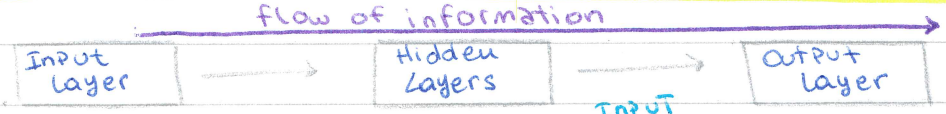
DEEP LEARNING (Machine Learning module II)

1. FEED FORWARD NEURAL NETWORK (FFNN)



DAG

FFNN: $f(x) = f^{(i)}(f^{(i-1)}(\dots f^{(2)}(f^{(1)}(x))\dots))$ → composition of functions

(output; input)

sequence of features, each of which represents a layer of the network.

Depth of the network is defined by the number of layers.

Input Layer: each node represents an input feature

Hidden Layers: perform non-linear computations on the input data (appl. of weight and activation function).

Output Layer: number of layer depends on the nature of the problem (e.g. binary classification has 2 output layers).

Feedforward NN overcome limitations of linear models:

(1) Do non linear transformation $x \rightarrow \varphi(x)$

(2) Apply LINEAR MODEL to $\varphi(x)$

(3) φ gives features or a representation for x

How we choose φ?

(1) GENERIC φ: for example infinite dimensional kernel used in RBF kernel.
Pros: enough capacity to fit training data
cons: poor generalization for highly variable objective functions.

(2) ENGINEER φ: manually engineering a function that can capture the relevant information in the input data. May be limited by the complexity of data.

(3) LEARNING φ FROM DATA: combine strengths of first 2 options. φ can be modelled very generically, and engineering can focus on designing neural networks' architectures capable of learning complex and informative representation from input data.

Thanks to FFNN we can overcome limitations of traditional linear models because they're able to learn complex, nonlinear functions from data, thus overcoming the inherent limitations of linear models in capturing complex relationships between input variables.

The main difference from linear models lies in the LOSS FUNCTION.
In NN Loss is non-convex leading to greater complexity and not guaranteeing convergence to global optimum.

Training NNs

- Optimize the parameters' values W (weights) in order to achieve $f(x,w)$ as close a the target function $f(x)$.
- Apply gradient descent → iterative approach to minimizing cost function
- Specify an appropriate cost function that accurately represents error between pred. output and des. output.
- Output representation, activation functions and network architecture significantly affects network peformances.

MODELING CHOICES

1. Cost function (Loss)

Measure the discepancy between model predictions and actual target value.

CLASSIFICATION → categorical cross-entropy
↘ Binary cross-entropy
} work with class probability. For models that produce prob. distr.

REGRESSION → MSE: strongly penalizes significant errors. Sensitive to outliers
→ MAE: less sensitive to outliers
↘ Huber loss: combines streughts of MAE and MSE. Avoids overfitting

CHOICE OF LOSS DEPENDS ON: goal of model training, nature of data, needs of the application.

2. Unit Types

Linear Units apply linear transformations to the inputs by combining the weights associated with the inputs values themselves. Output is a linear combination of input. $\hat{y} = W^T h + b$ (features from previous layer; bias)

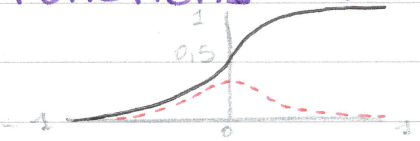
Softmax Units mostly used in Multiclass Classification. Transform the unnormalized scores from the last linear layer into a normalized probability distribution, allowing model to assign a probability to each membership class for a given input.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

(z_i: non-normalized score associated with class i; $\sum_j e^{z_j}$: sum of all exponents of un-normalized scores)

