Gradient regularization in discriminative and generative models

Dániel Varga

November 29, 2017

Hungarian Academy of Sciences
Alfréd Rényi Institute of Mathematics

Background

Regularization

Regularization: Any change to the model that makes it generalize better to unseen data.

- L2 weight decay (Plaut et al 1986)
- dropout (Srivastava et al 2014)
- label smoothing (Szegedy et al 2015)
- entropy regularization (Pereyra et al 2017)
- ...and hundreds of others

Data loss

Unregularized training: we are looking for optimal network parameters

$$\operatorname*{argmin}_{\Theta} L(f_\Theta(x), y)$$

where

- Θ are the network parameters,
- $f_\Theta(x)$ is the output of the network on input x,
- y is the expected output on input x.
- $L(\hat{y}, y)$ is the loss function that quantifies how much we like the network's output, the smaller the better.
- $(x, y) \sim (X, Y)$ is sampled from our training data.

Regularization term

Now we want to regularize, that is, improve generalization to unseen data. One standard class of methods works by adding a regularization term to the data loss that penalizes network settings that we want to avoid:

$$\underset{\Theta}{\operatorname{argmin}}[L(f_\Theta(x), y) + \lambda L_{reg}(\Theta)]$$

Weight decay

The archetypal regularization term, from 1986 but still widely used, is the *L2 weight decay*:

$$L_{reg}(\Theta) = \|\Theta\|_2^2$$

It punishes large network weights (strong neural connections), which helps to avoid memorizing superficial correlations.

Remark: For a single dense linear layer it is equal to the squared Frobenius norm of the weight matrix.

Method

Smoothness prior

- The core idea behind our project is to enforce a **smoothness** property of the neural network via a well-chosen regularization term.
- Tiny perturbations of the network input should not lead to large changes in network output.

Smoothness prior

- We formalize smoothness in several ways, leading to our *Spectral Regularizer*, that approximates the Frobenius norm of the Jacobian of the input-output mapping at the training examples:

$$L_{SpectReg}(x, \Theta) = \| \frac{\partial}{\partial x} f_\Theta(x) \|_F^2 = \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I^m)} [\| \epsilon^T \frac{\partial}{\partial x} f_\Theta(x) \|_2^2]$$

- and the *DataGrad* regularizer that penalizes large changes of the loss function at the training examples:

$$L_{DataGrad}(x, y, \Theta) = \| \frac{\partial}{\partial x} L(f_\Theta(x), y) \|_2^2$$

DataGrad

For lack of time we will only talk about DataGrad in this talk. To the best of our knowledge, the Spectral Regularizer is our contribution, but DataGrad has a more complex background story.

DataGrad

$$L_{DataGrad}(x, y, \Theta) = \| \frac{\partial}{\partial x} L(f_\Theta(x), y) \|_2^2$$

- The *DataGrad* regularization term was introduced by (Ororbia et al 2016).
- Intuitively, DataGrad penalizes large changes in the performance of the classifier at the training examples.
- Interestingly, the authors do not analyze or even implement this method, arguably making this a case of *flag-planting*.
- Instead, they work with an finite-difference approximation, based on *adversarial sampling*, and demonstrate that it can make models more robust to adversarial sampling.

DataGrad

In relation to (Ororbia et al 2016), our contributions are twofold:

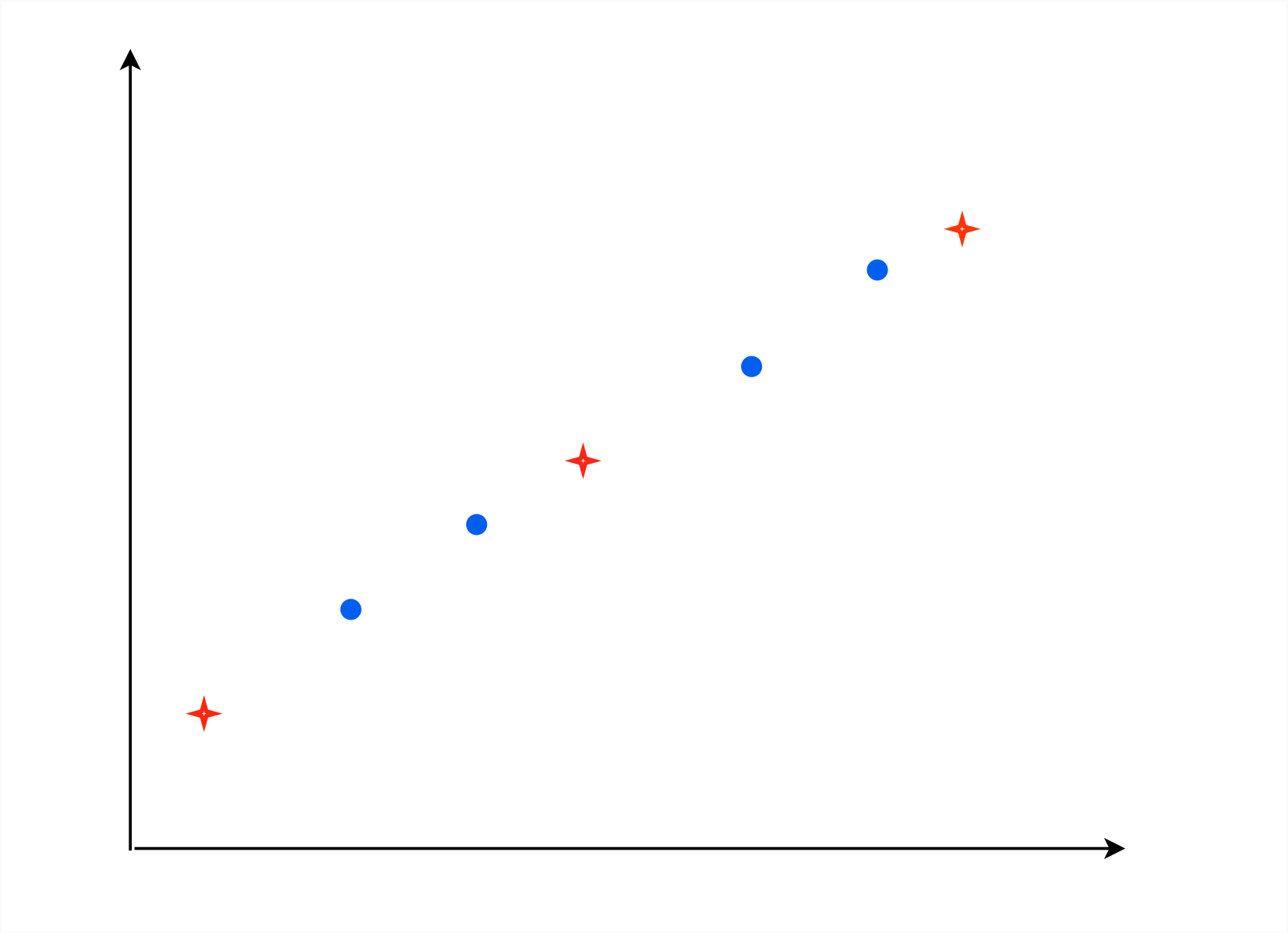
- We implement and investigate the symbolic, rather than adversarial sampling based version of DataGrad.
- We present detailed empirical evidence that the symbolic DataGrad method can improve classification accuracy, especially when the size of the training dataset is small.

Wait, this is not supposed to work!

