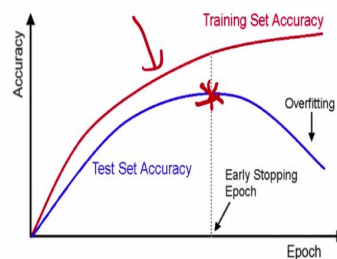


**Deep Learning for Computer Vision**  
**Professor Vineeth N Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture No 28**  
**Regularization in Neural Networks**

(Refer Slide Time: 00:13)

### Early Stopping

- **Simple idea:** keep monitoring cost function, and not let it become too low consistently; stop at an earlier iteration



Vineeth N B. (IIT-H) 54.4 Regularization



That was one regularization approach by imposing penalty of the  $L_2$  norm or  $L_1$  norm of the weights which is very popular. But even then you have to specify a coefficient for the  $L_2$  weight decay or  $L_1$  weight decay or any other weight decay that you choose. Another approach is known as Early stopping, which is perhaps the crudest simplest approach that you can use. The simple idea here is keep monitoring the cost function.

And do not let it become too low, consistently stop attempting earlier iteration. Rather, do not let your model over fit your training data. If it goes to 0 error, you are probably over fitting your data. And you do not want that to happen. So, you want to stop a little earlier. So here is a visual example. So if your y axis is accuracy, and x axis is the Epoch core training.

So, It is possible that your red curve here tells you that as you keep training, your training set, accuracy keeps increasing, or your training set loss keeps decreasing. They are both complementary. But It is possible that if you held aside some portion of your training data, called

it is a test set or a validation set and did not use it for your training, as in when you complete the Epoch, you can take that model and test it on that holdout set and see how it is performing.

As long as the performance from that holdout set or test set keeps improving, you keep training. When it starts dipping on that holdout set, it is probably time to stop. Even if you start going lower in training error. Even if your training set, performance keeps increasing. That time when you start when you when your performance on your holdout set, which you do not use for training decreases, it is time to stop training your Neural Network. This is the idea of early stopping.

(Refer Slide Time: 02:13)

Early Stopping

When to stop?

- Train n epochs; lower learning rate; train m epochs

NPTEL

IIT-H

Vineeth N B (IIT-H) 54.4 Regularization

This reminds us of the question that we asked last lecture, which we said we will cover some time here is when do you stop, we said we will talk about the stopping condition or the convergence criteria of these algorithms that we saw last lecture. So, when do you really stop, there are a few heuristics that you can use when working with them. The problem here is that when you work with computers, which are numerically approximated at some precision, it may not be possible to get an absolute 0 value for the grading.

You may have to ensure that you are Gradient say  $10^{-5}$ ,  $10^{-6}$  or something like that, and say I am going to stop here. But even that can lead to numerical errors. So, we are going to talk about a few heuristics, which we can use to decide when to stop training. One thing you can do is to



train n epochs, lower the learning rate, train m epochs epochs, so on and so forth. This is an approach.

(Refer Slide Time: 03:18)


**Early Stopping**

When to stop?

- Train n epochs; lower learning rate; train m epochs epochs → **Bad idea**: can't assume one-size-fits-all approach
- Error-change criterion:**
  - Stop when error isn't dropping over a window of, say, 10 epochs
  - Train for a fixed number of epochs after criterion is reached (possibly with lower learning rate)
- Weight-change criterion:**
  - Compare weights at epochs  $t - 10$  and  $t$  and test:  $\max_i \|w_i^t - w_i^{t-10}\| < \rho$



Vineeth N B. (IIT-H) 54.4 Regularization



But it is not a very advisable approach, because you do not know what n and m should be for all kinds of Neural Network models. Instead, what we can use are a couple of criteria, we can use a criteria known as the Error Change condition. In the Error Change condition, we keep checking for the error, and how it has been dropping over a window of epochs could be 10 epochs, 5 epochs, 3 epochs whatever.

And if the error is not dropping significantly across those epochs, or many batch iterations, you say it is time to stop your training. After you stopped, you can always train for a fixed number of iterations, you decide to stop because you still have not got to a critical point, which means your gradient is not absolute 0, you as well may just train a little bit more just to ensure that you probably get closer to the critical point. That is one heuristic you can use.

Another heuristic is a Weight Change criteria, you can compare the weights at an iteration on a epoch, that was at t-10 and at t which is your current iteration. And you can test if the maximum weight change is bounded by a value. Why do we say maximum weight change? If we simply take  $L_2$  norm of weights in the earlier iteration and  $L_2$  norm of weights in this iteration, that could be less than drop.

Because there were some weights that were not changing, but some weights were changing by a large amount. And you may have decided that the  $L_2$  norm of the weight change is fairly small, but by focusing on pairwise differences between weights in  $t-10$ . And now, and ensuring that the maximum pairwise difference between the weight at that epoch and this epoch is less than a constant is a good heuristic to stop training to.

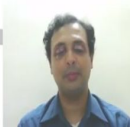


(Refer Slide Time: 05:13)

**Early Stopping**

When to stop?

- Train  $n$  epochs; lower learning rate; train  $m$  epochs → **Bad idea**: can't assume one-size-fits-all approach
- **Error-change criterion:**
  - Stop when error isn't dropping over a window of, say, 10 epochs
  - Train for a fixed number of epochs after criterion is reached (possibly with lower learning rate)
- **Weight-change criterion:**
  - Compare weights at epochs  $t-10$  and  $t$  and test:  $\max_i \|w_i^t - w_i^{t-10}\| < \rho$
  - Don't base on length of overall weight change vector
  - Possibly express as a percentage of the weight

Vineeth N B. (IIT-H) 54.4 Regularization



So in this case, it may be better not to base it on the length of the overall weight change vector, as I said, you could probably even this row could also be a percentage of your weight, you could also say that instead of row being a number saying less than 10th of - 3, you can say that it must be less than 10%, your overall norm of the weight or something like that. Those are ways in which or any other value for that matter. Those are ways in which you can use this as a stopping criterion for your Neural Network.

(Refer Slide Time: 05:48)

## Dataset Augmentation

- Creation of data using some knowledge of task
- Exploit the fact that certain transformations to image do not change label of image
- Methods:
  - Data jittering (E.g. Distortion and blurring of images)
  - Rotations
  - Color changes
  - Noise injection
  - Mirroring
- Helps increase data; is useful when training data provided is less (DNNs need large amounts of training data to work!)
- Also acts as regularizer (by avoiding overfitting to provided data)



Vineth N B. (IIT-H)

54.4 Regularization



Another regularization method is Dataset Augmentation, where, in addition to the data that is given to you in your training data, you also add more data through transformations on your original data, which, to some extent, exposes your Neural Network to data beyond the training data, and hence gives you better generalization performance. So, let us see how we go about this. So, let us see how you really augment your data in other ways.

