1 November, 2016

Neural Networks

Course 4: Making the neural network resistent to overfitting

Overview

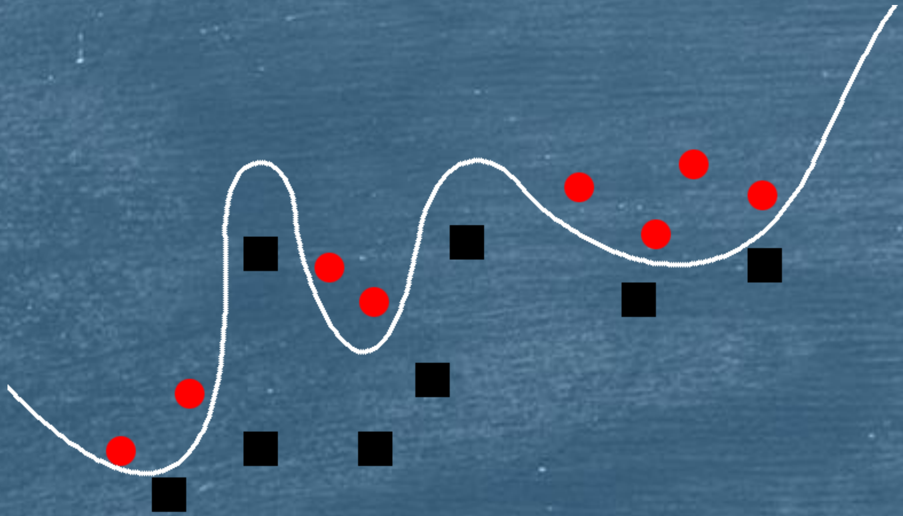
- Overfitting
- Regularization
- Dropout
- Max Norm Constraint
- Increase Dataset
- Conclusions

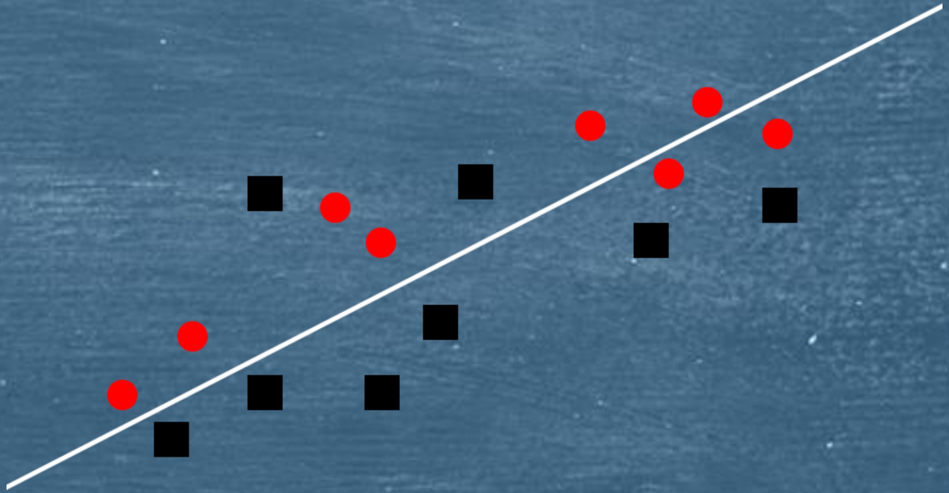
Overfitting

Overfitting

- Overfitting describes the process that happens when a model customizes itself too much to the data and does not generalize it. It tries to memorize is and not generalize on it

- Which of the following models is better?





Overfitting

- Experiment:
- We will train a neural network with the same architecture from the last course (784, 36, 10), but this time we will use a training data of 1000 images instead of 50000.

- We will use the same learning rate (0.3), the same mini batch size (10) and we will use cross-entropy as our cost function.

- But since we use a smaller dataset, we will train it more iterations: 500 instead of 30.

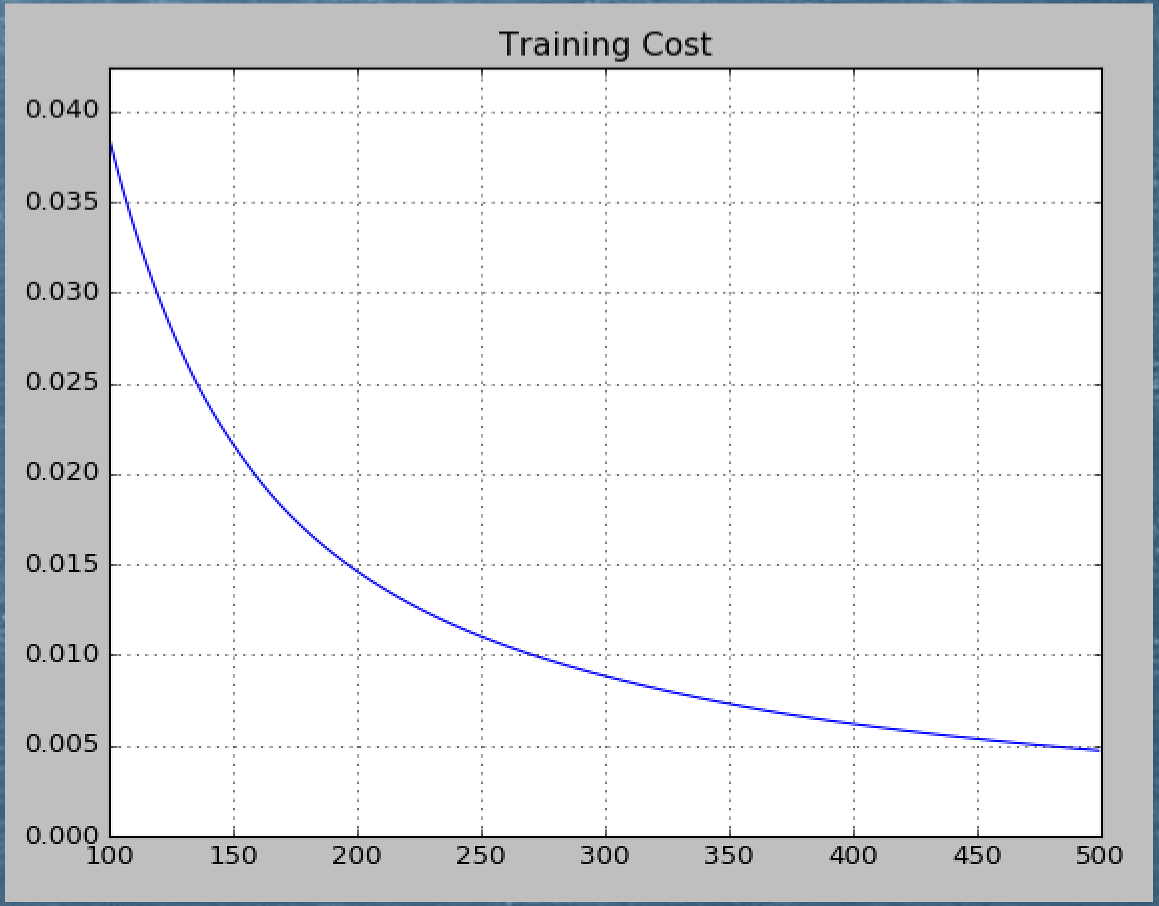
- (So nothing changes in the network but the training set size and the number of iterations)

