10707
Deep Learning: Spring 2020

Andrej Risteski

Machine Learning Department

Lecture 19:
Predictive/self-supervised learning: an overview

Unsupervised learning

Learning from data **without** labels.

What can we hope to do:

Task A: Fit a parametrized **structure** (e.g. clustering, low-dimensional subspace, manifold) to data to reveal something meaningful about data. (**Structure learning**)

Task B: Learn a (parametrized) **distribution** *close* to data generating distribution. (**Distribution learning**)

Task C: Learn a (parametrized) distribution that implicitly reveals an **"embedding"/"representation"** of data for downstream tasks. (**Representation/feature learning**)

Entangled! The "structure" and "distribution" often reveals an embedding.

Self-supervised/predictive learning

Given unlabeled data, design supervised tasks that induce a good representation for downstream tasks.

No good mathematical formalization, but the intuition is to "force" the predictor used in the task to learn something "semantically meaningful" about the data.

Self-supervised/predictive learning

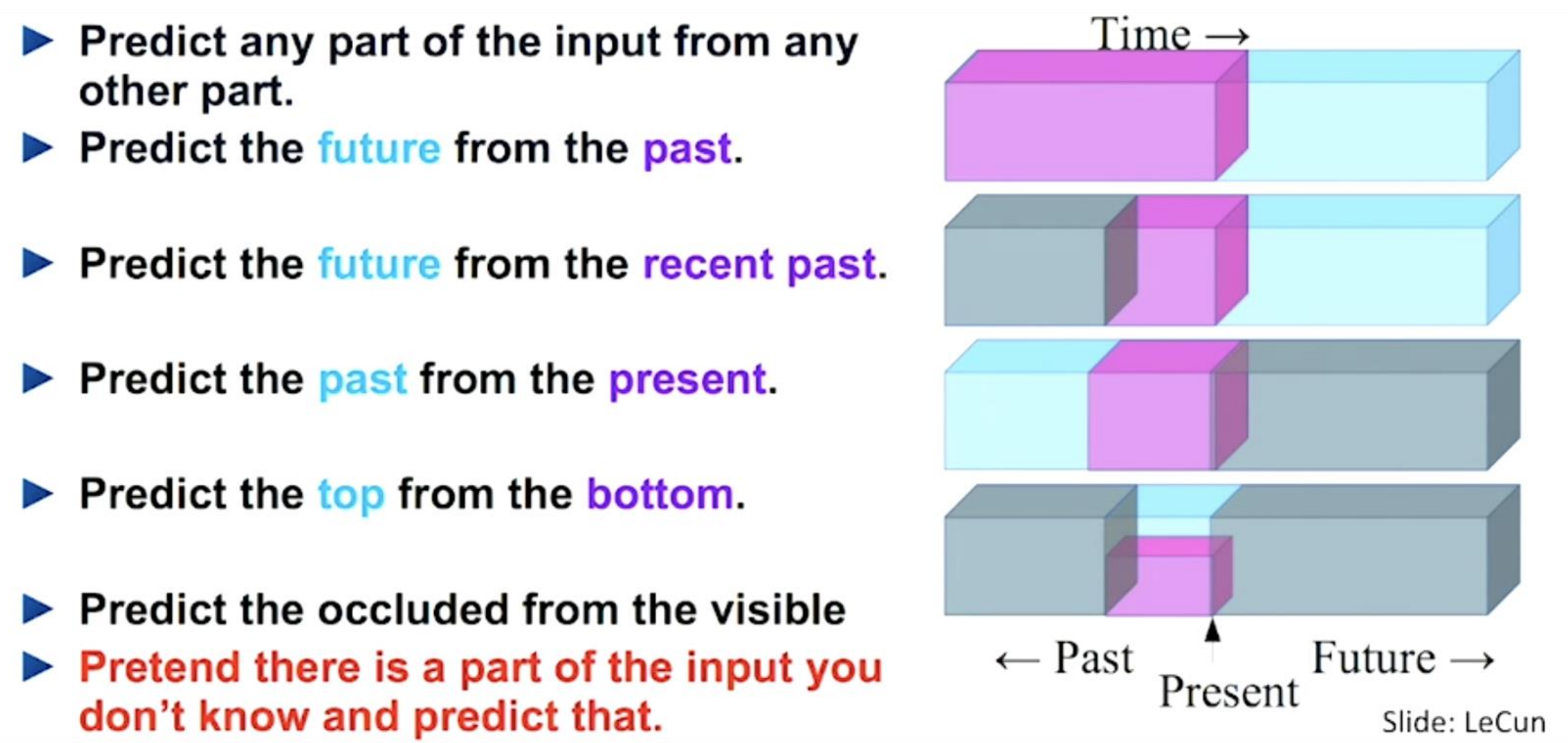


Figure by Yann LeCun

Self-supervised/predictive learning

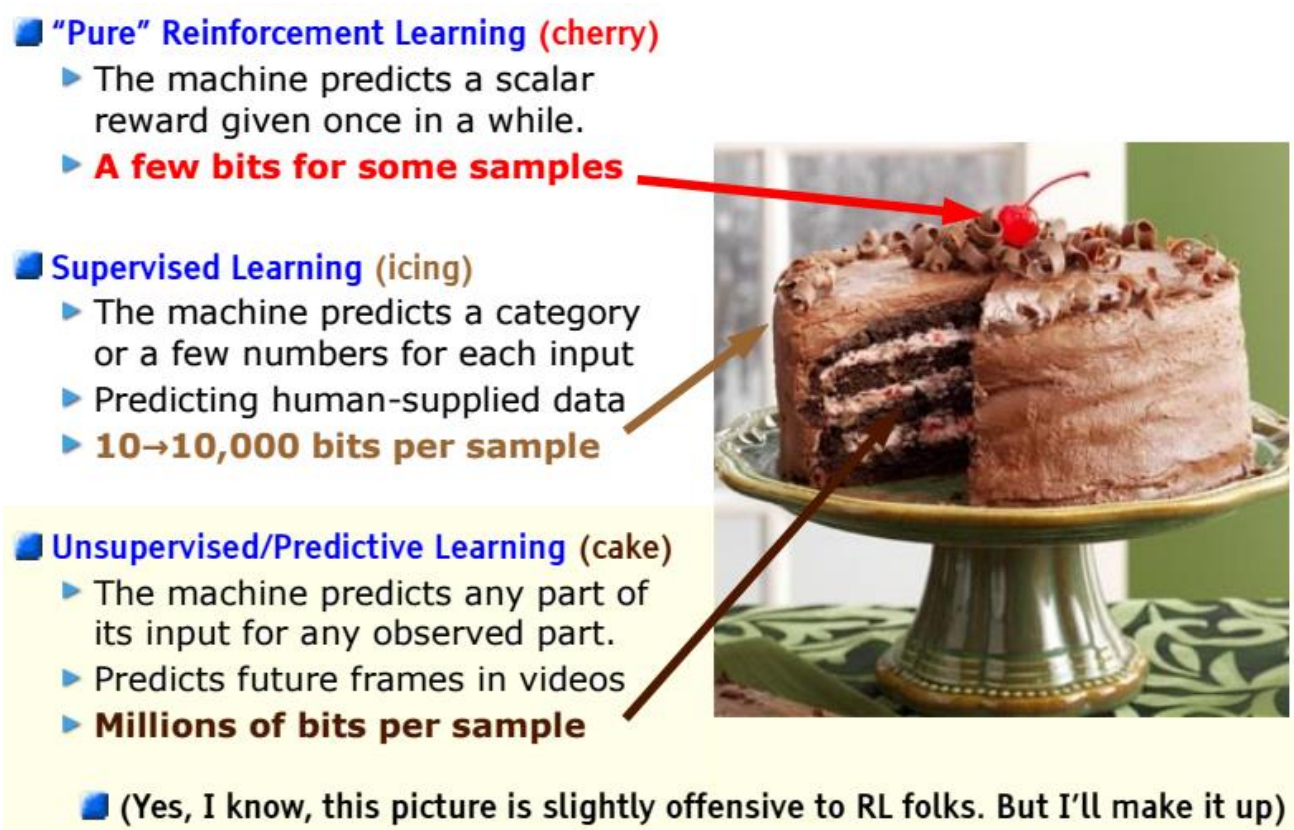
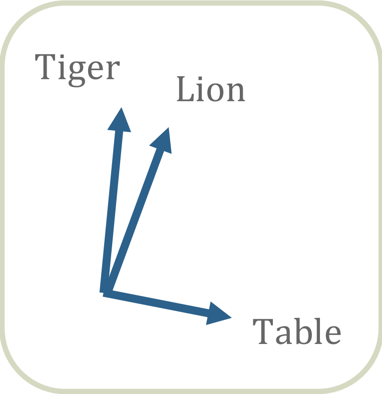