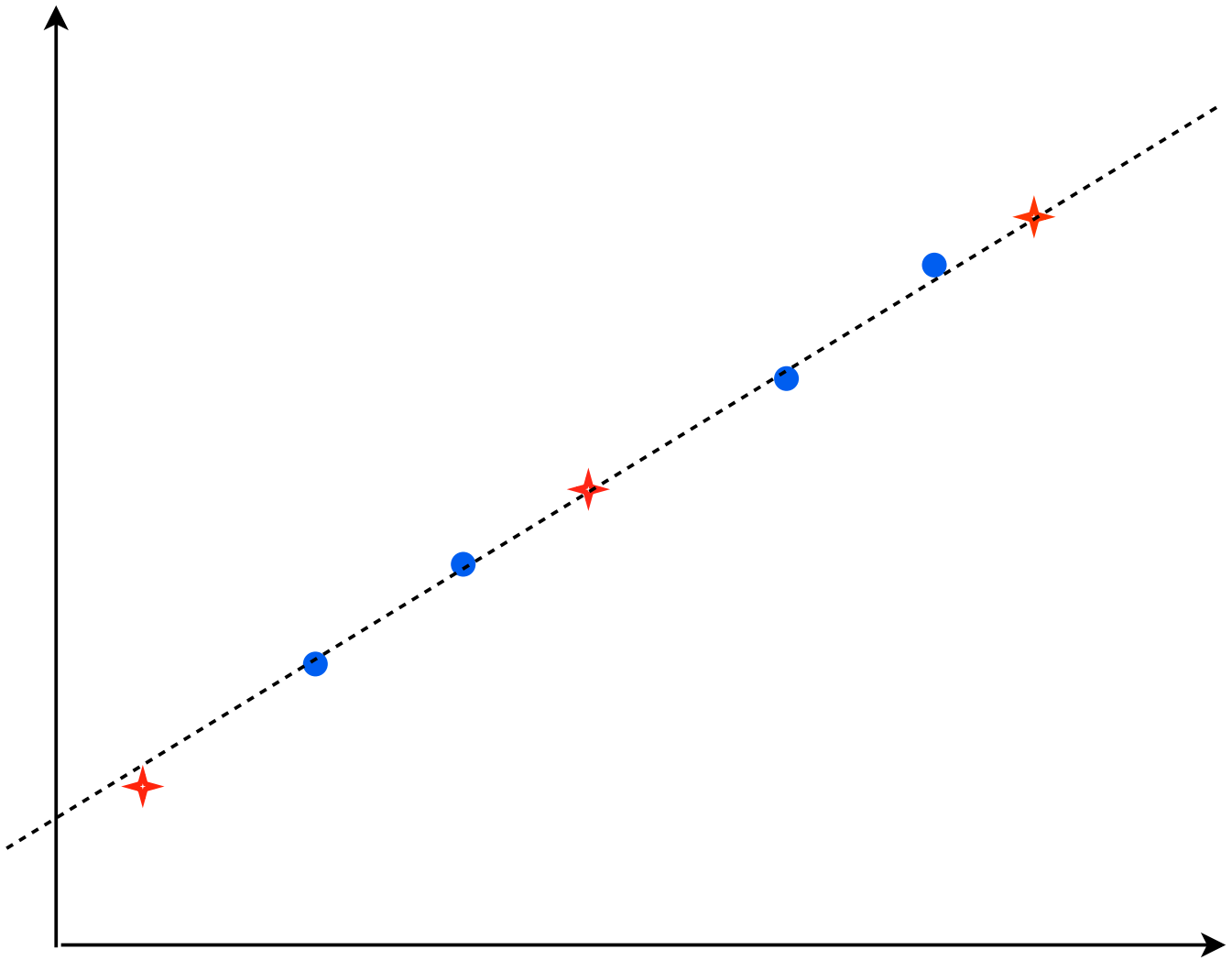
$$L_{DataGrad}(x, y, \Theta) = \| \frac{\partial}{\partial x} L(f_\Theta(x), y) \|_2^2$$

- Did I say that the gradient norm is only controlled at the training examples?
- Yes. (And standard tricks like controlling it at noise-perturbed training examples did not help.)

