Outline for today

- Convexified CNNs to removing the weight nonlinearity.
- Introduction to Generative Adversarial Networks
  - Inverse conv net for generating images
  - The adversarial game
  - Applications and improved training strategies, WGANs

Convexifying the parameters pt. 1 (Zhang et al. 16'[1])

Consider a two layer convolutional neural network composed of one convolutional layer followed by a fully connected layer.
Rather than working with x directly, form P vectors $z_p(x)$ for $p = 1, \ldots, P$ where $z_p(x)$ is the portion of x on patch p of the convolutional layer. Then the k^{th} component of $H(x, \theta)$ is given by

$$H(x,\theta)_k = \sum_{j=1}^{r} \sum_{p=1}^{p} \alpha_{k,j,p} \sigma(w_j^T z_p(x)).$$

Alternatively if we exclude the nonlinearity we can express this sum by

$$\sum_{j=1}^{r} \sum_{p=1}^{p} \alpha_{k,j,p} \sigma(w_j^T z_p(x)) = \sum_{j=1}^{r} Z(x) w_j$$

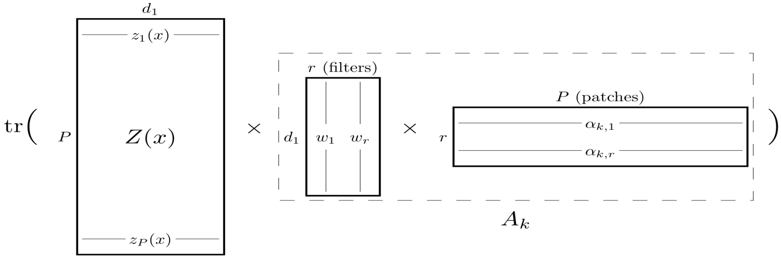
where $Z(x)$ has $z_p(x)$ as its p^{th} row.

[1] https://arxiv.org/pdf/1609.01000.pdf

Convexifying the parameters pt. 2 (Zhang et al. 16'[2])

Using the trace formula this can be further condensed to

$$H(x,\theta)_k = \operatorname{tr}\left(Z(x) \left(\sum_{j=1}^{r} w_j \alpha_{k,j}^T \right) \right) = \operatorname{tr}\left(Z(x) A_k \right)$$



The network parameters are given by A_k nonlinearity is imposed by the A_k having rank r, and we can express all of the parameters of the matrix by A which is similarly rank r.

[2] https://arxiv.org/pdf/1609.01000.pdf

Convexifying the parameters pt. 3 (Zhang et al. 16'[3])

One can impose the network structure through A, but remove the non-convex rank constraint by replacing a convexification, that is the sum of the singular values of A (Schatten-1, or nuclear, norm).

If the convolutional filters and fully connected rows are uniformly bounded in ℓ^2 by B_1 and B_2 respectively, then one can replace then the sum of the singular values of A are bounded by $B_1 B_2 r\sqrt{n}$ where n is the network output dimension and the network parameters can be considered by varying the nuclear norm bound between 0 and $B_1 B_2 r\sqrt{n}$.

The resulting learning programme is fully convex and can be efficiently solved. The above can be extended to nonlinear activations and multiple layers, learning one layer at a time.

[3] https://arxiv.org/pdf/1609.01000.pdf

Convexified CNN: MNIST (Zhang et al. 16'[4])

	basic	rand	rot	img	img+rot
SVM_{rbf} [44]	3.03%	14.58%	**11.11%**	22.61%	55.18%
NN-1 [44]	4.69%	20.04%	18.11%	27.41%	62.16%
CNN-1 (ReLU)	3.37%	9.83%	18.84%	14.23%	45.96%
CCNN-1	**2.38%**	**7.45%**	13.39%	**10.40%**	**42.28%**
TIRBM [38]	-	-	**4.20%**	-	35.50%
SDAE-3 [44]	2.84%	10.30%	9.53%	16.68%	43.76%
ScatNet-2 [8]	1.27%	12.30%	7.48%	18.40%	50.48%
PCANet-2 [9]	**1.06%**	6.19%	7.37%	10.95%	35.48%
CNN-2 (ReLU)	2.11%	5.64%	8.27%	10.17%	32.43%
CNN-2 (Quad)	1.75%	5.30%	8.83%	11.60%	36.90%
CCNN-2	1.38%	**4.32%**	6.98%	**7.46%**	**30.23%**

Table 1: Classification error on the basic MNIST and its four variations. The best performance within each block is bolded. The tag "ReLU" and "Quad" means ReLU activation and quadratic activation, respectively.

[4] https://arxiv.org/pdf/1609.01000.pdf

Convexified CNN: CIFAR10 (Zhang et al. 16'[5])

	Error rate
CNN-1	34.14%
CCNN-1	**23.62%**
CNN-2	24.98%
CCNN-2	**20.52%**
$\text{SVM}_{\text{Fastfood}}$ [27]	36.90%
PCANet-2 [9]	22.86%
CKN [30]	21.70%
CNN-3	21.48%
CCNN-3	**19.56%**

Table 3: Classification error on the CIFAR-10 dataset. The best performance within each block is bolded.