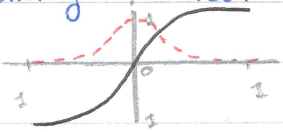
3. Activation Functions $h(z)$

Sigmoid $\frac{1}{1+e^{-z}}$



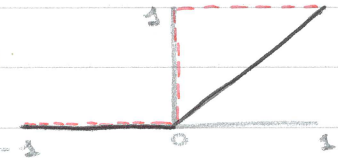
PROS	CONS
• squeezes output in range $[0,1]$ → useful with probability output	• Saturate across most of the domain ↳ causes gradient to tend toward zero.
• Differentiable and smooth. Ensure stable convergence during training	• Strongly sensitive only when input is near to zero

Hyperbolic tangent $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



PROS	CONS
• Differentiable and continuous.	• Saturate across most of the domain ↳ slower convergence
• Squeeze output in range $[-1,1]$. ↳ improve model's ability to learn and adapt to data distributions	

ReLU $\max(0,z)$



PROS	CONS
• Wide and consistent gradients when its active → NOT saturate	• Output not centered in zero
• Fast learning	• Dying ReLU: most of the inputs are in the negative range → gradient don't flow durig backprop → weights doesn't update
• Set zero when input its negative ↳ Fewer neurons activated: NETWORK SPARSITY	• Sensitive to large learning rate

Resolve with adjusting learning rate

Generalized ReLU obtain non-zero slope when $z_i < 0$. $h(z, a_i) = \max(0, z_i) + a_i \min(0, z_i)$

Exponential Linear Units (ELU): all adv. of ReLU. Prevents neuron dying. More Complexity.

Swish: Perform better on deeper models, improving performance on datasets like ImageNET

What matter most is the shape of DERIVATIVES of act. functions.

4. Architecture Design (Depth and Width)

Depends on several factors: complexity of the problem, availability of training data, computational resources.

CYBENKO'S THEOREM a 2-layer NN with linear output can approximate any continuous function over a compact domain with arbitrary precision, provided there are enough hidden units.

↳ NO indications of how large the network will be.

There are trade-offs between depth and width of neural networks.

Architecture design has a significant impact on performance.

2. BACKPROPAGATION

Gradient Descent Algorithm

To train NNs we employ (stochastic) Gradient Descent in combination with Backpropagation.

Backpropagation is a method for computing gradients. → update parameter.

Understanding the gradient descent: algorithm used to minimize the loss function adjusting parameters iteratively.

GOAL: find the optimal set of parameters that result in the lowest possible loss.

Each partial derivative measures how fast the loss changes in one direction.

Direction is determined by the negative gradient of the loss function.

Update rule for the parameters w in gradient descent: $\Delta_w L = \left[\frac{\partial L}{\partial w_1}, \dots, \frac{\partial L}{\partial w_N}\right]$

$\Delta_w L$ represents the change in the loss function L with respect to w

Algorithm proceeds by iteratively updating parameters according to the gradient direction until convergence is reached or a predefined number of iterations is completed.

The problem of Local Minima

In NNs, optimization problem is non-convex → exists a local minima.

↳ can be a good solution

The problem of Saddle Points

Some directions curve upwards and others downwards. At a saddle point, gradient is zero, even if the point is not a minimum.

↳ Become a problem only if the opt. problem becomes stuck exactly at saddle point.

Backpropagation Algorithm

Ideas to learn weight of a NN:

Randomly perturb one weight at time: evaluate if the change improve performance. Is very inefficient.

Perturbing all the weights simultaneously: correlating performance improvement with changes. It's inefficient.

Perturb only activations: are fewer in number respect to weights. It's still inefficient.

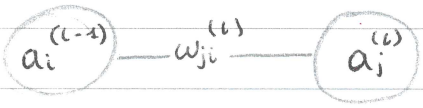
Are all inefficients, so to achieve a more efficient Learning we use Backpropagation

(1) FORWARD PROPAGATION: summation of inputs, production of activations and propagation of output through the network.

(2) ERROR ESTIMATION: comparison of labels with predictions obtained from the network.

(3) BACKPROPAGATION: of the error signal and using this signal to update the weights by computing gradients.

Exploring BP step by step



i,j indicating position of neurons within the layer

$(l-1)$, (l) indicates which layer it is in

w_{ji} indicates edge weight connecting neuron in layer i to the neuron in layer j.