Overfitting

Cost of training data

The cost of the training data decreases as we have expected

At the end of the training cycle, the error is very small (0.005)



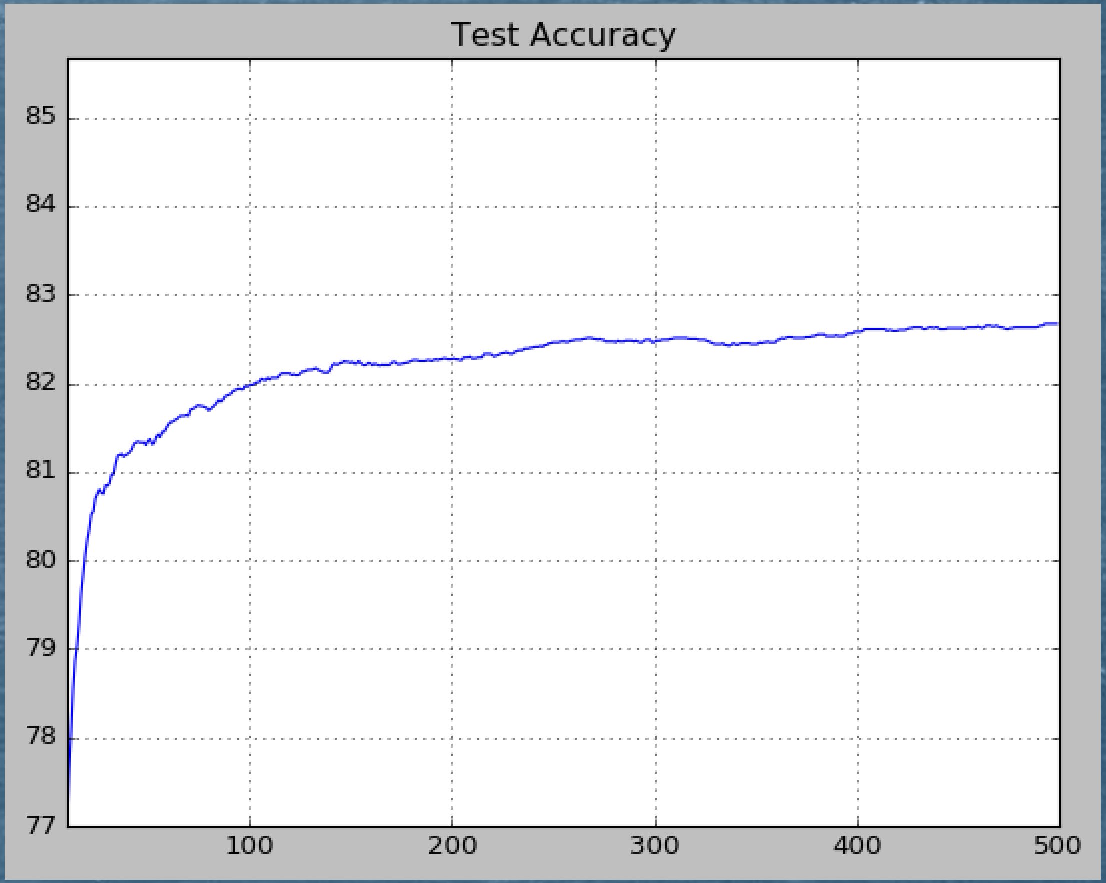
Overfitting

Accuracy on testing data

As it can be seen in the graph, the best accuracy is achieved at about iteration 270. After that it just fluctuates around the same point

Beyond that point, the network is overtraining or overfitting

We should have stopped at iteration 280

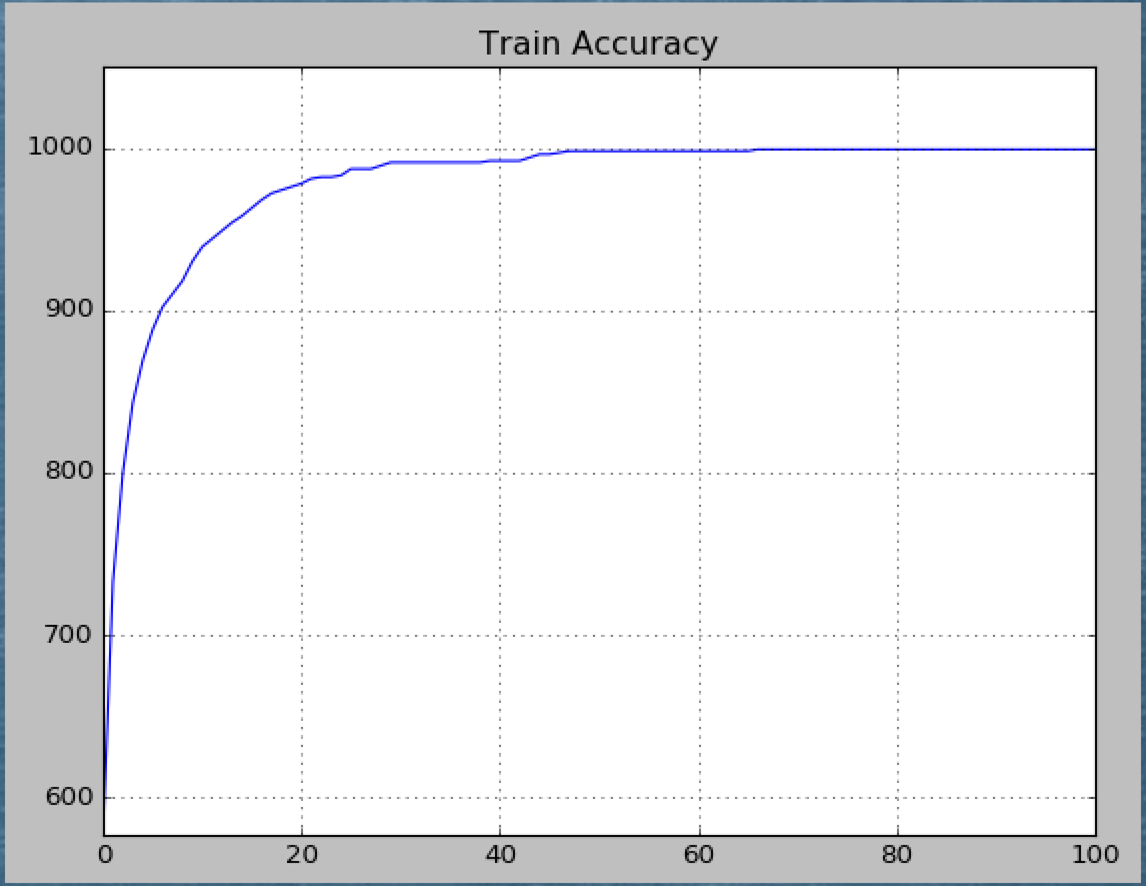


Overfitting

Accuracy on training data

And if we check the accuracy on the training data, from about iteration 42 is 1000 (100%).

