※ Generative Adversarial Networks (GANs) - Specialization

※ Course 1 - Build Basic Generative Adversarial Networks

- Generative Models
  - Discriminative Models: from a feature X, discriminate to which class Y this feature belongs.

    Features (X) → Class (Y) P(Y|X)

  - Generative Models: from a Y label and a random noise vector, it generates a feature X.

    Noise (ε), Class (Y) → Features (X) P(X,Y)

    ↳ Variational Autoencoders Encoder → Latent Space → Decoder → After training, the encoder is no longer needed.

    Latent Space → Decoder → Result

    ↳ Varies acess to data distribution to produce different results.

    ↳ Generative Adversarial Networks Generator Discriminator

    Compete with each other.

    Random Noise → Generator → Result

    ↳ The discriminator is only useful during training.

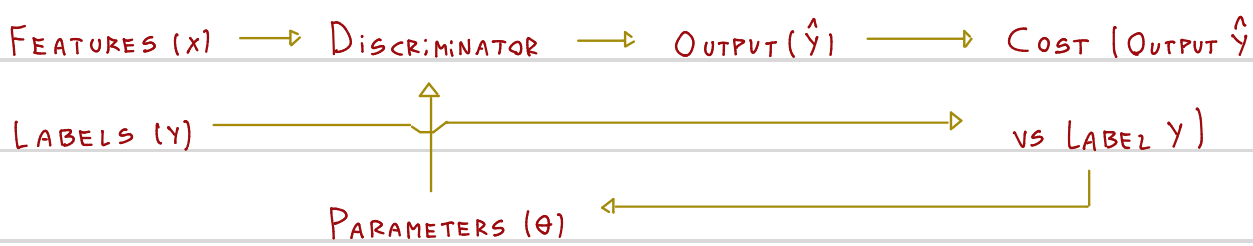
    ↳ Generative models learn to produce realistic examples and discriminative models distinguish between classes.

- Real Life GANs → GANs performance is rapidly improving.
  - Face generation progress: Black and white (2014) → Colored photo-realistic (2018)
  - Mimics the distribution of the training data. (StyleGAN2)
  - GANs for image translation: from one domain to another. (CycleGAN)
  - From a drawing, generate a realistic image. (GauGAN)
  - Animate face images, making them have reactions.
  - GANs for 3D Objects. (3D-GAN)
  - Companies Using GANs: Adobe (Next-gen Photoshop), Google (Text Generation), IBM (Data Augmentation), Snapchat / Tik Tok (Image Filters) and Disney (Super-resolution).

- Intuition Behind GANs
  - Generator learns to make fakes that look real. → Doesn't know how it should look.
  - Discriminator learns to distinguish real from fake.
  - The generator's goal is to fool the discriminator.
  - The discriminator's goal is to distinguish between real and fake.
  - They learn from the competition with each other.
  - At the end, fakes look real.

- Discriminator
  - Classifiers: distinguish between different classes.

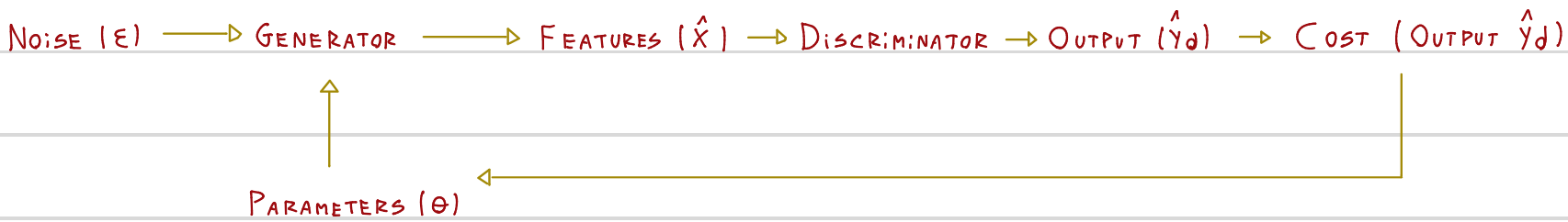
conditional probability
class y | features x
$P(y|x)$



  - It learns the probability of class y (real or fake) given features X.
  - The probabilities are the feedback for the generator.

