String Theory meets Machine Learning - Regularization and CNNs

Robin Schneider

Uppsala University

November 2020

Learning Hodge numbers

A physics problem

- There are 7890 distinct Complete Intersection Calabi Yau manifolds
- Described by configuration matrices

$$\mathcal{M} = \left[\begin{array}{c||ccc} n_0 & p_1^0 & \cdots & p_K^0 \\ \vdots & \vdots & \ddots & \vdots \\ n_r & p_1^r & \cdots & p_K^r \end{array} \right]_{\chi}^{h^{(1,1)}, h^{(2,1)}} . \tag{1}$$

- Want to learn Hodge numbers. We use a fully connected neural network for that.

Dense NN and CICYlist

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
m1flatten (Flatten)          (None, 180)               0
_________________________________________________________________
m1layer0 (Dense)             (None, 128)               23168
_________________________________________________________________
m1layer1 (Dense)             (None, 128)               16512
_________________________________________________________________
m1layer2 (Dense)             (None, 128)               16512
_________________________________________________________________
m1output (Dense)             (None, 19)                2451
=================================================================
Total params: 58,643
Trainable params: 58,643
Non-trainable params: 0
_________________________________________________________________
```

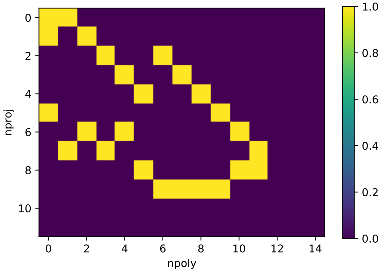


Figure: *On the left fully connected NN and on the right CICY with index 150 visualized as a 2d image.*

Overfitting, low bias high variance

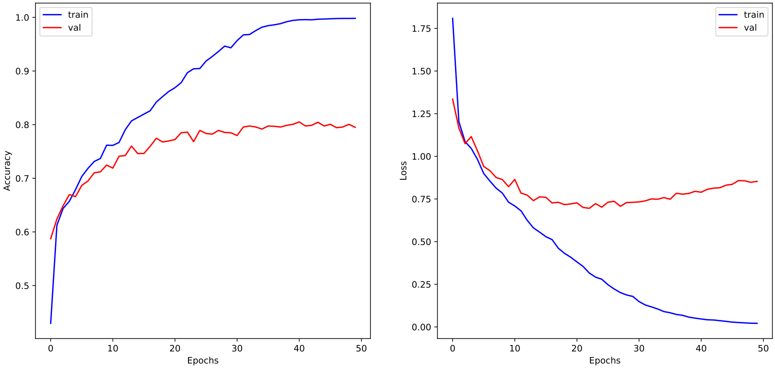


Figure: *Loss and accuracy plot of a fully connected NN learning $h^{1,1}$ of CICYs.*

Bias and Variance

Assume the true data of our model follows

$$y = f(x;\theta) + \epsilon \qquad \text{with } \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2). \tag{2}$$

Bias: $\mathbb{E}_D[f(x;\hat{\theta}_D)] - f(x;\theta)$.
Variance: $\mathbb{E}_D[(f(x;\hat{\theta}_D) - \mathbb{E}_D[f(x;\hat{\theta}_D)])^2]$
What does the expected error look like?

Bias and Variance

Assume the true data of our model follows

$$y = f(x;\theta) + \epsilon \qquad \text{with } \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2). \tag{2}$$

Bias: $\mathbb{E}_D[f(x;\hat{\theta}_D)] - f(x;\theta)$.