Figure by Yann LeCun

Part I:
Predictive learning in NLP

Word embeddings

Semantically meaningful **vector representations** of words



Example: Inner product (possibly scaled, i.e. cosine similarity) correlates with word similarity.

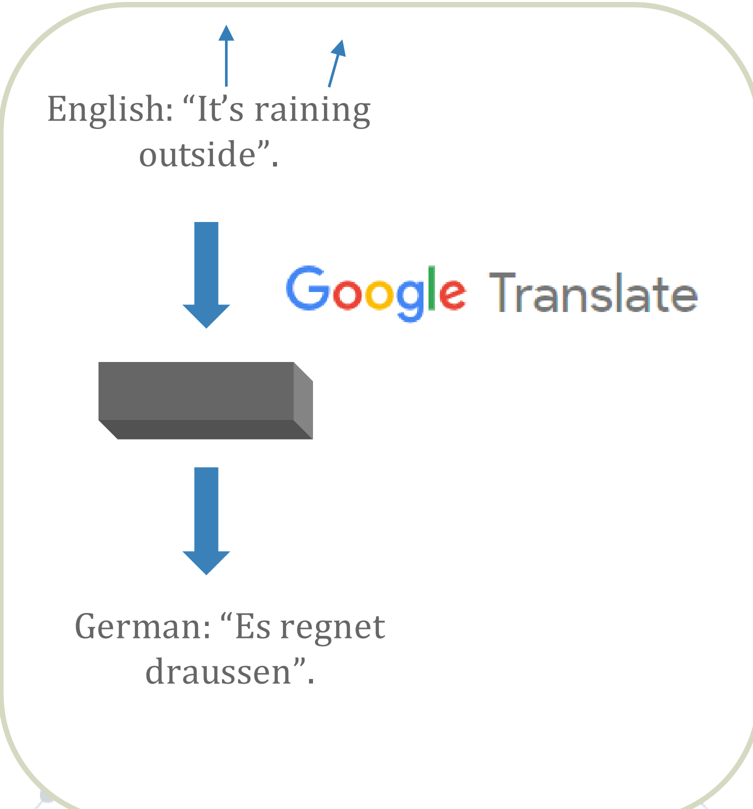
Word embeddings

Semantically meaningful **vector representations** of words

Example: Can use embeddings to do sentiment classification by training a simple (e.g. linear) classifier

Word embeddings

Semantically meaningful **vector representations** of words



Example: Can train a “simple” network that if fed word embeddings for two languages, can effectively translate.

Word embeddings via predictive learning

Basic task: predict the next word, given a few previous ones.

I	am	running	a	little	????

Late: 0.9
Early: 0.05
Tired: 0.04
Table: 0.01

In other words, optimize for

$$\max_{\theta} \sum_{t} \log \; p_{\theta}(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L})$$

Word embeddings via predictive learning

Basic task: predict the next word, given a few previous ones.

$$\max_{\theta} \sum_{t} \log p_{\theta}(x_t | x_{t-1}, x_{t-2}, \dots, x_{t-L})$$

Inspired by classical assumptions in NLP that the underlying distribution is Markov – that is, x_t only depends on the previous few words.

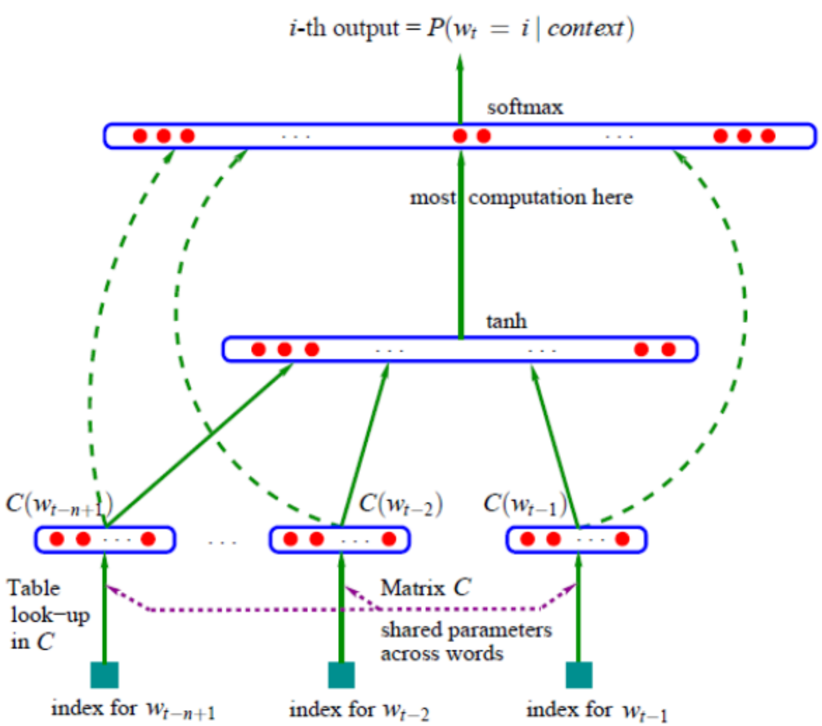
(Of course, this is violated if you wish to model long texts like paragraphs/books.)