You exploit the fact that certain transformations to the image do not change the label of the image. For example, if you had a cat in an image, and you want to call it a cat, whether the cat was small in size, large in size, rotated by 30 degrees rotated the other way, by 60 degrees, the cat is a cat. So why do not we take the original image, make these transformations, and then train the model with all of these transformations to hope that it will probably do well on some data that it has not seen so far.

Because that cat that comes tomorrow, and a new image could be rotated. So, there are different things that you can do, you can do Data Jittering, which means you can blur the image, you can distort the image a little bit to handle nice variations in your test data, you could rotate the image, as we just said, you could impose color changes in the image. How do you do color changes? Remember that every image has an R channel, G channel and B channel, it is common to have 3 channels as input to the Neural Network.

People generally do not use other color spaces, although you can. You can take the intensities in each of these channels and mix them up, you can take the intensities in the green channel and call it the direct channel, you can take the intensities in the blue channel and call it the green channel. And you can get several permutations combinations out of these, you could also inject noise in your data, you could inject some Gaussian noise across your image, you could also mirror your image. In cases where the mirroring does not change the label.

Whether you have if there are objects, I mean, if you see a cat one way or the other, It is still a cat. And you may still want your model to not change its decision based on how the cat looked this way or the other way. This helps increase data Firstly, because Neural Networks need large amounts of data for training. And in addition to increasing data, it also acts as a regularizer because it is exposing the model to newer kinds of data beyond the training data alone.

(Refer Slide Time: 08:39)



Dataset Augmentation: Example<sup>1</sup>

Original photo    Red color casting    Green color casting    Blue color casting

RGB all changed    Vignette    More vignette    Blue casting + vignette

<sup>1</sup>Wu, Ren, et al. "Deep image: Scaling up image recognition." arXiv 2015

Vineeth N B. (IIT-H)    S4.4 Regularization

Here are some examples of how these augmentations look. So, here is an Original photo, here is Red color casting, just by increasing the intensity of the red channel, Green color casting, similarly Blue color casting, RGB all changed as we said just keep flipping color channels, Vignette, more Vignette, Blue casting plus Vignette.

(Refer Slide Time: 09:05)

### Dataset Augmentation: Example<sup>1</sup>



<sup>1</sup>Wu, Ren, et al. "Deep image: Scaling up image recognition." arXiv 2015

Vineeth N B. (IIT-H)

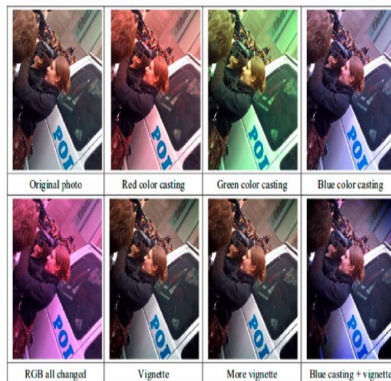
§4.4 Regularization



Here are some more examples, Left rotation and crop, see you rotated by left so let us see the original image again.

(Refer Slide Time: 09:13)

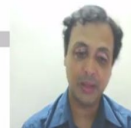
### Dataset Augmentation: Example<sup>1</sup>



<sup>1</sup>Wu, Ren, et al. "Deep image: Scaling up image recognition." arXiv 2015

Vineeth N B. (IIT-H)

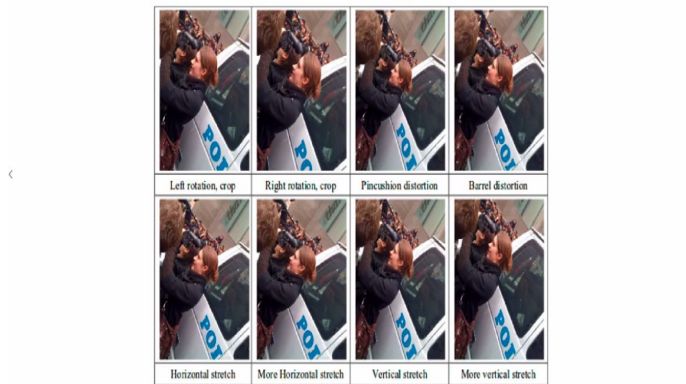
§4.4 Regularization



So, you can see the difference you can see. So, you can see that this was the original photo observed on the top left on the next slide.

(Refer Slide Time: 09:19)

### Dataset Augmentation: Example<sup>1</sup>



<sup>1</sup>Wu, Ren, et al. "Deep image: Scaling up image recognition." arXiv 2015

Vineeth N B. (IIT-H)

§4.4 Regularization



You can see that there was a left rotation and then what went outside the frame was cropped rather what went outside the size of Original image was cropped similarly Right rotation and crop, Pincushion distortion, Barrel distortion, Horizontal stretch, more Horizontal stretch, Vertical stretch, more Vertical stretch so on and so forth. Horizontal stretch is this way. Vertical stretch is this way. It is a good exercise for you to see which image processing operation would do all of these.

Remember, these were all byproducts of image processing operations that we talked about in the very beginning of this course, you can now try to see how you construct these distortions with simple image processing operations. If you need help, you can also look at this paper called Deep image scaling up recognition to understand what kind of augmentations and how you can arrive with them.



(Refer Slide Time: 10:22)

**Data Augmentation: Newer Methods**

**Mixup**


- Create virtual training examples as below ( $\lambda \in [0, 1]$ )

$$\tilde{x} = \lambda x_i + (1 - \lambda)x_j$$
$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j$$

where  $x_i, x_j$  are input vectors, and  $y_i, y_j$  are one-hot label encodings




- Variants: Manifold Mixup, AugMix

- Randomly mask out square regions of input during training



○ Variants: CutMix

Vineeth N B. (IIT-H)      54.4 Regularization



More newer methods in the last one or two years have done data augmentation in a very different manner. A very popular method today is called Mixup. What Mixup does for doing data set augmentation is create virtual training examples by interpolating data. If you have two data points  $x_i$  and  $x_j$  could be two images,  $x_i$  and  $x_j$ , you construct a new sample  $\tilde{x}$ , which is a convex combination between  $x_i$  and  $x_j$ .

So, the new image lies on a line drawn between  $x_i$  and  $x_j$ . And for the label to  $x_i$  may have a label  $y_i$ ,  $x_j$  may have a label  $y_j$ , remember that when you use it for training the Neural Network.  $y_i$  let us say you had three class labels,  $y_i$  would maybe be  $[1 \ 0 \ 0]$ . This could mean a dog could mean a cat. And this could mean a horse. So, you represent  $y_i$  for a dog to be  $[1 \ 0 \ 0]$  and  $y_j$  for a cat to be  $[0 \ 1 \ 0]$ .

