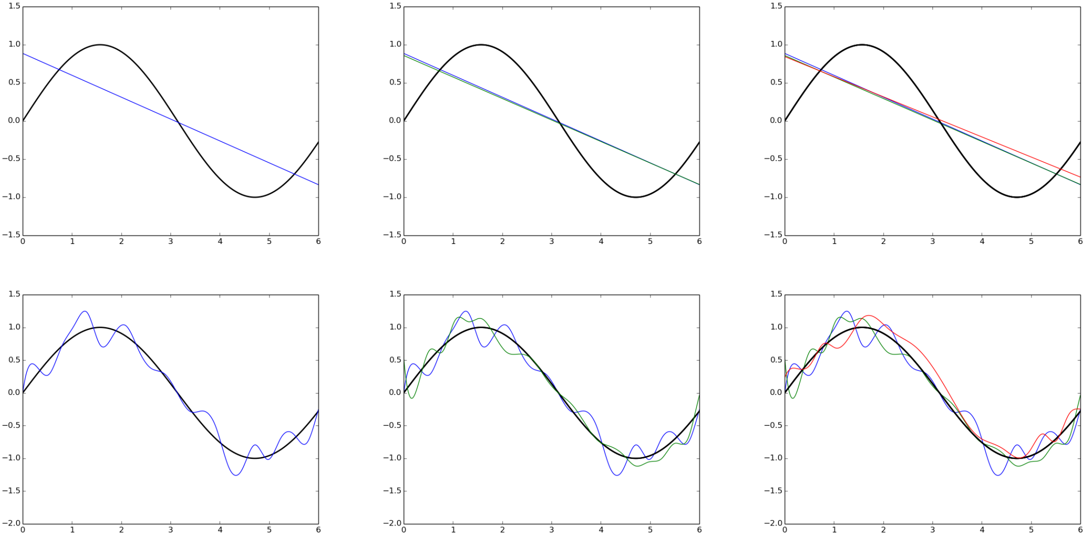
$$\underset{(degree:25)}{Complex} \quad y = \hat{f}(x) = \sum_{i=1}^{25} w_i x^i + w_0$$

- Note that in both cases we are making an assumption about how $y$ is related to $x$. We have no idea about the true relation $f(x)$
- The training data consists of 100 points

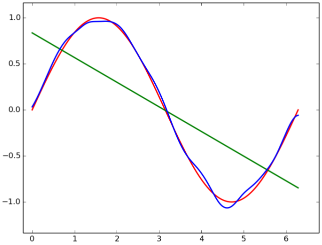



The points were drawn from a sinusoidal function (the true $f(x)$)

- We sample 25 points from the training data and train a simple and a complex model
- We repeat the process '$k$' times to train multiple models (each model sees a different sample of the training data)
- We make a few observations from these plots


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
1
2
3
4
5
6

- Simple models trained on different samples of the data do not differ much from each other
- However they are very far from the true sinusoidal curve (under fitting)
- On the other hand, complex models trained on different samples of the data are very different from each other (high variance)

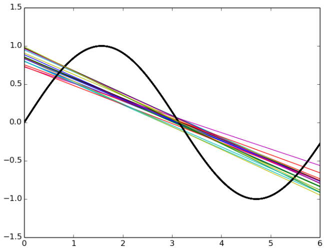


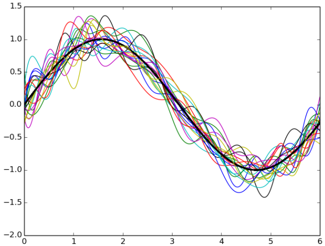

Green Line: Average value of $\hat{f}(x)$ for the simple model
Blue Curve: Average value of $\hat{f}(x)$ for the complex model
Red Curve: True model ($f$(x))

- Let $f(x)$ be the true model (sinusoidal in this case) and $\hat{f}(x)$ be our estimate of the model (simple or complex, in this case) then,

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

- $E[\hat{f}(x)]$ is the average (or expected) value of the model
- We can see that for the simple model the average value (green line) is very far from the true value $f(x)$ (sinusoidal function)
- Mathematically, this means that the simple model has a high bias
- On the other hand, the complex model has a low bias



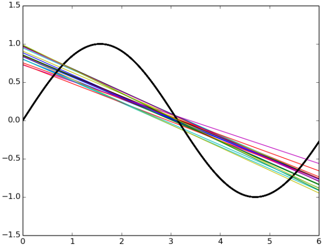

- We now define,

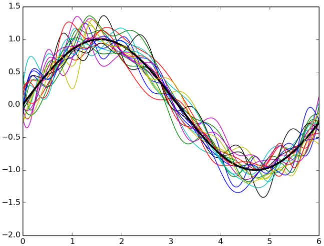
$$\text{Variance } (\hat{f}(x)) = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

(Standard definition from statistics)

- Roughly speaking it tells us how much the different $\hat{f}(x)$'s (trained on different samples of the data) differ from each other

- It is clear that the simple model has a low variance whereas the complex model has a high variance




- In summary (informally)
- Simple model: high bias, low variance
- Complex model: low bias, high variance
- There is always a trade-off between the bias and variance
- Both bias and variance contribute to the mean square error. Let us see how

# Module 8.2 : Train error vs Test error

- Consider a new point $(x, y)$ which was not seen during training
- If we use the model $\hat{f}(x)$ to predict the value of $y$ then the mean square error is given by