- Generator: generates examples of the class.

conditional probability
features x | class y
$P(x|y)$



  - Sampling noise vector: different outputs at every run.
  - Generates images that approximate with the data distribution of the training set.
  - The generator produces fake data takes as input noise (random features).
  - It learns the probability of features X.

- BCE Cost Function (Binary Cross Entropy)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h(x^{(i)},\theta) + (1-y^{(i)})\log(1-h(x^{(i)},\theta))\right]$$

- $-$: ensures that the cost is always greater or equal to 0
- $h(x^{(i)},\theta)$: prediction
- $x^{(i)}$: features
- $y^{(i)}$: label
- θ: parameters
- $\frac{1}{m}\sum_{i=1}^{m}$: Average loss of the whole batch

$y^{(i)}$	$h(x^{(i)},\theta)$	$y^{(i)}\log h(x^{(i)},\theta)$
0	Any	0
1	0,99	~0
1	~0	-oo

↳ Relevant when the label is 1

$y^{(i)}$	$h(x^{(i)},\theta)$	$(1-y^{(i)})\log(1-h(x^{(i)},\theta))$
1	Any	0
0	0,01	~0
0	~1	-oo

↳ Relevant when the label is 0

- Close to zero when the label and the prediction are similar.
- Approaches infinity when the label and the prediction are different.

- Putting It All Together
  - Discriminator learn getting feedback on if its classification was correct.
  - Generator learn using feedback from the discriminator.

Superior Discriminator → Fake as 100% fake → No way to improve.

↳ The two models should always be at a similar "skill" level.

Noise (ε) → Generator → $\hat{x}$ → Discriminator → Output ($\hat{y}$)

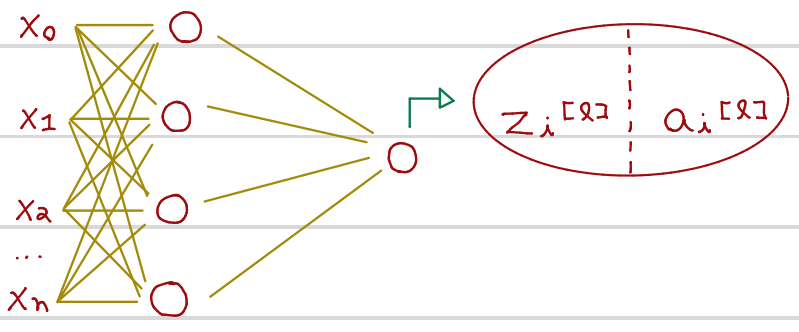
Reals (x) → Discriminator

Discriminator: Distinguish the fakes and the reals.

Generator: Make the fakes look like the reals.

- Intro to PyTorch
  - PyTorch: Imperative, computations on the go; Dynamic Computational Graphs;
  - TensorFlow: Symbolic, first define and then compile; Static Computational Graphs;
  - TensorFlow > 2.0 moves toward PyTorch by including Eager Execution.

- ACTIVATIONS (BASIC PROPERTIES)

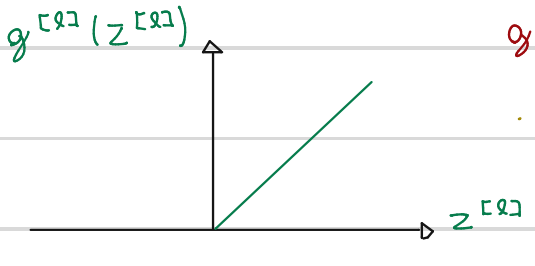


$$z_i^{[l]} = \sum_{i=0} w_i^{[l]} a_i^{[l-1]}$$

$$a_i^{[l]} = g^{[l]}(z_i^{[l]}) \rightarrow \text{DIFFERENTIABLE NON-LINEAR FUNCTION.}$$

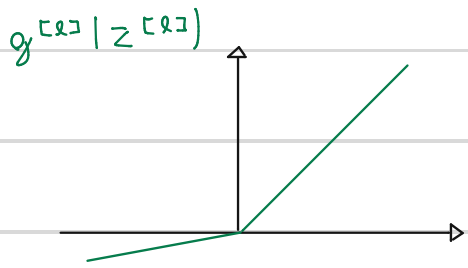
- DIFFERENTIABLE FOR BACKPROPAGATION.
- NON-LINEAR TO COMPUTE COMPLEX FEATURES.

