

Lecture 2: Deep Learning Pre-2017

CS 395T

Roadmap

I. Image modeling

- convolution, convnets, cross-entropy
- LeNet, AlexNet; MNIST / CIFAR / ImageNet
- deeper: VGG, ResNet
- generation: VAEs, GANs; how to measure quality

Roadmap

I. Image modeling

- convolution, convnets, cross-entropy
- LeNet, AlexNet; MNIST / CIFAR / ImageNet
- deeper: VGG, ResNet
- generation: VAEs, GANs; how to measure quality

II. Language modeling

- tokenization, RNNs, LSTMs
- tasks: language modeling, captioning, translation
- attention before Transformers; BLEU / ROUGE; datasets

Roadmap

I. Image modeling

- convolution, convnets, cross-entropy
- LeNet, AlexNet; MNIST / CIFAR / ImageNet
- deeper: VGG, ResNet
- generation: VAEs, GANs; how to measure quality

II. Language modeling

- tokenization, RNNs, LSTMs
- tasks: language modeling, captioning, translation
- attention before Transformers; BLEU / ROUGE; datasets

III. Adversarial examples

- adversarial examples and adversarial training

Part I. Image modeling

Why not just a fully-connected network?

Setup: input $x \in \mathbb{R}^{H \times W \times C}$, e.g. $224 \times 224 \times 3 = 150,528$ numbers.

Channels: Denoted by C above - R/G/B for images, internal layers of a CNN have $\gg 3$.

Cost of one dense (MLP) layer to 4096 hidden units:

$$150,528 \times 4096 \approx 6.2 \times 10^8 \text{ parameters (one layer!)}$$

Why not just a fully-connected network?

Setup: input $x \in \mathbb{R}^{H \times W \times C}$, e.g. $224 \times 224 \times 3 = 150,528$ numbers.

Channels: Denoted by C above - R/G/B for images, internal layers of a CNN have $\gg 3$.

Cost of one dense (MLP) layer to 4096 hidden units:

$$150,528 \times 4096 \approx 6.2 \times 10^8 \text{ parameters (one layer!)}$$

Problems:

- **Statistical:** #params $\gg$ #images $\Rightarrow$ overfitting.
- **Computational:** memory and FLOPs per layer are enormous.

Why not just a fully-connected network?

Setup: input $x \in \mathbb{R}^{H \times W \times C}$, e.g. $224 \times 224 \times 3 = 150,528$ numbers.

Channels: Denoted by C above - R/G/B for images, internal layers of a CNN have $\gg 3$.

Cost of one dense (MLP) layer to 4096 hidden units:

$$150,528 \times 4096 \approx 6.2 \times 10^8 \text{ parameters (one layer!)}$$

Problems:

- **Statistical:** #params $\gg$ #images $\Rightarrow$ overfitting.
- **Computational:** memory and FLOPs per layer are enormous.

But images are not arbitrary vectors: nearby pixels are correlated, and the same visual pattern can appear anywhere in the frame. We want to exploit this!

Convolutions: useful properties

Idea: use convolutions to generate “useful” features of images:

- **Locality:** a useful feature (edge, corner, texture) depends on a small neighborhood $\Rightarrow$ connect each unit only to a $k \times k$ patch.
- **Translation equivariance:** a cat is a cat wherever it sits $\Rightarrow$ *share* the same weights at every position.

Convolutions: useful properties

Idea: use convolutions to generate “useful” features of images:

- **Locality:** a useful feature (edge, corner, texture) depends on a small neighborhood $\Rightarrow$ connect each unit only to a $k \times k$ patch.
- **Translation equivariance:** a cat is a cat wherever it sits $\Rightarrow$ *share* the same weights at every position.

Mathematical perspective: for a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ (interpret as image), $K : [0, 1]^2 \rightarrow \mathbb{R}$ (interpret as weights/parameters), *convolution* of f and K is:

$$(f * K)(x) = \int_z K(z) \cdot f(x - z) dz.$$

Check translation invariance: $T_a f(x) := f(x - a)$, $(T_a f * K) = T_a(f * K)$.

Convolutions: useful properties

Idea: use convolutions to generate “useful” features of images:

- **Locality:** a useful feature (edge, corner, texture) depends on a small neighborhood $\Rightarrow$ connect each unit only to a $k \times k$ patch.
- **Translation equivariance:** a cat is a cat wherever it sits $\Rightarrow$ *share* the same weights at every position.

Mathematical perspective: for a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ (interpret as image), $K : [0, 1]^2 \rightarrow \mathbb{R}$ (interpret as weights/parameters), *convolution* of f and K is:

$$(f * K)(x) = \int_z K(z) \cdot f(x - z) dz.$$

Check translation invariance: $T_a f(x) := f(x - a)$, $(T_a f * K) = T_a(f * K)$.

Convolutional neural networks: implement something analogous to above...

The convolution operation

Image $x \in \mathbb{R}^{n \times n}$, kernel $w \in \mathbb{R}^{k \times k}$:

$$z_{i,j} = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} w_{a,b} x_{i+a,j+b}$$

The convolution operation

Image $x \in \mathbb{R}^{n \times n}$, kernel $w \in \mathbb{R}^{k \times k}$:

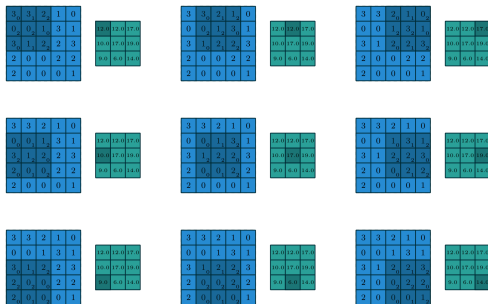
$$z_{i,j} = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} w_{a,b} x_{i+a,j+b}$$

Stride: Size of “jump” between consecutive values of i or j we compute $z_{i,j}$ for.

Padding: How much to enlarge edges of image (w/ 0's).

Output spatial size with stride s , padding p :

$$o = \left\lfloor \frac{n + 2p - k}{s} \right\rfloor + 1$$



A convolutional layer

Input $h^{(\ell-1)} \in \mathbb{R}^{n \times n \times C}$, **filters** $W^{(\ell)} \in \mathbb{R}^{k \times k \times C \times F}$, **biases** $b^{(\ell)} \in \mathbb{R}^F$, **stride** s :

$$z_{i,j,f}^{(\ell)} = \sum_{c=1}^C \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} W_{a,b,c,f}^{(\ell)} h_{si+a, sj+b, c}^{(\ell-1)} + b_f^{(\ell)}, \quad h^{(\ell)} = \sigma(z^{(\ell)})$$

A convolutional layer

Input $h^{(\ell-1)} \in \mathbb{R}^{n \times n \times C}$, **filters** $W^{(\ell)} \in \mathbb{R}^{k \times k \times C \times F}$, **biases** $b^{(\ell)} \in \mathbb{R}^F$, **stride** s :

$$z_{i,j,f}^{(\ell)} = \sum_{c=1}^C \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} W_{a,b,c,f}^{(\ell)} h_{si+a, sj+b, c}^{(\ell-1)} + b_f^{(\ell)}, \quad h^{(\ell)} = \sigma(z^{(\ell)})$$

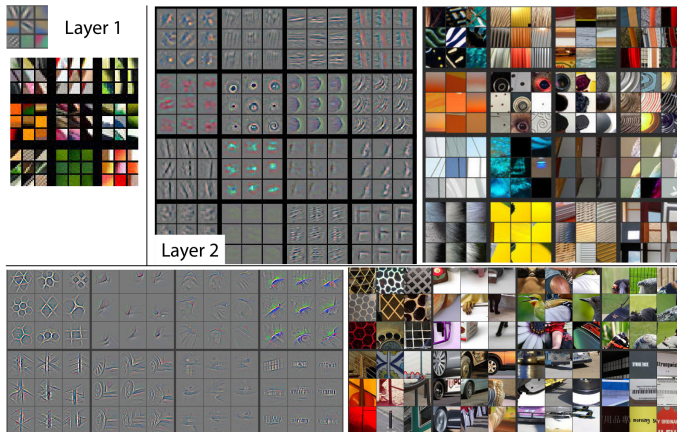
- **Nonlinearity:** $\sigma(u) = \max(u, 0)$ (ReLU); older nets used tanh / sigmoid.

Pooling / subsampling: shrink resolution (and create invariance to small translations)

$$h_{i,j,c}^{\text{pool}} = \max_{(a,b) \in \text{window}} h_{si+a, sj+b, c}.$$

- **Typical pattern:** resolution $\downarrow$, number of channels $\uparrow$, as depth grows.

What the shared filters learn



Layer 1: oriented edges, color blobs. **Layer 2:** corners, curves, texture. Deeper layers compose these into object parts.

From feature maps to a class prediction

Goal: Use CNN for image classification. **Backbone:** $h = f_{\theta}(x) \in \mathbb{R}^d$ after flattening / global pooling the last feature map.

Linear head + softmax over K classes:

$$z = Wh + b \in \mathbb{R}^K, \quad p_{\theta}(y = k \mid x) = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}.$$

From feature maps to a class prediction

Goal: Use CNN for image classification. **Backbone:** $h = f_{\theta}(x) \in \mathbb{R}^d$ after flattening / global pooling the last feature map.

Linear head + softmax over K classes:

$$z = Wh + b \in \mathbb{R}^K, \quad p_{\theta}(y = k \mid x) = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}.$$

- $z = \mathbf{logits}$; softmax turns them into a probability vector.
- Prediction $\hat{y} = \arg \max_k z_k$; the whole network is trained end-to-end.

The cross-entropy loss

Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, labels $y_i \in \{1, \dots, K\}$.

Loss on one example - cross entropy:

$$\mathcal{L}(\theta; x, y) = -\log p_{\theta}(y \mid x).$$

The cross-entropy loss

Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, labels $y_i \in \{1, \dots, K\}$.

Loss on one example - cross entropy:

$$\mathcal{L}(\theta; x, y) = -\log p_{\theta}(y \mid x).$$

Equivalent to **Maximum likelihood:** $\min_{\theta} \frac{1}{n} \sum_i \mathcal{L} = \max_{\theta} \log\text{-likelihood of the labels}.$

The cross-entropy loss

Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, labels $y_i \in \{1, \dots, K\}$.

Loss on one example - cross entropy:

$$\mathcal{L}(\theta; x, y) = -\log p_{\theta}(y \mid x).$$

Equivalent to **Maximum likelihood:** $\min_{\theta} \frac{1}{n} \sum_i \mathcal{L} = \max_{\theta} \log\text{-likelihood of the labels.}$

Gradient of cross-entropy w.r.t. the logits:

$$\frac{\partial \mathcal{L}}{\partial z_k} = p_{\theta}(y = k \mid x) - \mathbf{1}\{y = k\}.$$

The cross-entropy loss

Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, labels $y_i \in \{1, \dots, K\}$.

Loss on one example - cross entropy:

$$\mathcal{L}(\theta; x, y) = -\log p_{\theta}(y | x).$$

Equivalent to **Maximum likelihood:** $\min_{\theta} \frac{1}{n} \sum_i \mathcal{L} = \max_{\theta} \log\text{-likelihood of the labels.}$

Gradient of cross-entropy w.r.t. the logits:

$$\frac{\partial \mathcal{L}}{\partial z_k} = p_{\theta}(y = k | x) - \mathbf{1}\{y = k\}.$$

- Confidently wrong $\Rightarrow$ gradient of magnitude ≈ 1 : **no saturation**.
- Loss is convex in z (not in θ), and unbounded above: it punishes assigning probability $\rightarrow 0$ to the true class.