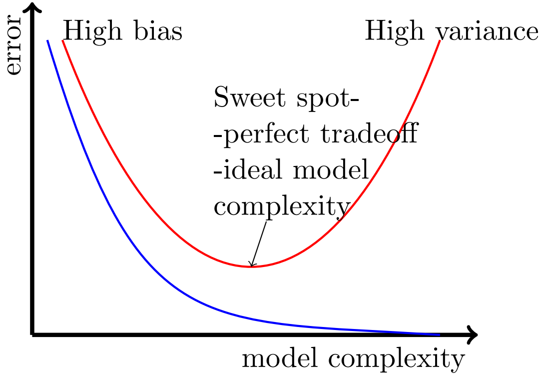
$$E[(y - \hat{f}(x))^2]$$

(average square error in predicting $y$ for many such unseen points)

- We can show that

$$\begin{aligned} E[(y - \hat{f}(x))^2] &= Bias^2 \\ &+ Variance \\ &+ \sigma^2 \text{ (irreducible error)} \end{aligned}$$

- See proof here




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

- The parameters of $\hat{f}(x)$ (all $w_i$'s) are trained using a training set $\{(x_i, y_i)\}_{i=1}^n$
- However, at test time we are interested in evaluating the model on a validation (unseen) set which was not used for training
- This gives rise to the following two entities of interest:
  $train_{err}$ (say, mean square error)
  $test_{err}$ (say, mean square error)
- Typically these errors exhibit the trend shown in the adjacent figure

## Intuitions developed so far

- Let there be $n$ training points and $m$ test (validation) points

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

$$test_{err} = \frac{1}{m}\sum_{i=n+1}^{n+m}(y_i - \hat{f}(x_i))^2$$

- As the model complexity increases $train_{err}$ becomes overly optimistic and gives us a wrong picture of how close $\hat{f}$ is to $f$
- The validation error gives the real picture of how close $\hat{f}$ is to $f$
- We will concretize this intuition mathematically now and eventually show how to account for the optimism in the training error

- Let D=$\{x_i, y_i\}_{i=1}^{m+n}$, then for any point $(x, y)$ we have,

$$y_i = f(x_i) + \varepsilon_i$$

- which means that $y_i$ is related to $x_i$ by some true function $f$ but there is also some noise $\varepsilon$ in the relation
- For simplicity, we assume

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

and of course we do not know $f$

- Further we use $\hat{f}$ to approximate $f$ and estimate the parameters using T $\subset$ D such that

$$y_i = \hat{f}(x_i)$$

- We are interested in knowing

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

but we cannot estimate this directly because we do not know $f$

- We will see how to estimate this empirically using the observation $y_i$ & prediction $\hat{y}_i$

$$
\begin{aligned}
E[(\hat{y}_i - y_i)^2] &= E[(\hat{f}(x_i) - f(x_i) - \varepsilon_i)^2] \quad (y_i = f(x_i) + \varepsilon_i) \\
&= E[(\hat{f}(x_i) - f(x_i))^2 - 2\varepsilon_i(\hat{f}(x_i) - f(x_i)) + \varepsilon_i^2] \\
&= E[(\hat{f}(x_i) - f(x_i))^2] - 2E[\varepsilon_i(\hat{f}(x_i) - f(x_i))] + E[\varepsilon_i^2]
\end{aligned}
$$

$$
\therefore E[(\hat{f}(x_i) - f(x_i))^2] = E[(\hat{y}_i - y_i)^2] - E[\varepsilon_i^2] + 2E[\, \varepsilon_i(\hat{f}(x_i) - f(x_i)) \,]
$$

We will take a small detour to understand how to empirically estimate an Expectation and then return to our derivation

- Suppose we have observed the goals scored($z$) in $k$ matches as $z_1 = 2,\ z_2 = 1,\ z_3 = 0,\ \ldots\ z_k = 2$

- Now we can empirically estimate $E[z]$ i.e. the expected number of goals scored as

$$E[z] = \frac{1}{k}\sum_{i=1}^{k} z_i$$

- Analogy with our derivation: We have a certain number of observations $y_i$ & predictions $\hat{y}_i$ using which we can estimate

$$E[(\hat{y}_i - y_i)^2] = \frac{1}{m}\sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

... returning back to our derivation

$$E[(\hat{f}(x_i) - f(x_i))^2] = E[(\hat{y}_i - y_i)^2] - E[\varepsilon_i^2] + 2E[\ \varepsilon_i(\hat{f}(x_i) - f(x_i))\ ]$$

- We can empirically evaluate R.H.S using training observations or test observations

**Case 1**: Using test observations

$$\underbrace{E[(\hat{f}(x_i) - f(x_i))^2]}_{true\, error} = \underbrace{\frac{1}{m}\sum_{i=n+1}^{n+m}(\hat{y}_i - y_i)^2}_{empirical\, estimation\, of\, error} - \underbrace{\frac{1}{m}\sum_{i=n+1}^{n+m}\varepsilon_i^2}_{small\, constant} + 2\ \underbrace{E[\ \varepsilon_i(\hat{f}(x_i) - f(x_i))\ ]}_{=\, covariance\, (\varepsilon_i, \hat{f}(x_i) - f(x_i))}$$

$$\begin{aligned} \because \text{covariance}(X, Y) &= E[(X - \mu_X)(Y - \mu_Y)] \\ &= E[(X)(Y - \mu_Y)] (\text{if } \mu_X = E[X] = 0) \\ &= E[XY] - E[X\mu_Y] = E[XY] - \mu_Y E[X] = E[XY] \end{aligned}$$

$$\underbrace{E[(\hat{f}(x_i) - f(x_i))^2]}_{true\ error}$$

$$= \underbrace{\frac{1}{m}\sum_{i=n+1}^{n+m}(\hat{y}_i - y_i)^2}_{empirical\ estimation\ of\ error} - \underbrace{\frac{1}{m}\sum_{i=n+1}^{n+m}\varepsilon_i^2}_{small\ constant} + 2\ \underbrace{E[\ \varepsilon_i(\hat{f}(x_i) - f(x_i))\ ]}_{=covariance\,(\varepsilon_i,\hat{f}(x_i)-f(x_i))}$$