That means we should have correctly identify all the numbers



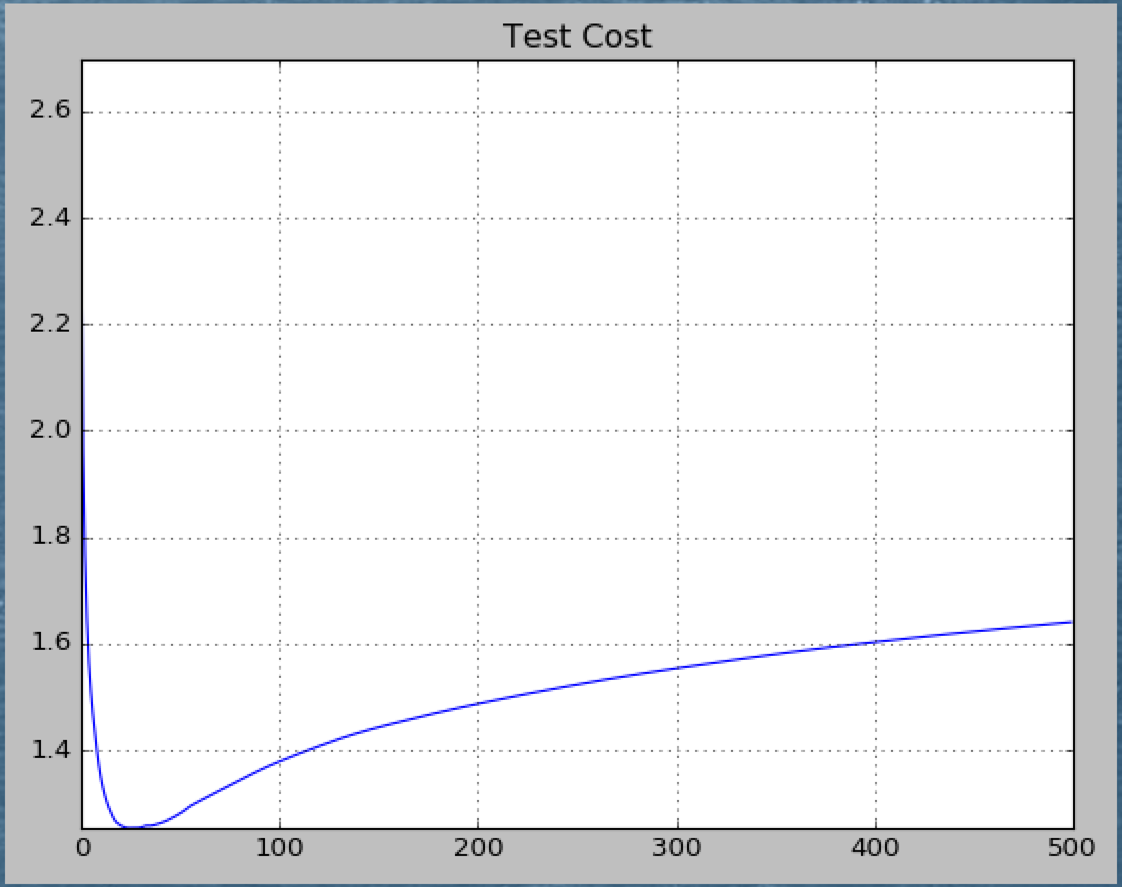
Overfitting

Cost of testing data

The cost of the testing data decreases until around iteration 20, after which it starts increasing

This shouldn't have happened if the model correctly approximated the data.

There should have been a continuous decrease in testing cost



Overfitting

- This clearly shows that the ANN learns particularities of the data and does not generalize.
- The main reason for why this happens is that the training data is to small compared to the size of the network.
- Our network has: 784*36 + 36*10 weights and 36+10 biases. That means it has 28584 parameters that it can use in order to model 1000 elements.
- How does the same network behave on the larger dataset (50000)

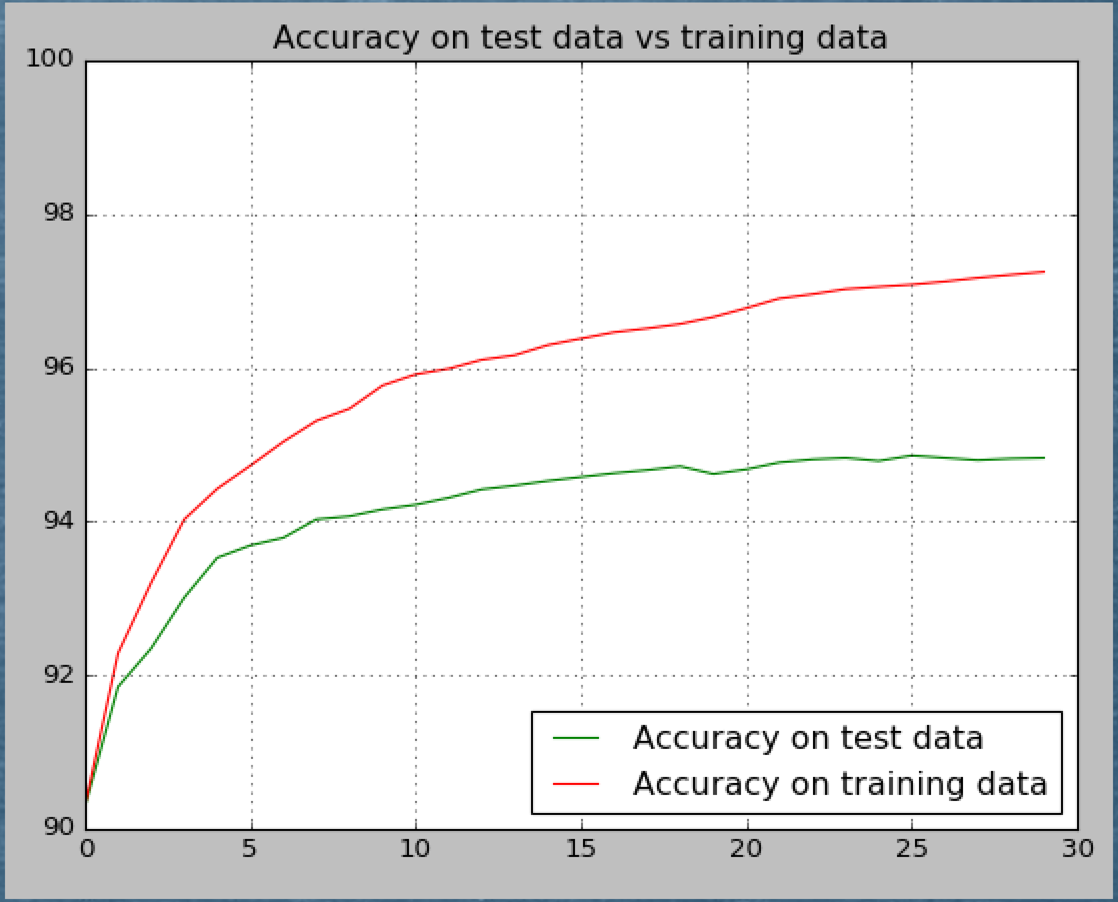
Overfitting

Accuracy on test data vs training data

Overfitting still happens here, but to lower degree.

Beyond iteration 7, the network isn't learning anymore.

However, the difference between the two accuracies is not that much. Only 2.42 %



Solutions for Overfitting

Solutions for Overfitting

- A good way to detect that you are not overtraining is to compare the accuracy on training data and accuracy on test data. These should be close with the accuracy on training being slightly higher

- So, increasing the training data does help to solve overfitting. With very large datasets, it becomes very difficult for a neural network to overfit.