The cross-entropy loss

Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, labels $y_i \in \{1, \dots, K\}$.

Loss on one example - cross entropy:

$$\mathcal{L}(\theta; x, y) = -\log p_{\theta}(y | x).$$

Equivalent to **Maximum likelihood:** $\min_{\theta} \frac{1}{n} \sum_i \mathcal{L} = \max_{\theta} \log\text{-likelihood of the labels.}$

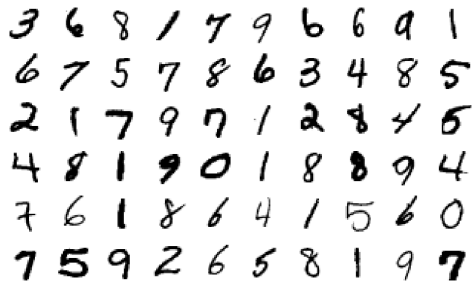
Gradient of cross-entropy w.r.t. the logits:

$$\frac{\partial \mathcal{L}}{\partial z_k} = p_{\theta}(y = k | x) - \mathbf{1}\{y = k\}.$$

- Confidently wrong $\Rightarrow$ gradient of magnitude ≈ 1 : **no saturation**.
- Loss is convex in z (not in θ), and unbounded above: it punishes assigning probability $\rightarrow 0$ to the true class.

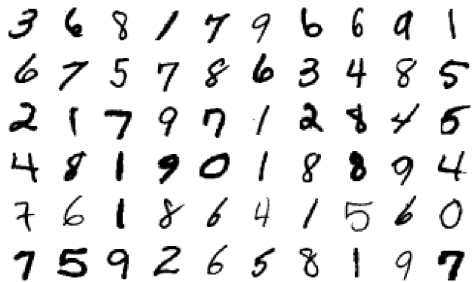
Training: SGD / momentum / Adam on mini-batches, gradients by backpropagation.

Evaluation: MNIST



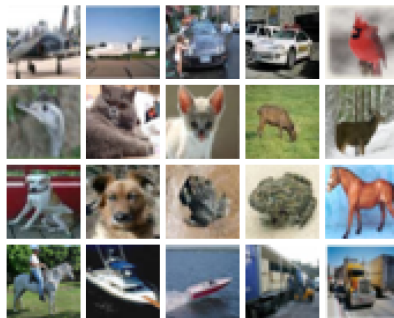
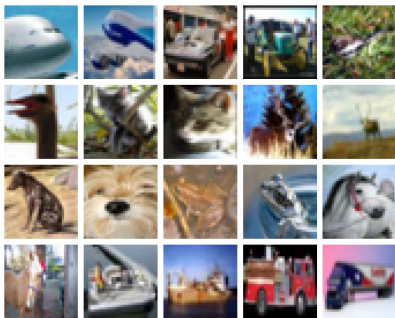
- Handwritten digits, 28×28 grayscale, 10 classes.
- 60,000 train / 10,000 test.

Evaluation: MNIST



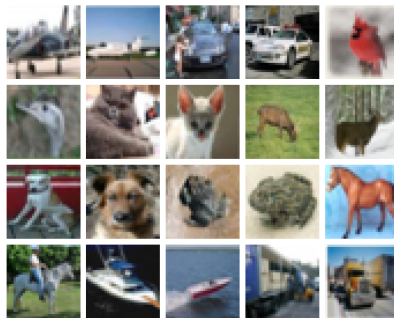
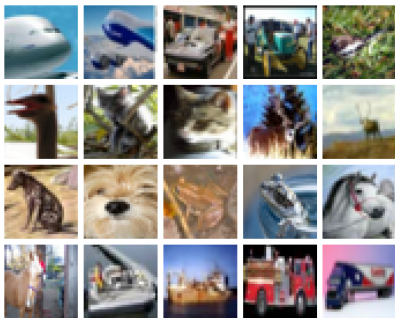
- Handwritten digits, 28×28 grayscale, 10 classes.
- 60,000 train / 10,000 test.
- The standard benchmark of the 1990s:
 $\approx 12\%$ error for linear models, 0.95% for LeNet-5.

Evaluation: CIFAR-10 / CIFAR-100



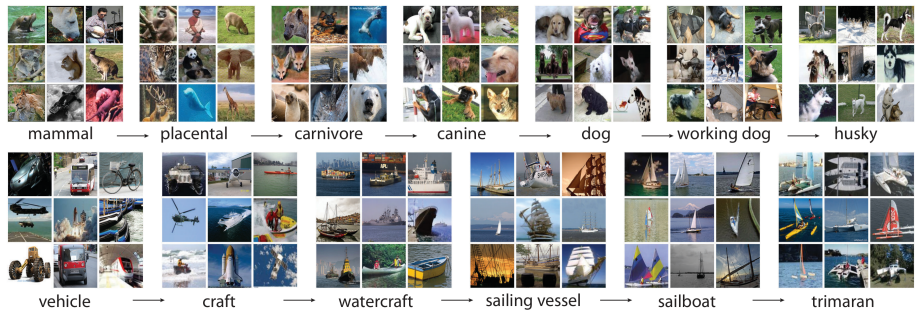
- 32×32 color images labeled from the 80M Tiny Images collection.
- **CIFAR-10**: 10 classes, 50K train / 10K test. **CIFAR-100**: 100 classes, 600 images each.

Evaluation: CIFAR-10 / CIFAR-100



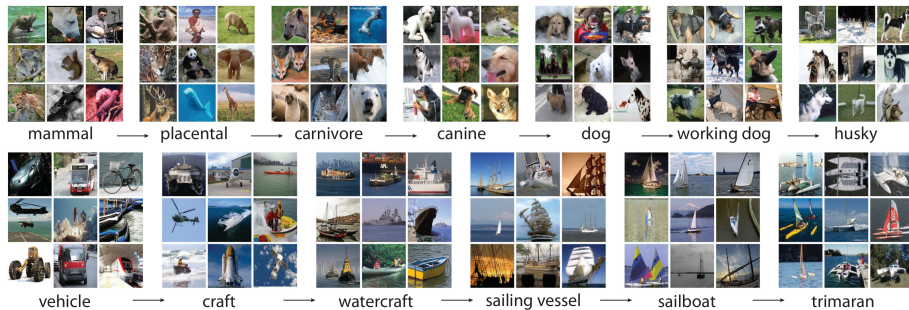
- 32×32 color images labeled from the 80M Tiny Images collection.
- **CIFAR-10:** 10 classes, 50K train / 10K test. **CIFAR-100:** 100 classes, 600 images each.
- Progress: $\approx 80\%$ accuracy (2010) $\rightarrow 93\%$ (ResNet) $\rightarrow > 99\%$ today.

Evaluation: ImageNet / ILSVRC



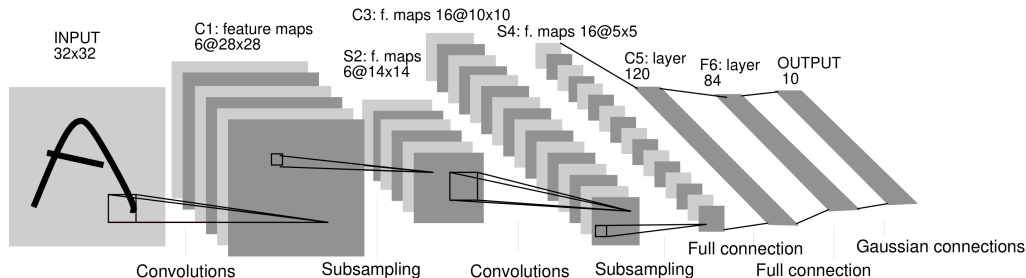
- ~14M images organized by the WordNet noun hierarchy (21K+ synsets).
- **ILSVRC-2012**: 1000 classes, 1.28M train / 50K val / 100K test.

Evaluation: ImageNet / ILSVRC



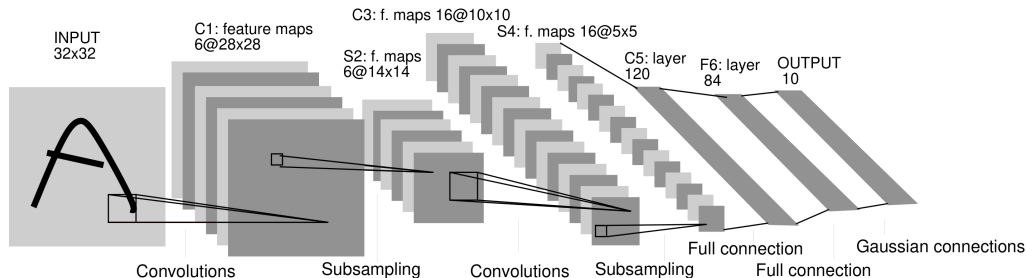
- ~14M images organized by the WordNet noun hierarchy (21K+ synsets).
- **ILSVRC-2012:** 1000 classes, 1.28M train / 50K val / 100K test.
- **Metric:** top-1 and top-5 error (credit if the true label is among the top 5).
- **Note:** fine-grained classes (120 dog breeds!).

LeNet-5 [LeCun et al., 1998]



- 32×32 input $\rightarrow \{\text{conv } w/s > 1\} \times 2 \rightarrow$ fully connected $\rightarrow 10$ outputs.
- $\approx 60\text{K}$ parameters; trained on MNIST with backpropagation.
- Test error 0.95% (LeNet-5), versus $\approx 8\%$ for a linear classifier.

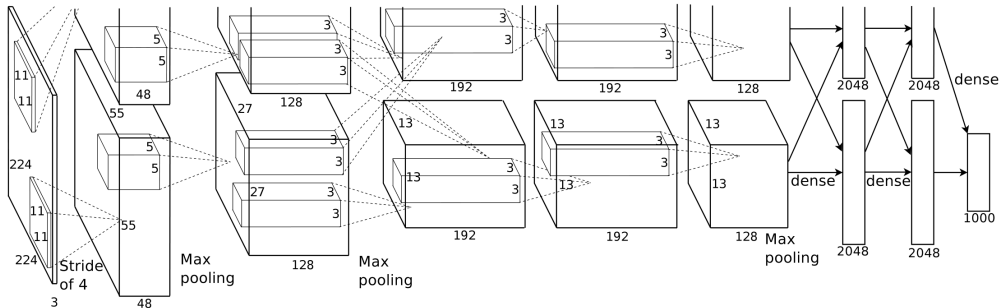
LeNet-5 [LeCun et al., 1998]



- 32×32 input $\rightarrow \{\text{conv w/ } s > 1\} \times 2 \rightarrow$ fully connected $\rightarrow$ 10 outputs.
- $\approx 60\text{K}$ parameters; trained on MNIST with backpropagation.
- Test error 0.95% (LeNet-5), versus $\approx 8\%$ for a linear classifier.

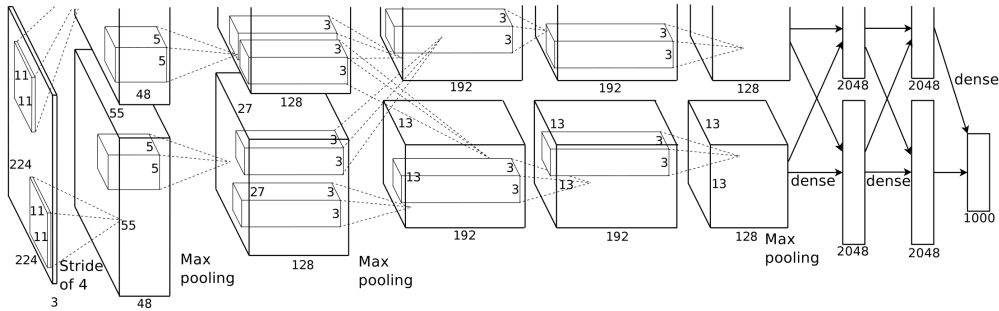
Fast-forward 14 years (more compute & data!)