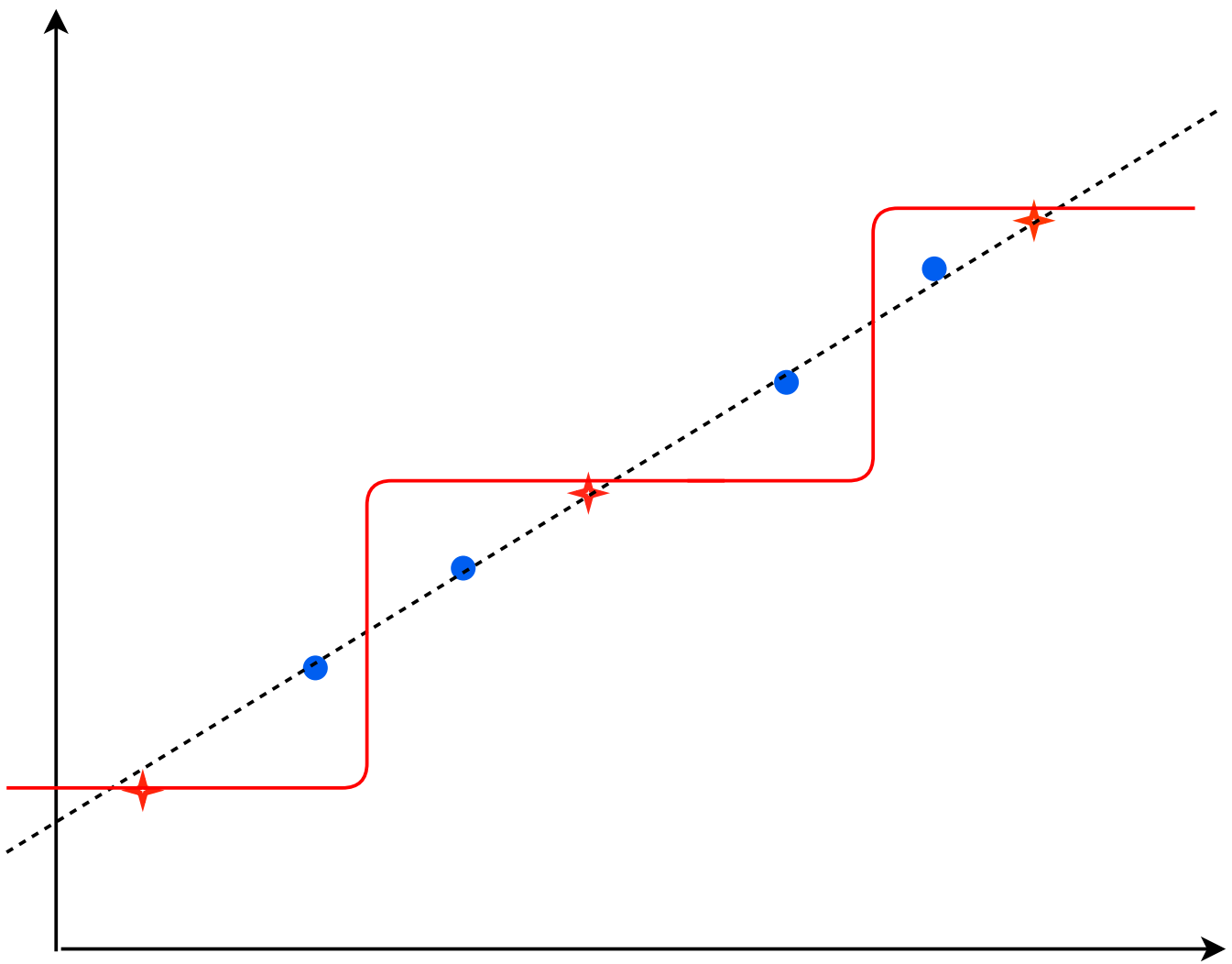
Wait, this is not supposed to work!



Wait, this is not supposed to work!



Wait, this is not supposed to work!



...But it does work.

- We are still in the process of understanding the phenomenon better,
- ...but apparently the low-dimensional intuition of the previous slides does not generalize to complex high-dimensional loss surfaces,
- ...especially not when our method of discovery of these surfaces is the gradient descent.
- Gradient descent does not converge to step function-like solutions such as seen on the previous slide.
- It is probably better to think of gradient regularization as "smarter weight decay". It influences the gradient norm at points far away from the training dataset.