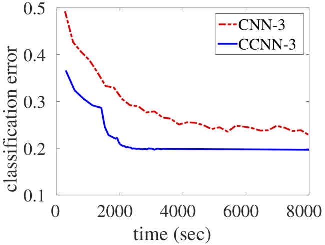


Figure 4: The convergence of CNN-3 and CCNN-3 on the CIFAR-10 dataset.

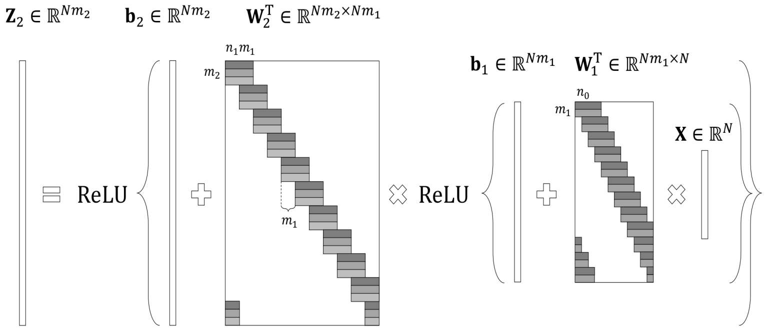
	CNN-1	CNN-2	CNN-3
Original	34.14%	24.98%	21.48%
Convexified	**23.62%**	**21.88%**	**18.18%**

Table 4: Comparing the original CNN and the one whose top convolution layer is convexified by CCNN. The classification errors are reported on CIFAR-10.

[5] https://arxiv.org/pdf/1609.01000.pdf

CNN model through sparse coding (Papyan et al. 16'[6])

Consider a deep conv. net composed of two convolutional layers:

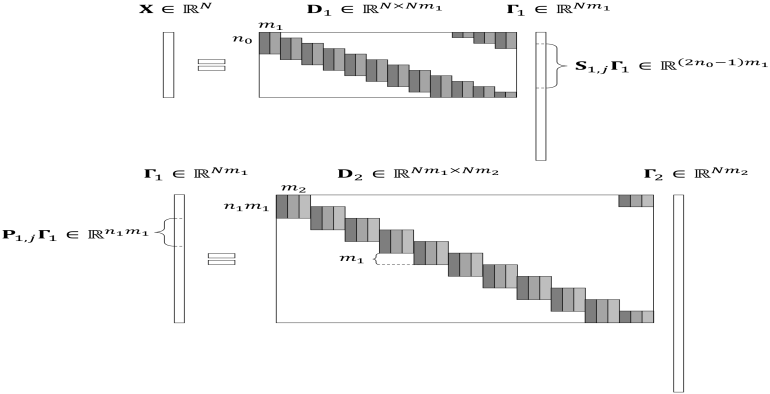


The forward map (note notation using transpose of $W^{(i)}$):

$$Z_2 = \sigma\left(b^{(2)} + (W^{(2)})^T \sigma\left(b^{(1)} + (W^{(1)})^T x\right)\right)$$

[6] https://arxiv.org/pdf/1607.08194.pdf

Deconvolutional NN data model (Papyan et al. 16'[7])



Two layer deconvolutional data model with weight matrices fixed, $W^{(i)} = D_i$, and $\Gamma_i \geq 0$ whose values compose data element X.

[7] https://arxiv.org/pdf/1607.08194.pdf

Generative deep nets (Goodfellow et al. 14'[9])

Example of a deep convolutional generator:

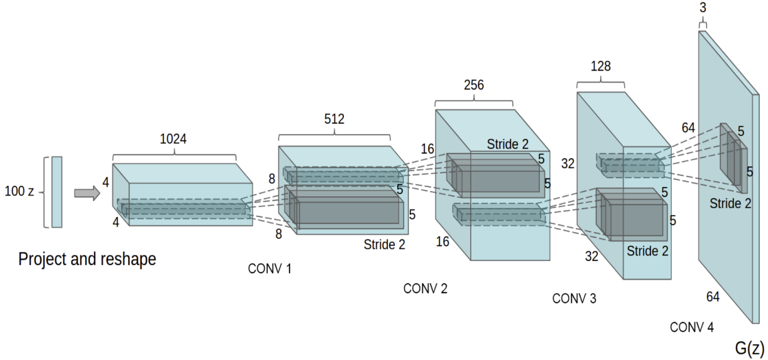


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. [8]

[8] https://arxiv.org/pdf/1511.06434.pdf

[9] https://arxiv.org/pdf/1406.2661.pdf

Generative deep nets (Goodfellow et al. 14'[10])

Train the two network parameters using the objective

$$\min_G \max_D n^{-1} \sum_{\mu=1}^{n} \log(D(x_\mu, y_\mu)) + p^{-1} \sum_{p} \log(1 - D(G(z_p), y_p))$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

[10] https://arxiv.org/pdf/1406.2661.pdf

Generative deep nets (Radford et al. 16'[11])

Figure 2: Generated bedrooms after one training pass through the dataset. Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate.