- COMMON ACTIVATION FUNCTIONS
  - ACTIVATIONS: ReLU = RECTIFIED LINEAR UNIT



$$g^{[l]}(z^{[l]}) = \max(0, z^{[l]})$$

- DYING ReLU PROBLEM

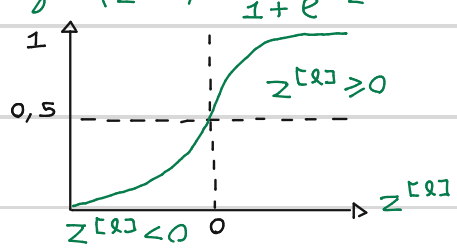


$$g^{[l]}(z^{[l]}) = \max(a z^{[l]}, z^{[l]})$$

- SOLVES THE DYING ReLU PROBLEM
- WHEN $z \leq 0$, THE DERIVATES EQUALS TO ZERO, CAUSING SOME NEURONS TO GET STUCK AND STOP LEARNING.

  - ACTIVATIONS: SIGMOID

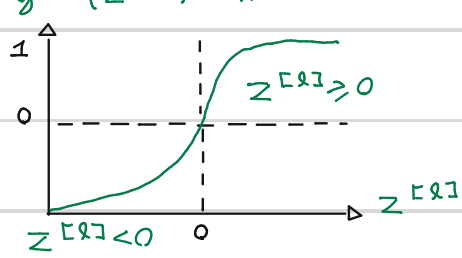
$$g^{[l]}(z^{[l]}) = \frac{1}{1+e^{-z^{[l]}}}$$



- VALUES BETWEEN 0 AND 1.
- VANISHING GRADIENT AND SATURATION PROBLEMS.

  - ACTIVATIONS: TANH

$$g^{[l]}(z^{[l]}) = \tanh(z^{[l]})$$



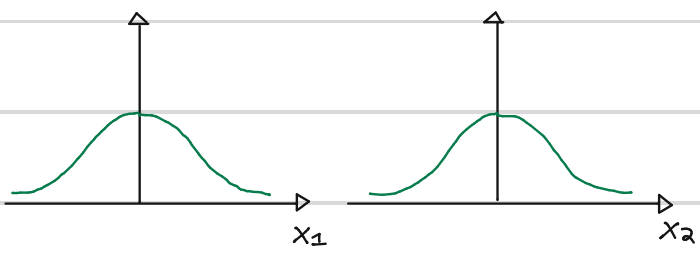
- VALUES BETWEEN -1 AND 1.
- KEEPS THE SIGN OF THE INPUT.
- SAME ISSUES AS SIGMOID.

- VANISHING GRADIENT PROBLEM: ACTIVATION FUNCTIONS LIKE THE SIGMOID HAVE A DERIVATE THAT IS BOUNDED BETWEEN 0 AND 1, MEANING THAT THE GRADIENT DECREASES EXPONENTIALLY AS INFORMATION IS PROPAGATED THROUGH THE VARIOUS LAYERS OF THE NEURAL NETWORKS. AS A RESULT, THE DEEPER LAYERS OF THE NEURAL NETWORKS CAN RECEIVE VERY SMALL GRADIENTS, MAKING IT DIFFICULT OR IMPOSSIBLE TO UPDATE THEIR WEIGHTS IN A MEANINGFUL WAY.

· SATURATION GRADIENT PROBLEM: OCCURS WHEN THE ACTIVATION FUNCTION OF A LAYER OF THE NEURAL NETWORK "SATURATES" OR APPROACHES ITS UPPER OR LOWER LIMIT, IN SUCH A WAY THAT THE GRADIENT BECOMES TOO SMALL TO UPDATE THE WEIGHTS OF THAT LAYER EFFECTIVELY.

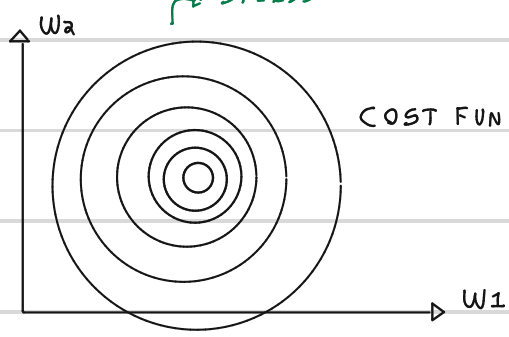
· BATCH NORMALIZATION (EXPLAINED)

NORMALIZATION AND ITS EFFECTS



X1 AND X2 AROUND MEAN AT 0 AND STD. AT 1.

