Neural networks

Regularization

The problem of overfitting

With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.
John von Neumann

- Large number of parameters can describe whatever the data is.
- The goal is not memorizing th training data, but generalization of new examples
- Von Neumann was worried about models with 4 parameters. Do not tell him about our MNIST classifier with thousands of parameters!

Strategies to overfitting

- Increase the amount of data
  - More data adds variability and complicate the memorization process
  - Nevertheless, it is difficult to get data in general
- Reduce the size of the network
  - This can be a contradiction, large networks are preferred because of their potential and power

- How do we avoid overfitting without more data and keeping the size of the model?

Regularization

L_2 Regularization

- Also called weight decay
- It adds an extra term in the cost function

$$\hat{C} = C_0 + \frac{\lambda}{2n}\sum_{w} w^2$$

Where

- C_0 is the original cost function (MSE or cross-entropy loss)
- λ is the regularization parameter
- n is the size of the mini-batch

L_2 Regularization

$$\hat{C} = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

- The effect is to make the network learn small weights
- The derivative is straightforward to compute

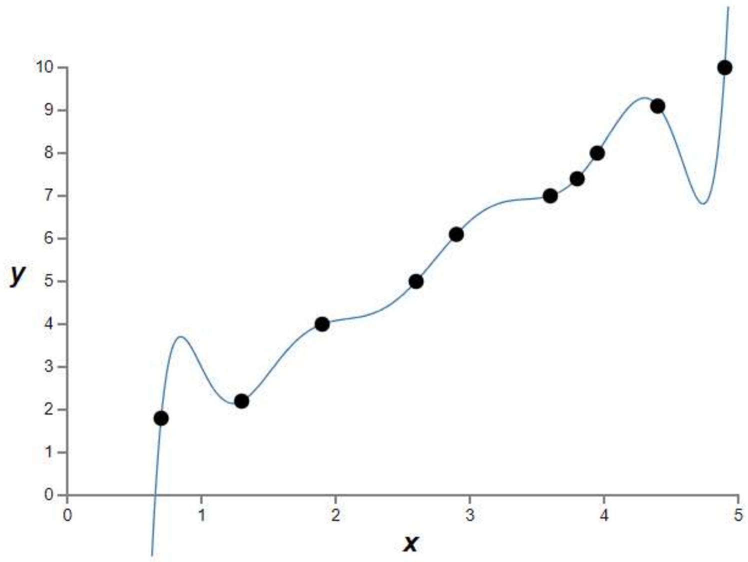
$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

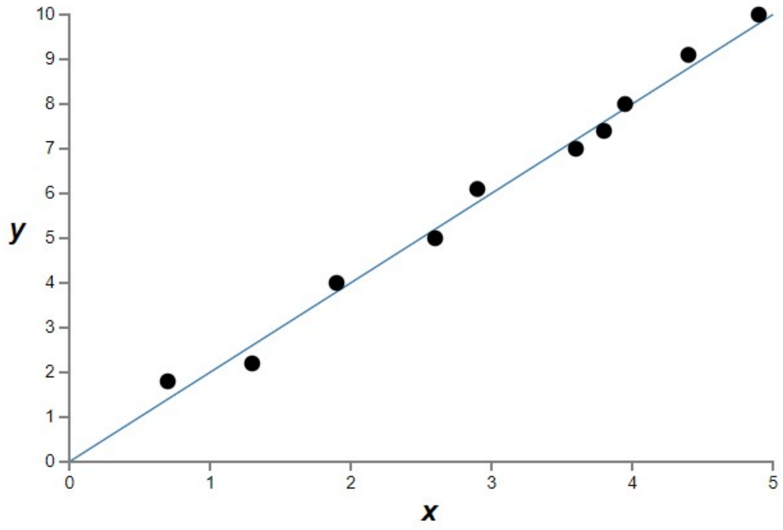
- The update rule will be

$$w = w - \eta\left(\frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w\right) \quad \Longrightarrow \quad w = \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$$

Weight decay

L_2 Regularization





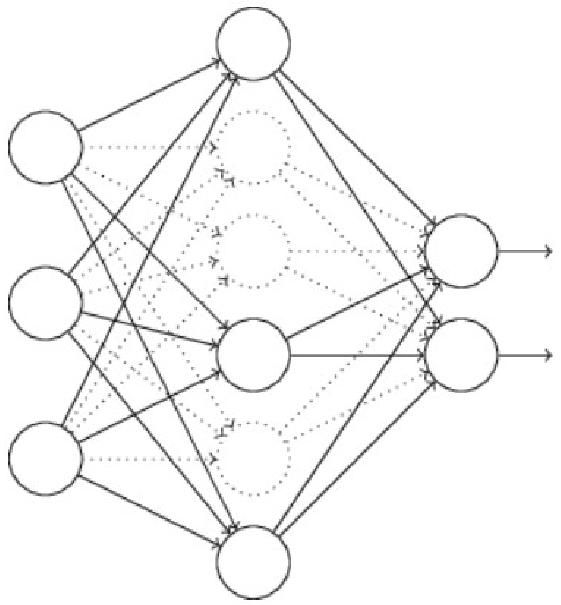
Which model describes better the data?