Variance: $\mathbb{E}_D[(f(x;\hat{\theta}_D) - \mathbb{E}_D[f(x;\hat{\theta}_D)])^2]$
What does the expected error look like?

$$\begin{aligned}\mathbb{E}_{D,\epsilon}[J] &= \mathbb{E}_{D,\epsilon}\left[\sum_{i=1}^{n}(y_i - f(x_i;\hat{\theta}_D))^2\right] \\ &= \sum_i^n \Bigg[\underbrace{\sigma_\epsilon^2}_{\text{Noise}} + \underbrace{(\mathbb{E}_D[f(x_i;\hat{\theta}_D)] - f(x_i;\theta))^2}_{\text{Bias}^2} + \\ &\qquad \underbrace{\mathbb{E}_D[(f(x_i;\hat{\theta}_D) - \mathbb{E}_D[f(x_i;\hat{\theta}_D)])^2]}_{\text{Variance}}\Bigg]\end{aligned} \tag{3}$$

Bias and Variance tradeoff

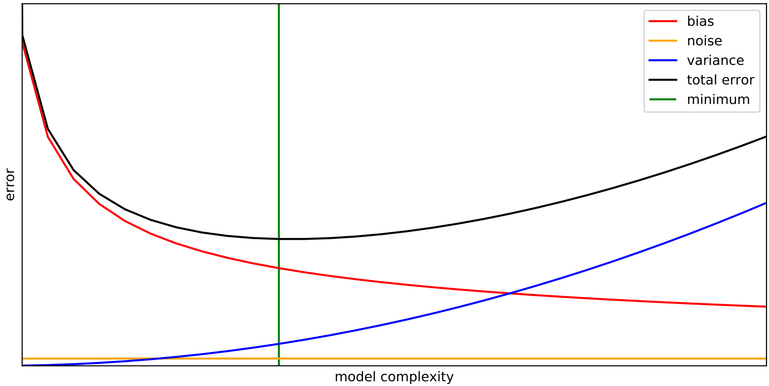


Figure: *Model error and its decomposition as model complexity grows.*

Regularization

How to decrease overfitting?

Regularization

How to decrease overfitting?
Introduce **regularization**. Simplest way is to add penalty term to cost function

$$J_{\text{total}}(\theta) = J_{\text{reg}}(\theta) + \lambda J_{\text{penalty}}(\theta). \tag{4}$$

Usually the penalty term takes the form

$$J_{\text{penalty}}(\theta) = \frac{1}{2} \sum_i |\theta_i|^q. \tag{5}$$

L2 - Ridge, weight decay

$q = 2$: L2 (LASSO, weight decay) regression.
Assume Gaussian prior $p(\theta) = \mathcal{N}(\theta|0, \alpha^{-1})$ and likelihood with precision parameter β^{-1}. Then maximize with respect to log posterior (recall Bayes theorem: $p(\theta|D) \propto p(D|\theta)p(\theta)$):

$$\hat{\theta} = \arg\max_{\theta}(\log(p(D|\theta)p(\theta))) \tag{6}$$

L2 - Ridge, weight decay

$q = 2$: L2 (LASSO, weight decay) regression.
Assume Gaussian prior $p(\theta) = \mathcal{N}(\theta|0, \alpha^{-1})$ and likelihood with precision parameter β^{-1}. Then maximize with respect to log posterior (recall Bayes theorem: $p(\theta|D) \propto p(D|\theta)p(\theta)$):

$$\hat{\theta} = \arg\max_{\theta}(\log(p(D|\theta)p(\theta))) \tag{6}$$

which results in the following loss

$$J(\theta) = \frac{\beta}{2}\sum_{i=1}^{n}(f(x_n;\theta) - y_n)^2 + \frac{\alpha}{2}\theta^2 \tag{7}$$

such that $\lambda = \frac{\alpha}{\beta}$. One can solve this and finds that each component shrinks by $\hat{\theta}^{L2} = \frac{d_j^2}{d_j^2+\lambda}\hat{\theta}^{LS}$.

L1 - LASSO, sparse

$q = 1$: L1 (LASSO, Sparse) regularization with loss function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(x_n;\theta) - y_n)^2 + \frac{\lambda}{2}|\theta| \tag{8}$$