[11] https://arxiv.org/pdf/1511.06434.pdf

Generative deep nets (Radford et al. 16'[12])

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

[12] https://arxiv.org/pdf/1511.06434.pdf

Wasserstein GAN (Arjovsky et al. 17'[14])

One of the central challenges with GANs is the ability to train the parameters. Improvements have been made through choice of generative architecture (DC-GAN of Radford) and through different training objective functions (W-GAN)

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
    for $t = 1, ..., n_{\text{critic}}$ **do**
        for $i = 1, ..., m$ **do**
            Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
            $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
            $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
            $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
        end for
        $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$
    end for
    Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p(\boldsymbol{z})$.
    $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

[13]

[13] https://arxiv.org/pdf/1704.00028.pdf
[14] https://arxiv.org/pdf/1701.07875.pdf

Wasserstein GAN (Arjovsky et al. 17'[15])

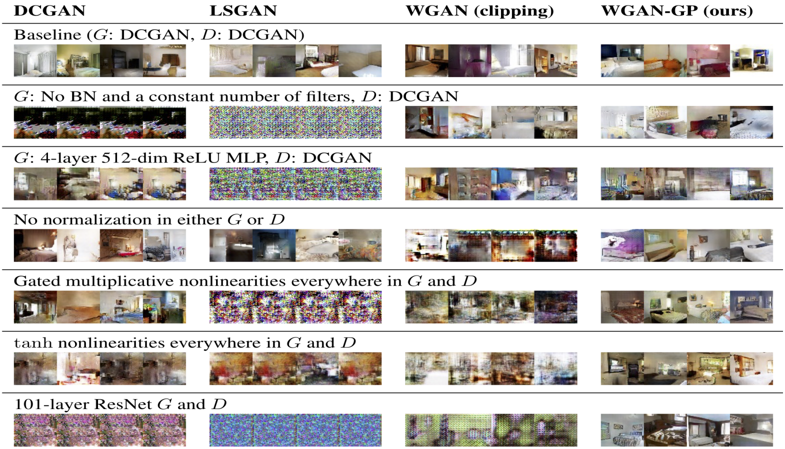


Figure 2: Different GAN architectures trained with different methods. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP.

[15] https://arxiv.org/pdf/1704.00028.pdf

Wasserstein GAN (Arjovsky et al. 17'[16])

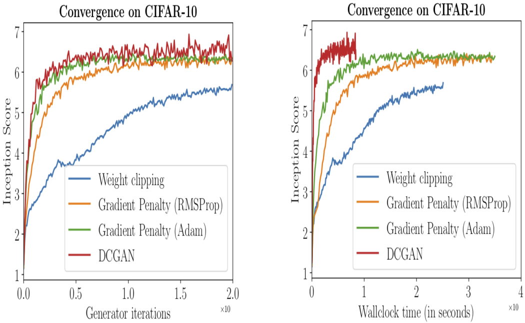


Figure 3: CIFAR-10 Inception score over generator iterations (left) or wall-clock time (right) for four models: WGAN with weight clipping, WGAN-GP with RMSProp and Adam (to control for the optimizer), and DCGAN. WGAN-GP significantly outperforms weight clipping and performs comparably to DCGAN.

[16] `https://arxiv.org/pdf/1704.00028.pdf`

Large scale WGAN (Karras et al. 18'[17])

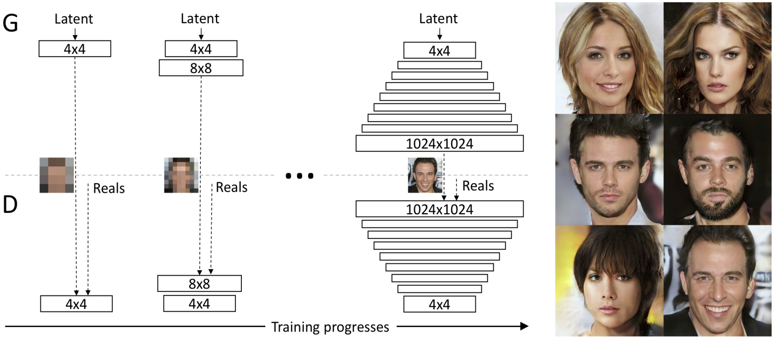


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

[17] https://arxiv.org/abs/1710.10196

Large scale WGAN (Karras et al. 18'[18])

Figure 10: Top: Our CELEBA-HQ results. Next five rows: Nearest neighbors found from the training data, based on feature-space distance. We used activations from five VGG layers, as suggested by Chen & Koltun (2017). Only the crop highlighted in bottom right image was used for comparison in order to exclude image background and focus the search on matching facial features.

[18] https://arxiv.org/abs/1710.10196