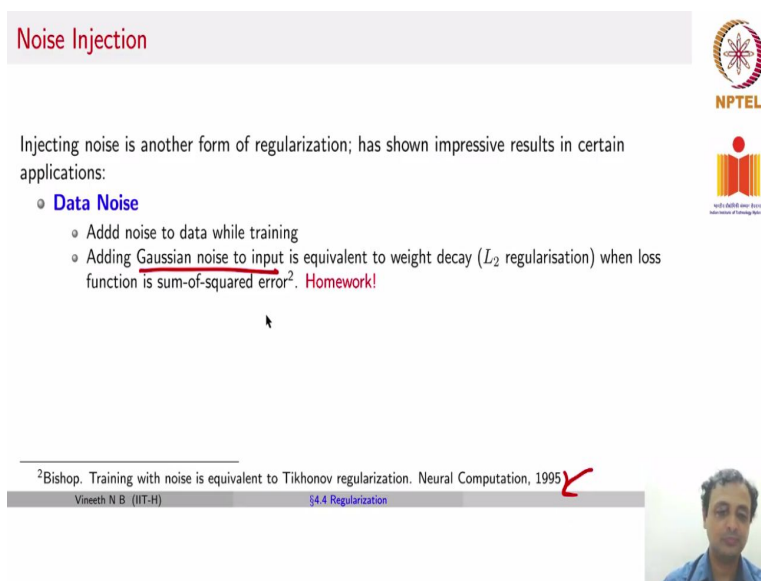
So, now you could take combinations of these, you can say  $\lambda y_i + (1 - \lambda)y_j$ . And that will give you a new combination, the values need not be 1 and 0, it can lie between 1 and 0. And that is the purpose here.  $\tilde{y}$  will give, for if you mix up the image in a particular way, you also mix up the label in the same way, that is what Mixup does and Mixup for the last couple of years has led to several variants like Manifold Mixup, AugMix, CutMix, so on and so forth.

This is one interesting and effective way of performing Dataset augmentation today. Another popular method today is known as Cut Out, where you randomly mask out square regions of

input, you can see these gray boxes here. So, these are all boxes of gray, which have masked out certain regions during training. That is the reason it is called cut out, we just cut out certain portions and give the rest of the image you fill in black in that location and fill in the rest of the image and fill in the image with that black portion. And let the Neural Network train on that.

Even these kinds of augmentations have shown fairly good performance, fairly good generalization performance when you train Neural Networks. More recent variant of a cutout is also called CutMix. And there are many other follow up methods, but these are the broad, broad ones, which you perhaps should know.

(Refer Slide Time: 13:13)



The slide is titled "Noise Injection" in red text. Below the title, it states: "Injecting noise is another form of regularization; has shown impressive results in certain applications:". This is followed by a bulleted list under the heading "Data Noise":

- Add noise to data while training
- Adding Gaussian noise to input is equivalent to weight decay ( $L_2$  regularisation) when loss function is sum-of-squared error<sup>2</sup>. Homework!

On the right side of the slide, there are two logos: the NPTEL logo (National Programme on Technology Enhanced Learning) and the IIT-H logo (Indian Institute of Technology Hyderabad). At the bottom of the slide, there is a footer with the text: "<sup>2</sup>Bishop. Training with noise is equivalent to Tikhonov regularization. Neural Computation, 1995" with a red arrow pointing to the citation. Below the footer, there is a small video inset showing a man speaking.

Another approach as we said for regularization is Injection of Noise. So, this noise can be injected at a data level and can also be indicated at a label level or a gradient level as we will see. In data noise, you add noise to the data while training. And you can mathematically show that adding Gaussian noise to the input is equivalent into doing  $L_2$  weight decay, when the loss function is the sum of squared error.

That is an interesting connection. We are going to leave it to us to do homework. So, this is the paper that actually showed it way back in the 90s. You can look at that paper if you want to, if you want to try to understand how to do it, but we will review it next, next lecture please try on your own to see if you can prove this.

(Refer Slide Time: 14:08)

## Noise Injection

Injecting noise is another form of regularization; has shown impressive results in certain applications:

- **Data Noise**
  - Add noise to data while training
  - Adding Gaussian noise to input is equivalent to weight decay ( $L_2$  regularisation) when loss function is sum-of-squared error<sup>2</sup>. **Homework!**
  - Can be interpreted as form of **dataset augmentation**.
- **Label Noise**

<sup>2</sup>Bishop. Training with noise is equivalent to Tikhonov regularization. Neural Computation, 1995



So, just adding noise to your input data. As I just said, adding Gaussian noise to input is equivalent doing  $L_2$  weight decay for a particular kind of a loss function. But which means adding noise is also a regularizer and also becomes a certain kind of Augmentation. We can also do Label noise and Gradient noise.

(Refer Slide Time: 14:31)

## Regularization through Label Noise<sup>3</sup>

- Disturb each training sample with probability  $\alpha$ .
- For each disturbed sample, label randomly drawn from uniform distribution over  $\{1, 2, \dots, C\}$  regardless of true label

### Algorithm 1 DisturbLabel

- 1: **Input:**  $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ , noise rate  $\alpha$ .
- 2: **Initialization:** a network model  $M: f(x; \theta_0) \in \mathbb{R}^C$ ;
- 3: **for** each mini-batch  $\mathcal{D}_t = \{(x_m, y_m)\}_{m=1}^M$  **do**
- 4:   **for** each sample  $(x_m, y_m)$  **do**
- 5:     Generate a disturbed label  $\tilde{y}_m$  with Eqn (2);
- 6:   **end for**
- 7:   Update the parameters  $\theta_t$  with Eqn (1);
- 8: **end for**
- 9: **Output:** the trained model  $M': f(x; \theta_T) \in \mathbb{R}^C$ .

$$\begin{cases} \tilde{c} \sim \mathcal{P}(\alpha), \\ \tilde{y}_{\tilde{c}} = 1, \\ \tilde{y}_i = 0, \quad \forall i \neq \tilde{c}. \end{cases} \quad (2)$$

<sup>3</sup>Xie, Lingxi, et al. "DisturbLabel: Regularizing CNN on the Loss Layer." CVPR 2016.



And let us see how that is done. In case of label noise, you disturb each training sample with some probability alpha. And for each disturb sample, you label you the, you draw the label

randomly from a uniform distribution, regardless of the true label. So, here is a more this is the algorithm drawn from this paper known as discrete disturbed label, just published in 2016. You generated Disturbed Label so for a particular training sample, which picked up with probability alpha, you instead of taking its correct label, you uniformly sample from any of the labels that you have in your set of categories when you do a classification problem. This just adds some label noise. This is clearly an incorrect label. But this, you hope will keep the Neural Network from over fitting to training data, and hence, generalize with. This is an idea, but it is not used too often, in practice.