The main problem: The trivial way of parametrizing $p_{\theta}(x_t | x_{t-1}, x_{t-2}, \dots, x_{t-L})$ is a “lookup table” with V^L entries.

Word embeddings via predictive learning

Basic task: predict the next word, given a few previous ones.

$$\max_{\theta} \sum_{t} \log p_{\theta}(x_t | x_{t-1}, x_{t-2}, \dots, x_{t-L})$$



[*Bengio-Ducharme-Vincent-Janvin '03*]: A neural parametrization of the above probabilities.

Main ingredients:

Embeddings: A word embedding $C(w)$ for all words w in dictionary.

Non-linear transforms: Potentially deep network taking as inputs i, $C(x_{t-1}), C(x_{t-2}), \dots, C(x_{t-L})$, and outputting some vector o. Can be recurrent net too.

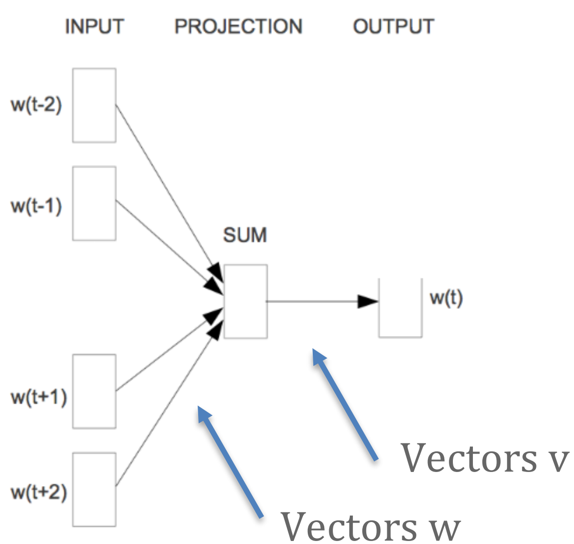
Softmax: Softmax distribution for x_t with parameters given by o.

Word embeddings via predictive learning

Related: predict *middle* word in a sentence, given *surrounding* ones

$$\max_{\theta} \sum_{t} \log p_{\theta}(x_t | x_{t-L}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+L})$$

CBOW (Continuous Bag of Words): proposed by *Mikolov et al.* ‘13



Parametrization is chosen s.t.

$$p_{\theta}(x_t | x_{t-L}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+L}) \propto$$

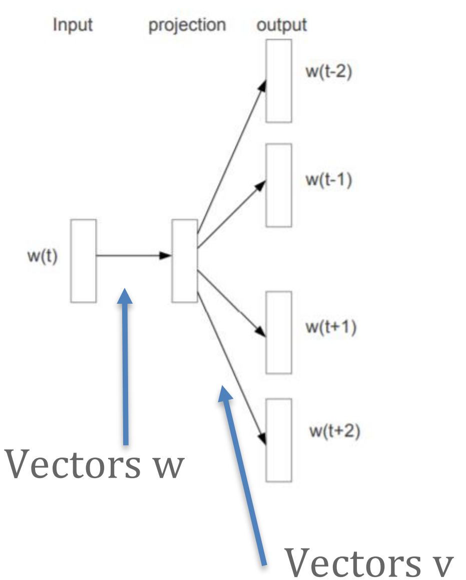
$$\exp\left(v_{x_t}, \sum_{i=t-L}^{t+L} w_{t_i} \right)$$

Word embeddings via predictive learning

Related: predict surrounding words, *given* middle word

$$\max_{\theta} \sum_{t} \sum_{i=t-L, i \neq t}^{t+L} \log \; p_{\theta}(x_i | x_t)$$

Skip-Gram: (also) proposed by *Mikolov et al.* '13



Parametrization is s.t. $p_{\theta}(x_i | x_t) \propto \exp(v_{x_i}, w_{x_t})$

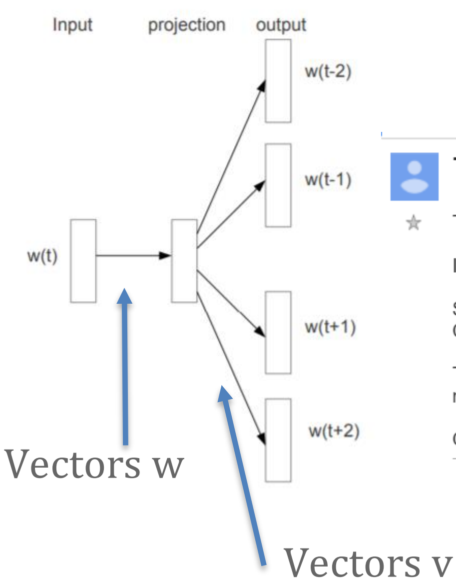
In practice, lots of other tricks are tacked on to deal with the slowest part of training: the softmax distribution (partition function sums over entire vocabulary).
Common ones are *negative sampling, hierarchical softmax*, etc.

Word embeddings via predictive learning

Related: predict surrounding words, *given* middle word

$$\max_{\theta} \sum_{t} \sum_{i=t-L, i\neq t}^{t+L} \log\ p_{\theta}(x_i | x_t)$$

Skip-Gram: (also) proposed by *Mikolov et al.* '13



Parametrization is s.t. $p_{\theta}(x_i | x_t) \propto \exp(v_{x_i}, w_{x_t})$

Tomas Mikolov 10/7/13

There are quite a few differences between the skip-gram and the CBOW models. However, if you have a lot of training data, their performance should be comparable.

If you want to see a list of advantages of each model, then my current experience is:

Skip-gram: works well with small amount of the training data, represents well even rare words or phrases
CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words