- Another obvious solution would be to reduce the network size

- The solutions above are sometimes impractical:
  - Can not get more training data
  - It may take much too long to train on a larger data set
  - Large networks tend to be more powerful than smaller ones

Solutions for Overfitting: Regularization

- Understanding: bias and variance

- **Bias**:

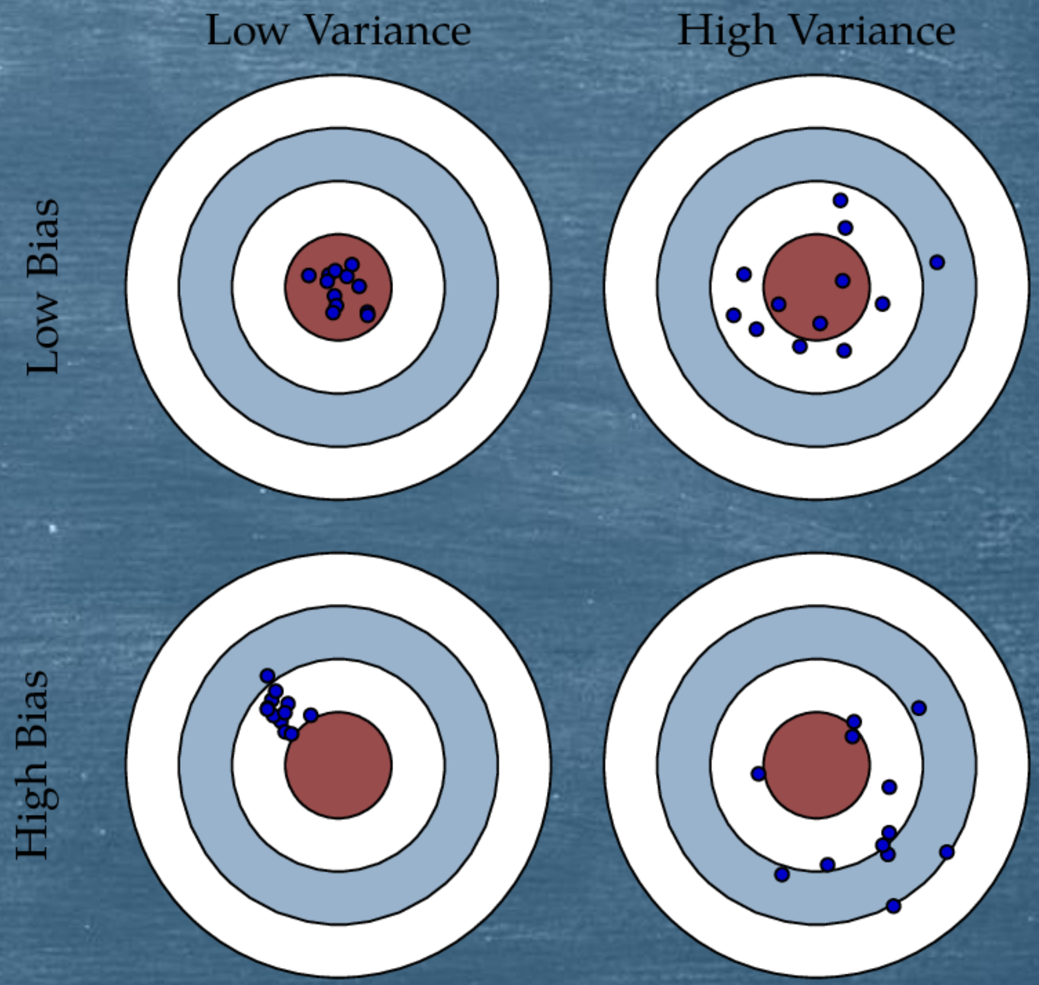
  The bias measures how far are the estimated elements (their average) from the target
- **Variance:**

  The variance measures how spread are the estimated elements from their mean

Solutions for Overfitting: Regularization

- Let's suppose there are some darts players who want to hit the center of the target (approximate a point) and each players shoots many darts.

- This is how the target would look depending on how biased are the estimated shoots from the target or how spread are they (variance).



Solutions for Overfitting: Regularization

- It can be shown, that the Mean Squared Error is actually making a tradeoff between variance and bias.

- Let's say we have some points (x, y) and there is a relation between x and y such that $y = f(x)$. We want to find the function f so we will try to approximate it, by g.

- In this case:
  - $bias(g(x)) = E[g(x)] - f(x)$
  - $var(g) = E[(g(x) - E[g(x)])^2]$
  - $MSE_D(f) = E[\left(g(x) - f(x)\right)^2]$

  Where $E = expected\ value\ (mean)$

Solutions for Overfitting: Regularization

- We will use the following lema:

  $$\mathrm{E}[\mathrm{X}^2] = E[(X - E[X])^2] + E[X]^2$$

- Proof:

  $$E[(X - E[X])^2] = E\left[X^2 + E[X]^2 - 2XE[X]\right] = E[X^2] + E[E[X]^2] - 2E[XE[X]] =$$

  $$=E[X^2] + E[X]^2 - 2E[X]^2$$

  $$+E[X]^2$$

$$= E[X^2]$$

Solutions for Overfitting: Regularization

$$E\left[\left(g(x)-f(x)\right)^2\right] =$$

$$= E[g(x)^2 - 2g(x)f(x) + f(x)^2]$$

$$= E[g(x)^2] - 2E\left[g(x)f(x)\right] + E[f(x)^2]]$$

$$= \ E[(g(x) - E[g(x)])^2] + E[g(x)]^2 - 2E[g(x)f(x)] + E[(f(x) - E[f(x)])^2] + E[f(x)]^2 =$$

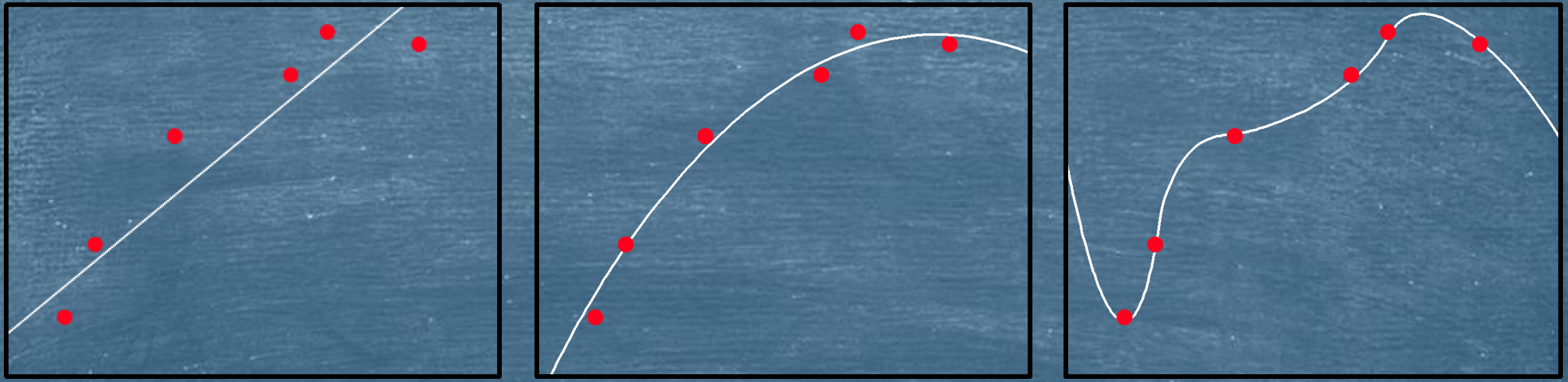
$$= E[(g(x) - E[g(x)])^2] + \left(E[g(x)] - f(x)\right)^2 + E\left[\left(f(x) - E[f(x)]]\right)^2\right] =$$

$$= variance + bias^2 + \sigma^2(noise)$$

Solutions for Overfitting: Regularization

- $MSE(f) = bias^2 + variance + \sigma^2(noise)$
- What this tells us is that for a value of the error we have to tradeoff between variance and bias (since the noise doesn't depend on the estimator)
- Both high bias and high variance are negative properties for the model. A model with high bias, tends to underfit the data while a model with high variance tends to overfit
- The purpose is to find an equilibrium of those two.