(Refer Slide Time: 15:34)

### Regularization through Gradient Noise<sup>4</sup>

- **Idea:** Add noise to gradient

$$g_t \leftarrow g_t + N(0, \sigma_t^2)$$

- Annealed Gaussian noise by decaying variance




$$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

- Showed significant improvement in performance in certain applications

---

<sup>4</sup>Neelakantan, Arvind, et al. "Adding gradient noise improves learning for very deep networks." arXiv preprint arXiv:1511.06807 (2015)

Vineeth N B. (IIT-H) 54.4 Regularization

An idea that is used in practice is known as Gradient noise. In this case, you add noise to the gradient, instead of the input or the output, you add noise to the gradient while training. And how do you do that, you take the gradient of any weight in the Neural Network when you are doing back propagation, and to that gradient, before you update your parameter add some noise. Remember, you would have formerly propagated, got an output, got an error, got a gradient.

And then you would ideally use that gradient to back propagate, before you back propagate and update your weights. You add noise, what noise Gaussian noise with mean 0 and variance  $\sigma_t^2$ . And this work also suggests that you anneal the Gaussian noise by decaying its variance over the iterations. So, the  $\sigma_t^2$  can be written as  $\frac{\eta}{(1+t)^\gamma}$ , where  $\eta$ ,  $\gamma$  are user specified constants.

You have to specify them to use this method. But what it is doing now is, as you keep training over time, you are trying to reduce the variance of this Gaussian, which means you are trying to keep the nice, lesser and lesser and less as you go through training. This does seem to show significant improvement in performance in certain applications?


(Refer Slide Time: 17:03)

**Ensemble Methods**

- Train several different models separately, then have all models vote on the output for test examples
- An example of a general strategy in machine learning called **model averaging**
- Models can correspond to different classifiers, or could be different instances of same classifier trained with:
  - different hyperparameters
  - different features
  - different samples of the training data

Credit: Ali Ghodsi, Univ of Waterloo; Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) 4.4 Regularization



A gentle approach to achieve regularization in Machine Learning is Ensemble methods. In ensemble methods, you train several different models separately, and then have all models vote on the output. Why is this a regularizer? Because if one of those models over fit your training data, you still hope that the other models would not have over fit. And they would perhaps help you do well on test data in the generalization setting.

A standard example of a strategy for assembling is model averaging. So, you have  $k$  different models, and you get all their outputs an average the outputs and the hope is that not all of them would over fit your training. So, these different models that you choose for assembling could be different models that you get from different hyper parameters, you could use different learning rates or different number of layers in a Neural Network.

It could be different features in terms of input that you give to these models, or it could be different samples of your training data. So, if you have 10,000 data points in your training data

you can take 1000 of them, and then train one model, take another 1000 train another model, we will see that in a slide from now.

(Refer Slide Time: 18:29)

**Ensemble Methods**

- **Bagging** (short for bootstrap aggregating): reduces generalization error by forming an ensemble using different instances of same classifier
- For e.g., consider a set of  $k$  logistic regression models
- Given a dataset, construct multiple training sets by sampling with replacement ( $T_1, T_2, \dots, T_k$ )
- Train  $i^{\text{th}}$  instance of classifier using training set  $T_i$

Credit: Ali Ghodsi, Univ of Waterloo; Mitesh Khapra, IIT Madras

The diagram shows three parallel logistic regression models. Each model takes a subset of the training data  $y$  as input and produces a prediction  $y_{1r1}$ ,  $y_{1r2}$ , and  $y_{1r3}$  respectively. These predictions are then aggregated to produce the final prediction  $y_{final}$ . Each model is labeled "Logistic Regression".

NPTEL  
National Programme on Technology Enhanced Learning

Vineeth N B (IIT-H) §4.4 Regularization

So, here is that example, bagging with stats, which stands for bootstrap aggregating is an ensemble method in traditional Machine Learning, which does one ensemble using the training data set. How does it do this, if you had, say,  $k$  logistic regression models or any machine learning models for that matter, we just take an example of logistic regression here.

Given a data set, a training data set, you construct multiple training sets, by sampling with replacement. So, you construct  $k$  different data sets from your original data set by sampling with replacement. So, you take if you had a data set with 10,000 data points, you take thousand of them, train a model, replace them and take another thousand so you just keep you can train as many models as you would like by when you start sampling with replacement. And each such model here is trained with the corresponding training data set. So, you can get  $k$  such models now using  $k$  samples of your training data.

(Refer Slide Time: 19:39)

## Ensemble Methods

- Expected squared error of ensemble predictor is:
- Suppose that each model makes an error  $\epsilon_i$  on a test example
- Let  $\epsilon_i$  be drawn from a zero-mean multivariate normal distribution with:
  - Variance =  $\mathbb{E}[\epsilon_i^2] = V$
  - Covariance =  $\mathbb{E}[\epsilon_i \epsilon_j] = C$
- Error made by average prediction of all models is  $\frac{1}{k} \sum_i \epsilon_i$

$$\begin{aligned} \text{MS E} &= \mathbb{E} \left[ \left( \frac{1}{k} \sum_i \epsilon_i \right)^2 \right] \\ &= \frac{1}{k^2} \mathbb{E} \left[ \sum_{i,j} \epsilon_i \epsilon_j + \sum_i \epsilon_i^2 \right] \\ &= \frac{1}{k^2} \mathbb{E} \left[ \sum_i \epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j \right] \\ &= \frac{1}{k^2} (kV + k(k-1)C) \\ &= \frac{1}{k} V + \frac{k-1}{k} C \end{aligned}$$

*Credit: Ali Ghodsi, Univ of Waterloo; Mitesh Khapra, IIT Madras*



Now, let us try to analyze why this can be useful. Suppose each model makes an error  $\epsilon_i$  on a test sample. Let us make this assumption. Let us to be able to mathematically analyze this. Let us assume that  $\epsilon_i$  is drawn from a 0 mean multivariate normal distribution with variance of  $\epsilon_i^2$  to be given us  $V$ , and covariance  $\epsilon_i, \epsilon_j$  that is the variance between these models to be given by quantity called  $C$ .

The error made by the average prediction of, of these models is  $\frac{1}{k}$  summation over  $i$   $\epsilon_i$ . That is the average error made by all models for the given training example test example. Let us now look at the expected squared error of this ensemble predictor. The mean square error of the ensemble is going to be the expectation over  $\frac{1}{k}$  summation  $\epsilon_i^2$ . That is what you have here.