AlexNet [Krizhevsky et al., 2012]



- 5 convolutional + 3 fully-connected layers, $\approx 60\text{M}$ parameters, 1000-way softmax.
- ILSVRC-2012 top-5 error **15.3%** vs. **26.2%** for the runner-up.

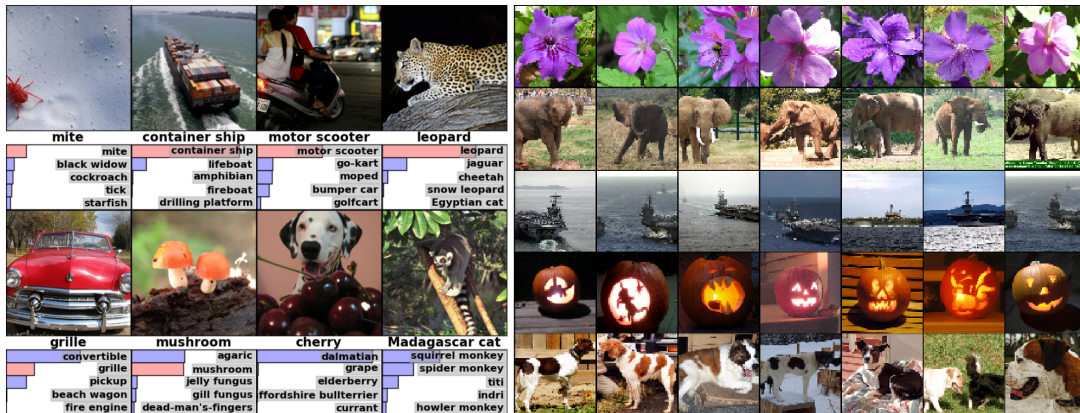
AlexNet [Krizhevsky et al., 2012]



- 5 convolutional + 3 fully-connected layers, $\approx 60\text{M}$ parameters, 1000-way softmax.
- ILSVRC-2012 top-5 error **15.3%** vs. **26.2%** for the runner-up.

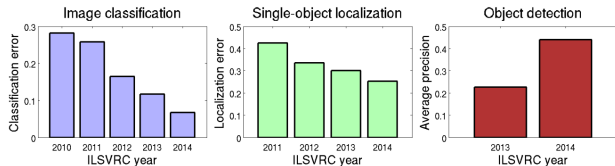
Some aspects: ReLU nonlinearities (fast to train), split network onto two(!) GPUs, dropout (regularization technique), data augmentation.

AlexNet: qualitative results



Left: top-5 predictions. **Right:** in each row, left-most column shows test image; remaining show nearest neighbors in last-hidden-layer feature space.

Progress on ILSVRC over time

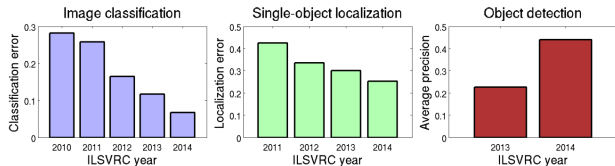


Winning entries, top-5

classification error:

2010	28.2% (hand-crafted)
2011	25.8% (hand-crafted)
2012	16.4% AlexNet
2013	11.7% ZFNet
2014	6.7% GoogLeNet
2015	3.6% ResNet

Progress on ILSVRC over time



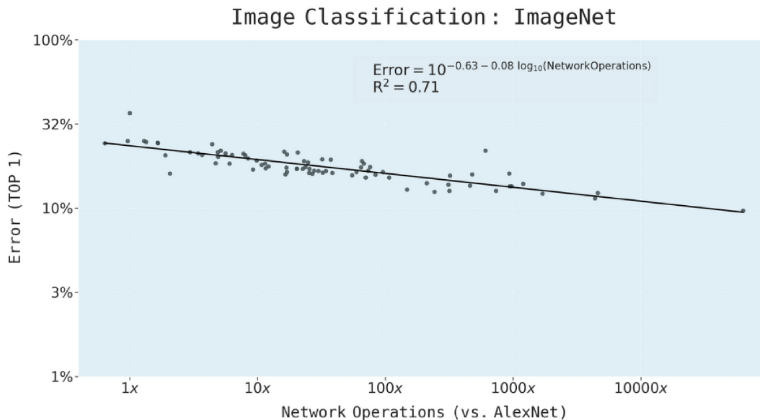
Winning entries, top-5

classification error:

2010	28.2% (hand-crafted)
2011	25.8% (hand-crafted)
2012	16.4% AlexNet
2013	11.7% ZFNet
2014	6.7% GoogLeNet
2015	3.6% ResNet

Progress buys accuracy with computation ("scaling laws")

b)



ImageNet top-1 error against network operations (relative to AlexNet), across published models. Error falls roughly polynomially in compute – each increment costs more than the last.

Deeper networks: VGG [Simonyan & Zisserman, 2014]

A	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers
input (224×224 RGB image)				
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool				
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool				
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512
maxpool				
FC-4096				
FC-4096				
FC-1000				
soft-max				

Rough idea: use only 3×3 convolutions (stride 1) and 2×2 max-pooling; go deep (11-19 weight layers).

Deeper networks: VGG [Simonyan & Zisserman, 2014]

A	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers
input (224 × 224 RGB image)				
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool				
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool				
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool				
FC-4096				
FC-4096				
FC-1000				
soft-max				

Rough idea: use only 3×3 convolutions (stride 1) and 2×2 max-pooling; go deep (11–19 weight layers).

Why stacks of 3×3 ? Two stacked 3×3 layers see a 5×5 window and three see 7×7 , with fewer parameters and more nonlinearities:

$$3(3^2 C^2) = 27C^2 < 49C^2.$$

VGG: results and cost

- ILSVRC-2014: 7.3% top-5 error.
- Depth helps here: 11 layers → 19 layers steadily reduces error.

VGG: results and cost

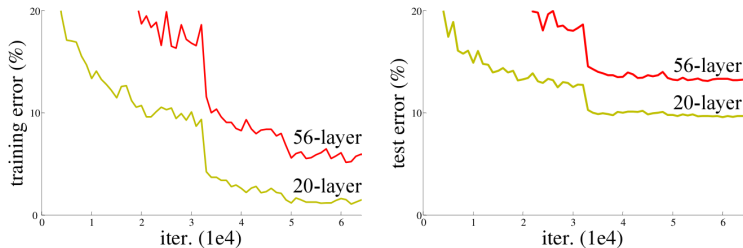
- ILSVRC-2014: 7.3% top-5 error.
- Depth helps here: 11 layers $\rightarrow$ 19 layers steadily reduces error.
- **Size:** VGG-16 has $\approx$ 138M parameters.
- **Note:** VGG features are good image representation for various other tasks.

VGG: results and cost

- ILSVRC-2014: 7.3% top-5 error.
- Depth helps here: 11 layers $\rightarrow$ 19 layers steadily reduces error.
- **Size:** VGG-16 has $\approx$ 138M parameters.
- **Note:** VGG features are good image representation for various other tasks.

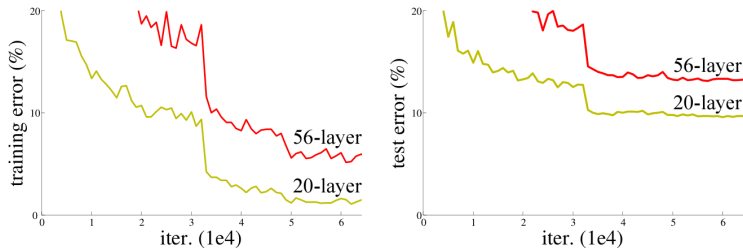
Open question after VGG: can we just keep stacking layers?

The degradation problem [He et al., 2015]



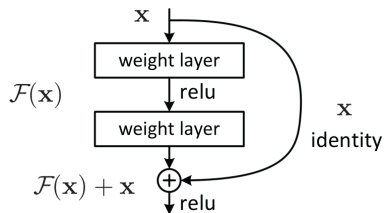
- A 56-layer plain net has **higher training** error than a 20-layer one.
- Not overfitting (training error is worse), not (only) vanishing gradients – batch norm was already used.

The degradation problem [He et al., 2015]



- A 56-layer plain net has **higher training** error than a 20-layer one.
- Not overfitting (training error is worse), not (only) vanishing gradients – batch norm was already used.
- Optimization difficulty: deeper plain nets struggle to learn to even represent the identity.

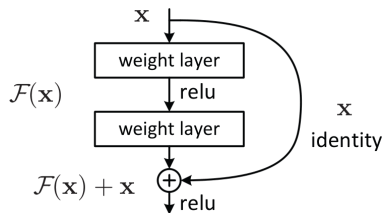
ResNet: residual learning



Instead of fitting desired mapping (call it $\mathcal{H}(x)$) directly, fit the **residual** $\mathcal{F}(x) = \mathcal{H}(x) - x$:

$$y = \mathcal{F}(x, \{W_i\}) + x.$$

ResNet: residual learning



Instead of fitting desired mapping (call it $\mathcal{H}(x)$) directly, fit the **residual** $\mathcal{F}(x) = \mathcal{H}(x) - x$:

$$y = \mathcal{F}(x, \{W_i\}) + x.$$

- E.g., in diagram: $\mathcal{F}(x) = W_2 \cdot \sigma(W_1 \cdot x)$.
- Identity is now free ($\mathcal{F} = 0$), so extra depth can never hurt in principle.
- Use a learned projection $W_s x$ instead of identity if dims change.

ResNet: architecture

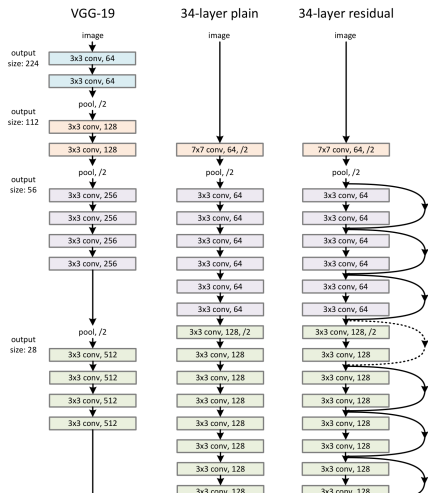


Figure: VGG-19 (left), 34-layer plain (middle), 34-layer residual (right).

ResNet: architecture

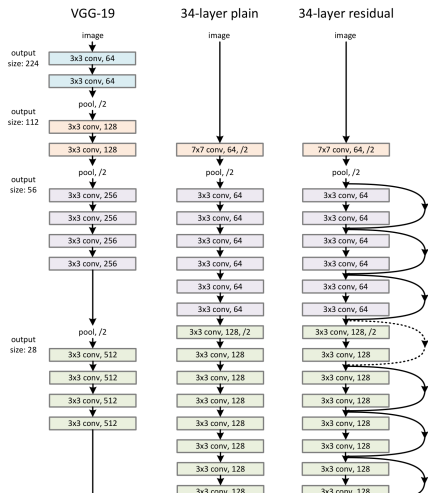


Figure: VGG-19 (left), 34-layer plain (middle), 34-layer residual (right).