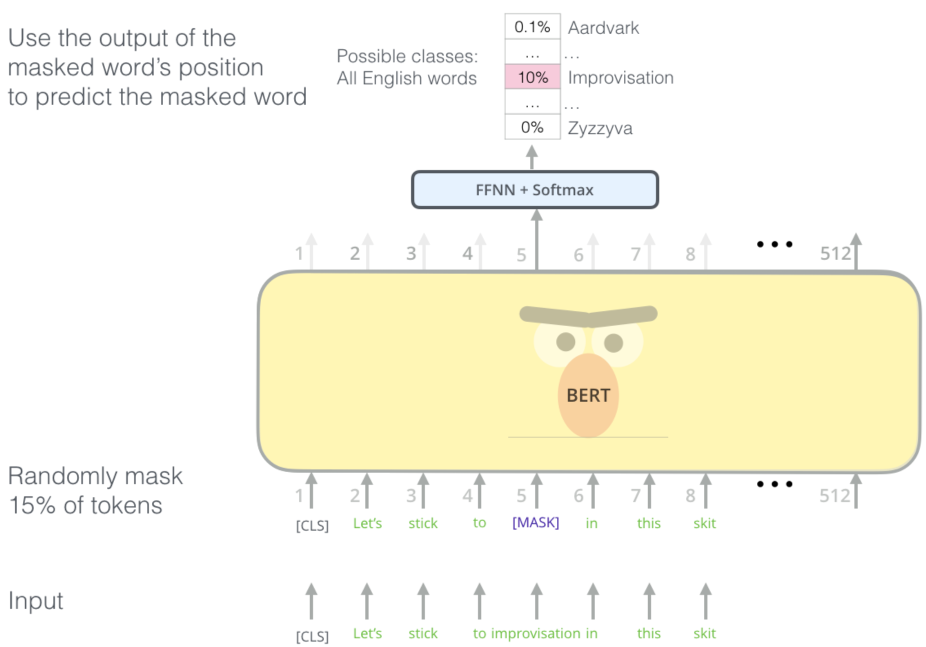
This can get even a bit more complicated if you consider that there are two different ways how to train the models: the normalized hierarchical softmax, and the un-normalized negative sampling. Both work quite differently.

Overall, the best practice is to try few experiments and see what works the best for you, as different applications have different requirements.
- show quoted text -

Word embeddings via predictive learning

Related: predict random *15%* of the words, given the rest

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. ’18.



Pretty much all-across-the-board best-performing representations for most downstream tasks. (Pre fine-tuning, of course.)

Evaluating word embeddings

First variant (predict next word, given previous ones) can be used as a **generative model** for text. (Also called *language model*.) The other ones cannot.

In former case, a natural measure is the cross-entropy

$$-\mathbb{E}_{x_1,x_2,\ldots,x_T} \log p_\theta(x_{\leq T}) = \mathbb{E}_{x_1,x_2,\ldots,x_T} \sum_t \log\ p_\theta(x_t | x_{<t})$$

For convenience, we often take exponential of this (called *perplexity*)

If we do not have a generative model, we have to use **indirect** means.

Evaluating word embeddings

Intrinsic tasks: Test performance of word embeddings on tasks measuring their "semantic" properties. Examples include solving "which is the most similar word" queries, analogy queries (i.e. "man is to woman as king is to ??"

Extrinsic tasks: How well can we "finetune" the word embeddings to solve some (supervised) downstream task. "Finetune" usually means train a (relatively small) feedforward network. Examples of such tasks include:

Part-of-Speech Tagging (determine whether a word is noun/verb/…),
Named Entity Recognition (recognizing named entities like persons, places) – e.g. label a sentence as Picasso$_{\text{[person]}}$ died in France$_{\text{[country]}}$, many others.

Semantic similarity

Observation: similar words tend to have larger (renormalized) inner products (also called cosine similarity).

Precisely, if we look at the word embeddings for words i,j

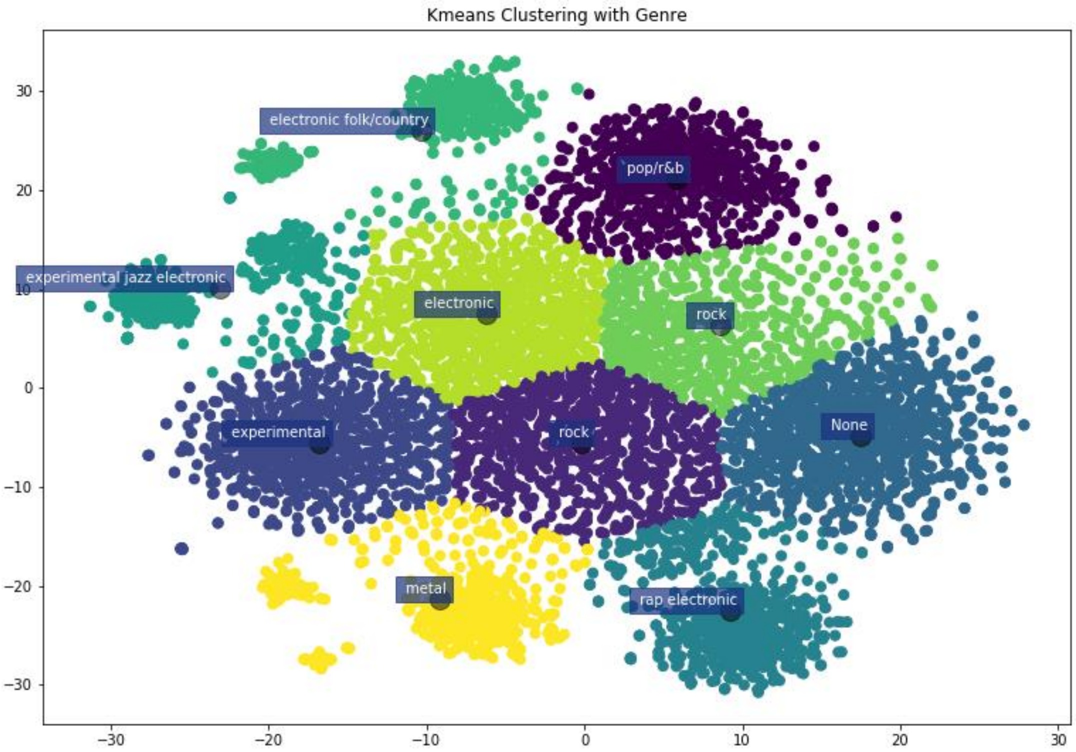
$\left\langle \frac{w_i}{||w_i||}, \frac{w_j}{||w_j||} \right\rangle = \cos(w_i, w_j)$ tends to be larger for similar words i,j

Example: the nearest neighbors to "Frog" look like

To solve semantic similarity query like "which is the most similar word to", output the word with the highest cosine similarity.

Semantic clustering

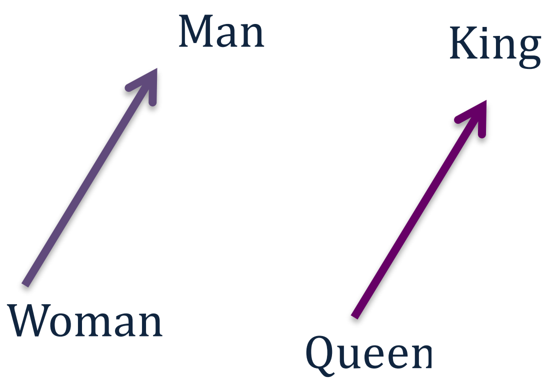
Consequence: clustering word embeddings should give “semantically” relevant clusters.



t-SNE projection of word embeddings for artists (clustered by genre). Image from https://medium.com/free-code-camp/learn-tensorflow-the-word2vec-model-and-the-tsne-algorithm-using-rock-bands-97c99b5dcb3a

Analogies

Observation: You can solve *analogy* queries by linear algebra.