is equivalent to linear regression with the additional condition

$$\hat{\theta} = \arg\min_{\theta}((f(x;\theta) - y)^2) \text{ and } \lambda \geq |\theta| \tag{9}$$

We can't solve this exactly.

L1 - LASSO, sparse

$q = 1$: L1 (LASSO, Sparse) regularization with loss function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(x_n;\theta) - y_n)^2 + \frac{\lambda}{2}|\theta| \tag{8}$$

is equivalent to linear regression with the additional condition

$$\hat{\theta} = \arg\min_{\theta}((f(x;\theta) - y)^2) \text{ and } \lambda \geq |\theta| \tag{9}$$

We can't solve this exactly. However assuming x is orthogonal one can analyze it using subgradient methods to find

$$\hat{\theta}^{LASSO} = \text{sign}(\hat{\theta}^{LS})(|\hat{\theta}^{LS}| - \lambda)_+ \tag{10}$$

$\hat{\theta}^{LS}$ optimal least square fit value. The subscript $+$ denotes positive part. Thus λ determines a threshold which sets small parameters to zero.

L1 and L2 in a picture

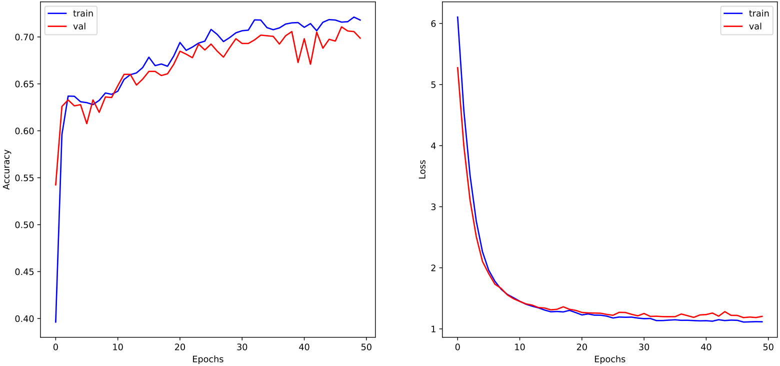


Figure: *Accuracy and loss plot of a fully connected neural network learning $h^{1,1}$ of CICYs with l1, l2 values of* $(0.001, 0.001)$.

Dropout

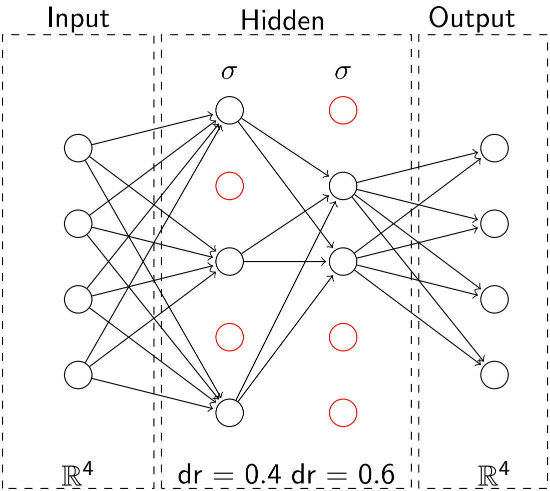


Figure: *A fully connected Neural Network of a classification problem. In red dropped nodes during training.*

Dropout in a picture

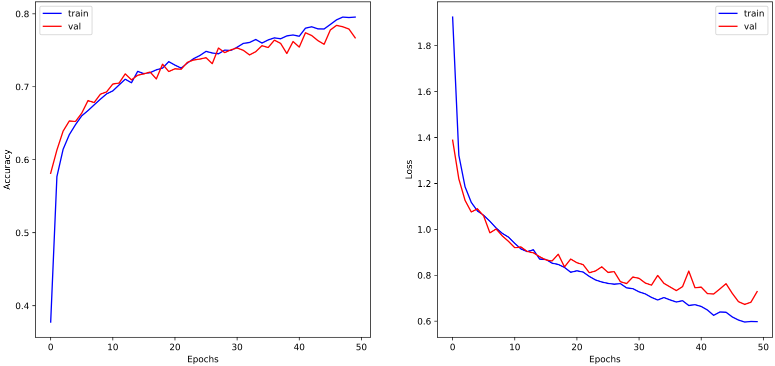


Figure: *Accuracy and loss plot of a fully connected neural network learning* $h^{1,1}$ *of CICYs with a dropout rate of* 0.2.

Convolutional Neural Networks

CNNs gained wide popularity with the publishing of AlexNET which won the ImageNet competition in 2012

- Higher accuracy (less prone to overfitting; AlexNET had $> 10\%$ accuracy over runner up)
- less parameters (e.g. 3mio weights for single dense neuron in 1000x1000px)
- More natural (learns features and filters of the image)

CNN

kernel

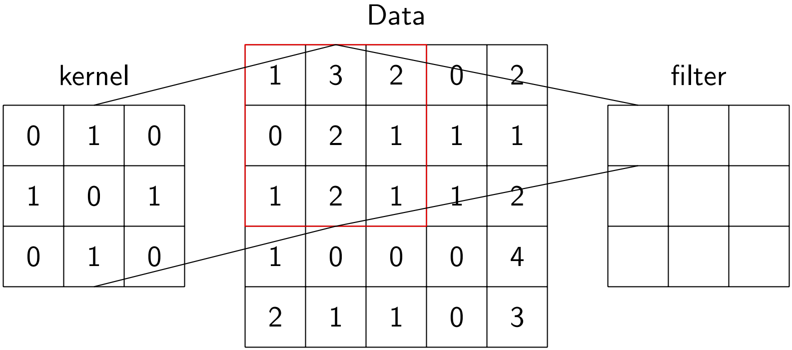
0	1	0
1	0	1
0	1	0

Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

- First, define kernel shape
- Depth is the number of kernels scanning over the data
- Stride is the scan length
- Padding is number of zeros cols/rows added to the boundaries

CNN


Data
kernel
filter
0 1 0
1 0 1
0 1 0
1 3 2 0 2
0 2 1 1 1
1 2 1 1 2
1 0 0 0 4
2 1 1 0 3

CNN

kernel

0	1	0
1	0	1
0	1	0

Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

filter

6		