- Dotted shortcuts = dimensions change.
- ResNet-152 won ILSVRC-2015 with 3.57% top-5 error.
- Residual connections are now standard everywhere – including Transformers.

Image generation

From classification to generation

Classification: model $p(y \mid x)$ – a low-dimensional output.

Generation: model $p(x)$ itself, and sample new x (very high-dimensional!).

From classification to generation

Classification: model $p(y \mid x)$ – a low-dimensional output.

Generation: model $p(x)$ itself, and sample new x (very high-dimensional!).

Latent-variable recipe: pick a simple prior $z \sim p(z) = \mathcal{N}(0, I)$ on $\mathbb{R}^d$, $d \ll \dim(x)$, and learn a *generative model* $p_\theta(x \mid z)$ parametrized by θ ; then

$$p_\theta(x) = \int p_\theta(x \mid z) p(z) dz.$$

From classification to generation

Classification: model $p(y \mid x)$ – a low-dimensional output.

Generation: model $p(x)$ itself, and sample new x (very high-dimensional!).

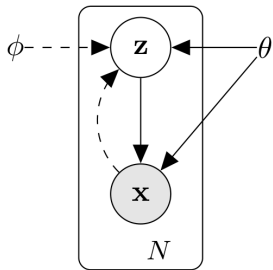
Latent-variable recipe: pick a simple prior $z \sim p(z) = \mathcal{N}(0, I)$ on $\mathbb{R}^d$, $d \ll \dim(x)$, and learn a *generative model* $p_\theta(x \mid z)$ parametrized by θ ; then

$$p_\theta(x) = \int p_\theta(x \mid z) p(z) dz.$$

Difficulty: this integral (and hence the likelihood) is intractable; so hard to, e.g., optimize θ to do MLE!.

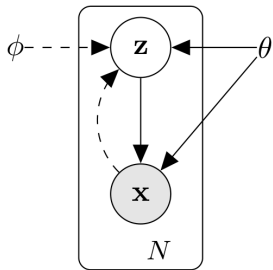
- **VAE:** optimize a tractable *lower bound* on $\log p_\theta(x)$.
- **GAN:** give up on likelihood; match distributions through a two-player game.

Variational autoencoders [Kingma & Welling, 2013]



Idea: in order to train generative $p_{\theta}(x|z)$, need to find z corresponding to data x .

Variational autoencoders [Kingma & Welling, 2013]



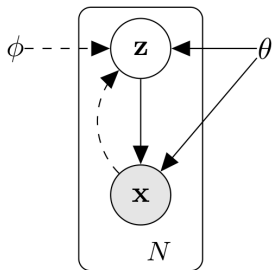
Idea: in order to train generative $p_{\theta}(x|z)$, need to find z corresponding to data x .

Generative "decoder" model (solid arrows), parameters θ :

$$z \sim \mathcal{N}(0, I), \quad x \sim p_{\theta}(x | z).$$

$p_{\theta}(x | z)$ is a neural net: e.g. $x \sim \mathcal{N}(\mu_{\theta}(z), \sigma_{\theta}^2(z)I)$.

Variational autoencoders [Kingma & Welling, 2013]



Idea: in order to train generative $p_{\theta}(x|z)$, need to find z corresponding to data x .

Generative "decoder" model (solid arrows), parameters θ :

$$z \sim \mathcal{N}(0, I), \quad x \sim p_{\theta}(x | z).$$

$p_{\theta}(x | z)$ is a neural net: e.g. $x \sim \mathcal{N}(\mu_{\theta}(z), \sigma_{\theta}^2(z)I)$.

Inference "encoder" model (dashed), parameters ϕ :

$$q_{\phi}(z | x) = \mathcal{N}(\mu_{\phi}(x), \text{diag } \sigma_{\phi}^2(x))$$

approximates the intractable posterior $p_{\theta}(z | x)$.

Encoder q_{ϕ} , decoder p_{θ} – trained jointly.

The evidence lower bound (ELBO)

For any $q_\phi(z \mid x)$,

$$\log p_\theta(x) = \underbrace{\mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z \mid x)} \right]}_{\mathcal{L}(\theta, \phi; x) = \text{ELBO}} + \text{KL}(q_\phi(z \mid x) \parallel p_\theta(z \mid x)) .$$

The evidence lower bound (ELBO)

For any $q_\phi(z \mid x)$,

$$\log p_\theta(x) = \underbrace{\mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z \mid x)} \right]}_{\mathcal{L}(\theta, \phi; x) = \text{ELBO}} + \text{KL}(q_\phi(z \mid x) \parallel p_\theta(z \mid x)).$$

Since $\text{KL} \geq 0$: $\log p_\theta(x) \geq \mathcal{L}(\theta, \phi; x)$, with equality iff $q_\phi = p_\theta(\cdot \mid x)$.

The evidence lower bound (ELBO)

For any $q_\phi(z \mid x)$,

$$\log p_\theta(x) = \underbrace{\mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z \mid x)} \right]}_{\mathcal{L}(\theta, \phi; x) = \text{ELBO}} + \text{KL}(q_\phi(z \mid x) \parallel p_\theta(z \mid x)).$$

Since $\text{KL} \geq 0$: $\log p_\theta(x) \geq \mathcal{L}(\theta, \phi; x)$, with equality iff $q_\phi = p_\theta(\cdot \mid x)$.

Rewriting the bound:

$$\mathcal{L}(\theta, \phi; x) = \underbrace{\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x \mid z)]}_{\text{reconstruction}} - \underbrace{\text{KL}(q_\phi(z \mid x) \parallel p(z))}_{\text{regularizer toward the prior}}.$$

Maximizing $\mathcal{L}$ over (θ, ϕ) raises a lower bound on the data log-likelihood *and* tightens it.

Training a VAE: the reparameterization trick

$$\mathcal{L}(\theta, \phi; x) = \underbrace{\mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x | z)]}_{\text{reconstruction}} - \underbrace{\text{KL}(q_{\phi}(z | x) \parallel p(z))}_{\text{regularizer toward the prior}} .$$

Problem: computing $\nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} [f(z)]$ – distribution we average over depends on ϕ .

Training a VAE: the reparameterization trick

$$\mathcal{L}(\theta, \phi; x) = \underbrace{\mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x | z)]}_{\text{reconstruction}} - \underbrace{\text{KL}(q_{\phi}(z | x) \parallel p(z))}_{\text{regularizer toward the prior}}.$$

Problem: computing $\nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} [f(z)]$ – distribution we average over depends on ϕ .

Fix: write the sample as a deterministic function of noise,

$$z = \mu_{\phi}(x) + \sigma_{\phi}(x) \odot \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I),$$

so that $\mathbb{E}_{q_{\phi}}[f(z)] = \mathbb{E}_{\varepsilon}[f(\mu_{\phi} + \sigma_{\phi} \odot \varepsilon)]$ and gradients pass straight through.

Training a VAE: the reparameterization trick

$$\mathcal{L}(\theta, \phi; x) = \underbrace{\mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x | z)]}_{\text{reconstruction}} - \underbrace{\text{KL}(q_{\phi}(z | x) \| p(z))}_{\text{regularizer toward the prior}}.$$

Problem: computing $\nabla_{\phi} \mathbb{E}_{q_{\phi}(z|x)} [f(z)]$ – distribution we average over depends on ϕ .

Fix: write the sample as a deterministic function of noise,

$$z = \mu_{\phi}(x) + \sigma_{\phi}(x) \odot \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I),$$

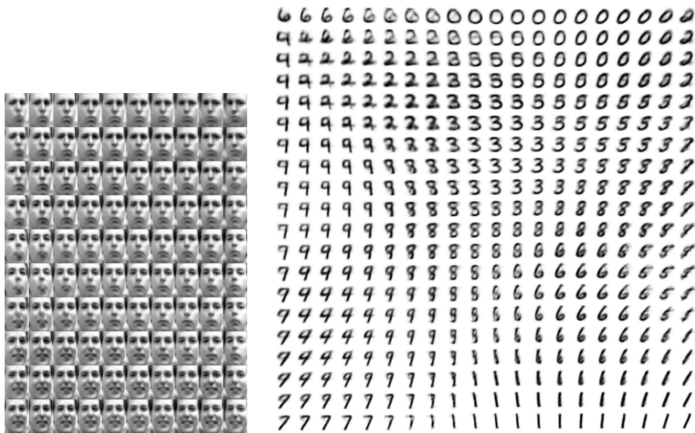
so that $\mathbb{E}_{q_{\phi}}[f(z)] = \mathbb{E}_{\varepsilon}[f(\mu_{\phi} + \sigma_{\phi} \odot \varepsilon)]$ and gradients pass straight through.

Monte-Carlo objective: sample ε and get z as above, and define below loss which we can differentiate (KL term is closed-form for Gaussians):

$$\tilde{\mathcal{L}} = \log p_{\theta}(x | \mu_{\phi}(x) + \sigma_{\phi}(x) \odot \varepsilon) - \frac{1}{2} \sum_{j=1}^d \left(\mu_{\phi,j}^2 + \sigma_{\phi,j}^2 - \log \sigma_{\phi,j}^2 - 1 \right).$$

Train with plain SGD/Adam on (θ, ϕ) .

VAE samples



Decoder outputs as z sweeps a 2-D latent grid (Frey faces, MNIST). **Characteristic failure:** samples are smooth and blurry – recall $p_{\theta}(x | z)$ was $\mathcal{N}(\mu_{\theta}(z), \sigma_{\theta}^2(z))$.

Generative adversarial networks [Goodfellow et al., 2014]

Two players:

- **Generator** $G_\theta : \mathbb{R}^d \rightarrow \mathcal{X}$ turns noise $z \sim p_z$ into a (G_θ deterministic now, should be less blurry!).
- **Discriminator** $D_\omega : \mathcal{X} \rightarrow (0, 1)$ estimates “probability this came from the data”.

Generative adversarial networks [Goodfellow et al., 2014]

Two players:

- **Generator** $G_\theta : \mathbb{R}^d \rightarrow \mathcal{X}$ turns noise $z \sim p_z$ into a (G_θ deterministic now, should be less blurry!).
- **Discriminator** $D_\omega : \mathcal{X} \rightarrow (0, 1)$ estimates “probability this came from the data”.

Minimax objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))].$$

Generative adversarial networks [Goodfellow et al., 2014]

Two players:

- **Generator** $G_\theta : \mathbb{R}^d \rightarrow \mathcal{X}$ turns noise $z \sim p_z$ into a (G_θ deterministic now, should be less blurry!).
- **Discriminator** $D_\omega : \mathcal{X} \rightarrow (0, 1)$ estimates “probability this came from the data”.

Minimax objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))].$$

- D wants to separate real from fake; G wants to make them indistinguishable.

In practice: alternate k gradient steps on D with one on G .