For prediction purposes, the right model is more resilient to noisy inputs

Weight decay helps to build simpler models, hence reducing overfitting

Dropout

- Simple strategy
  - In every run of our network, drop some neurons in the hidden layers with probability p



The effect of randomly dropping some neurons is like averaging the responses of many networks
In every run we are training a different neural network.

Data augmentation

- More data always reduce overfitting, because we introduce variability
- A priori knowledge needs to be used in order to introduce meaningful augmentation
- In the case of images, typical transformation are
  - Small rotations
  - Flipping
  - Random crops

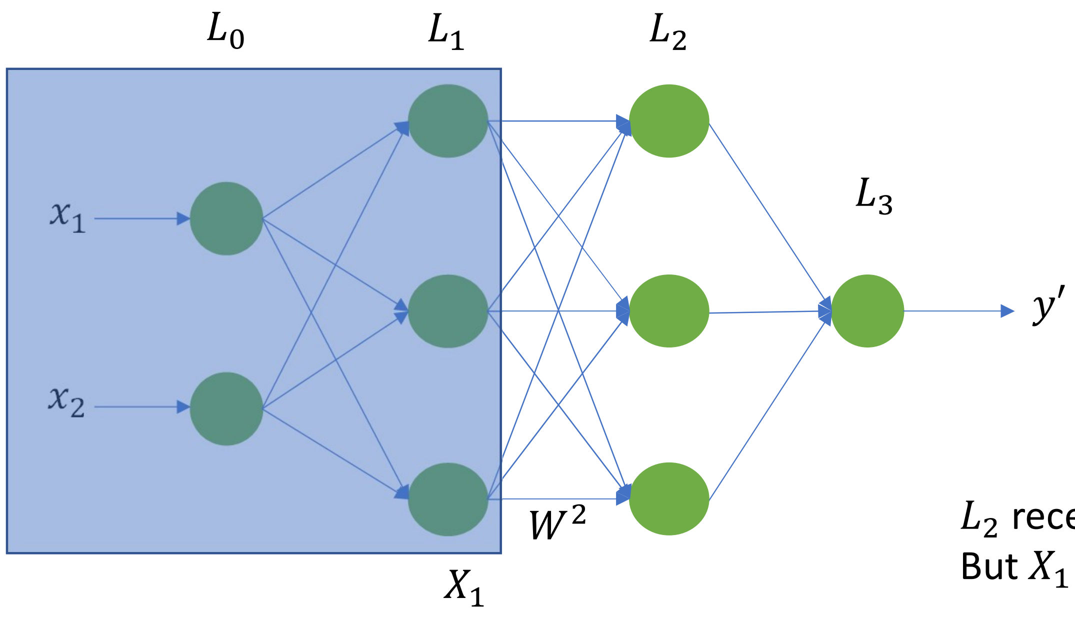
Summary

- Avoid overfitting is a major problem in Deep models
- We have alternatives: regularization, dropout, data agumentation

Neural networks

Bonus Track: Batch normalization

Introduction



The computation in L_2 depends on

$$X_2 = \sigma(X_1 \cdot W^2 + b^2)$$

and

$$X_1 = \sigma(X_0 \cdot W^1 + b^1)$$

L_2 receives the data X_1, so L_2 learns the distribution of X_1
But X_1 changes in every iteration!