Now, that can be expanded since  $\frac{1}{k}$  is constant,  $\frac{1}{k^2}$ , because  $\frac{1}{k}$  was inside the brackets, it comes out as  $\frac{1}{k^2}$ . And the term here can be expanded this way, a simple way to understand why that is correct is remember, a plus b, the whole square is a square plus b squared plus ab plus ba. So that is exactly the way we are writing it. So, we are writing out a summation over  $i$   $\epsilon_i^2$ , which is which is equivalent to saying something like  $\epsilon_1^2 + \epsilon_2^2 + \dots + \epsilon_k^2$ , which is like a plus b whole square is a square plus b squared plus ab plus ba.

So, in this case, we are writing  $\epsilon_i \epsilon_j$ , when  $i$  and  $j$  are the same, those will give you the square term, and  $\epsilon_i \epsilon_j$ , when  $i$  and  $j$  are not equal, so that will be the  $ab$   $ba$  terms you can. Now work out for higher dimensions, but that is how the expansion comes. So, you then have since  $j$  is equal to  $i$ , this can be summarized as summation over  $i$   $\epsilon_i$  square. And the rest, the other term remains the same.

Now, by our definition is here. And the linearity of expectation, over expectation can be of the term can be because of the linearity of it, you can say expectation of  $a + b$  is expectation of  $a$  plus expectation of  $b$ . Using that you can write this expectation of sum of  $i$   $\epsilon_i$  square, which would be sum of  $i$   $\epsilon_i$  expectation  $\epsilon_i$  square so let me write that out to make it clear.

So, this would be expectation of summation  $\epsilon_i$  square plus will have an expectation of the other term. Now, but this term is can also be written as summation over  $i$  expectation,  $\epsilon_i$  square, once again, because of the linearity of expectation, you can write this way. So, when you do that, you would have an expectation of  $\epsilon_i$  square, which is  $V$ , and you will take a summation  $k$  times over  $V$  so you will have  $kV$ .

Similarly, in this case, you would have  $k$  into  $k - 1$   $C$ , why is that the case you would have  $i$  going from  $1$  to  $k$ ,  $j$  not equal to  $i$ . So, you will get the  $k - 1$  values in the second submission. So,  $k$  into  $k - 1$  into  $\epsilon_i \epsilon_j$ , the expectation is given by  $C$ . So you will have that to be  $C$ . Now cancelling out case, you will be left with  $1$  by  $kV$  plus  $k - 1$   $k$  by  $C$ .



(Refer Slide Time: 23:56)

### Ensemble Methods

- Expected squared error of ensemble predictor:

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

What does this tell you?

- If errors of model are perfectly correlated, then  $V = C$  and  $MSE = V$ , i.e. bagging does not help: the MSE of ensemble is as bad as individual models



Vineeth N B (IIT-H) 54.4 Regularization



Why, what are we going to do with this? Let us again write that out. So, we are saying that the mean squared error of this ensemble is given by  $MSE = \frac{1}{k}V + \frac{k-1}{k}C$ . What does this tell us? This does tell us something interesting. It tells us that if the errors of the model are perfectly correlated, which means  $V$  and  $C$  are the same.

(Refer Slide Time: 24:20)

### Ensemble Methods

- Expected squared error of ensemble predictor is:

- 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 with:

Variance =  $\mathbb{E}[\varepsilon_i^2] = V$

Covariance =  $\mathbb{E}[\varepsilon_i \varepsilon_j] = C$

- Error made by average prediction of all models is  $\frac{1}{k} \sum_i \varepsilon_i$

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

*Handwritten notes:  $\sum_i \mathbb{E}[\varepsilon_i^2]$ ,  $\sum_{i \neq j} \mathbb{E}[\varepsilon_i \varepsilon_j]$ ,  $(\varepsilon_1 + \varepsilon_2 + \dots + \varepsilon_k)^2$*

Credit: Ali Ghodsi, Univ of Waterloo; Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) 54.4 Regularization



So, remember, in this case, they are perfectly correlated, which means all of them are exactly correlated the same way.

(Refer Slide Time: 24:26)

### Ensemble Methods

- Expected squared error of ensemble predictor:

$$MSE = \frac{1}{k}V + \frac{k-1}{k}V = V$$

What does this tell you?

- If errors of model are perfectly correlated, then  $V = C$  and  $MSE = V$ , i.e. bagging does not help: the MSE of ensemble is as bad as individual models



Vineeth N B. (IIT-H) 54.4 Regularization



So then  $V$  is equal to  $C$ , then when you substitute  $V$  is equal to  $C$  here, so you replace this with  $V$ , you will end up with  $V$  itself to be the answer. Rather, bagging really does not help.

(Refer Slide Time: 24:46)

### Ensemble Methods

- Expected squared error of ensemble predictor is:

- 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 with:

$$\text{Variance} = \mathbb{E}[\varepsilon_i^2] = V$$

$$\text{Covariance} = \mathbb{E}[\varepsilon_i \varepsilon_j] = C$$

- Error made by average prediction of all models is  $\frac{1}{k} \sum_i \varepsilon_i$

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

Credit: Ali Ghodsi, Univ of Waterloo; Mitesh Khapra, IIT Madras  
Vineeth N B. (IIT-H) 54.4 Regularization



Because we are saying now that  $V$  is the variance of any one model. And it is also the variance of between the models.

(Refer Slide Time: 24:50)

### Ensemble Methods

- Expected squared error of ensemble predictor:

$$MSE = \frac{1}{k}V + \frac{k-1}{k}C \quad \text{with } C = V \Rightarrow MSE = V$$

What does this tell you?

- If errors of model are perfectly correlated, then  $V = C$  and  $MSE = V$ , i.e. bagging does not help: the MSE of ensemble is as bad as individual models



Vineeth N B (IIT-H) 54.4 Regularization



And our ensemble also has the same variance from the mean squared error. So, it does not really help much, when all the errors of the model are all exactly the same, they are all correlated in exactly the same way.

(Refer Slide Time: 25:03)

### Ensemble Methods

- Expected squared error of ensemble predictor:

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

What does this tell you?

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

Credit: Ali Ghodsi, Univ of Waterloo; Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H) 54.4 Regularization



However, if the errors of the model are independent or uncorrelated, especially if  $C$  is equal to 0 number  $C$  was the correlation between  $\epsilon_i$  and  $\epsilon_j$ . So, if two different models errors are uncorrelated, and  $C$  will be 0, and your mean squared error will be  $\frac{1}{k}$  times  $V$ . So, which

means the ensemble will have significantly lesser variance, the MSE of the ensemble will have significantly lesser variance. Another way of summarizing this discussion is that the ensemble will perform at least as well as its individual members.