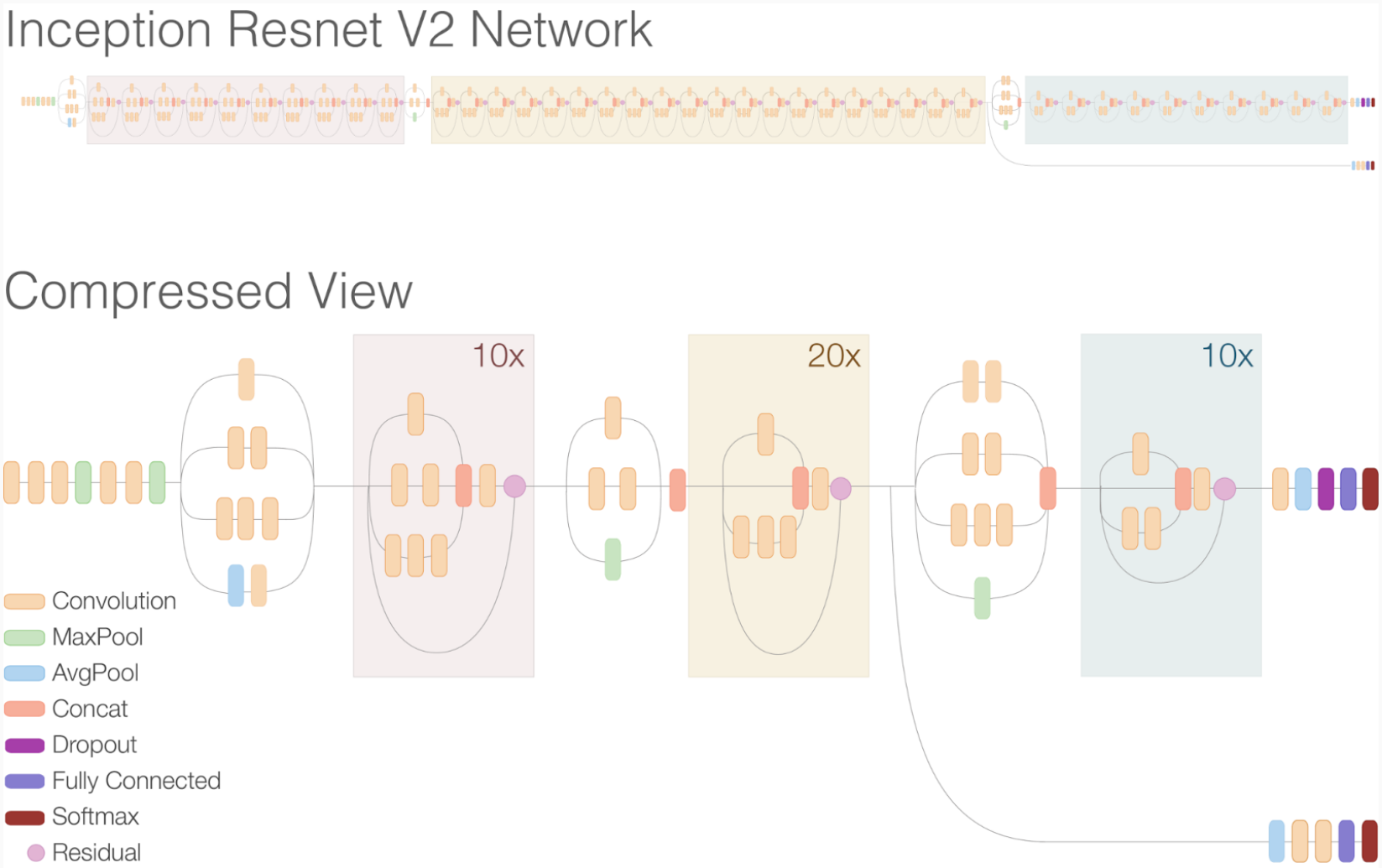
Symbolic differentiation

An implementation issue

- How do we calculate the gradient norms required for our Symbolic DataGrad and SpectReg regularizations?

Symbolic differentiation

Modern neural networks are big and complex beasts:



(Szegedy et al 2016)

Symbolic differentiation

- ...but at the end of the day they are still just formulae (or more precisely, straight line programs).
- They are built up from a small class of operations such as matrix-vector multiplication, vector addition, and elementwise maximum of vectors.
- Modern tensor software libraries such as Tensorflow and PyTorch are designed to manipulate such formulae symbolically. We can build the formula for $\frac{\partial}{\partial x} f(x)$ with a single operation:

```
tf.gradients(f, [x])
```

Symbolic differentiation

```
datagrad_loss = tf.reduce_sum(
    tf.square(tf.gradients(loss, [x])[0]), axis=1)

gradients = tf.gradients(
    logits * tf.random_normal((OUTPUT_DIM,)), [x])[0]
spectreg_loss = tf.reduce_sum(
    tf.square(gradients), axis=1)
```

Experimental Results

MNIST – training on 2000 randomly chosen samples, interaction with weight decay

Weight decay	NoGR	SpectReg	DataGrad
LeNet			
no WD	97.15	97.55 ($\lambda = 0.03$)	**97.93** ($\lambda = 20$)
WD=0.0005	97.32	97.67 ($\lambda = 0.05$)	**97.93** ($\lambda = 50$)