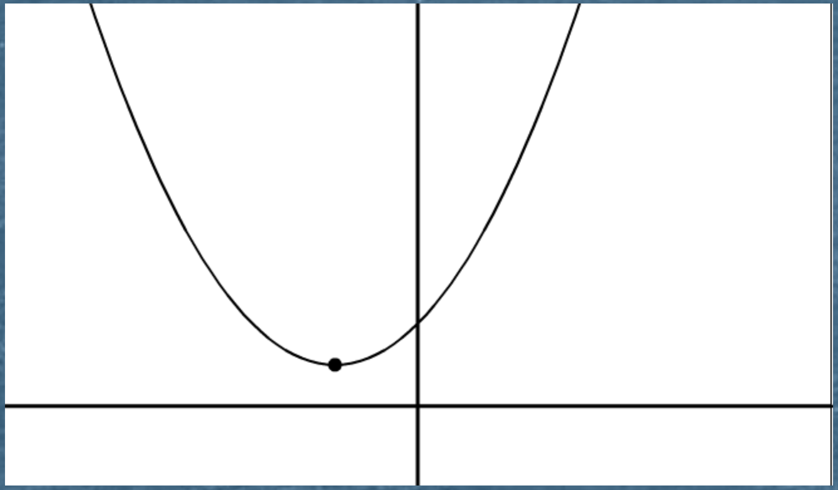
Solutions for Overfitting: Regularization



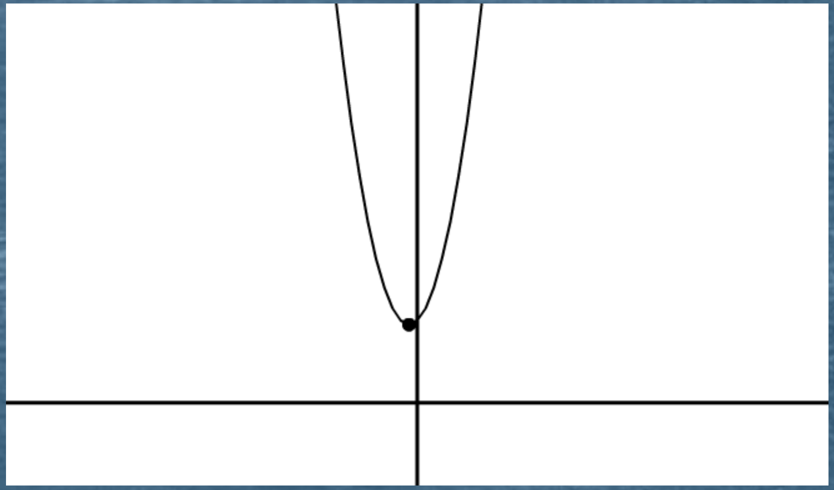
- The model on the left underfits the data. It has high bias (big distance from the targets), but low variance. If we move one point, the line would probably not change very much
- The model in the right, overfits the data. It has low bias (there is almost an exact match of the target points) but it has high variance. If we move one point there will probably be a big change in the model
- The model in the center is the right one since it has lower bias than the one on the left and a lower variance from the one from the right

Solutions for Overfitting: Regularization

- In order for a model to have high variance, it needs two things:
  - A large number of weights (for example, the line in the first image has less weights then the polynomial in the right image)
  - Large values for the weights. This is needed in order to be able to make rapid shifts. Think of a parabola $(ax^2 + bx + c)$ The greater a is, the more narrow the parabola will be



$0.5x^2 + x + 1$



$5x^2 + x + 1$

Solutions for Overfitting: Regularization

- So, in order to make the function smooth (low variance) we need to have low weights (even zero).

- We modify the cost function by introducing an element that penalizes large weights.

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \qquad where\ C_0\ is\ the\ original\ costfunction$$

$$and\ \lambda\ is\ called\ the\ regularization\ parameter$$

So, for the cross entropy will look like this:

$$C = \frac{1}{n} \Sigma_{xj} [t_j lny_j^L + (1 - t_j) \ln(1 - y_j^L)] + \frac{\lambda}{2n} \Sigma_w w^2$$

And the Mean Square Error (Quadratic cost):

$$C = \frac{1}{2n} \Sigma_{xj} (t_j - y_j^L)^2 + \frac{\lambda}{2n} \Sigma_w w^2$$

Solutions for Overfitting: Regularization

- The λ (regularization parameter) in $C = C_0 + \frac{\lambda}{2n}\sum_w w^2$ controls how much bias vs variance we want.
  - If $\lambda = 0$, then we have the standard Cost Function
  - The bigger λ is, the more bias (and less variance) we want. If λ is too big, we can go into under fitting
  - This is called L2 regularization and it is also known as weight decay

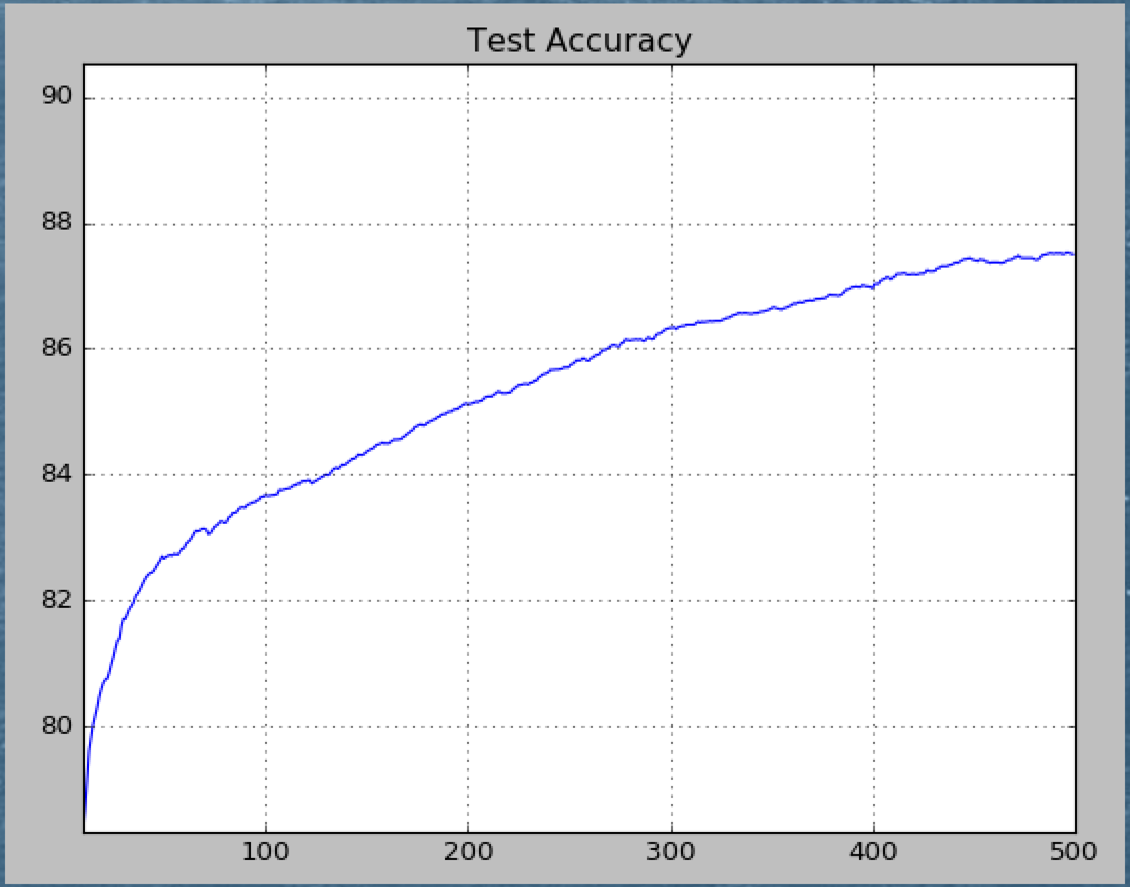
Solutions for Overfitting: Regularization