Precisely, w = queen will be the solution to:

$$\operatorname{argmin}_w \|v_w - v_{\text{king}} - (v_{\text{woman}} - v_{\text{man}})\|^2$$

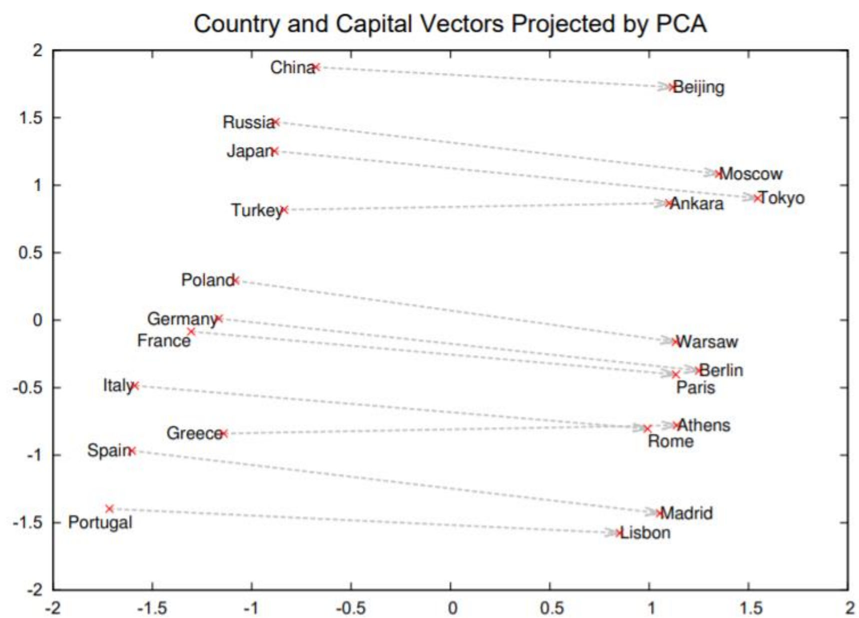


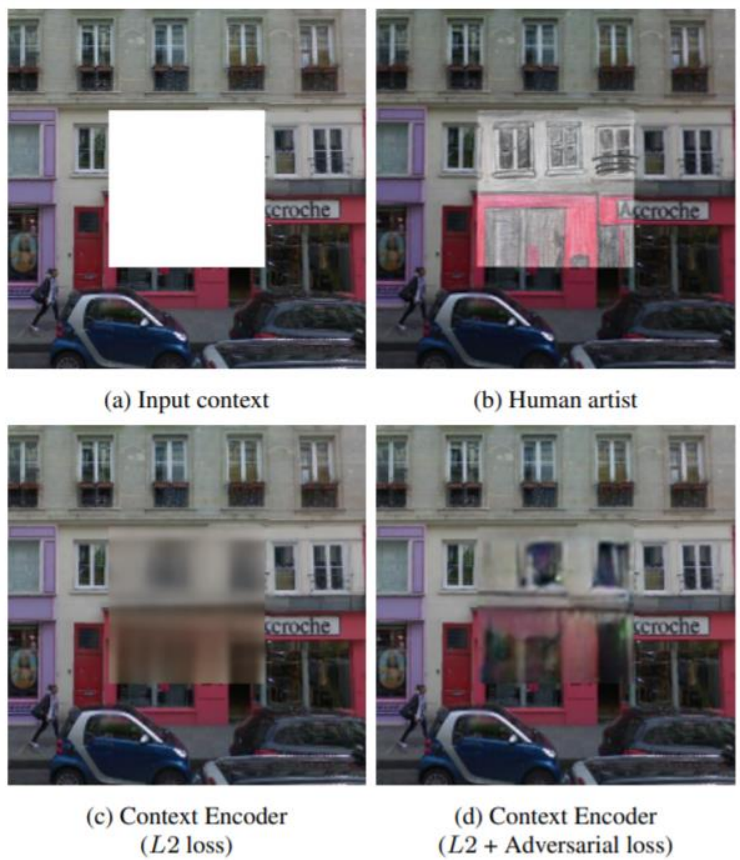
Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Part II:
Predictive learning in vision

Inpainting

The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting



Inpainting

The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

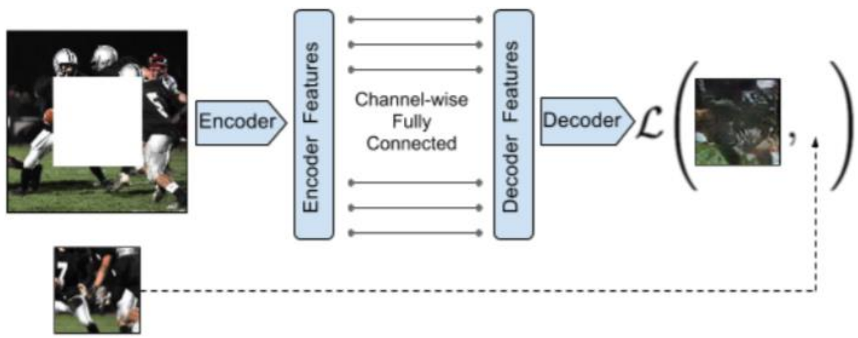


Figure 2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 3.1. The decoder then produces the missing regions in the image.

Architecture:
An encoder E takes a part of image, constructs a representation.

A decoder D takes representation, tries to reconstruct missing part.

Much trickier than in NLP:

As we have seen, meaningful losses for vision are much more difficult to design.

Choice of region to mask out is much more impactful.

Inpainting

The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

If reconstruction loss is l_2: tendency to produce blurry images.

Remember: one of the usefulness of GANs is to provide a better loss for images.

(c) Context Encoder
($L2$ loss)

(d) Context Encoder
($L2$ + Adversarial loss)

Inpainting

The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

If reconstruction loss is l_2: tendency to produce blurry images.

Remember: one of the usefulness of GANs is to provide a better loss for images.