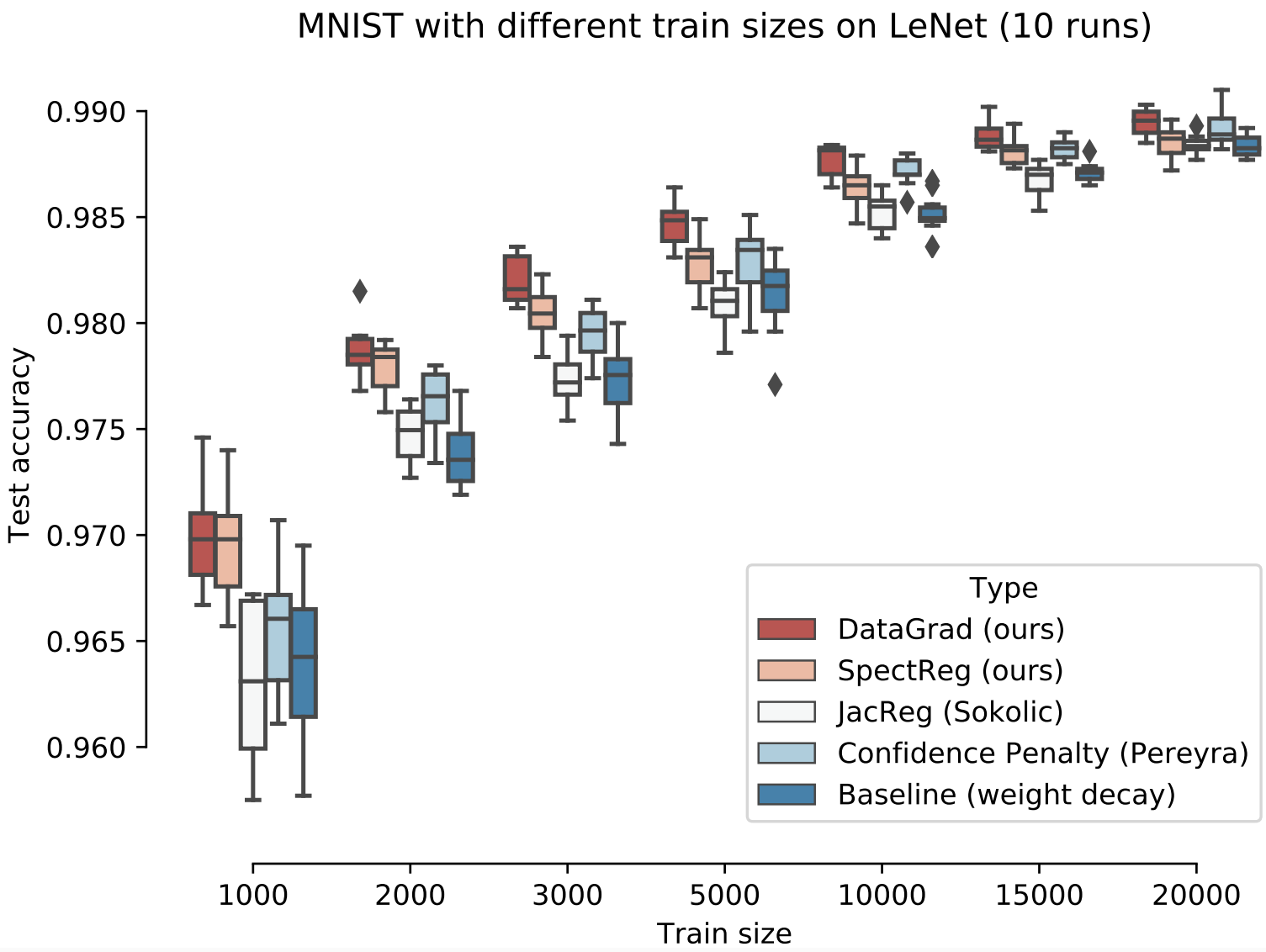
Dropout is on in all of these runs, dropout rate 0.5.