- None of the test observations participated in the estimation of $\hat{f}(x)$[the parameters of $\hat{f}$(x) were estimated only using training data]

$$\therefore \varepsilon \perp (\hat{f}(x_i) - f(x_i))$$

$$\therefore E[\varepsilon_i \cdot (\hat{f}(x_i) - f(x_i))] = E[\varepsilon_i] \cdot E[\hat{f}(x_i) - f(x_i))] = 0 \cdot E[\hat{f}(x_i) - f(x_i))] = 0$$

$$\therefore \text{true error} = \text{empirical test error} + \text{small constant}$$

- Hence, we should always use a validation set(independent of the training set) to estimate the error

**Case 2**: Using training observations

$$\underbrace{E[(\hat{f}(x_i) - f(x_i))^2]}_{true\ error}$$

$$= \underbrace{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}_{empirical\ estimation\ of\ error} - \underbrace{\frac{1}{n}\sum_{i=1}^{n}\varepsilon_i^2}_{small\ constant} + 2\ \underbrace{E[\ \varepsilon_i(\hat{f}(x_i) - f(x_i))\ ]}_{=\ covariance\ (\varepsilon_i, \hat{f}(x_i) - f(x_i))}$$

Now, $\varepsilon \not\perp \hat{f}(\mathrm{x})$ because $\varepsilon$ was used for estimating the parameters of $\hat{f}(x)$

$$\therefore E[\varepsilon_i \cdot (\hat{f}(x_i) - f(x_i))] \neq E[\varepsilon_i] \cdot E[\hat{f}(x_i) - f(x_i))] \neq 0$$

Hence, the empirical train error is smaller than the true error and does not give a true picture of the error

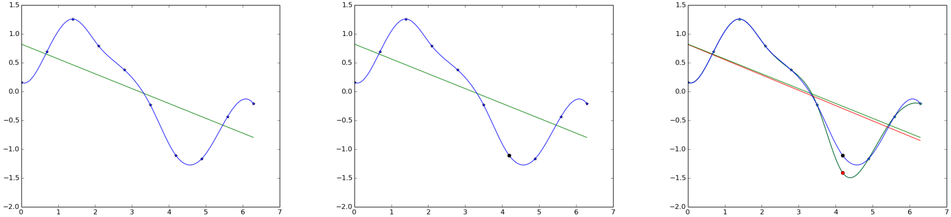
But how is this related to model complexity? Let us see

# Module 8.3 : True error and Model complexity

Using Stein's Lemma (and some trickery) we can show that

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

- 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($\hat{f}$)
- Can you link this to model complexity?
- Yes, indeed a complex model will be more sensitive to changes in observations whereas a simple model will be less sensitive to changes in observations
- Hence, we can say that
  true error = empirical train error + small constant + $\Omega$(model complexity)

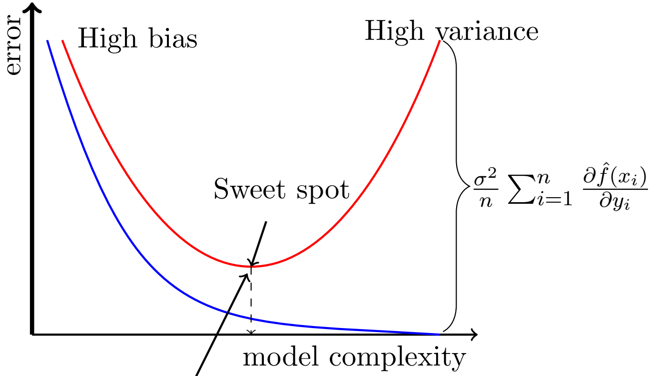

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
1
2
3
4
5
6
7

- Let us verify that indeed a complex model is more sensitive to minor changes in the data
- We have fitted a simple and complex model for some given data
- We now change one of these data points
- The simple model does not change much as compared to the complex model

- Hence while training, instead of minimizing the training error $\mathscr{L}_{train}(\theta)$ we should minimize

$$\min_{w.r.t\ \theta} \mathscr{L}_{train}(\theta) + \Omega(\theta) = \mathscr{L}(\theta)$$

- Where $\Omega(\theta)$ would be high for complex models and small for simple models
- $\Omega(\theta)$ 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 for all regularization methods
- We can show that $l_1$ regularization, $l_2$ regularization, early stopping and injecting noise in input are all instances of this form of regularization.




$\Omega(\theta)$ should ensure that model has reasonable complexity

- Why do we care about this bias variance tradeoff and model complexity?

- Deep Neural networks are highly complex models.
- Many parameters, many non-linearities.
- It is easy for them to overfit and drive training error to 0.
- Hence we need some form of regularization.

## Different forms of regularization

- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout

# Module 8.4 : $l_2$ regularization

## Different forms of regularization

- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout

- For $l_2$ regularization we have,

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

- For SGD (or its variants), we are interested in

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

- Update rule:

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

- Requires a very small modification to the code
- Let us see the geometric interpretation of this

- Assume $w^*$ is the optimal solution for $\mathscr{L}(w)$ [not $\widetilde{\mathscr{L}}(w)$] i.e. the solution in the absence of regularization ($w^*$ optimal $\rightarrow \nabla \mathscr{L}(w^*) = 0$)
- Consider $u = w - w^*$. Using Taylor series approximation (upto $2^{nd}$ order)

$$
\begin{aligned}
\mathscr{L}(w^* + u) &= \mathscr{L}(w^*) + u^T \nabla \mathscr{L}(w^*) + \frac{1}{2} u^T H u \\
\mathscr{L}(w) &= \mathscr{L}(w^*) + (w - w^*)^T \nabla \mathscr{L}(w^*) + \frac{1}{2}(w - w^*)^T H (w - w^*) \\
&= \mathscr{L}(w^*) + \frac{1}{2}(w - w^*)^T H (w - w^*) \qquad (\because \nabla L(w^*) = 0\ ) \\
\nabla \mathscr{L}(w) &= \nabla \mathscr{L}(w^*) + H(w - w^*) \\
&= H(w - w^*)
\end{aligned}
$$