(Refer Slide Time: 25:44)

**Dropout**

*Hebbian*

- One major issue in learning large neural networks is **co-adaptation**
  - As network is trained iteratively, powerful connections are learned more while weaker ones are ignored
  - After many iterations, only a fraction of node connections participate
  - Therefore, expanding neural network size may not help

Vineeth N B (IIT-H) 54.4 Regularization



Now, why did we talk about ensembles? We want to now see, how do you bring that idea of ensembles to regularize a Neural Network that an ensemble of Neural Networks for that, for that matter. So, one major issue before we design ensembling for Neural Networks, a major issue that occurs when you learn large Neural Networks is what is known as co-adaptation. Co-adaptation is, as it should remind you of what we talked about as Hebbian learning at the start of this week, that as Network is trained iteratively powerful connections are learned more and more while weaker ones are ignored.

You could ask me what is wrong, that is what Hebbian learning does. Hebbian learning is written this way by a person called Donald Hebb. So Hebbian learning seems to say the same thing. It is all right in general. But when the same connections get to learn more and more and other weaker connections are ignored, you may have to be concerned that your model may be over fitting to your training data.

Because you do want to ensure that your models, as we said earlier, It is okay if you make a few mistakes on your training data, but you want to do well on your test set. So, It is possible that

after many iterations, only a fraction of node connections actually participate in generating the output, and just increasing your Neural Network size is not really going to help because the situation will continue to proliferate even then.

(Refer Slide Time: 27:28)

## Dropout

- One major issue in learning large neural networks is **co-adaptation**
  - As network is trained iteratively, powerful connections are learned more while weaker ones are ignored
  - After many iterations, only a fraction of node connections participate
  - Therefore, expanding neural network size may not help
- Dropout**: Regularization method to address this issue
- Training Phase**: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction,  $p$ , of nodes (and corresponding activations)



Vineeth N B. (IIT-H)

§4.4 Regularization



So, what can we do here? We try to use a method called Dropout, which is a regularization method. What does dropout do in the training phase for each hidden layer, for each training sample, you ignore a random fraction of the nodes.

(Refer Slide Time: 27:49)

## Dropout

- One major issue in learning large neural networks is **co-adaptation**
  - As network is trained iteratively, powerful connections are learned more while weaker ones are ignored
  - After many iterations, only a fraction of node connections participate
  - Therefore, expanding neural network size may not help
- Dropout**: Regularization method to address this issue
- Training Phase**: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction,  $p$ , of nodes (and corresponding activations)
- Test Phase**: Use all activations, but reduce them by factor  $p$  (to account for missing activations during training)



Vineeth N B. (IIT-H)

§4.4 Regularization



And in the test phase, you use all the activations, but reduce them by a factor of  $p$ .  $p$  was the probability with which you pick nodes. And the test phase, you multiply all the activations by  $p$ .  $p$  could be 0.5 for that matter. So when, when  $p$  is 0.5, in every layer, you drop 50 percent of the nodes in each mini batch iteration, when you train a Neural Network using SGD. So, you do that for each mini batch iteration, you probably do thousand iterations.

And each iteration, your Neural Network could be slightly changing, because you are randomly dropping of nodes in each layer. So, at the end after training, what is the model that you have to use for testing? You take the old original full model, but you multiply the activations that you get from every layer by 0.5, in this case. Why do you need to do that, because each node could be participated 50 percent of the time, that is how it was sampled. So, you probably should give only 50 percent weightage, to its activation.

(Refer Slide Time: 28:58)

Dropout

(a) Standard Neural Net

(b) After applying dropout.

<sup>5</sup>Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Vineeth N B. (IIT-H) 54.4 Regularization

So, here is the illustration of how Dropout works. This was proposed in 2014. So, you have a standard Neural Network. In this case, this is the input layer. And here is the output layer. So, you see here that when you any apply Dropout in each layer, there are certain neurons that you simply do not consider for that mini batch iteration when you forward propagate. You only consider the other weights. It is almost as if those weights were 0 or do not exist, you could also do dropout in the input layer. So certain input features are just not considered at all in that iteration. Why do we do this?

(Refer Slide Time: 29:42)

## Dropout

- Recall ensemble methods: with  $H$  hidden units, each of which can be dropped, we can have  $2^H$  possible models



Vineeth N B (IIT-H)

§4.4 Regularization



Let us examine this a bit more. And we will also give you an intuition of how this how this can be canceled. Remember, we said that ensemble methods are good regularizes because they can minimize the variance of a model and hence do well on future test data. Ensemble is generally more robust than a single model. So, we want to see how to do it with a Neural Network. And if a Neural Network has  $H$  hidden units, each of which can be dropped, you can now have  $2^H$  possible models. That is the total number of possible models that you can have the Neural Network to create an ensemble.

(Refer Slide Time: 30:23)

## Dropout

- Recall ensemble methods: with  $H$  hidden units, each of which can be dropped, we can have  $2^H$  possible models
- This is prohibitively large and we cannot possibly train so many networks
- **Trick:** Each of the  $2^{H-1}$  models that includes hidden unit  $h$  must share the same weight for the unit
  - serves as a form of regularization
  - makes the models cooperate



Vineeth N B. (IIT-H)

54.4 Regularization



So, how do we go about creating such an ensemble, you probably do not want to train  $2^H$  models, and then ensemble them by voting or any other means. So, what we do now is impose one small constraint, we say that, for a particular hidden unit  $H$ , there are  $2^{H-1}$  models, which can vary when that hidden unit is fixed. Because there are  $H-1$ , so  $2^{H-1}$  models. Apart from this node can be changed when this unit is fixed. We are saying now that across all of those  $2^{H-1}$  models, this unit  $H$  must have the same weight, this does not change, its weight is fixed, and only the other weights can change. This is the trick that we are going to use.



(Refer Slide Time: 31:15)

## Dropout

- Recall ensemble methods: with  $H$  hidden units, each of which can be dropped, we can have  $2^H$  possible models
- This is prohibitively large and we cannot possibly train so many networks
- Trick:** Each of the  $2^{H-1}$  models that includes hidden unit  $h$  must share the same weight for the unit
  - serves as a form of regularization
  - makes the models cooperate
- Including all hidden units at test time with scaling of  $\frac{1}{2}$  is equivalent to computing geometric mean of all  $2^H$  models.



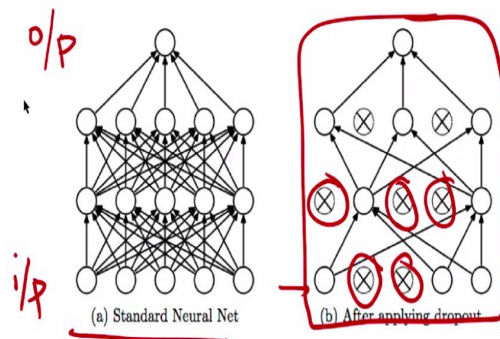
Vineeth N B. (IIT-H) §4.4 Regularization



And how does that actually work? It happens that if you do this, if you drop out with a probability of a particular probability  $p$  of 0.5, during training in every mini batch iteration, test time, simply multiplying your activations by half, or  $p$  for that matter, is equivalent to the geometric mean of all your  $2^H$  models. We are going to prove this. But before we prove, let us try to understand intuitively why this could be useful as a regularizer.