CNN

kernel

0	1	0
1	0	1
0	1	0

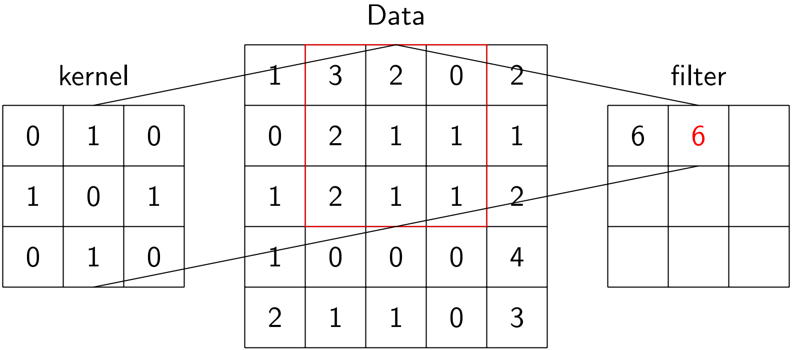
Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

filter

6		

CNN


Data
kernel
filter
0 1 0
1 0 1
0 1 0
1 3 2 0 2
0 2 1 1 1
1 2 1 1 2
1 0 0 0 4
2 1 1 0 3
6 6

CNN

Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

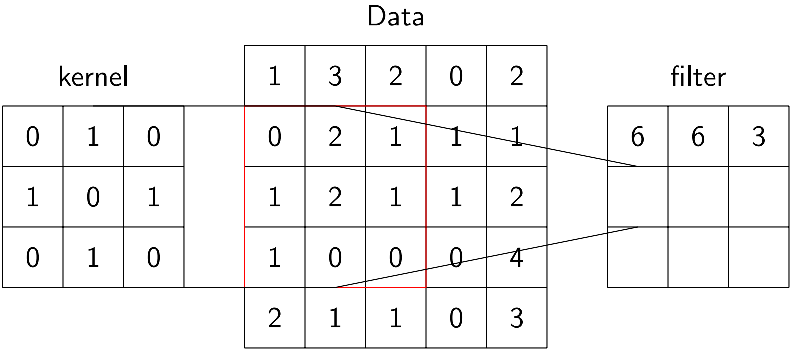
kernel

0	1	0
1	0	1
0	1	0

filter

6	6	3

CNN


Data
kernel
filter
1 3 2 0 2
0 1 0
0 2 1 1 1
6 6 3
1 0 1
1 2 1 1 2
0 1 0
1 0 0 0 4
2 1 1 0 3

CNN

kernel

0	1	0
1	0	1
0	1	0

Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

filter

6	6	3
4		

CNN

Data

kernel

0	1	0
1	0	1
0	1	0

Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

filter

6	6	3
4	4	4
4	2	5

Pooling

Variations:

- Max pooling - returns max value
- Average pooling - returns mean value

Similar to convolutional blocks:

- Define, pooling size.
- Define, stride length.
- Define, padding.

Pooling

Variations:

- Max pooling - returns max value
- Average pooling - returns mean value

Similar to convolutional blocks:

- Define, pooling size.
- Define, stride length.
- Define, padding.

Advantages:

- Reduces dimension
- Preserves translational invariance

CNN + Pooling

Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

filter

6	6	3
4	4	4
4	2	5

max pooling

Figure: *Combined 3x3 convolutional kernel and 2x2 max pooling with zero padding and stride of one.*

CNN + Pooling

Data

1	3	2	0	2
0	2	1	1	1
1	2	1	1	2
1	0	0	0	4
2	1	1	0	3

filter

6	6	3
4	4	4
4	2	5

max pooling

6	6
4	5

Figure: *Combined 3x3 convolutional kernel and 2x2 max pooling with zero padding and stride of one.*

Inception Architectures

GoogleNet [1409.4842] state of the art algorithm in 2014 at ImageNet.
Utilizes Inception blocks:

- Have several convolutional blocks with different kernels in parallel
- Concatenate results

Inception Architectures

GoogleNet [1409.4842] state of the art algorithm in 2014 at ImageNet. Utilizes Inception blocks:

- Have several convolutional blocks with different kernels in parallel
- Concatenate results
- (Optional) Use pooling
- (Optional) Use batch normalization [1502.03167], such that mean activation is about 0 and variance close to 1:

$$\hat{x}^k = \frac{x^k - \mu_B^k}{\sqrt{(\sigma_B^k)^2 + \epsilon}} \tag{11}$$

Then transform

$$y^k = \gamma^k \hat{x}^k + \beta^k \tag{12}$$

Inception Block