- Now,

$$
\begin{aligned}
\nabla \widetilde{\mathscr{L}}(w) &= \nabla \mathscr{L}(w) + \alpha w \\
&= H(w - w^*) + \alpha w
\end{aligned}
$$

- Let $\widetilde{w}$ be the optimal solution for $\widetilde{L}(w)$ [i.e regularized loss]

$$\because \nabla \widetilde{L}(\widetilde{w}) = 0$$

$$H(\widetilde{w} - w^*) + \alpha\widetilde{w} = 0$$
$$\therefore (H + \alpha\mathbb{I})\widetilde{w} = Hw^*$$
$$\therefore \widetilde{w} = (H + \alpha\mathbb{I})^{-1}Hw^*$$

- Notice that if $\alpha \to 0$ then $\widetilde{w} \to w^*$ [no regularization]
- But we are interested in the case when $\alpha \neq 0$
- Let us analyse the case when $\alpha \neq 0$

- If H is symmetric Positive Semi Definite

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

$$\begin{aligned}
\widetilde{w} &= (H + \alpha\mathbb{I})^{-1}Hw^* \\
&= (Q\Lambda Q^T + \alpha\mathbb{I})^{-1}Q\Lambda Q^T w^* \\
&= (Q\Lambda Q^T + \alpha Q\mathbb{I}Q^T)^{-1}Q\Lambda Q^T w^* \\
&= [Q(\Lambda + \alpha\mathbb{I})Q^T]^{-1}Q\Lambda Q^T w^* \\
&= Q^{T^{-1}}(\Lambda + \alpha\mathbb{I})^{-1}Q^{-1}Q\Lambda Q^T w^* \\
&= Q(\Lambda + \alpha\mathbb{I})^{-1}\Lambda Q^T w^* \qquad (\because Q^{T^{-1}} = Q) \\
\widetilde{w} &= QDQ^T w^*
\end{aligned}$$

where $D = (\Lambda + \alpha\mathbb{I})^{-1}\Lambda$, is a diagonal matrix which we will see in more detail soon

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

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

$$D = (\Lambda + \alpha\mathbb{I})^{-1}\Lambda$$

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

- So what is happening here?
- $w^*$ first gets rotated by $Q^T$ to give $Q^T w^*$
- However if $\alpha = 0$ then $Q$ rotates $Q^T w^*$ back to give $w^*$
- If $\alpha \neq 0$ then let us see what $D$ looks like
- So what is happening now?

$$\begin{aligned}\widetilde{w} &= Q(\Lambda + \alpha\mathbb{I})^{-1}\Lambda Q^T w^* \\ &= QDQ^T w^*\end{aligned}$$

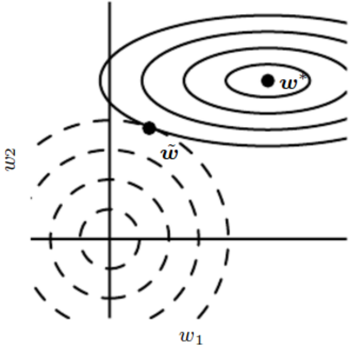
$$(\Lambda + \alpha\mathbb{I})^{-1} = \begin{bmatrix} \frac{1}{\lambda_1+\alpha} & & & \\ & \frac{1}{\lambda_2+\alpha} & & \\ & & \ddots & \\ & & & \frac{1}{\lambda_n+\alpha} \end{bmatrix}$$

$$D = (\Lambda + \alpha\mathbb{I})^{-1}\Lambda$$

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

- Each element $i$ of $Q^T w^*$ gets scaled by $\frac{\lambda_i}{\lambda_i+\alpha}$ before it is rotated back by $Q$
- if $\lambda_i >> \alpha$ then $\frac{\lambda_i}{\lambda_i+\alpha} = 1$
- if $\lambda_i << \alpha$ then $\frac{\lambda_i}{\lambda_i+\alpha} = 0$
- Thus only significant directions (larger eigen values) will be retained.

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

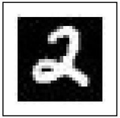



- The weight vector($w^*$) is getting rotated to ($\tilde{w}$)
- All of its elements are shrinking but some are shrinking more than the others
- This ensures that only important features are given high weights

# Module 8.5 : Dataset augmentation

## Different forms of regularization

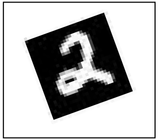
- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout



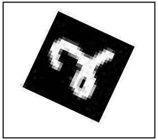
label = 2

[given training data]

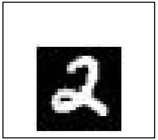
We exploit the fact that certain transformations to the image do not change the label of the image.