(Refer Slide Time: 32:02)

## Dropout



<sup>5</sup>Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2011

Vineeth N B. (IIT-H) §4.4 Regularization



An intuitive example is let us assume that there was a leader of an organization who had 1000 employees. Now each of these neurons here are like employees of an organization. And the CEO puts together these employees in certain configurations to achieve a certain task. Now, the way a Neural Network learns to perform a task is not through knowledge, is by repetitively doing the same thing again, and again and again.

And it is effectively trial and error, where you keep changing the weights and effectively get there get to performing well on your task. So, if an organization had to run this way, what could happen is, over time, certain people in the organizations gather certain specializations and they keep getting better and better at that, while maybe others who are not exposed to the task did not do so well, in that kind of task.

This is what we mentioned as co-adaptation. And this could be dangerous, because after finishing a particular task, if the CEO had to take up a new project, and certain employees left, then the CEO would be left with certain people who may not really know certain tasks well. So, what does the CEO do? The CEO simply takes an interesting decision of asking employees to randomly show up every week, only 50 percent of the employees show up every week.

And they have to do all the tasks between them, even the tasks that the other 50 percent was doing. How does this help? Now each employee is even learning other tasks, which they may not have specialized on. And this builds a robust organization that can handle newer tasks in the future. This should give you the intuition of how Dropout works. Now let us see mathematically about how dropout ensembles.

(Refer Slide Time: 34:05)

### Dropout for Single Non-Linear Unit

- Consider a logistic sigmoidal function on input  $x$ :

$$o = \sigma(x) = \frac{1}{1 + ce^{-\lambda x}} \quad c \geq 0$$



Vineeth N B (IIT-H) 34.4 Regularization



So, let us consider a single example. Let us consider a logistic sigmoidal function on an input  $x$ . So, the output is equal to  $\sigma(x)$ , which for the moment, let us define it as something like this, let us call it  $\frac{1}{1+ce^{-\lambda x}}$  where  $c \geq 0$ . So, this is a valid logistic sigmoidal function.

(Refer Slide Time: 34:30)

### Dropout for Single Non-Linear Unit

- Consider a logistic sigmoidal function on input  $x$ :

$$o = \sigma(x) = \frac{1}{1 + ce^{-\lambda x}} \quad c \geq 0$$

- $2^n$  possible sub-networks indexed by  $k$



Vineeth N B (IIT-H) 34.4 Regularization



Let us assume now that there are  $2^n$  possible sub networks, which are indexed by  $k$ .

(Refer Slide Time: 34:41)

### Dropout for Single Non-Linear Unit

- Consider a logistic sigmoidal function on input  $x$ :

$$o = \sigma(x) = \frac{1}{1 + ce^{-\lambda x}} \quad c >= 0$$

- $2^n$  possible sub-networks indexed by  $k$
- Geometric mean of outputs from all sub-networks:  $G = \prod_k o_k^{\frac{1}{2^n}}$



Vineeth N B. (IIT-H) 54.4 Regularization



$k$  is the index we could use for covering all of those to power  $m$  sub Networks. Let us define the geometric mean of the outputs from all of these sub networks as  $G$ . This is your geometric mean formula for all those  $k$  models. Remember,  $k$  is indexing  $2^n$  possible sub networks for all  $2^n$  possible sub network models.  $G$  is the geometric mean of the outputs.

(Refer Slide Time: 35:03)

### Dropout for Single Non-Linear Unit

- Consider a logistic sigmoidal function on input  $x$ :

$$o = \sigma(x) = \frac{1}{1 + ce^{-\lambda x}} \quad c >= 0$$

- $2^n$  possible sub-networks indexed by  $k$
- Geometric mean of outputs from all sub-networks:  $G = \prod_k o_k^{\frac{1}{2^n}}$
- Geometric mean of complementary outputs:  $G' = \prod_k (1 - o_k)^{\frac{1}{2^n}}$



Vineeth N B. (IIT-H) 54.4 Regularization



Similarly, you could also get the geometric mean of the complimentary output instead of  $o_k$ . You can also take  $1$  minus,  $o_k$ . So, if you have a sigmoid, it is going to give you a value between  $0$

and 1. So, if you get this is typically used for binary classification setting, where if your output is 0.6, you are telling that this is probability with a probability of 0.6, you are saying it is class 1. But that also means the probability of class 2 is 0.4. That is the information that we are using here to get the complimentary outputs geometric mean.

(Refer Slide Time: 35:41)

### Dropout for Single Non-Linear Unit

- Consider a logistic sigmoidal function on input  $x$ :  

$$o = \sigma(x) = \frac{1}{1 + ce^{-\lambda x}} \quad c > 0$$
- $2^n$  possible sub-networks indexed by  $k$
- Geometric mean of outputs from all sub-networks:  $G = \prod_k o_k^{\frac{1}{2^n}}$
- Geometric mean of complementary outputs:  
 $G' = \prod_k (1 - o_k)^{\frac{1}{2^n}}$

Normalized geometric mean  $NGM = \frac{G}{G+G'}$



Hence:


$$NGM = \frac{1}{1 + \left( \prod_k \frac{1 - \sigma(x_k)}{\sigma(x_k)} \right)^{\frac{1}{2^n}}}$$

$$= \frac{1}{1 + \left( \prod_k e^{-\lambda x_k} \right)^{\frac{1}{2^n}}} \left( \frac{1 - \sigma(x)}{\sigma(x)} = ce^{-\lambda x} \right)$$

$$= \frac{1}{1 + c [e^{-\lambda \sum_k x_k / 2^n}]} = \sigma(\mathbb{E}[x])$$

where  $\mathbb{E}[x] = \sum_k x_k / 2^n$

Vineeth N B. (IIT-H)
54.4 Regularization


The normalized geometric mean now is given by  $G/(G+G')$  prime, where  $G$  and  $G'$  are as defined here, this can be written as this will turn out to be  $1/(1+G'/G)$ , and substituting for  $G'$  and  $G$ , which are given here and substituting for  $o$ , you get this term here. So,  $G'$  by  $G$  is  $(1 - O_k)/O_k$ . and you have the product that goes outside. And instead of  $O_k$ , you replace it as  $\sigma(x)$  and instead of  $1 - O_k$  you replace it by  $1 - \sigma(x)$ .