Why the minimax game matches distributions

Minimax objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))].$$

Why the minimax game matches distributions

Minimax objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))].$$

Step 1 – optimal discriminator for fixed G (with p_g = distribution of $G(z)$):

$$D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Why the minimax game matches distributions

Minimax objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))].$$

Step 1 – optimal discriminator for fixed G (with p_g = distribution of $G(z)$):

$$D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Step 2 – substitute back:

$$\max_D V(D, G) = 2 \cdot (\text{KL}(p_{\text{data}} \parallel (p_g + p_{\text{data}})/2) + \text{KL}(p_g \parallel (p_g + p_{\text{data}})/2)) - \log 4.$$

Why the minimax game matches distributions

Minimax objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))].$$

Step 1 – optimal discriminator for fixed G (with p_g = distribution of $G(z)$):

$$D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Step 2 – substitute back:

$$\max_D V(D, G) = 2 \cdot (\text{KL}(p_{\text{data}} \parallel (p_g + p_{\text{data}})/2) + \text{KL}(p_g \parallel (p_g + p_{\text{data}})/2)) - \log 4.$$

Step 3: $\text{KL} \geq 0$ with equality iff $p_g = p_{\text{data}}$, so the global optimum is $p_g = p_{\text{data}}$.

Why the minimax game matches distributions

Minimax objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))].$$

Step 1 – optimal discriminator for fixed G (with p_g = distribution of $G(z)$):

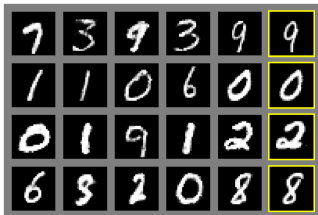
$$D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Step 2 – substitute back:

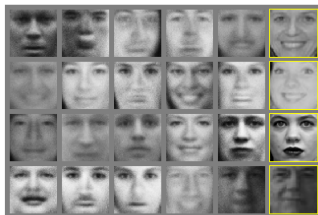
$$\max_D V(D, G) = 2 \cdot (\text{KL}(p_{\text{data}} \parallel (p_g + p_{\text{data}})/2) + \text{KL}(p_g \parallel (p_g + p_{\text{data}})/2)) - \log 4.$$

Step 3: $\text{KL} \geq 0$ with equality iff $p_g = p_{\text{data}}$, so the global optimum is $p_g = p_{\text{data}}$.

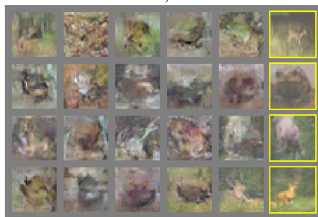
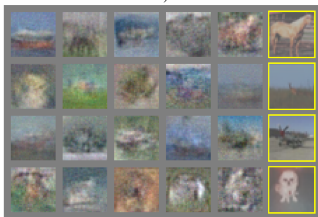
GAN samples: 2014



a)

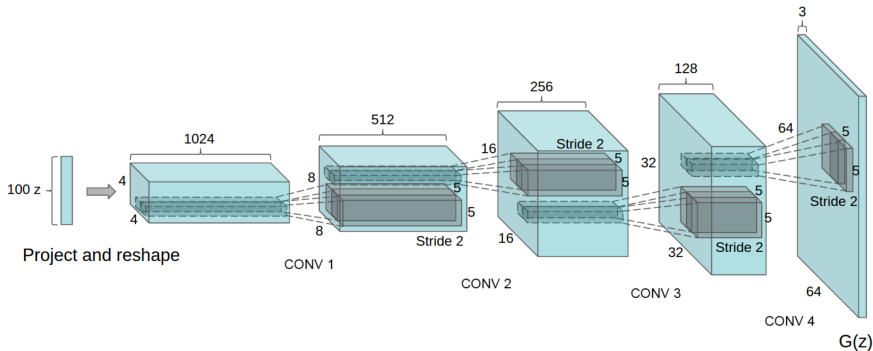


b)



MNIST, Toronto Face Database, CIFAR-10. Rightmost column: nearest training example, to show the model is not memorizing.

DCGAN [Radford et al., 2015]



Idea: improve performance of GANs, with a bunch of empirical “tricks”:

- Replace pooling with strided convolutions (has same effect on image size).
- Batch normalization, ReLU in G (tanh output), LeakyReLU in D ; no fully-connected layers.

DCGAN samples (LSUN bedrooms)



Scaling GANs up: 1024×1024 faces



Progressive growing [Karras et al., 2018]



StyleGAN [Karras et al., 2019]

Bunch of empirical tricks to deal with instabilities during training (min-max optimization is hard!)

- Large batches, wide layers, and a truncation trick at sampling time: truncate latent z to bounded domain.

Class-conditional GANs at ImageNet scale



- Conditioning on a class label y : $G(z, y)$, $D(x, y)$.

Class-conditional GANs at ImageNet scale



- Conditioning on a class label y : $G(z, y)$, $D(x, y)$.
- By 2018–2019 GAN samples at 512×512 were routinely hard to distinguish from photographs – but training remained unstable and mode coverage hard to verify.

How do we measure image quality?

The problem: no likelihood for GANs, and likelihood is a poor proxy for perceptual quality anyway. We want a number that rewards **fidelity** *and* **diversity**.

¹Architecture we did not cover.

How do we measure image quality?

The problem: no likelihood for GANs, and likelihood is a poor proxy for perceptual quality anyway. We want a number that rewards **fidelity** and **diversity**.

Inception Score (IS). Push samples through a pretrained Inception¹ classifier:

$$\text{IS}(G) = \exp \left(\mathbb{E}_{x \sim p_g} \text{KL}(p(y | x) \| p(y)) \right).$$

- $p(y | x)$ peaked (recognizable object) **and** $p(y) = \mathbb{E}_x p(y | x)$ near-uniform (diverse) $\Rightarrow$ high.
- **Higher is better.** Ignores the real data entirely – only the classifier's opinion counts.

¹Architecture we did not cover.

Fréchet Inception Distance (FID)

Embed real and generated images with an Inception network (pool3, 2048-d) and fit a Gaussian to each:

$$\text{FID} = \|\mu_r - \mu_g\|_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}\right).$$

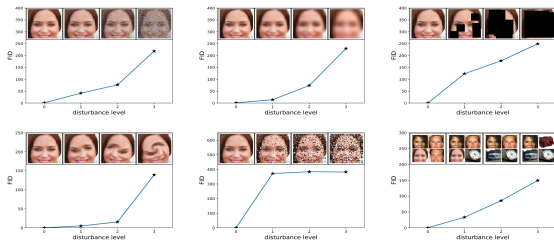
- Fréchet (Wasserstein-2) distance between the two Gaussians. **Lower is better**; 0 iff the moments match.
- Unlike IS, it compares against real data.

Fréchet Inception Distance (FID)

Embed real and generated images with an Inception network (pool3, 2048-d) and fit a Gaussian to each:

$$\text{FID} = \|\mu_r - \mu_g\|_2^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}).$$

- Fréchet (Wasserstein-2) distance between the two Gaussians. **Lower is better**; 0 iff the moments match.
- Unlike IS, it compares against real data.



FID rises monotonically as noise, blur, occlusion, or swirl are applied.

Part II. Language modeling

How to turn text into a vector

Two things must be decided before any architecture:

- **Tokenization:** what are the atomic symbols? Split the string into a sequence $w_1, \dots, w_T$ drawn from a finite vocabulary $\mathcal{V}$.
- **Representation:** each token id $w_t \in \{1, \dots, |\mathcal{V}|\}$ becomes a vector by table lookup,

$$x_t = E_{w_t, :} \in \mathbb{R}^d, \quad E \in \mathbb{R}^{|\mathcal{V}| \times d} \text{ (learned).}$$

How to turn text into a vector

Two things must be decided before any architecture:

- **Tokenization:** what are the atomic symbols? Split the string into a sequence $w_1, \dots, w_T$ drawn from a finite vocabulary $\mathcal{V}$.
- **Representation:** each token id $w_t \in \{1, \dots, |\mathcal{V}|\}$ becomes a vector by table lookup,

$$x_t = E_{w_t, :} \in \mathbb{R}^d, \quad E \in \mathbb{R}^{|\mathcal{V}| \times d} \text{ (learned)}.$$

Then: a model that maps a variable-length sequence $x_{1:T}$ to predictions – the job of an RNN, LSTM, transformer, etc.

Tokenization before Transformers

Word-level (dominant through ~2015):

- Split on whitespace/punctuation, lowercase, map rare words to <unk>.
- Penn Treebank: vocabulary capped at 10K. One Billion Word: 800K types kept.
- **Problem:** fixed vocabulary $\Rightarrow$ out-of-vocabulary words, huge softmax, no morphology (*run* / *running* unrelated), and names/numbers are unlearnable.

Tokenization before Transformers

Word-level (dominant through ~2015):

- Split on whitespace/punctuation, lowercase, map rare words to <unk>.
- Penn Treebank: vocabulary capped at 10K. One Billion Word: 800K types kept.
- **Problem:** fixed vocabulary $\Rightarrow$ out-of-vocabulary words, huge softmax, no morphology (*run* / *running* unrelated), and names/numbers are unlearnable.

Character-level:

- $|\mathcal{V}| \approx 100$, no OOV – but sequences become $5\times$ longer and long-range dependencies harder.

Tokenization before Transformers

Word-level (dominant through ~ 2015):

- Split on whitespace/punctuation, lowercase, map rare words to $\langle \text{unk} \rangle$.
- Penn Treebank: vocabulary capped at 10K. One Billion Word: 800K types kept.
- **Problem:** fixed vocabulary $\Rightarrow$ out-of-vocabulary words, huge softmax, no morphology (*run* / *running* unrelated), and names/numbers are unlearnable.

Character-level:

- $|\mathcal{V}| \approx 100$, no OOV – but sequences become $5\times$ longer and long-range dependencies harder.

Cost of the output layer: predicting the next word costs $O(|\mathcal{V}|d)$ per step.

Subword tokenization: byte-pair encoding [Sennrich et al., 2016]

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i], symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Idea: start from characters; repeatedly merge the most frequent adjacent symbol pair; stop after M merges.

Subword tokenization: byte-pair encoding [Sennrich et al., 2016]

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i], symbols[i+1]] += freq
    return pairs

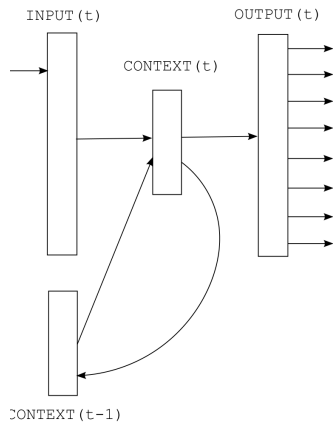
def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
        'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Idea: start from characters; repeatedly merge the most frequent adjacent symbol pair; stop after M merges.

- Frequent words end up as single tokens; rare words are split into pieces:
lower → low er,
Abendessen → Abend essen.
- **Open vocabulary** with a fixed table size (30K–60K).
- Introduced for neural machine translation; still what Transformers use today.

Recurrent neural networks [Mikolov et al., 2010]

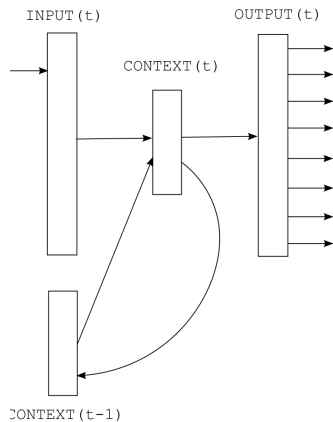


A hidden state carries information forward in time:

$$h_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t + b_h),$$

$$\hat{y}_t = \text{softmax}(W_{hy} h_t + b_y).$$

Recurrent neural networks [Mikolov et al., 2010]



A hidden state carries information forward in time:

$$h_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t + b_h),$$

$$\hat{y}_t = \text{softmax}(W_{hy} h_t + b_y).$$

- Same weights (W_{hh} , W_{xh} , W_{hy}) at every step: **parameter sharing across time**, exactly as convolution shares across space.
- Handles **any** sequence length; h_t is a fixed-size summary of $x_{1:t}$.
- $h_0 = 0$ (or learned).