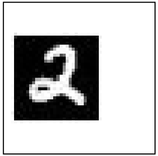
rotated by 20°



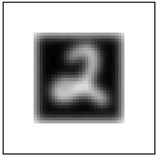
rotated by 65°



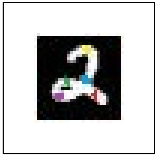
shifted vertically



shifted horizontally



blurred



changed some pixels

label = 2

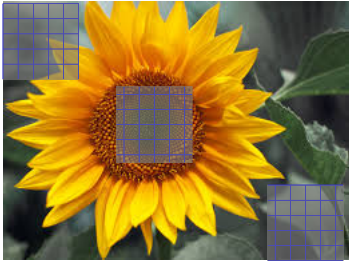
[augmented data = created using some knowledge of the task]

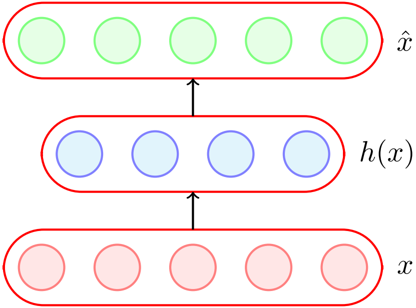
- Typically, More data = better learning
- Works well for image classification / object recognition tasks
- Also shown to work well for speech
- For some tasks it may not be clear how to generate such data

# Module 8.6 : Parameter Sharing and tying

## Other forms of regularization

- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout






## Parameter Sharing

- Used in CNNs
- Same filter applied at different positions of the image
- Or same weight matrix acts on different input neurons

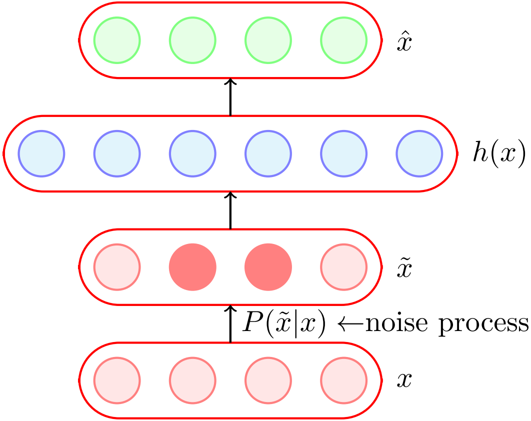
## Parameter Tying

- Typically used in autoencoders
- The encoder and decoder weights are tied.

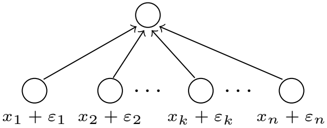
# Module 8.7 : Adding Noise to the inputs

## Other forms of regularization

- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout




- We saw this in Autoencoder
- We can show that for a simple input output neural network, adding Gaussian noise to the input is equivalent to weight decay ($L_2$ regularisation)
- Can be viewed as data augmentation




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

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

$$\widehat{y} = \sum_{i=1}^{n} w_i x_i$$

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

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

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

We are interested in $E[(\widetilde{y} - y)^2]$

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

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

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

$$= E\left[(\widehat{y} - y)^2\right] + 0 + E\left[\sum_{i=1}^{n} w_i^2 \varepsilon_i^2\right]$$

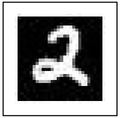
($\because$ $\varepsilon_i$ is independent of $\varepsilon_j$ and $\varepsilon_i$ is independent of ($\widehat{y}$-$y$) )

$$= (E\left[(\widehat{y} - y)^2\right] + \sigma^2 \sum_{i=1}^{n} w_i^2$$ (same as $L_2$ norm penalty)

# Module 8.8 : Adding Noise to the outputs

## Other forms of regularization

- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Hard targets |

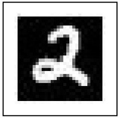
$$\text{minimize} : \sum_{i=0}^{9} p_i \log q_i$$

$$\text{true distribution} : p = \{0, 0, 1, 0, 0, 0, 0, 0, 0, 0\}$$

$$\text{estimated distribution} : q$$

**Intuition**

- Do not trust the true labels, they may be noisy
- Instead, use 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 |

$$\varepsilon = \text{small positive constant}$$

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

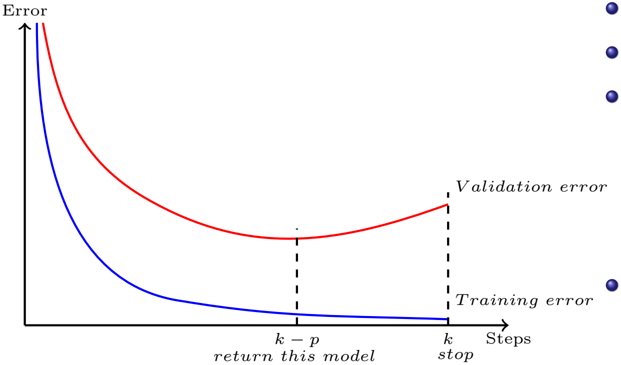
$$\text{true distribution + noise} : p = \left\{\frac{\varepsilon}{9}, \frac{\varepsilon}{9}, 1-\varepsilon, \frac{\varepsilon}{9}, \dots\right\}$$

$$\text{estimated distribution} : q$$

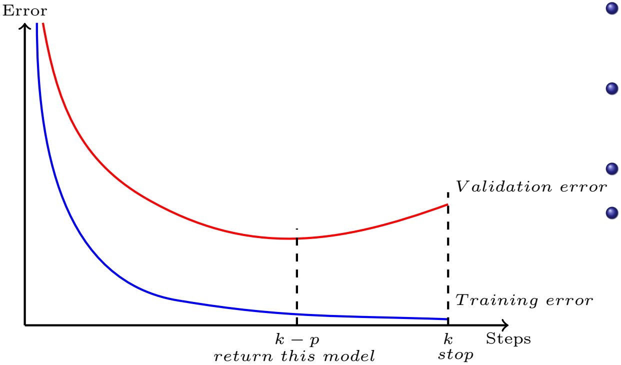
# Module 8.9 : Early stopping

## Other forms of regularization

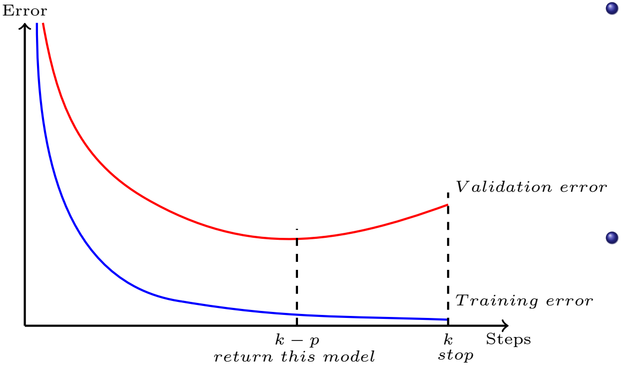
- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout




- Track the validation error
- Have a patience parameter $p$
- If you are at step $k$ and there was no improvement in validation error in the previous $p$ steps then stop training and return the model stored at step $k-p$
- Basically, stop the training early before it drives the training error to 0 and blows up the validation error




- Very effective and the mostly widely used form of regularization
- Can be used even with other regularizers (such as $l_2$)
- How does it act as a regularizer ?
- We will first see an intuitive explanation and then a mathematical analysis




- Recall that the update rule in SGD is

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

- Let $\tau$ be the maximum value of $\nabla w_i$ then

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

- Thus, $t$ controls how far $w_t$ can go from the initial $w_0$
- In other words it controls the space of exploration

We will now see a mathematical analysis of this

- Recall that the Taylor series approximation for $\mathscr{L}(w)$ is

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

$$= \mathscr{L}(w^*) + \frac{1}{2}(w - w^*)^T H (w - w^*) \qquad [\ w^* \ \textit{is optimal so} \ \nabla \mathscr{L}(w^*) \ \textit{is} \ 0\ ]$$

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

Now the SGD update rule is:

$$\begin{aligned} w_t &= w_{t-1} - \eta \nabla \mathscr{L}(w_{t-1}) \\ &= w_{t-1} - \eta H(w_{t-1} - w^*) \\ &= (I - \eta H) w_{t-1} + \eta H w^* \end{aligned}$$

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

- Using EVD of $H$ as $H = Q\Lambda Q^T$, we get:

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

- If we start with $w_0 = 0$ then we can show that (See Appendix)

$$w_t = Q[I - (I - \varepsilon\Lambda)^t]Q^T w^*$$

- Compare this with the expression we had for optimum $\tilde{W}$ with $L_2$ regularization

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

- We observe that $w_t = \tilde{w}$, if we choose $\varepsilon$,$t$ and $\alpha$ such that

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

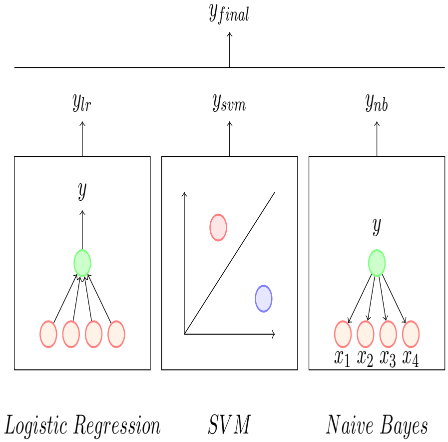
### Things to be remember

- Early stopping only allows $t$ updates to the parameters.
- If a parameter $w$ corresponds to a dimension which is important for the loss $\mathscr{L}(\theta)$ then $\frac{\partial \mathscr{L}(\theta)}{\partial w}$ will be large
- However if a parameter is not important ($\frac{\partial \mathscr{L}(\theta)}{\partial w}$ is small) then its updates will be small and the parameter will not be able to grow large in '$t'$ steps
- Early stopping will thus effectively shrink the parameters corresponding to less important directions (same as weight decay).

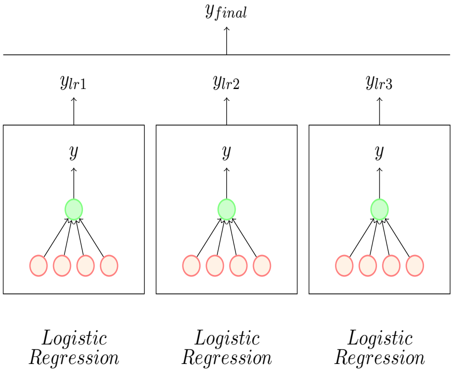
# Module 8.10 : Ensemble methods

## Other forms of regularization

- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout




- Combine the output of different models to reduce generalization error
- The models can correspond to different classifiers
- It could be different instances of the same classifier trained with:
  - different hyperparameters
  - different features
  - different samples of the training data




Each model trained with a different sample of the data (sampling with replacement)

- Bagging: form an ensemble using different instances of the same classifier
- From a given dataset, construct multiple training sets by sampling with replacement $(T_1, T_2, ..., T_k)$
- Train $i^{th}$ instance of the classifier using training set $T_i$

- The error made by the average prediction of all the models is $\frac{1}{k}\sum_i \varepsilon_i$
- The expected squared error is :

$$
\begin{aligned}
mse =& E[(\frac{1}{k}\sum_i \varepsilon_i)^2] \\
=& \frac{1}{k^2}E[\sum_i \sum_{i=j} \varepsilon_i \varepsilon_j + \sum_i \sum_{i\neq j} \varepsilon_i \varepsilon_j] \\
=& \frac{1}{k^2}E[\sum_i \varepsilon_i^2 + \sum_i \sum_{i\neq j} \varepsilon_i \varepsilon_j] \\
=& \frac{1}{k^2}(\sum_i E[\varepsilon_i^2] + \sum_i \sum_{i\neq j} E[\varepsilon_i \varepsilon_j]) \\
=& \frac{1}{k^2}(kV + k(k-1)C) \\
=& \frac{1}{k}V + \frac{k-1}{k}C
\end{aligned}
$$