In each epoch, a layer has to learn a new distribution: covariate shift

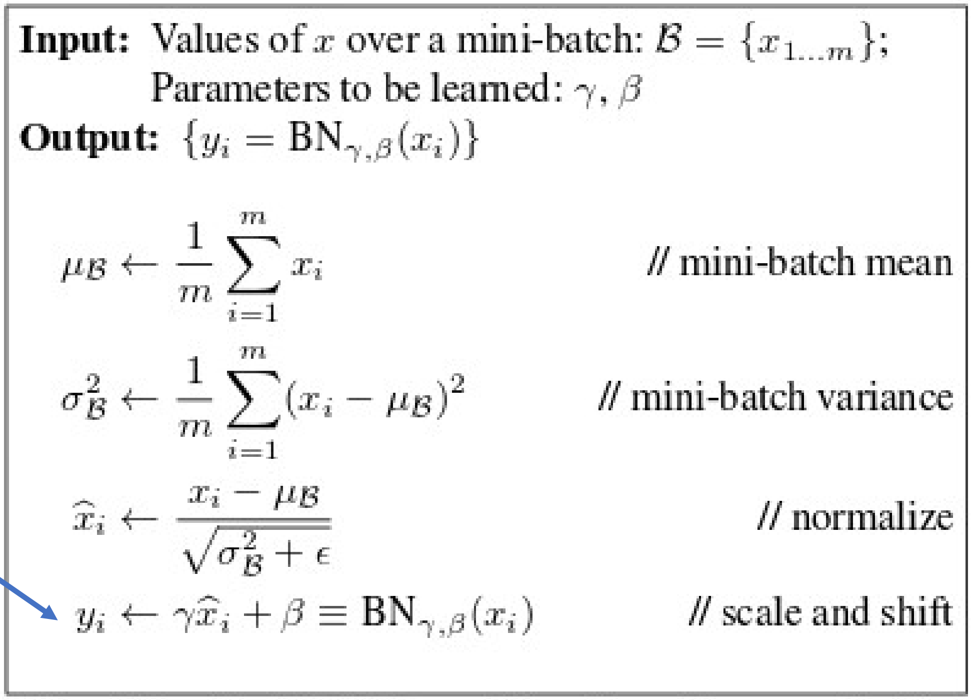
The consequence is a slow convergence during the optimization

Batch normalization

Batch normalization: normalize the activation of the previous layer to have zero mean and unit variance

The normalization is on the batch

After normalization, we need to give the network the opportunity of correcting the distribution if needed



```
Input:  Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
        Parameters to be learned: $\gamma$, $\beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$                              // mini-batch mean
    $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$          // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$                      // normalize
    $y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$               // scale and shift
```

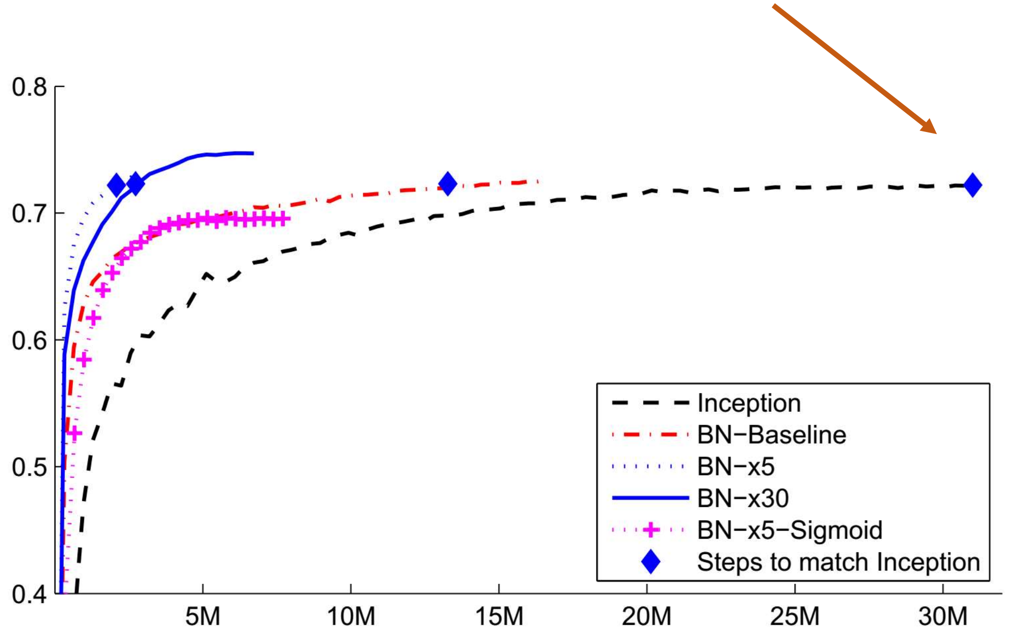
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch normalization

As we have better convergence, we can increase the learning rate

- A baseline CNN took 30M iterations to get 72.2% of accuracy for classification

- BN with the same learning rate: 13M to reach 72.2%
- BN with 5*learning_rate: 2M to reach 72.2%
- BN with 30*learning_rate: 2.7M to reach 72.2%


5M
10M
15M
20M
25M
30M
Inception
BN−Baseline
BN−x5
BN−x30
BN−x5−Sigmoid
Steps to match Inception