```
Model: "inception"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              [(None, 12, 15, 1)]  0
__________________________________________________________________________________________________
b0_12x1_conv (Conv2D)           (None, 12, 15, 32)   384         input[0][0]
__________________________________________________________________________________________________
b0_1x15_conv (Conv2D)           (None, 12, 15, 32)   480         input[0][0]
__________________________________________________________________________________________________
b0_12x1_bn (BatchNormalization) (None, 12, 15, 32)   96          b0_12x1_conv[0][0]
__________________________________________________________________________________________________
b0_1x15_bn (BatchNormalization) (None, 12, 15, 32)   96          b0_1x15_conv[0][0]
__________________________________________________________________________________________________
b0_12x1 (Activation)            (None, 12, 15, 32)   0           b0_12x1_bn[0][0]
__________________________________________________________________________________________________
b0_1x15 (Activation)            (None, 12, 15, 32)   0           b0_1x15_bn[0][0]
__________________________________________________________________________________________________
block0 (Concatenate)            (None, 12, 15, 64)   0           b0_12x1[0][0]
                                                                 b0_1x15[0][0]
__________________________________________________________________________________________________
```

Figure: *Inception block applied to learning CICY hodge numbers.*

GoogleNet

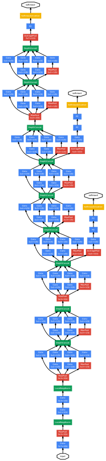


Figure 3: GoogLeNet network with all the bells and whistles

7

Application: Learning Hodge numbers

Reproduce some results of the early literature. We will predict Hodge numbers of Complete Intersection Calabi Yau 3-folds

- Kernel methods and dense NN by He et al. [1706.02714,1806.03121,1903.03113]
- Using Inception architecture by Erbin and Finotello [2007.13379]
- Recently four folds were investigated by He and Lukas [2009.02544]