↳ SPEEDS UP LEARNING.



· TRAINING DATA USES BATCH STATS.

· TEST DATA USES TRAINING STATS.

↳ REDUCTION OF COVARIATE SHIFT. → COVARIATE SHIFT OCCURS WHEN TRAINING AND TEST DATA HAVE DIFFERENT DISTRIBUTIONS, WHICH CAN LEAD TO INACCURATE OR UNEXPECTED MODEL RESULTS.

↳ SMOOTHS THE COST FUNCTION.

· BATCH NORMALIZATION (PROCEDURE)

$$\hat{Z}_i^{[\ell]} = \frac{Z_i^{[\ell]} - \mu Z_i^{[\ell]}}{\sqrt{\sigma^2_{Z_i^{[\ell]}} + \epsilon}}$$

$\mu Z_i^{[\ell]}$ → BATCH MEAN OF $Z_i^{[\ell]}$

$\sigma^2_{Z_i^{[\ell]}}$ → BATCH STD OF $Z_i^{[\ell]}$

↳ FOR EVERY EXAMPLE IN THE BATCH TRAINING

$$Y_i^{[\ell]} = \gamma \hat{Z}_i^{[\ell]} + \beta$$

β → SHIFT FACTOR

γ → SCALE FACTOR → LEARNABLE PARAMETERS TO GET THE OPTIMAL DIST.

$$\hat{Z}_i^{[\ell]} = \frac{Z_i^{[\ell]} - E(Z_i^{[\ell]})}{\sqrt{VAR(Z_i^{[\ell]}) + \epsilon}}$$

· RUNNING MEAN AND RUNNING STD FROM TRAINING.

· FRAMEWORKS LIKE TENSORFLOW AND PYTORCH KEEP TRACK OF THEM.

↳ FOR EVERY EXAMPLE IN THE BATCH TEST

· REVIEW OF CONVOLUTIONS

· CONVOLUTIONS ARE USEFUL LAYERS FOR PROCESSING IMAGES.

· THEY SCAN THE IMAGE TO DETECT USEFUL FEATURES.

· JUST ELEMENT-WISE PRODUCTS AND SUMS.

· PADDING AND STRIDE

· STRIDE DETERMINES HOW THE FILTER SCANS THE IMAGE.

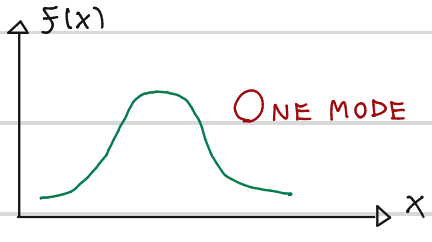
· PADDING LIKE A FRAME ON THE IMAGE.

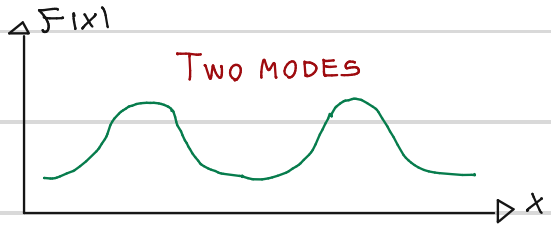
· PADDING GIVES SIMILAR IMPORTANCE TO THE EDGES AND THE CENTER.

- Pooling and Upsampling
  - Types of Pooling: max pooling, average pooling, min pooling, ...
  - Types of Upsampling: nearest neighbors, linear interpolation, Bi-linear interpolation, ...
  - No learnable parameters.

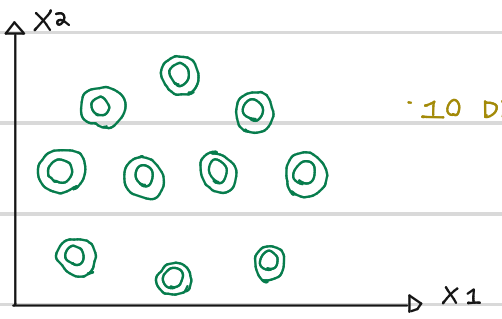
- Transposed Convolutions
  - Transposed convolutions upsample.
  - They have learnable parameters.
  - Problem: Results have a checkerboard pattern.