Composition of encoder+decoder

Mask

$$\mathcal{L}_{rec}(x) = \|\hat{M} \odot (x - F((1 - \hat{M}) \odot x))\|_2^2,$$

DC-GAN objective

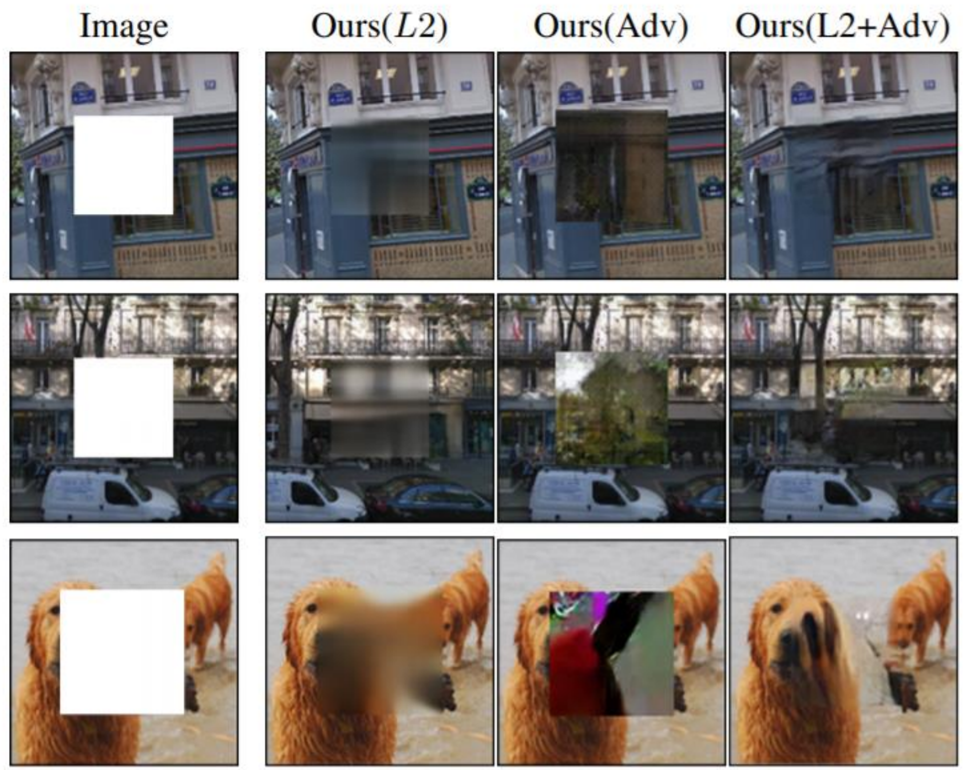
$$\begin{aligned}\mathcal{L}_{adv} = \max_{D} \quad & \mathbb{E}_{x \in \mathcal{X}}[\log(D(x)) \\ & + \log(1 - D(F((1 - \hat{M}) \odot x)))],\end{aligned}$$

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}_{adv}.$$

Inpainting

The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

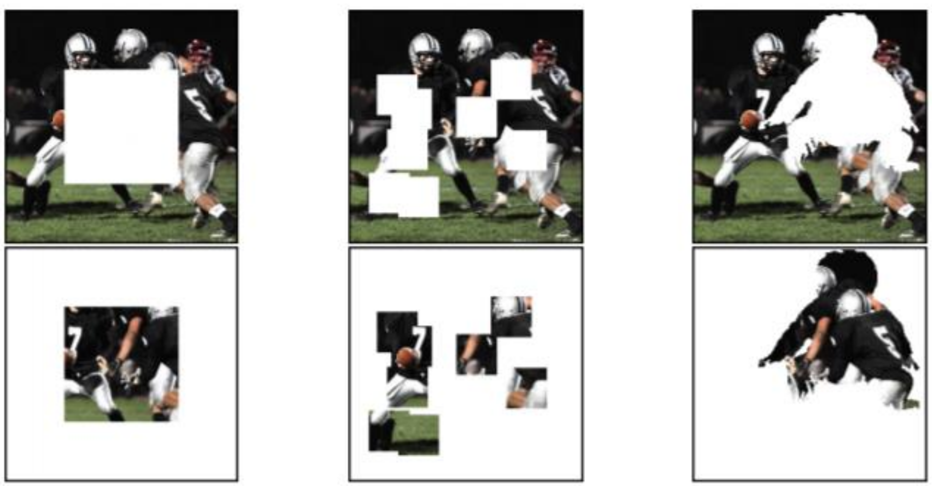


Inpainting

The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

How to choose the region?



(a) Central region (b) Random block (c) Random region

Figure 3: An example of image x with our different region masks $\hat{M}$ applied, as described in Section 3.3.

Task should be "solvable", but not "too easy".

Fixed (central region): tends to produce less generalizeable representations

Random blocks: slightly better, but square borders still hurt.

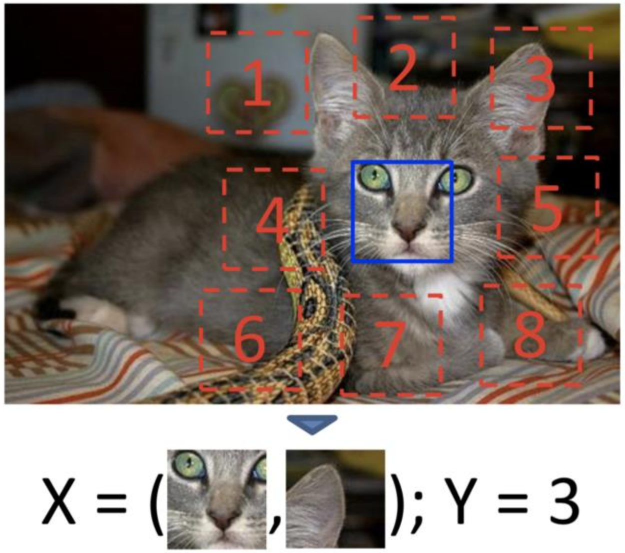
Random silhouette (fully random doesn't make sense – prediction task is too ill-defined) – even better!

Jigsaw puzzles

In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

Doersch et al. '16: Unsupervised Visual Representation Learning by Context Prediction

Task: Predict ordering of two randomly chosen pieces from the image.



Representation: penultimate layer of a neural net used to solve task.

Intuition: understanding relative positioning of pieces of an image requires some understanding of how images are composed.

Jigsaw puzzles

In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

Doersch et al. '16: Unsupervised Visual Representation Learning by Context Prediction

Quite finnicky: one needs to make sure the predictor cannot take any obvious "shortcuts".

Boundary texture continuity is a big clue: include gaps in tiles.

Long lines spanning tiles are a clue: jitter location of tiles.

Chromatic aberration (some cameras tend to focus different wavelengths at different position – e.g. green shifts towards center of image): randomly drop 2 of the 3 channels.

Predicting rotations

In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

Task: predict one of 4 possible rotations of an image.