Unrolling and backpropagation through time

Unroll the recurrence into a feed-forward graph of depth T with tied weights:

$$x_1 \rightarrow h_1 \rightarrow h_2 \rightarrow \cdots \rightarrow h_T, \quad \mathcal{L} = \sum_{t=1}^T \mathcal{L}_t(\hat{y}_t, y_t).$$

Unrolling and backpropagation through time

Unroll the recurrence into a feed-forward graph of depth T with tied weights:

$$x_1 \rightarrow h_1 \rightarrow h_2 \rightarrow \cdots \rightarrow h_T, \quad \mathcal{L} = \sum_{t=1}^T \mathcal{L}_t(\hat{y}_t, y_t).$$

Gradient of the loss at time T w.r.t. an earlier state:

$$\frac{\partial \mathcal{L}_T}{\partial h_t} = \frac{\partial \mathcal{L}_T}{\partial h_T} \prod_{k=t+1}^T \frac{\partial h_k}{\partial h_{k-1}}, \quad \frac{\partial h_k}{\partial h_{k-1}} = W_{hh}^\top \text{diag}(\sigma'(\cdot)).$$

Unrolling and backpropagation through time

Unroll the recurrence into a feed-forward graph of depth T with tied weights:

$$x_1 \rightarrow h_1 \rightarrow h_2 \rightarrow \cdots \rightarrow h_T, \quad \mathcal{L} = \sum_{t=1}^T \mathcal{L}_t(\hat{y}_t, y_t).$$

Gradient of the loss at time T w.r.t. an earlier state:

$$\frac{\partial \mathcal{L}_T}{\partial h_t} = \frac{\partial \mathcal{L}_T}{\partial h_T} \prod_{k=t+1}^T \frac{\partial h_k}{\partial h_{k-1}}, \quad \frac{\partial h_k}{\partial h_{k-1}} = W_{hh}^\top \text{diag}(\sigma'(\cdot)).$$

- A product of $T - t$ Jacobians. Truncated BPTT (backprop over the last k steps) is used in practice.

Vanishing and exploding gradients

Roughly, $\left\| \prod_{k=t+1}^T \partial h_k / \partial h_{k-1} \right\| \sim \lambda^{T-t}$ where λ is the relevant singular value of W_{hh} :

- $\lambda < 1$: gradient $\rightarrow 0$ – the model **cannot learn** dependencies more than ~ 10 steps back.
- $\lambda > 1$: gradient blows up – training diverges.

Vanishing and exploding gradients

Roughly, $\left\| \prod_{k=t+1}^T \partial h_k / \partial h_{k-1} \right\| \sim \lambda^{T-t}$ where λ is the relevant singular value of W_{hh} :

- $\lambda < 1$: gradient $\rightarrow 0$ – the model **cannot learn** dependencies more than ~ 10 steps back.
- $\lambda > 1$: gradient blows up – training diverges.

Partial fix:

- **Gradient clipping** rescales $g \leftarrow g \cdot \min(1, c/\|g\|)$; handles explosion, not vanishing.

Vanishing and exploding gradients

Roughly, $\left\| \prod_{k=t+1}^T \partial h_k / \partial h_{k-1} \right\| \sim \lambda^{T-t}$ where λ is the relevant singular value of W_{hh} :

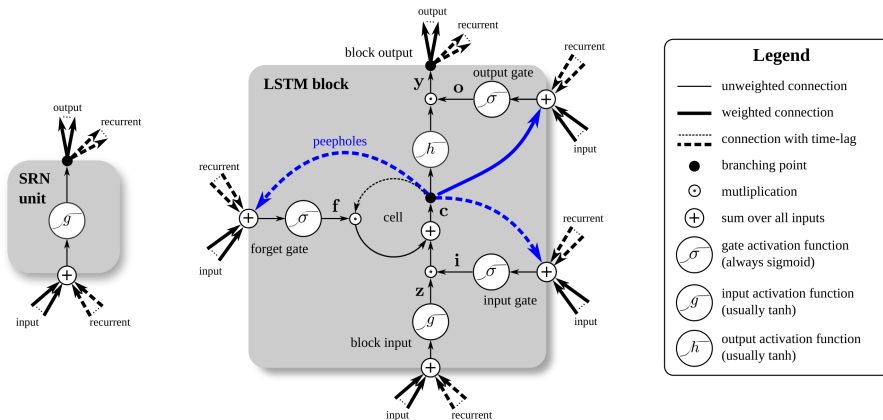
- $\lambda < 1$: gradient $\rightarrow 0$ – the model **cannot learn** dependencies more than ~ 10 steps back.
- $\lambda > 1$: gradient blows up – training diverges.

Partial fix:

- **Gradient clipping** rescales $g \leftarrow g \cdot \min(1, c/\|g\|)$; handles explosion, not vanishing.

More promising fix: change the recurrence so that there is a path along which the gradient is multiplied by ≈ 1 .

LSTM [Hochreiter & Schmidhuber, 1997]



Left: a plain recurrent unit. **Right:** an LSTM block – a cell with input, forget, and output gates.

LSTM equations

Idea: Maintain a memory “cell” c_t over time.

At step t – Input to LSTM layer: x_t ; **Output to LSTM layer:** h_t

LSTM equations

Idea: Maintain a memory “cell” c_t over time.

At step t – Input to LSTM layer: x_t ; **Output to LSTM layer:** h_t

LSTM layer computes several auxiliary variables:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad \text{input gate}$$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad \text{forget gate}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad \text{output gate}$$

$$g_t = \tanh(W_g[h_{t-1}, x_t] + b_g) \quad \text{candidate update}$$

$$\boxed{c_t = f_t \odot c_{t-1} + i_t \odot g_t} \quad h_t = o_t \odot \tanh(c_t).$$

LSTM equations

Idea: Maintain a memory “cell” c_t over time.

At step t – Input to LSTM layer: x_t ; **Output to LSTM layer:** h_t

LSTM layer computes several auxiliary variables:

$$\begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) && \text{input gate} \\ f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) && \text{forget gate} \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) && \text{output gate} \\ g_t &= \tanh(W_g[h_{t-1}, x_t] + b_g) && \text{candidate update} \end{aligned}$$

$$\boxed{c_t = f_t \odot c_{t-1} + i_t \odot g_t} \quad h_t = o_t \odot \tanh(c_t).$$

Why it fixes vanishing gradients: the cell update $c_{t-1} \rightarrow c_t$ is **additive**, so

$$\frac{\partial c_t}{\partial c_{t-1}} = \text{diag}(f_t) \approx I \text{ when the forget gate is open (i.e., } f_t \approx \mathbf{1})$$

Task 1: language modeling

Goal: model the distribution over token sequences by the chain rule; for embedding matrix $E_o \in \mathbb{R}^{V \times d}$ and network F_θ (e.g., RNN/LSTM),

$$p_\theta(w_1, \dots, w_T) = \prod_{t=1}^T p_\theta(w_t \mid w_{1:t-1}), \quad p_\theta(w_t \mid w_{1:t-1}) = \text{softmax}(E_o \cdot F_\theta(w_{1:t-1})).$$

Training: exactly cross-entropy again, summed over positions.

Task 1: language modeling

Goal: model the distribution over token sequences by the chain rule; for embedding matrix $E_o \in \mathbb{R}^{V \times d}$ and network F_θ (e.g., RNN/LSTM),

$$p_\theta(w_1, \dots, w_T) = \prod_{t=1}^T p_\theta(w_t \mid w_{1:t-1}), \quad p_\theta(w_t \mid w_{1:t-1}) = \text{softmax}(E_o \cdot F_\theta(w_{1:t-1})).$$

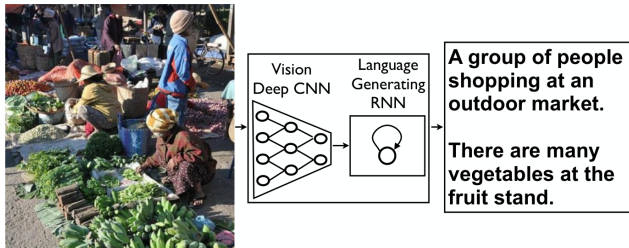
Training: exactly cross-entropy again, summed over positions.

Metric – perplexity:

$$\text{PPL} = \exp\left(-\frac{1}{T} \sum_{t=1}^T \log p(w_t \mid w_{1:t-1})\right) = e^{\text{average cross-entropy}}.$$

Interpretation: effective number of equally likely choices per token. Lower is better.

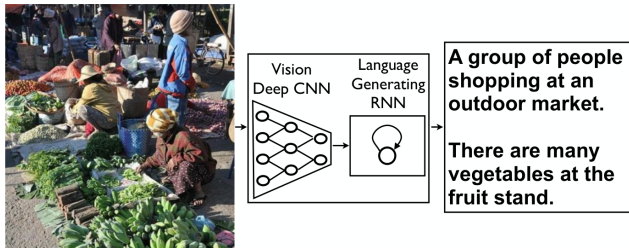
Task 2: image captioning



Encoder-decoder across modalities [Vinyals et al., 2015]: a CNN produces an image vector, an LSTM generates the sentence.

$$\theta^* = \arg \max_{\theta} \sum_{(I, S)} \log p_{\theta}(S | I), \quad \log p_{\theta}(S | I) = \sum_{t=0}^N \log p_{\theta}(S_t | I, S_{0:t-1}).$$

Task 2: image captioning



Encoder-decoder across modalities [Vinyals et al., 2015]: a CNN produces an image vector, an LSTM generates the sentence.

$$\theta^* = \arg \max_{\theta} \sum_{(I, S)} \log p_{\theta}(S | I), \quad \log p_{\theta}(S | I) = \sum_{t=0}^N \log p_{\theta}(S_t | I, S_{0:t-1}).$$

The image is fed in as the first “word”.

Captioning: outputs

A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



A skateboarder does a trick on a ramp.



A dog is jumping to catch a frisbee.



A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



A little girl in a pink hat is blowing bubbles.



A refrigerator filled with lots of food and drinks.



A herd of elephants walking across a dry grass field.



A close up of a cat laying on a couch.



A red motorcycle parked on the side of the road.



A yellow school bus parked in a parking lot.