- Mode Collapse



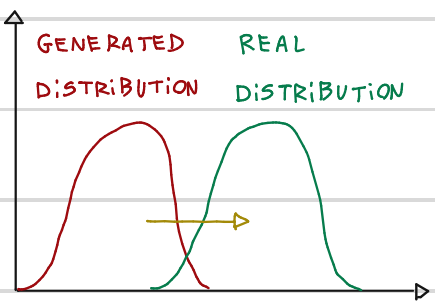


- Any peak on the density function is mode!



- 10 different modes, 1 per digit
- Modes are peaks in the distribution of features.
- Typical with real-world datasets.
- Mode collapse happens when the generator gets stuck in one mode (the generator gets stuck in a local minima). The discriminator will eventually learn to differentiate the generator's fakes when this happens and outskill it, ending the model's learning.

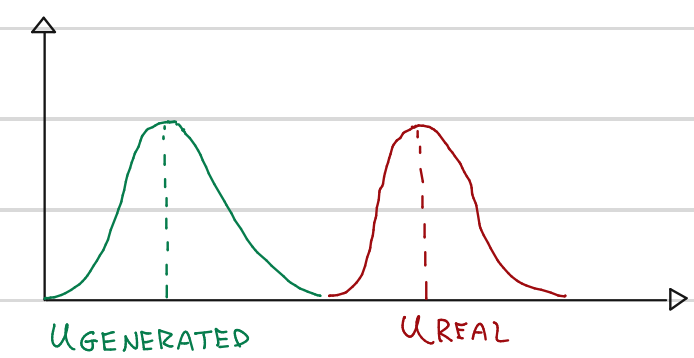
- Problem with BCE Loss
  - Generator: maximize cost.
  - Discriminator: minimize cost.



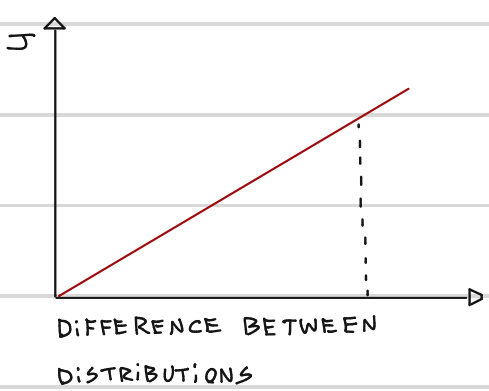
- Make the generated and real distributions look similar.
- Discriminator: single output, easier to train than the generator. → criticizing is more straightforward.
- Generator: complex output, difficult to train.
  - ↳ Often, the discriminator gets better than the generator.

- GANs try to make the real and generated distributions look similar.
- When the discriminator improves too much, the function approximated by BCE Loss will contain flat regions.
- Flat regions on the cost function = vanishing gradients.

Earth Mover's Distance



- Effort to make the generated distribution equal to the real distribution.
- Depends on the distance and amount moved.



- Gradient not close to zero even for very different distributions.

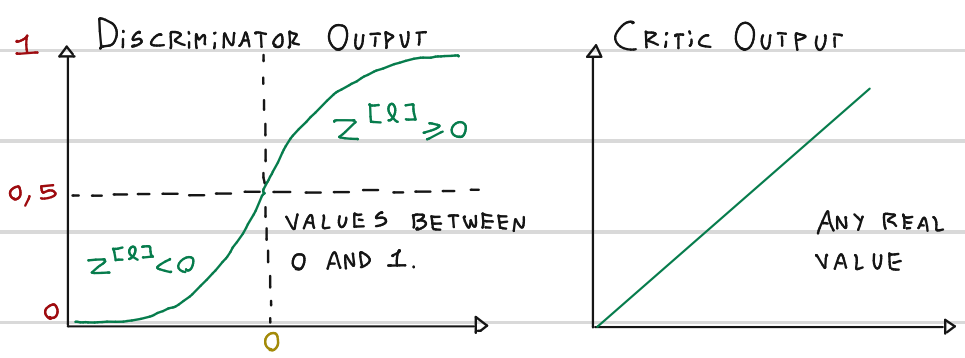
- Earth mover's distance (EMD) is a function of amount and distance.
- Doesn't have flat regions when the distributions are very different.
- Approximating EMD solves the problems associated with BCE.