- When would bagging work?
- Consider a set of $k$ LR models
- Suppose that each model makes an error $\varepsilon_i$ on a test example
- Let $\varepsilon_i$ be drawn from a zero mean multivariate normal distribution
- $Variance = E[\varepsilon_i^2] = V$
- $Covariance = E[\varepsilon_i \varepsilon_j] = C$

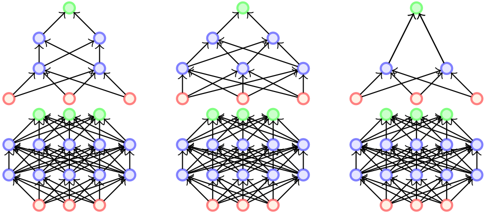
$$mse = \frac{1}{k}V + \frac{k-1}{k}C$$

- When would bagging work ?
- If the errors of the model are perfectly correlated then $V = C$ and $mse = V$ [bagging does not help: the mse of the ensemble is as bad as the individual models]
- If the errors of the model are independent or uncorrelated then $C = 0$ and the mse of the ensemble reduces to $\frac{1}{k}V$
- On average, the ensemble will perform at least as well as its individual members

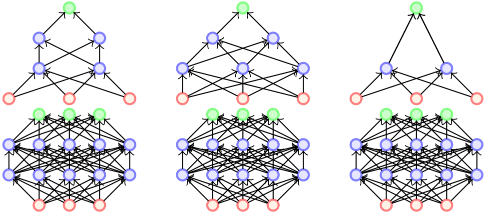
# Module 8.11 : Dropout

## Other forms of regularization

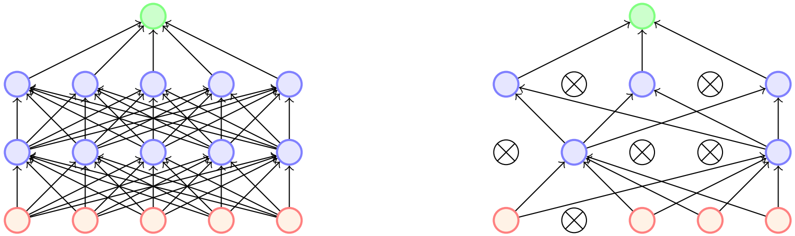
- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout



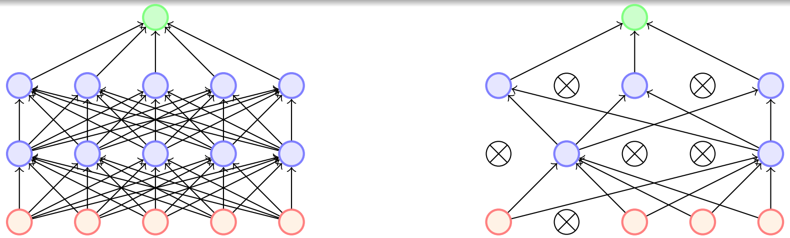
- Typically model averaging(bagging ensemble) always helps
- Training several large neural networks for making an ensemble is prohibitively expensive
- Option 1: Train several neural networks having different architectures(obviously expensive)
- Option 2: Train multiple instances of the same network using different training samples (again expensive)
- Even if we manage to train with option 1 or option 2, combining several models at test time is infeasible in real time applications



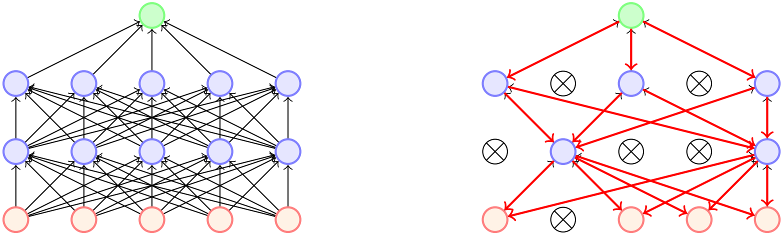
- Dropout is a technique which addresses both these issues.
- Effectively it allows training several neural networks without any significant computational overhead.
- Also gives an efficient approximate way of combining exponentially many different neural networks.



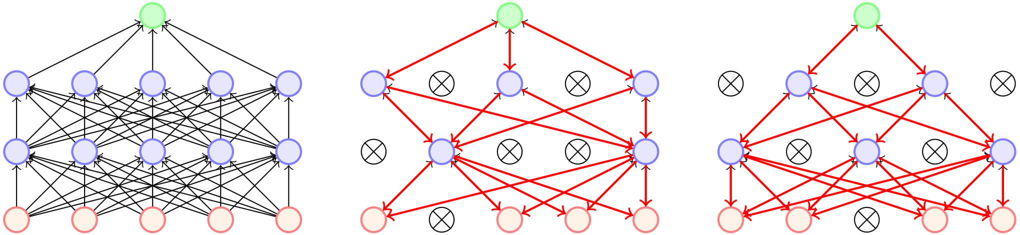
- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically $p = 0.5$) for hidden nodes and $p = 0.8$ for visible nodes



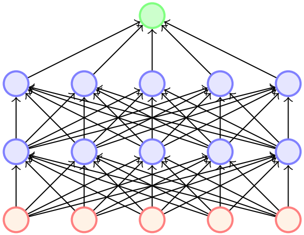
- Suppose a neural network has $n$ nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of $n$ nodes, what are the total number of thinned networks that can be formed? $2^n$
- Of course, this is prohibitively large and we cannot possibly train so many networks
- **Trick:** (1) Share the weights across all the networks
  (2) Sample a different network for each training instance
- Let us see how?