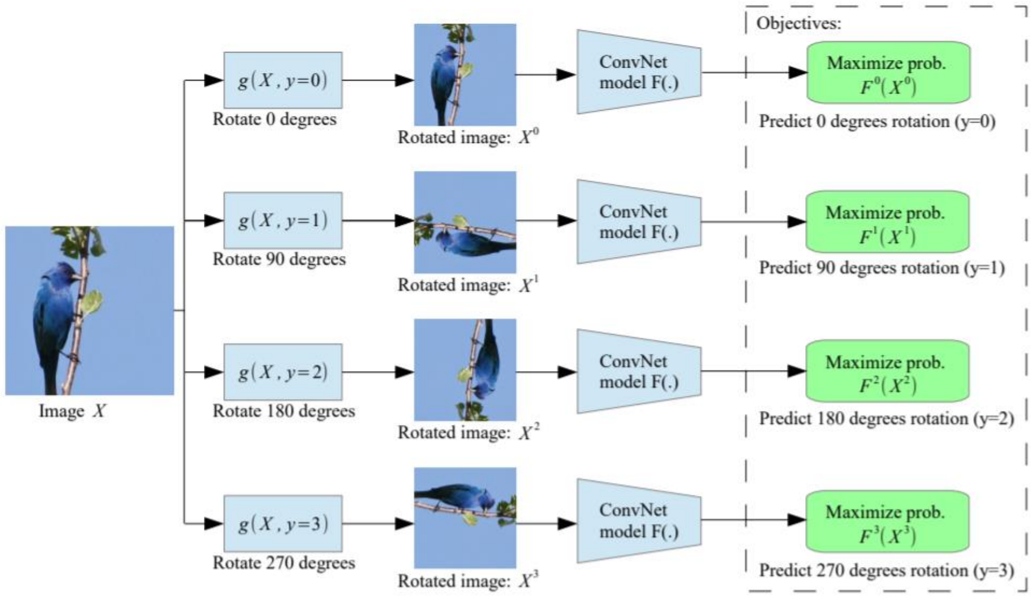


Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation y predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation y^*.

Predicting rotations

In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

Task: predict one of 4 possible rotations of an image.

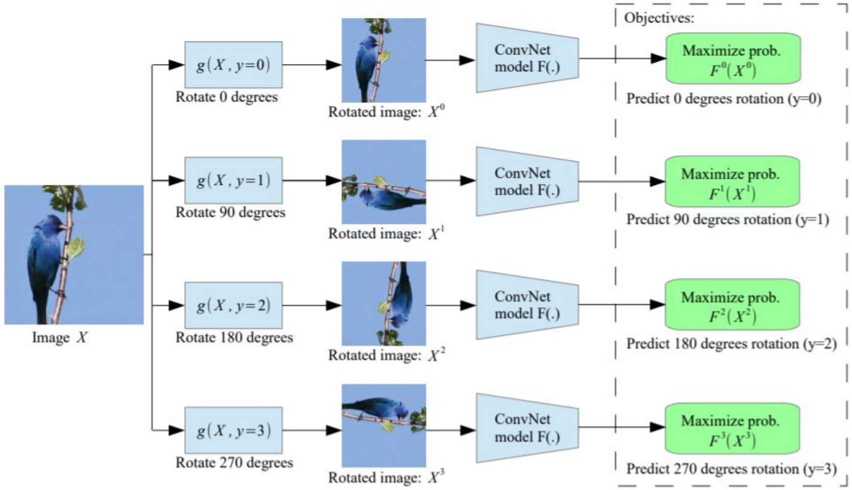


Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation y predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation y^*.

Representation: penultimate layer of a neural net used to solve task.

Intuition: a rotation is a global transformation. ConvNets are much better at capturing local transformations (as convolutions are local), so there is no obvious way to "cheat".

Predicting rotations

In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

Task: predict one of 4 possible rotations of an image.

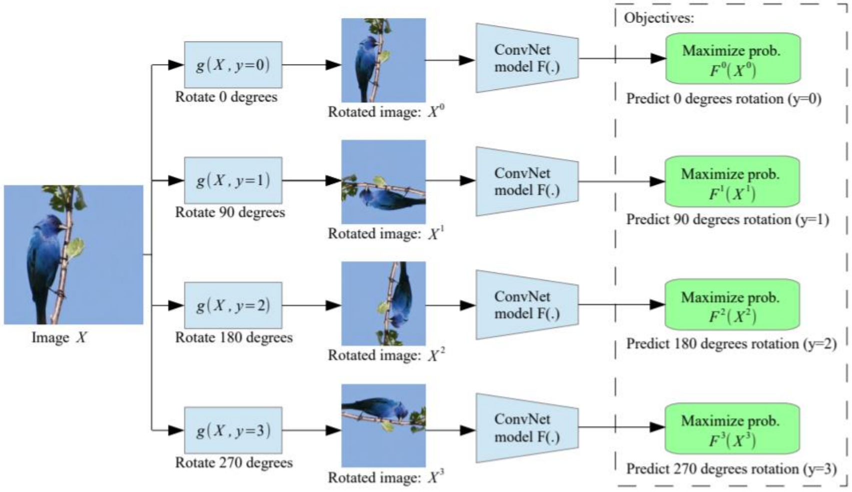


Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation y predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation y^*.

Less finicky to get right: no obvious artifacts the model can make use of to cheat.
The 90 deg. rotations also don't introduce any additional artifacts due to discretization.

Contrastive divergence

Another natural idea: if features are "semantically" relevant, a "distortion" of an image should produce similar features. Some instances of distortions:

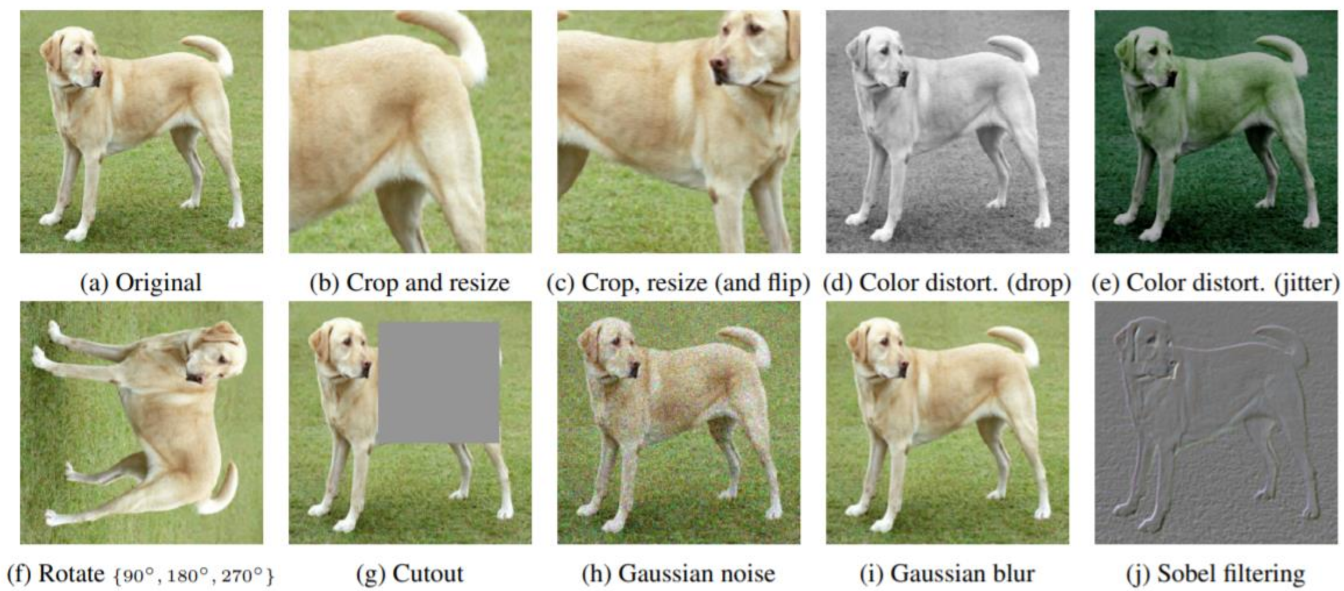


Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

Contrastive divergence

Another natural idea: if features are "semantically" relevant, a "distortion" of an image should produce similar features. Some instances of distortions:

Contrastive divergence framework:

For every training sample, produce multiple *augmented* samples by applying various transformations.