Describes without errors

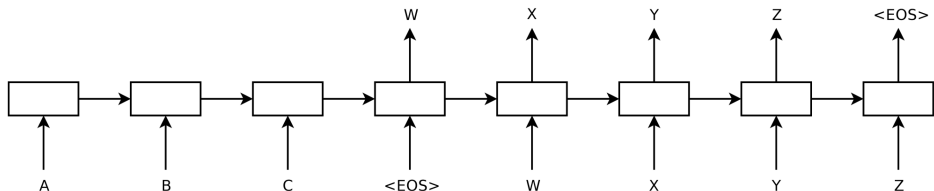
Describes with minor errors

Somewhat related to the image

Unrelated to the image

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

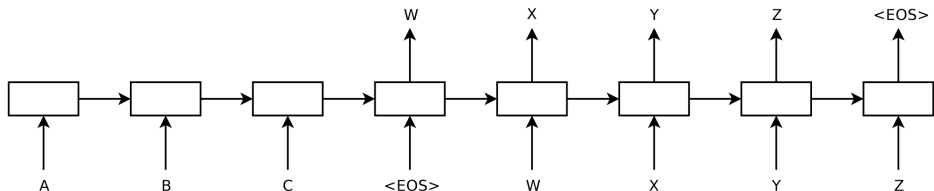
Task 3: machine translation



Sequence to sequence [Sutskever et al., 2014]: read the source with one LSTM into a fixed vector $c = h_T$, then generate the target with a second LSTM,

$$p(y_1, \dots, y_{T'} \mid x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t \mid c, y_{1:t-1}).$$

Task 3: machine translation



Sequence to sequence [Sutskever et al., 2014]: read the source with one LSTM into a fixed vector $c = h_T$, then generate the target with a second LSTM,

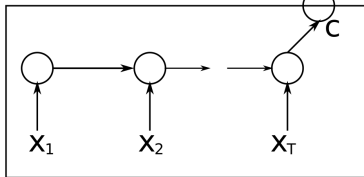
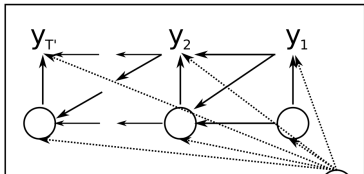
$$p(y_1, \dots, y_{T'} \mid x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t \mid c, y_{1:t-1}).$$

- 4 layers, 1000 units, 384M parameters
- Neat trick: reversing the source sentence gave a large gain.

The encoder-decoder bottleneck

Everything the decoder knows about the source must fit in **one fixed vector c** .

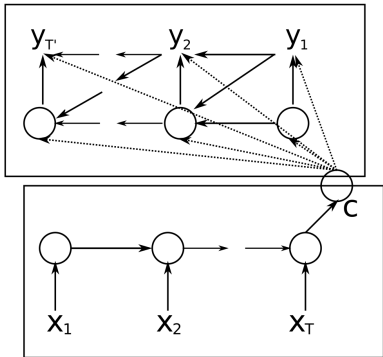
Decoder



Encoder

The encoder-decoder bottleneck

Decoder



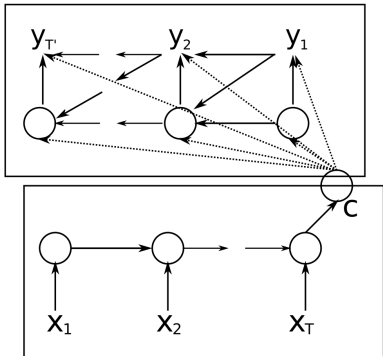
Encoder

Everything the decoder knows about the source must fit in **one fixed vector c** .

- Translation quality degrades sharply as sentences get longer than the training distribution.
- The information a decoder needs at step t is usually **local** to a few source words – but c has already averaged everything together.

The encoder-decoder bottleneck

Decoder



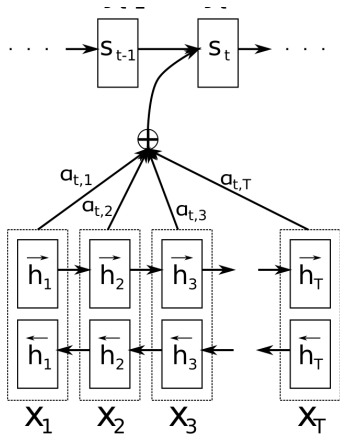
Encoder

Everything the decoder knows about the source must fit in **one fixed vector c** .

- Translation quality degrades sharply as sentences get longer than the training distribution.
- The information a decoder needs at step t is usually **local** to a few source words – but c has already averaged everything together.

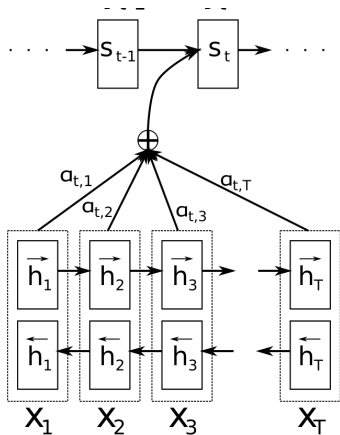
Idea: let the decoder look back at all source states, and learn where to look.

Attention [Bahdanau et al., 2015]



Encoder: a **bidirectional** RNN gives one annotation per source word, $h_j = [\vec{h}_j; \overleftarrow{h}_j]$.

Attention [Bahdanau et al., 2015]



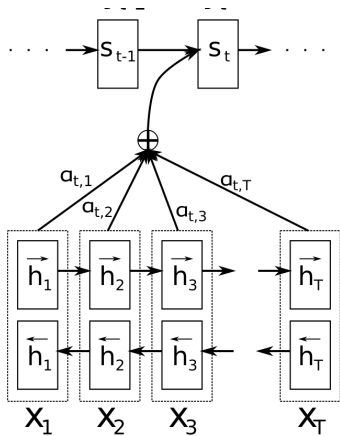
Encoder: a **bidirectional** RNN gives one annotation per source word, $h_j = [\vec{h}_j; \overleftarrow{h}_j]$.

Decoder: RNN that maintains a state s_i . Get information from encoder using **context vector** c_i that *attends* to each state h_j :

$$e_{ij} = v^T \tanh(Ws_{i-1} + Uh_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^T \exp(e_{ik})}, \quad c_i = \sum_{j=1}^T \alpha_{ij} h_j.$$

Attention [Bahdanau et al., 2015]



Encoder: a **bidirectional** RNN gives one annotation per source word, $h_j = [\vec{h}_j; \overleftarrow{h}_j]$.

Decoder: RNN that maintains a state s_i . Get information from encoder using **context vector** c_i that *attends* to each state h_j :

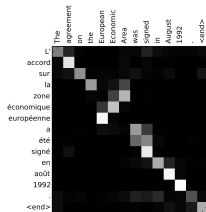
$$e_{ij} = v^T \tanh(Ws_{i-1} + Uh_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^T \exp(e_{ik})}, \quad c_i = \sum_{j=1}^T \alpha_{ij} h_j.$$

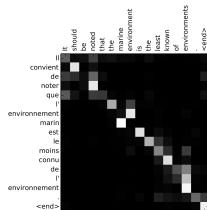
$$s_i = \text{RNN}(s_{i-1}, y_{i-1}, c_i), \quad p(y_i | y_{<i}, x) = g(y_{i-1}, s_i, c_i).$$

The weights α_{ij} form a soft alignment.

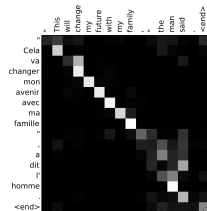
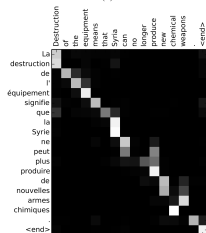
What attention learns



(a)

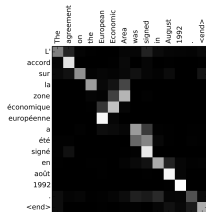


(b)

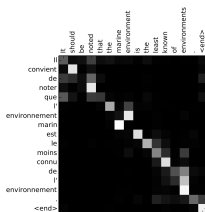


Attention weight α_{ij} (white = 1) for English $\rightarrow$ French.

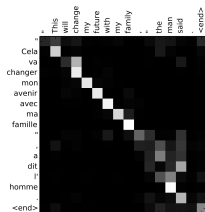
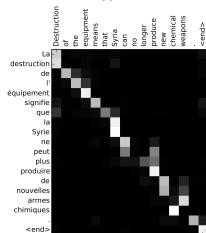
What attention learns



(a)



(b)



Attention weight α_{ij} (white = 1) for English→French.

- Mostly monotonic, but correctly non-monotonic where the languages reorder (*European Economic Area* → *zone économique européenne*).
- Learned with **no alignment supervision**.
- Removes the length degradation: performance on 50-word sentences no longer collapses.

Attention variants [Luong et al., 2015]

Scoring functions between decoder state s_t and source state h_j :

$$\text{score}(s_t, h_j) = \begin{cases} s_t^\top h_j & \text{dot} \\ s_t^\top W_a h_j & \text{general} \\ v_a^\top \tanh(W_a[s_t; h_j]) & \text{concat (Bahdanau-style)} \end{cases}$$

Attention variants [Luong et al., 2015]

Scoring functions between decoder state s_t and source state h_j :

$$\text{score}(s_t, h_j) = \begin{cases} s_t^\top h_j & \text{dot} \\ s_t^\top W_a h_j & \text{general} \\ v_a^\top \tanh(W_a[s_t; h_j]) & \text{concat (Bahdanau-style)} \end{cases}$$

- **Global** attention attends over all source positions; **local** attention restricts to a learned window.

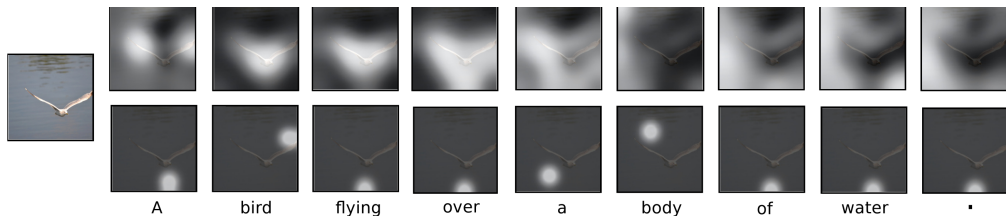
Attention variants [Luong et al., 2015]

Scoring functions between decoder state s_t and source state h_j :

$$\text{score}(s_t, h_j) = \begin{cases} s_t^\top h_j & \text{dot} \\ s_t^\top W_a h_j & \text{general} \\ v_a^\top \tanh(W_a[s_t; h_j]) & \text{concat (Bahdanau-style)} \end{cases}$$

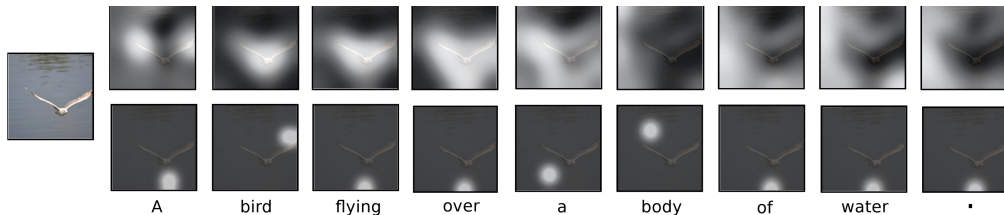
- **Global** attention attends over all source positions; **local** attention restricts to a learned window.

Attention for images: show, attend and tell [Xu et al., 2015]



- Annotations h_j = feature vectors at spatial locations of a CNN feature map (e.g. 14×14).
- At each generated word, α_{ij} is a spatial map – the model looks at the object it is naming.

Attention for images: show, attend and tell [Xu et al., 2015]



- Annotations h_j = feature vectors at spatial locations of a CNN feature map (e.g. 14×14).
- At each generated word, α_{ij} is a spatial map – the model looks at the object it is naming.
- **Soft** attention (differentiable, weighted average) vs. **hard** attention (sample one location, train with REINFORCE).

Early datasets: language modeling

- **Penn Treebank** (WSJ portion): $\sim 1\text{M}$ words, vocabulary truncated to 10K, standard train/valid/test split. Tiny by modern standards – and heavily overfit by the community.
- **Text8 / enwik8**: 100MB of Wikipedia, character-level; measured in bits per character.

Early datasets: language modeling

- **Penn Treebank** (WSJ portion): $\sim 1\text{M}$ words, vocabulary truncated to 10K, standard train/valid/test split. Tiny by modern standards – and heavily overfit by the community.
- **Text8 / enwik8**: 100MB of Wikipedia, character-level; measured in bits per character.
- **One Billion Word Benchmark** (2013): 0.8B words of news crawl, 800K vocabulary – the first LM benchmark where model capacity clearly paid off.