Wasserstein Loss

- With BCE Loss: Discriminator → Minimize Cost, Generator → Maximize Cost.
- W-Loss approximates the Earth Mover's Distance. $\min_{g} \max_{c} \mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))$

Generator → minimize the distance.

Critic → maximize the distance.



- BCE Loss → Discriminator outputs between 0 and 1
- W-Loss → Critic outputs any number

$$-[\mathbb{E}(\log(d(x))) + \mathbb{E}(1 - \log(d(g(z))))]$$

$$\mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))$$

- W-loss helps with mode collapse and vanishing gradient problems.

· CONDITION ON WASSERSTEIN CRITIC

$$\min_{g} \max_{c} \mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))$$

→ NEEDS TO BE 1-LIPSCHITZ CONTINUOUS

· THE NORM OF THE GRADIENT SHOULD BE AT MOST 1 FOR EVERY POINT.

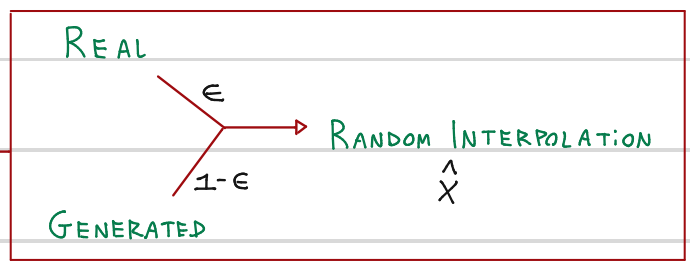
· NEEDED FOR TRAINING STABLE NEURAL NETWORKS WITH W-LOSS. → THIS CONDITION ENSURES THAT W-LOSS IS VALIDLY APPROXIMATING EARTH MOVER'S DISTANCE.

· 1-LIPSCHITZ CONTINUITY ENFORCEMENT

· WEIGHT CLIPPING FORCES THE WEIGHTS OF THE CRITIC TO A FIXED INTERVAL.

· GRADIENT DESCENT TO UPDATE WEIGHTS → CLIP THE CRITIC'S WEIGHTS. → LIMITS THE LEARNING ABILITY OF THE CRITIC.

· GRADIENT PENALTY: $\min_{g} \max_{c} \mathbb{E}(c(x)) - \mathbb{E}(c(g(z))) + L_{reg}$ → REGULARIZATION OF THE CRITIC'S GRADIENT.



$(\|\nabla c(\hat{x})\|_2 - 1)^2$ → REGULARIZATION TERM

$\epsilon x + (1-\epsilon) g(z)$ → INTERPOLATION

· PUTTING IT ALL TOGETHER: $\min_{g} \max_{c} \underbrace{\mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))}_{(a)} + \underbrace{\lambda \mathbb{E}(\|\nabla c(\hat{x})\|_2 - 1)^2}_{(b)}$

· (a) MAKES THE GAN LESS PRONE TO MODE COLLAPSE AND VANISHING GRADIENT.

· (b) TRIES TO MAKE THE CRITIC BE 1-L CONTINUOUS, FOR THE LOSS FUNCTION TO BE CONTINUOUS AND DIFFERENTIABLE.

· WEIGHT CLIPPING AND GRADIENT PENALTY ARE WAYS TO ENFORCE 1-L CONTINUITY.

→ USING THE GRADIENTS ON AN INTERMEDIATE IMAGE WITH RESPECT TO THE CRITIC IS AN APPROXIMATION FOR ENFORCING THE GRADIENT NORM TO BE 1 ALMOST EVERYWHERE. → SINCE CHECKING THE CRITIC'S GRADIENT AT EACH POINT OF THE FEATURE SPACE IS VIRTUALLY IMPOSSIBLE, YOU CAN APPROXIMATE THIS BY USING INTERPOLATED IMAGES.