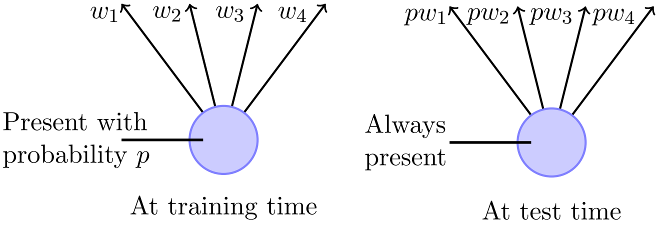


- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active

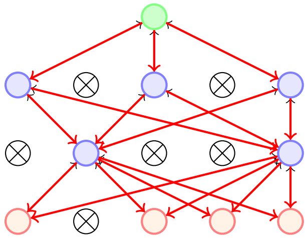


- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights
- If the weight was active for both the training instances then it would have received two updates by now
- If the weight was active for only one of the training instances then it would have received only one updates by now
- Each thinned network gets trained rarely (or even never) but the parameter sharing ensures that no model has untrained or poorly trained parameters

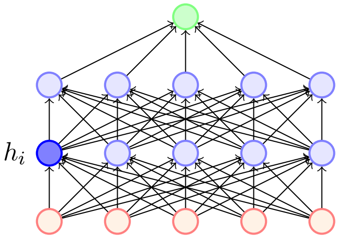





- What happens at test time?
- Impossible to aggregate the outputs of $2^n$ thinned networks
- Instead we use the full Neural Network and scale the output of each node by the fraction of times it was on during training



- Dropout essentially applies a masking noise to the hidden units
- Prevents hidden units from co-adapting
- Essentially a hidden unit cannot rely too much on other units as they may get dropped out any time
- Each hidden unit has to learn to be more robust to these random dropouts




- Here is an example of how dropout helps in ensuring redundancy and robustness
- 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

## Recap

- $l_2$ regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout

# Appendix

- To prove: The below two equations are equivalent

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

- Proof by induction:
- Base case: $t = 1$ and $w_0$=0:
- $w_1$ according to the first equation:

$$\begin{aligned} w_1 &= (I - \eta Q\Lambda Q^T)w_0 + \eta Q\Lambda Q^T w^* \\ &= \eta Q\Lambda Q^T w^* \end{aligned}$$

- $w_1$ according to the second equation:

$$\begin{aligned} w_1 &= Q(I - (I - \eta\Lambda)^1)Q^T w^* \\ &= \eta Q\Lambda Q^T w^* \end{aligned}$$

- Induction step: Let the two equations be equivalent for $t^{th}$ step

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

- Proof that this will hold for $(t+1)^{th}$ step

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

(using $w_t = Q[I - (I - \varepsilon \Lambda)^t] Q^T w^*$)

(using $w_t = Q[I - (I - \varepsilon \Lambda)^t] Q^T w^*$)

$$= (I - \eta Q \Lambda Q^T) Q(I - (I - \eta \Lambda)^t) Q^T w^* + \eta Q \Lambda Q^T w^*$$

$$= (I - \eta Q \Lambda Q^T) Q(I - (I - \eta \Lambda)^t) Q^T w^* + \eta Q \Lambda Q^T w^*$$

$$= (I - \eta Q \Lambda Q^T) Q(I - (I - \eta \Lambda)^t) Q^T w^* + \eta Q \Lambda Q^T w^*$$

(Opening this bracket)

$$= IQ(I - (I - \eta \Lambda)^t) Q^T w^* - \eta Q \Lambda Q^T Q(I - (I - \eta \Lambda)^t) Q^T w^* + \eta Q \Lambda Q^T w^*$$

- Continuing

$$
\begin{aligned}
w_{t+1} &= Q(I-(I-\eta\Lambda)^t)Q^Tw^* - \eta Q\Lambda Q^TQ(I-(I-\eta\Lambda)^t)Q^Tw^* + \eta Q\Lambda Q^Tw^* \\
&= Q(I-(I-\eta\Lambda)^t)Q^Tw^* - \eta Q\Lambda(I-(I-\eta\Lambda)^t)Q^Tw^* + \eta Q\Lambda Q^Tw^* (\because Q^TQ = I \\
&= Q(I-(I-\eta\Lambda)^t)Q^Tw^* - \eta Q\Lambda(I-(I-\eta\Lambda)^t)Q^Tw^* + \eta Q\Lambda Q^Tw^* \\
&= Q\big[(I-(I-\eta\Lambda)^t) - \eta\Lambda(I-(I-\eta\Lambda)^t) + \eta\Lambda\big]Q^Tw^* \\
&= Q(I-(I-\eta\Lambda)^t)Q^Tw^* - \eta Q\Lambda(I-(I-\eta\Lambda)^t)Q^Tw^* + \eta Q\Lambda Q^Tw^* \\
&= Q\big[(I-(I-\eta\Lambda)^t) - \eta\Lambda(I-(I-\eta\Lambda)^t) + \eta\Lambda\big]Q^Tw^* \\
&= Q\big[(I-(I-\eta\Lambda)^t) - \eta\Lambda(I-(I-\eta\Lambda)^t) + \eta\Lambda\big]Q^Tw^* \\
&= Q\big[I-(I-\eta\Lambda)^t + \eta\Lambda(I-\eta\Lambda)^t\big]Q^Tw^* \\
&= Q\big[I-(I-\eta\Lambda)^t + \eta\Lambda(I-\eta\Lambda)^t\big]Q^Tw^* \\
&= Q\big[I-(I-\eta\Lambda)^t + \eta\Lambda(I-\eta\Lambda)^t\big]Q^Tw^* \\
&= Q\big[I-(I-\eta\Lambda)^t(I-\eta\Lambda)\big]Q^Tw^* \\
&= Q\big[I-(I-\eta\Lambda)^t(I-\eta\Lambda)\big]Q^Tw^*
\end{aligned}
$$