Gradient regularization's interaction with DO and BN

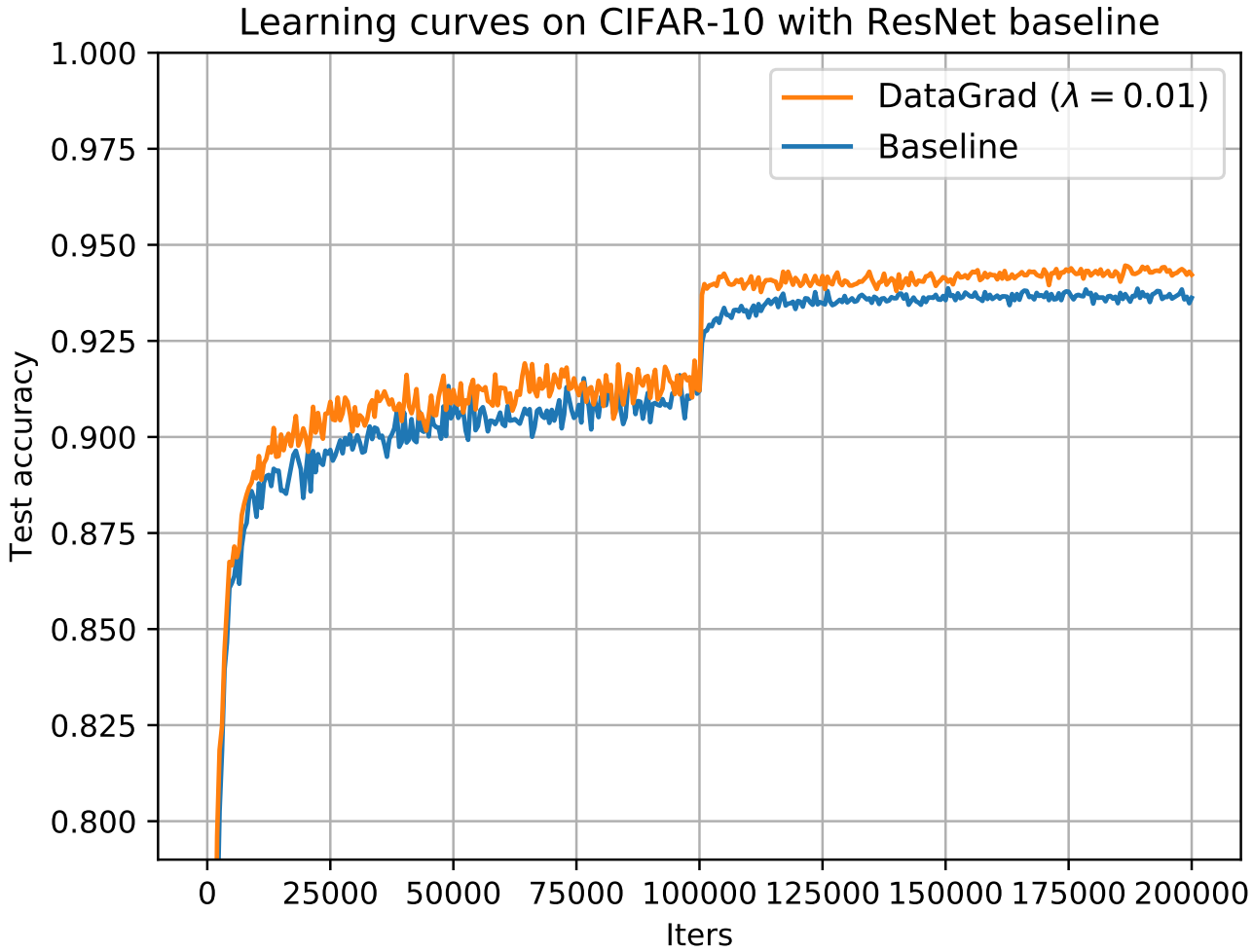
	NoGR	SpectReg	DataGrad
LeNet unreg	96.99	*97.59*	97.56
Lenet BatchNorm	96.89	*96.94*	96.89
Lenet Dropout	97.29	97.67	**97.93**

Comparison of dropout, batch normalization and two variants of gradient regularization: symbolic DataGrad and SpectReg. Train size was set to 2000. Each hyperparameter was tuned individually on a development set.

Comparison of various regularization methods on MNIST



DataGrad learning curve on full CIFAR-10



Improvements even on full CIFAR-10 with data augmentation.

Generative models

Gradient regularization in generative models

Unfortunately I do not have the time to talk about gradient regularization in generative models, but aren't these artificial bedrooms nice?

Summary

Summary

- We propose the symbolic *DataGrad* and *SpectReg* regularization methods,
- present them in a more general framework of Jacobian-based regularizers,
- and experimentally demonstrate that they very consistently improve test accuracy on variations of standard low-resolution image recognition benchmarks,
- especially when the size of the training set is small.