Early datasets: language modeling

- **Penn Treebank** (WSJ portion): $\sim 1\text{M}$ words, vocabulary truncated to 10K, standard train/valid/test split. Tiny by modern standards – and heavily overfit by the community.
- **Text8 / enwik8**: 100MB of Wikipedia, character-level; measured in bits per character.
- **One Billion Word Benchmark** (2013): 0.8B words of news crawl, 800K vocabulary – the first LM benchmark where model capacity clearly paid off.

Example (PTB, preprocessed):

pierre <unk> N years old will join the board as a nonexecutive director
nov. N

(numbers mapped to N, rare words to <unk>, all lowercase, no punctuation)

Early datasets: translation

- **WMT** shared tasks (annual since 2006) define the standard benchmarks.
 - WMT'14 En-Fr: ~ 36 M sentence pairs (Europarl, Common Crawl, UN, news commentary).
 - WMT'14 En-De: ~ 4.5 M pairs – small enough that it became the default testbed.
- Test sets are ~ 3 K professionally translated news sentences (`newstest2014`, ...).
- **IWSLT**: TED talk subtitles, ~ 200 K pairs, used for fast iteration.

Early datasets: translation

- **WMT** shared tasks (annual since 2006) define the standard benchmarks.
 - WMT'14 En-Fr: ~ 36 M sentence pairs (Europarl, Common Crawl, UN, news commentary).
 - WMT'14 En-De: ~ 4.5 M pairs – small enough that it became the default testbed.
- Test sets are ~ 3 K professionally translated news sentences (`newstest2014`, ...).
- **IWSLT**: TED talk subtitles, ~ 200 K pairs, used for fast iteration.

Example pair (Europarl):

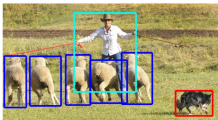
en: Resumption of the session

fr: Reprise de la session

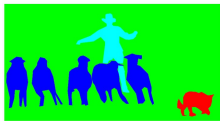
Early datasets: vision + language



(a) Image classification



(b) Object localization



(c) Semantic segmentation



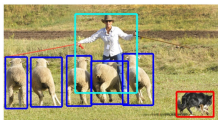
(d) This work

- **Flickr8k / Flickr30k**: 8K / 31K images, 5 crowd-sourced captions each.
- **MSCOCO** [Lin et al., 2014]: 328K images, 2.5M labeled object instances, 91 categories, 5 captions per image.

Early datasets: vision + language



(a) Image classification



(b) Object localization



(c) Semantic segmentation



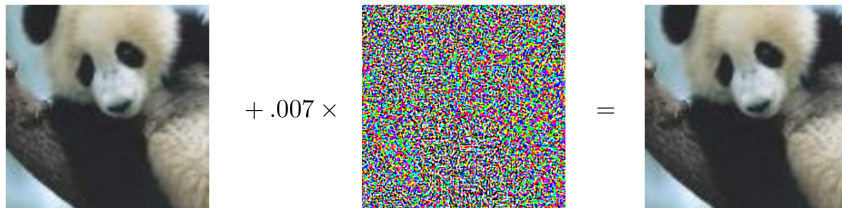
(d) This work

- **Flickr8k / Flickr30k**: 8K / 31K images, 5 crowd-sourced captions each.
- **MSCOCO** [Lin et al., 2014]: 328K images, 2.5M labeled object instances, 91 categories, 5 captions per image.
- Non-iconic images (objects in context, occluded, off-center) – deliberately harder than ImageNet-style single-object photos.
- Supported detection, segmentation, and captioning with one annotation effort.

Sample captions: "A man riding a wave on top of a surfboard." / "Two dogs play in the grass."

Part III. Adversarial examples

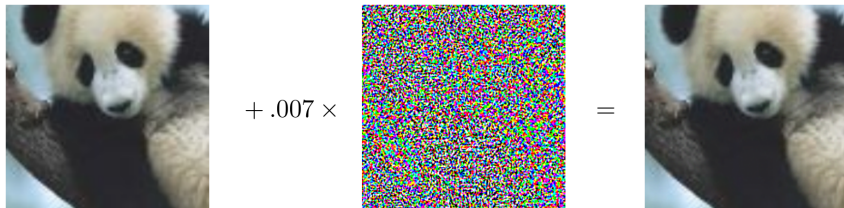
Adversarial examples [Szegedy et al., 2013; Goodfellow et al., 2014]



Definition. Given a classifier f_θ and a correctly classified (x, y) , find

$$x' = x + \delta, \quad \|\delta\|_\infty \leq \epsilon, \quad \text{such that} \quad \arg \max_k f_\theta(x')_k \neq y.$$

Adversarial examples [Szegedy et al., 2013; Goodfellow et al., 2014]



Definition. Given a classifier f_θ and a correctly classified (x, y) , find

$$x' = x + \delta, \quad \|\delta\|_\infty \leq \epsilon, \quad \text{such that} \quad \arg \max_k f_\theta(x')_k \neq y.$$

Fast gradient sign method: $x' = x + \epsilon \operatorname{sign}(\nabla_x \mathcal{L}(\theta, x, y))$ – one gradient step.

- $\epsilon = 8/255$ is invisible to a human, and flips most undefended networks.
- Perturbations **transfer** across architectures and training sets.

Adversarial training as a min-max problem [Madry et al., 2017]

Replace empirical risk minimization by a **saddle-point** problem:

$$\min_{\theta} \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} \mathcal{L}(\theta, x + \delta, y) \right], \quad \mathcal{S} = \{\delta : \|\delta\|_{\infty} \leq \epsilon\}.$$

- **Inner max:** the attacker finds the worst perturbation inside the threat model.
- **Outer min:** the learner fits parameters robust to that attacker.

Adversarial training as a min-max problem [Madry et al., 2017]

Replace empirical risk minimization by a **saddle-point** problem:

$$\min_{\theta} \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} \mathcal{L}(\theta, x + \delta, y) \right], \quad \mathcal{S} = \{\delta : \|\delta\|_{\infty} \leq \epsilon\}.$$

- **Inner max:** the attacker finds the worst perturbation inside the threat model.
- **Outer min:** the learner fits parameters robust to that attacker.

Inner problem solved by projected gradient descent (PGD):

$$x^{t+1} = \Pi_{x+\mathcal{S}} \left(x^t + \alpha \operatorname{sign}(\nabla_x \mathcal{L}(\theta, x^t, y)) \right), \quad x^0 = x + \text{uniform noise}.$$

Adversarial training as a min-max problem [Madry et al., 2017]

Replace empirical risk minimization by a **saddle-point** problem:

$$\min_{\theta} \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} \mathcal{L}(\theta, x + \delta, y) \right], \quad \mathcal{S} = \{\delta : \|\delta\|_{\infty} \leq \epsilon\}.$$

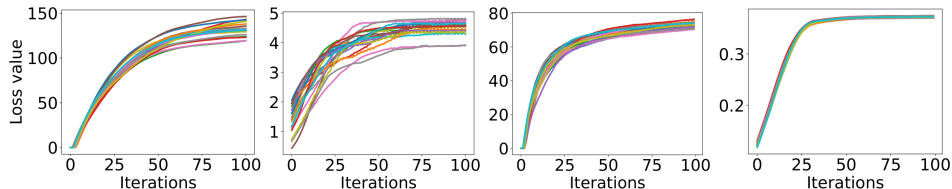
- **Inner max:** the attacker finds the worst perturbation inside the threat model.
- **Outer min:** the learner fits parameters robust to that attacker.

Inner problem solved by projected gradient descent (PGD):

$$x^{t+1} = \Pi_{x+\mathcal{S}} \left(x^t + \alpha \operatorname{sign}(\nabla_x \mathcal{L}(\theta, x^t, y)) \right), \quad x^0 = x + \text{uniform noise}.$$

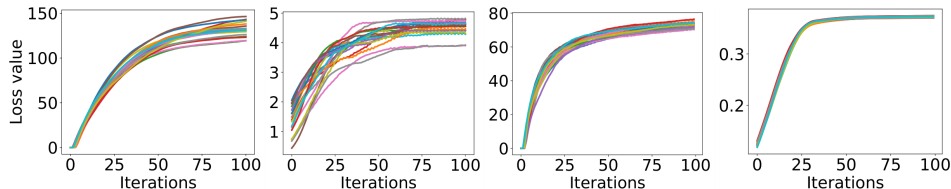
Training: at each SGD step, run k PGD steps to build $x + \delta$, then take one gradient step on θ . Cost: $\approx k+1$ times normal training.

Does the inner maximization behave?



Cross-entropy loss during 20 PGD runs, each from a uniformly random start in the ϵ -ball (MNIST / CIFAR-10, standard and adversarially trained models).

Does the inner maximization behave?



Cross-entropy loss during 20 PGD runs, each from a uniformly random start in the ϵ -ball (MNIST / CIFAR-10, standard and adversarially trained models).

- Values concentrate: many distinct local maxima, but all of similar loss (checked over 10^5 restarts).
- Suggests PGD is a good enough “universal” first-order adversary – so a model robust to PGD is robust to first-order attacks.
- **Price:** lower clean accuracy, much higher training cost, robustness only inside the chosen $\mathcal{S}$.

Summary

- **Convolution** = locality + weight sharing. Convnets + cross-entropy + SGD, scaled up on ImageNet, replaced hand-designed vision pipelines in a single year.
- **Depth** was the lever (VGG), and **residual connections** made depth trainable (ResNet).

Summary

- **Convolution** = locality + weight sharing. Convnets + cross-entropy + SGD, scaled up on ImageNet, replaced hand-designed vision pipelines in a single year.
- **Depth** was the lever (VGG), and **residual connections** made depth trainable (ResNet).
- **Generation:** VAEs optimize a variational lower bound (stable, blurry); GANs play a minimax game (sharp, unstable). Quality is measured indirectly, by IS and FID.
- **Robustness:** tiny worst-case perturbations break these models; adversarial training states the defense as a saddle-point problem.

Summary

- **Convolution** = locality + weight sharing. Convnets + cross-entropy + SGD, scaled up on ImageNet, replaced hand-designed vision pipelines in a single year.
- **Depth** was the lever (VGG), and **residual connections** made depth trainable (ResNet).
- **Generation:** VAEs optimize a variational lower bound (stable, blurry); GANs play a minimax game (sharp, unstable). Quality is measured indirectly, by IS and FID.
- **Robustness:** tiny worst-case perturbations break these models; adversarial training states the defense as a saddle-point problem.
- **Language:** tokenization → RNN → LSTM → encoder-decoder → **attention**. Evaluated with perplexity, BLEU, ROUGE.

Summary

- **Convolution** = locality + weight sharing. Convnets + cross-entropy + SGD, scaled up on ImageNet, replaced hand-designed vision pipelines in a single year.
- **Depth** was the lever (VGG), and **residual connections** made depth trainable (ResNet).
- **Generation**: VAEs optimize a variational lower bound (stable, blurry); GANs play a minimax game (sharp, unstable). Quality is measured indirectly, by IS and FID.
- **Robustness**: tiny worst-case perturbations break these models; adversarial training states the defense as a saddle-point problem.
- **Language**: tokenization → RNN → LSTM → encoder-decoder → **attention**. Evaluated with perplexity, BLEU, ROUGE.

Next: attention was a component. What happens when you make it the whole model?