That is how you get the first term there. Now,  $(1 - \sigma(x))/\sigma(x)$  is equal to  $c e^{-\lambda x}$ . we are going to, let us leave that as an exercise for you to work out. But that comes by simple substitution of terms here using  $\sigma(x)$  here to substitute those terms, you will find that this is the answer. So, that is what we are replacing here. Now, this product can be converted, converted to a summation inside the exponential term. Remember,  $e^a \cdot e^b = e^{a+b}$ . Using that idea, this product can be converted to a summation inside the  $e$  power term. And that is how you go from here to here.

(Refer Slide Time: 35:14)

### Dropout for Single Non-Linear Unit

- Consider a logistic sigmoidal function on input  $x$ :
 
$$o = \sigma(x) = \frac{1}{1 + ce^{-\lambda x}} \quad c > 0$$
- Normalized geometric mean  $NGM = \frac{G}{G+G'}$   $\frac{1}{1+G'}$ 

Hence:



$$NGM = \frac{1}{1 + \left( \prod_k \frac{1 - \sigma(x_k)}{\sigma(x_k)} \right)^{\frac{1}{2^n}}}$$


$$= \frac{1}{1 + \left( \prod_k ce^{-\lambda x_k} \right)^{\frac{1}{2^n}}} \left( \frac{1 - \sigma(x)}{\sigma(x)} = ce^{-\lambda x} \right)$$

$$= \frac{1}{1 + c[e^{-\lambda \sum_k x_k / 2^n}]} = \sigma(\mathbb{E}[x])$$

$NGM(\sigma(x)) = \sigma(\mathbb{E}[x])$  where  $\mathbb{E}[x] = \sum_k x_k / 2^n$

$e^a \cdot e^b = e^{a+b}$
- $2^n$  possible sub-networks indexed by  $k$
- Geometric mean of outputs from all sub-networks:  $G = \prod_k o_k^{\frac{1}{2^n}}$
- Geometric mean of complementary outputs:  $G' = \prod_k (1 - o_k)^{\frac{1}{2^n}}$



And now, let us define this, this if you look at this carefully, this looks very similar to the original definition of  $\sigma$  just the input  $x$  is different. So, this is given by  $\sigma(\mathbb{E}(x))$ , that expectation over  $x$  is this term inside. But what is this telling us? You could now summarize this as the normalized geometric mean of  $\sigma(x)$  can be written as  $\sigma(\mathbb{E}(x))$ . So, why is this important? We are saying now that if you know sample these  $2^n$  models in subway, probably you consider only half of them at a particular point in time, simply multiplying the values of  $x$  by half will give you the same value.

(Refer Slide Time: 38:15)

## Dropout for Single Non-Linear Unit

- Consider a logistic sigmoidal function on input  $x$ :

$$o = \sigma(x) = \frac{1}{1 + ce^{-\lambda x}} \quad c > 0$$

- $2^n$  possible sub-networks indexed by  $k$
- Geometric mean of outputs from all sub-networks:  $G = \prod_k o_k^{\frac{1}{2^n}}$
- Geometric mean of complementary outputs:  $G' = \prod_k (1 - o_k)^{\frac{1}{2^n}}$

- Normalized geometric mean  $NGM = \frac{G}{G+G'}$ .

Hence:

$$\begin{aligned} NGM &= \frac{1}{1 + \left(\prod_k \frac{1 - \sigma(x_k)}{\sigma(x_k)}\right)^{\frac{1}{2^n}}} \\ &= \frac{1}{1 + \left(\prod_k ce^{-\lambda x_k}\right)^{\frac{1}{2^n}}} \left(\dots \frac{1 - \sigma(x)}{\sigma(x)} = ce^{-\lambda x}\right) \\ &= \frac{1}{1 + c[e^{-\lambda \sum_k x_k / 2^n}]} = \sigma(\mathbb{E}[x]) \end{aligned}$$

$$\text{where } \mathbb{E}[x] = \sum_k x_k / 2^n$$

NGM is equivalent to output of overall network with weights divided by two

Vineeth N B. (IIT-H) 54.4 Regularization



So, we say that the NGM or the normalized geometric mean is equivalent to the output of the overall Network with weights divided by 2. Rather, this tells us that doing dropout by dropping nodes in each layer with a certain probability becomes equivalent to taking the overall Network and multiplying the activations by the same probability at test time. And this makes this ensemble easy to work in each mini batch iteration, you drop a few neurons. And then at the end of training, you really are not taking  $2^n$  different models and averaging, you are using just one model, you are multiplying each layers outputs by  $p$ , and you are done with your ensemble result. That is why it is a powerful tool.

(Refer Slide Time: 39:08)

## Homework



### Readings

- Deep Learning book, [Chapter 7](#), Sections 7.1-7.5, 7.8, 7.12
- [Tutorial on Dropout](#) for more information (Optional)

### Exercise

- Show that adding Gaussian noise (with zero mean) to input is equivalent to  $L_2$  weight decay when loss function is MSE.

Vineeth N B. (IIT-H)

§4.4 Regularization









With that, your homework for this lecture is Chapter 7, Sections as given here. And if you would like a tutorial on Dropout, to understand it better, please look at this link. And as we already said earlier, your exercise for this lecture is show that adding Gaussian noise with 0 mean to the input is equivalent to  $L_2$  weight decay. When your loss function is mean squared errors. Give it a try.



(Refer Slide Time: 39:38)

## References I

-  Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580.
-  Pierre Baldi and Peter Sadowski. "The dropout learning algorithm". In: *Artificial intelligence* 210 (2014), pp. 78–122.
-  Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.
-  David Warde-Farley et al. "An empirical analysis of dropout in piecewise linear networks". In: *CoRR* abs/1312.6197 (2014).
-  Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
-  Terrance DeVries and Graham W Taylor. "Improved Regularization of Convolutional Neural Networks with Cutout". In: *arXiv preprint arXiv:1708.04552* (2017).



Vineeth N B. (IIT-H)

§4.4 Regularization



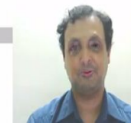
## References II

-  Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. 2018.
-  Ghodsi, Ali, *STAT 946 - Deep Learning (Fall 2015)*. URL: [https://uwaterloo.ca/data-analytics/sites/ca.data-analytics/files/uploads/files/f15\\_stat946\\_deep\\_learning\\_outline\\_1.pdf](https://uwaterloo.ca/data-analytics/sites/ca.data-analytics/files/uploads/files/f15_stat946_deep_learning_outline_1.pdf) (visited on 06/05/2020).
-  Khapra, Mitesh M., *CS 7015 - Deep Learning (Spring 2019)*. URL: <https://www.cse.iitm.ac.in/~miteshk/CS7015.html> (visited on 06/03/2020).



Vineeth N B. (IIT-H)

§4.4 Regularization



Here are references.