- The easiest way to introduce regularization into backpropagation is to adjust how the cost varies in respect to the weights: $\frac{\partial C}{\partial w}$ (since the weights are the only ones affected)
- $\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$
- $w = w - \eta \frac{\partial C_0}{\partial w} - \eta \frac{\lambda}{n} w = \left(1 - \eta \frac{\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$
- Since we'll most likely use SGD, the equation will use is:

$$w = \left(1 - \eta \frac{\lambda}{n}\right) w - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial w}$$

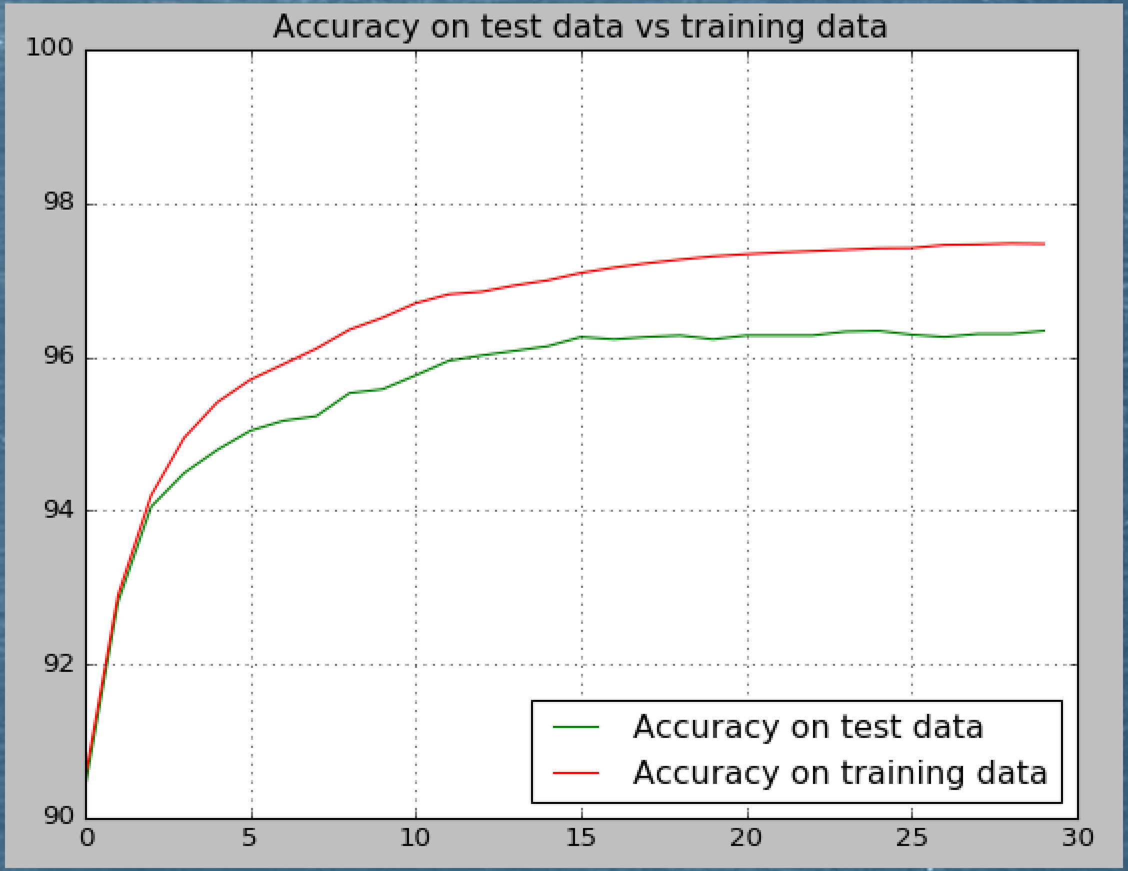
Solutions for Overfitting: Regularization

- Let's train the ANN again on the 1000 training set, but this time with regularization. $\lambda = 0.1$
- As it can be seen, the test accuracy continues to increase (even after iteration 300).
- We have also achieved a greater accuracy than before 87.5 vs 82.6



Solutions for Overfitting: Regularization

- The same experiment, but this time using the entire data set, but using $\lambda = 5$
- The difference between the test accuracy and training accuracy is also smaller (1.13%)
- The overall accuracy is greater than before (96.34% vs 94.86%



Solutions for Overfitting: Regularization

- Another variant of regularization is L1 where instead of using squares of the weights, we're using absolute values

$$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

In order to use this in our backpropagation algorithm, we must first discover how to adjust the cost

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} sgn(w) \qquad \textit{where } sgn(w) \textit{is the sign of } w$$

$$w = w - \eta\frac{\lambda}{\mathrm{n}} sgn(w) - \eta\frac{\partial C_0}{\partial w}$$

Or, if we use SGD:

$$w = w - \eta\frac{\lambda}{\mathrm{n}} sgn(w) - \frac{\eta}{m}\sum_x \frac{\partial C_0}{\partial w}$$

Solutions for Overfitting: Regularization

- So, how is L1 different from L2?
  - L1 $w = w - \eta \frac{\lambda}{\mathrm{n}} sgn(w) - \eta \frac{\partial C_0}{\partial w}$
  - L2 $w = \left(1 - \eta \frac{\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$

L1 drives the weights down by a constant amount:

- that means if some weights are very big, they won't be reduced too much
- if some weights are small, they will probably be driven to zero

L2 drives the weights down by an amount that depends on the value of w:

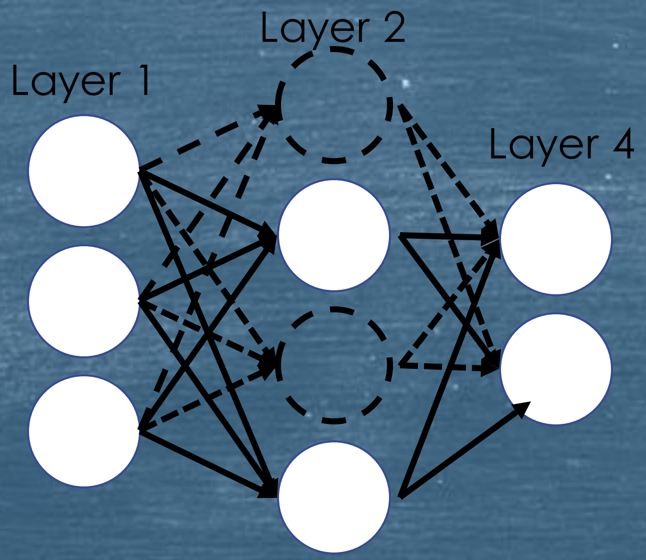
- for large weights, this means large amount of decrease
- for small weights, this means a small mount of decrease

L1 tends to build the model on certain weights while L2 uses more weights but with smaller values

Solutions for Overfitting: Dropout

- The idea of this technique is to make the network use less weights during training

- On each minibatch that we train, we randomly select ½ from the hidden neurons and make them invisible (of course, including the weights that go into or out of them)



Solutions for Overfitting: Dropout

- We update the visible weights as before, using backpropagation
- We restore the invisible hidden neurons and their connection
- We then select another minibatch where we randomly select another ½ from the hidden neurons that we will make them invisible

After we have finished the training, we restore all the hidden neurons and halve the hidden weights from the hidden neurons to the output neurons

Solutions for Overfitting: Dropout

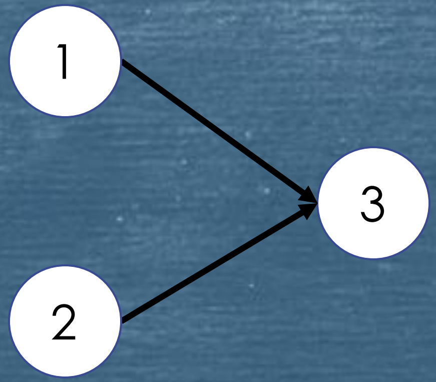
- If neuron 1 outputs the right value 80% of time
- If neuron 2 outputs only random value

Then neuron 3 will most likely compute:

$$activation1 * 1 + activation2 \ * 0$$

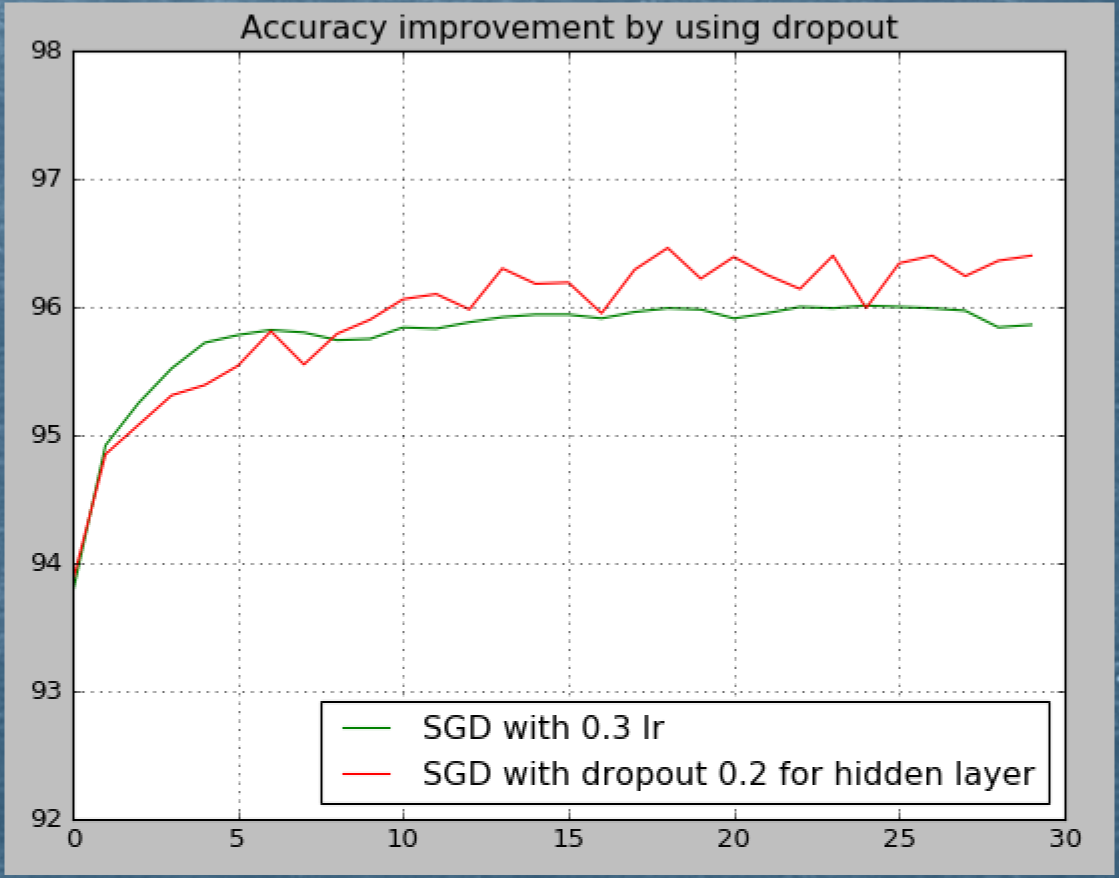
The gradients will get propagated only through neurons 3 and 1

This means the network uses less than its potential (since neuron 2 will never be used)



Solutions for Overfitting: Dropout

- Improvement of the accuracy by using 20% dropout in the hidden layer when used on the entire dataset



Solutions for Overfitting: Dropout

Other Advantages:

- makes the network robust to the loss of any individual neuron
- Makes it more efficient since more neurons will participate in learning

How it is usually implemented:

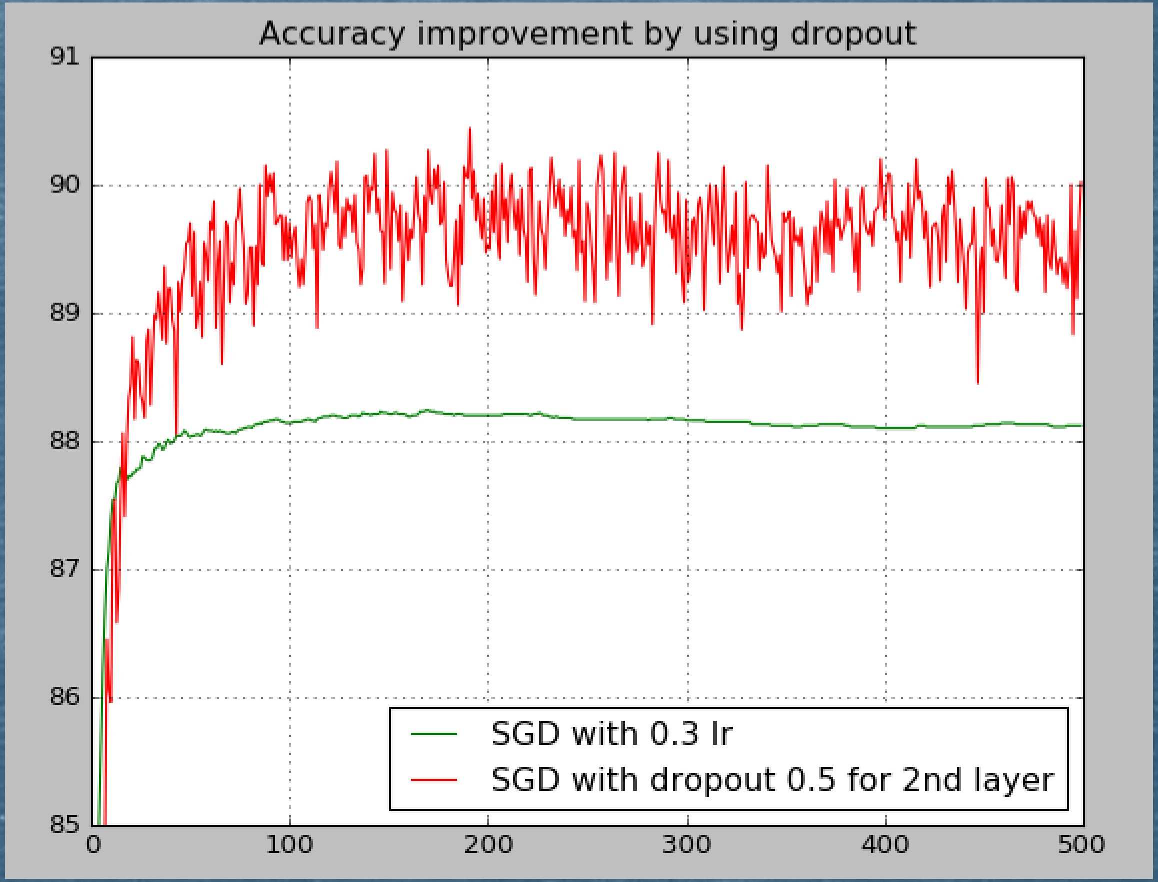
During training:

1. Given the n activations from the previous layer, generate n numbers of 1 (Bernoulli distribution) with a probability of (1-p)

2. Multiply the activations with the generated vector (element-wise) and also multiply by $\frac{1}{1-p}$

Solutions for Overfitting: Dropout

- Improvement of the accuracy by using
- 50% dropout in the hidden layer,
- On a dataset of just 1000 elements



Solutions for Overfitting: Maxnorm

Maxnorm tries to limit large weights, but on each individual neuron.

1. The weight updates are performed as usual

2. If $||\mathrm{w}||_2 = \sqrt{\sum_i w_i^2} >$ c, then

    3. adjust the weights:

$$w = w * \frac{c}{||w||_2 + \epsilon},$$

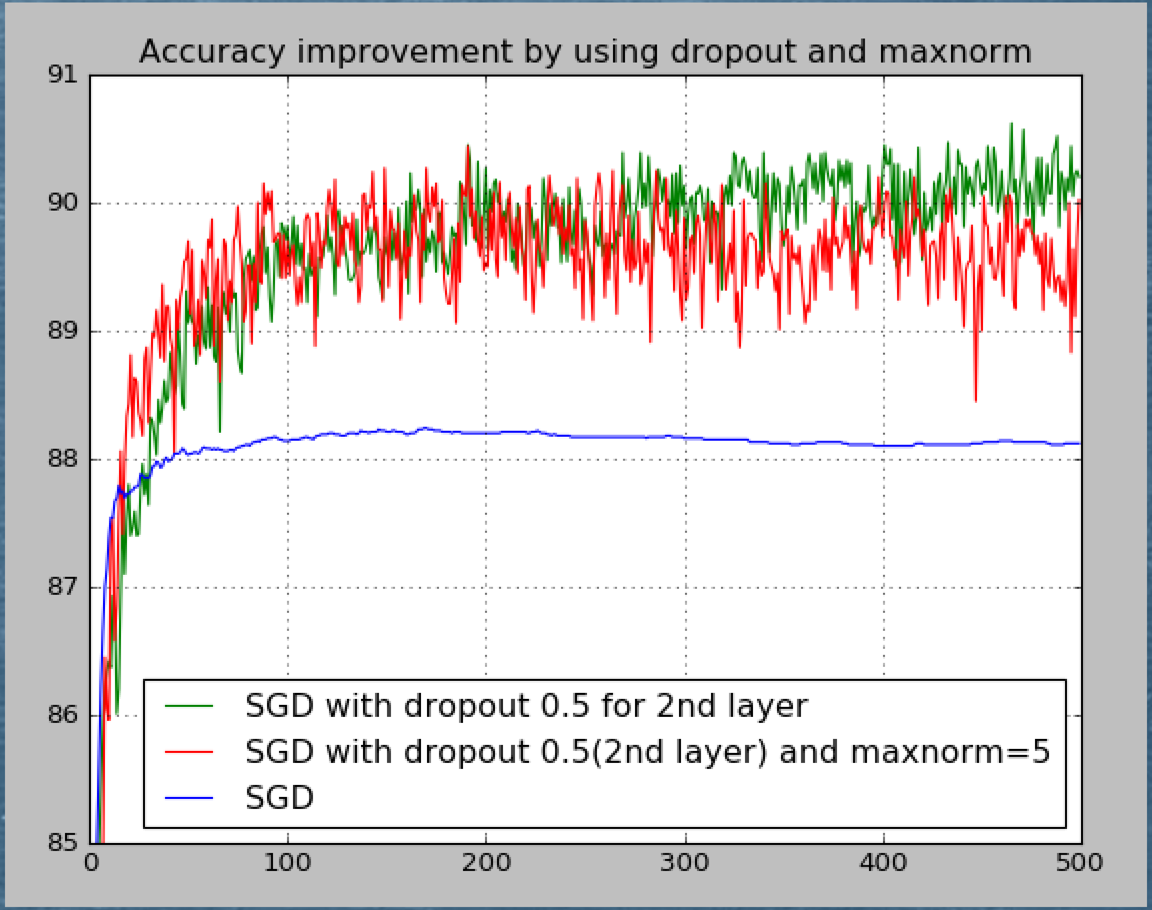
$$where\ \epsilon = 10^{-8}\ (a\ small\ value\ to\ avoid\ division\ by\ zero)$$

Maxnorm allows the usage of a bigger learning rate (since the weights will not explode)

Solutions for Overfitting: Maxnorm

- Improvement of the accuracy, on a dataset of just 1000 elements, by using :

- 50% dropout in the hidden layer
- Maxnorm of 5 for hidden layer + 50% dropout



Solutions for Overfitting: Increase Dataset

As we've seen earlier, when using a bigger dataset, our network didn't overfit that bad

So, another idea is to increase the data.

Since, sometimes it is difficult to find training data, a solution is to add small noise to our existing data set using different functions, like slightly rotating the image

Solutions for Overfitting: Increase Dataset

In "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis, by Patrice Simard, Dave Steinkraus, and John Platt (2003) the authors were using a neural network with 800 hidden neurons to classify MNIST digits.

They achieved an increase of accuracy from 98.4% to 98.9% by:

- Rotations
- Translating
- Skewing the image

By using "elastic distortions" (that should emulate random oscillations found in human muscles) they increased the accuracy to 99.3%

Questions & Discussion

Demo

Framework created by Andrej Karpathy
http://cs.stanford.edu/

Bibliography

http://neuralnetworksanddeeplearning.com/

http://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html

Nitish Srivastava , Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: “A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research 15, (https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf)

http://blog.fliptop.com/blog/2015/03/02/bias-variance-and-overfitting-machine-learning-overview/