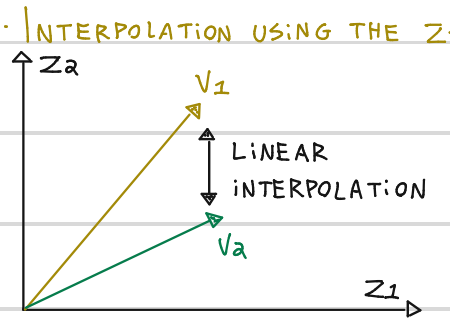
- Conditional Generation: Intuition
  - Unconditional Generation: You get outputs from a random class. Input → Generator → Generated Example
  - Conditional Generation: You get what you ask for. Input → Generator → Generated Example
    Class
  - Conditional:
    - Examples from the classes you want.
    - Training dataset needs to be labeled.
  - Unconditional:
    - Examples from random classes.
    - Training dataset doesn't need to be labeled.

- Conditional Generation: Inputs
  - Noise Vector + Class (one-hot) vector → Generator → Generated Example
    - Control in the generation;
    - Randomness in the generation;
  - Image + Label → Discriminator → Output Prevision
  - Discriminator Input: Image + Class (one-hot) → matrices full of zeros, if not a class, and matrix full of ones if is a class;
    - 3 channels (or 1 if grayscale);

- Controllable Generation
  - Change specific features of the output.
  - Tweak the input noise vector to get different features on the output.

  Noise Vector (z) → Generator → Controlled Output
  - Controllable:
    - Examples with the features that you want.
    - Training dataset doesn't need to be labeled.
    - Manipulate the z vector input.
  - Conditional:
    - Examples from the classes you want.
    - Training dataset needs to be labeled.
    - Append a class vector to the input.

- Vector Algebra in The Z-Space
  - Interpolation using the z-space: how an image morphs into another.



  - Z-space with Noise Vectors

    Intermediate images using the z-space
  - To control output features, you need to find directions in the z-space.
  - To modify your output, you move around in the z-space.

- CHALLENGES WITH CONTROLLABLE GENERATION
  - FEATURE CORRELATION
    - UNCORRELATED FEATURES — WOMAN → ADD BEARD → WOMAN WITH BEARD
    - CORRELATED FEATURES — WOMAN → ADD BEARD → MAN WITH BEARD
      - ↑ MAKE MORE MASCULINE
  - Z-SPACE ENTANGLEMENT: IT IS NOT POSSIBLE TO CONTROL SINGLE OUTPUT FEATURES.
  - WHEN TRYING TO CONTROL ONE FEATURE, OTHERS THAT ARE CORRELATED CHANGE.
  - Z-SPACE ENTANGLEMENT MAKES CONTROLLABILITY DIFFICULT, IF NOT IMPOSSIBLE.
  - ENTANGLEMENT HAPPENS WHEN Z DOES NOT HAVE ENOUGH DIMENSIONS.

- CLASSIFIER GRADIENTS

DIRECTION TO MAKE THE NOISE VECTOR UNTIL THE FEATURE EMERGES.

  - CLASSIFIERS CAN BE USED TO FIND DIRECTIONS IN THE Z-SPACE.
  - TO FIND DIRECTIONS, THE UPDATES ARE DONE JUST TO THE NOISE VECTOR.

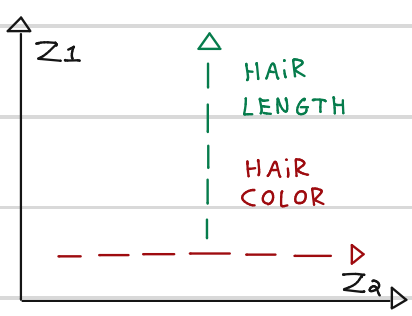
- DISENTANGLEMENT
  - DISENTANGLED Z-SPACES LET YOU CONTROL INDIVIDUAL FEATURES BY CORRESPONDING Z VALUES DIRECTLY TO THEM.
  - THERE ARE SUPERVISED AND UNSUPERVISED METHODS TO ACHIEVE DISENTANGLEMENT.
  - DISENTANGLED Z-SPACES:

z_1 z_2

$V_1 = [1, 2, 3, \ldots]$

$V_2 = [5, 6, 7, \ldots]$

HAIR COLOR, HAIR LENGTH → LATENT FACTORS OF VARIATION.



Z-SPACE WITH NOISE VECTORS

    - CHANGES TO ONE FEATURE DON'T AFFECT THE OTHERS.
    - DIFFICULT FOR CONTINUOUS CLASSES.

$L_{NEW} = L_{ORIGINAL} + reg$

ORIGINAL LOSS — REGULARIZATION

- CAN BE ANY LOSS FUNCTION (e.g. BCE, W-LOSS).