Train an encoder E (i.e. map that produces features) to predict whether two samples are augmentations of the same base sample.

A common way is to train E to make $\langle E(x), E(x') \rangle$ big if x, x' are two augmentations from same sample, small otherwise, e.g.

$$l_{x,x'} = -\log\left(\frac{\exp(\tau\langle E(x), E(x')\rangle}{\sum_{x,x'} \exp(\tau\langle E(x), E(x')\rangle)}\right)$$

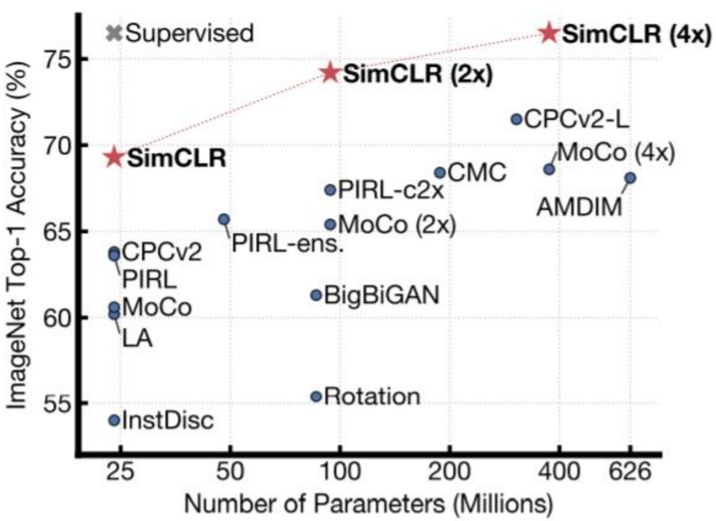
$$\min \sum_{x,x' \text{ augments of each other}} l_{x,x'}$$

Contrastive divergence

Another natural idea: if features are “semantically” relevant, a “distortion” of an image should produce similar features. Some instances of distortions:

Many works follow this framework, starting with Oord ‘18: Representation Learning with Contrastive Predictive Learning.

Current state of the art for self-supervised learning is in fact using this framework: *Chen, Kornblith, Norouizi, Hinton ‘20: A Simple Framework for Contrastive Learning of Visual Representations*



Several tricks needed to gain this improvement.

Most important one seems to be that augmentations that work best are compositions of a geometric one (e.g. crop/rotation/..) and an appearance one (color distortion/blur/..)

Troubling fact: architecture of classifier matters

Kolesnikov et al. '19: Revisiting Self-Supervised Visual Representation Learning

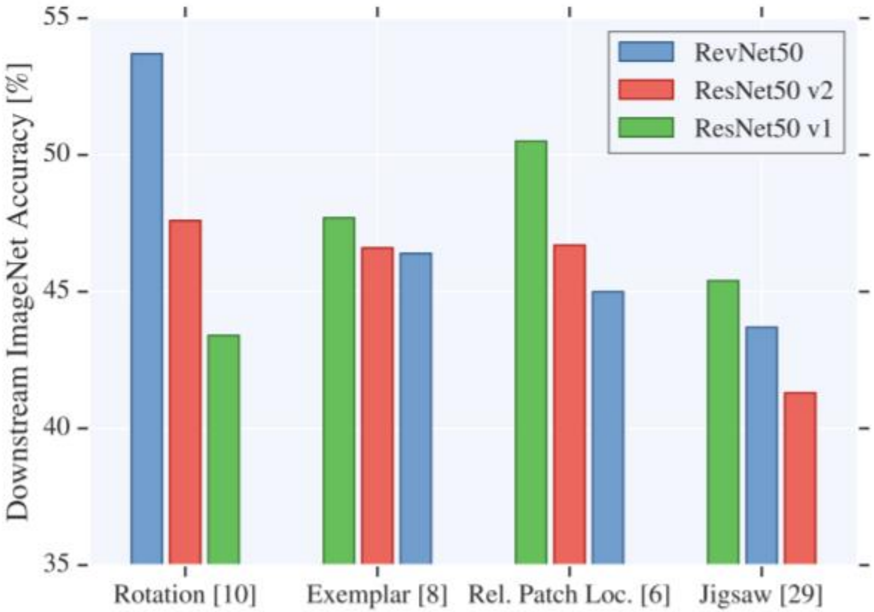


Figure 1. Quality of visual representations learned by various self-supervised learning techniques significantly depends on the convolutional neural network architecture that was used for solving the self-supervised learning task. In our paper we provide a large scale in-depth study in support of this observation and discuss its implications for evaluation of self-supervised models.

Troubling fact: architecture of classifier matters

Kolesnikov et al. '19: Revisiting Self-Supervised Visual Representation Learning

Table 1. Evaluation of representations from self-supervised techniques based on various CNN architectures. The scores are accuracies (in %) of a linear logistic regression model trained on top of these representations using *ImageNet* training split. Our validation split is used for computing accuracies. The architectures marked by a "(-)" are slight variations described in Section 3.1. Sub-columns such as 4× correspond to widening factors. Top-performing architectures in a column are bold; the best pretext task for each model is underlined.

Model	Rotation				Exemplar			RelPatchLoc		Jigsaw	
	4×	8×	12×	16×	4×	8×	12×	4×	8×	4×	8×
RevNet50	**47.3**	**50.4**	**53.1**	**<u>53.7</u>**	**42.4**	45.6	46.4	40.6	45.0	40.1	43.7
ResNet50 v2	43.8	47.5	47.2	<u>47.6</u>	**43.0**	45.7	46.6	42.2	46.7	38.4	41.3
ResNet50 v1	41.7	43.4	43.3	43.2	**42.8**	**46.9**	**47.7**	**46.8**	**<u>50.5</u>**	**42.2**	**45.4**
RevNet50 (-)	45.2	**51.0**	**52.8**	**<u>53.7</u>**	38.0	42.6	44.3	33.8	43.5	36.1	41.5
ResNet50 v2 (-)	38.6	44.5	47.3	<u>48.2</u>	33.7	36.7	38.2	38.6	43.4	32.5	34.4
VGG19-BN	16.8	14.6	16.6	22.7	26.4	28.3	<u>29.0</u>	28.5	<u>29.4</u>	19.8	21.1