A crucial aspect is the **sensivity of the loss function wrt activations of neurons of previous layers**. → How a small change in a neuron activations can change error of the network.

Preceding neurons influence the cost function through **multiple pathways**.

↳ Change of cost caused by a variation of activation in one specific neuron, must take care of the effect of that neuron on all neurons of successive layers.

↳ To do this we sum the influences along each of these pathways.

Sum multiple expressions of different CHAIN RULES corresponding to each pathway of influence.

$$\frac{\partial L}{\partial a_i^{(l-1)}} = \sum_j \frac{\partial z_j^{(l)}}{\partial a_i^{(l-1)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial L}{\partial a_j^{(l)}}$$

Quanto il costo L cambia al variare dell'attivazione del neurone i nel layer prec.

Somma per tutti i neuroni

Quanto cambia l'input del neurone j al variare dell'attivazione del neurone i

Quanto cambia l'attivazione di j al variare del suo input

→ sum over layer l

sensibilità del costo rispetto all'attivazione del neurone j

3. OPTIMIZATION

Types of Gradient Descent

Batch Gradient Descent (BGD)

Use the entire dataset (train) to calculate the gradient of the loss function.

More computational resources, More stable and accurate convergence

- Compute gradient estimate on N training samples $\{(x^{(1)}, y^{(1)}), \dots (x^{(N)}, y^{(N)})\}$
- $\hat{g} \leftarrow \frac{1}{N} \nabla_w \sum_N \mathcal{L}(f(x^{(i)}, w), y^{(i)})$

Stochastic Gradient Descent (SGD)

Update for each individual training sample. Less comp. resources, Less stable.

- Compute gradient estimate on a random training sample $(x^{(i)}, y^{(i)})$
- $\hat{g} \leftarrow \nabla_w \mathcal{L}(f(x^{(i)}, w), y^{(i)})$ ↳ CAN BE NOISY → use MiniBatches

Stochastic Gradient with Momentum

PROBLEM: when surface is almost flat, SGD moves slowly; in regions where gradient changes rapidly SGD can fluctuate a lot.

SOLUTION: SGD with Momentum → introduce **velocity (v)**, represents an exponentially decaying moving average of negative gradients. It acts as a kind of "inertia" in the movement of the model parameters.

Direction and speed by which the parameters move as the learning progresses

- Compute Gradient estimate on a random training example $(x^{(i)}, y^{(i)})$
- $\hat{g} \leftarrow \nabla_w \mathcal{L}(f(x^{(i)}, w), y^{(i)})$
- Update velocity: $v \leftarrow \alpha v - \eta \hat{g}$ $\quad \alpha \in [0, 1[$ $\quad \eta$ = learning rate

  New hyperparameter usually set to 0.5 / 0.9
- Update parameters: $w \leftarrow w + v$

(α Parameter controls the contribution of the previous velocity in the current parameter update.)

Stochastic Gradient with Nesterov Momentum

Adds a correction to the previous momentum **before** calculating the gradient. → The gradient term is computed from an intermediate position. If the momentum term points in the wrong direction or overshoots, the gradient can still "go back" and correct it in the same update step.

- Update parameters **temporarily** $\tilde{w} \leftarrow w + \alpha v$
- $\hat{g} \leftarrow \nabla_{\tilde{w}} \mathcal{L}(f(x^{(i)}, \tilde{w}), y^{(i)})$
- Update velocity $v \leftarrow \alpha v - \eta \hat{g}$
- Update parameters $w \leftarrow w + v$

PROBLEM: so far we have assigned the same learning rate to all features
↳ all features are equally important? NOT ALWAYS

SOLUTION: Adaptive Learning Rate Methods

Adagrad (Adaptive Gradient Algorithm)

Scales the model parameters by the square root of the sum of the squares of all historical gradient values. → Automatically adapt the learning rate for each parameter according to its update rate

- $\hat{g} \leftarrow \nabla_w L\left(f(x^{(i)}, w), y^{(i)}\right)$
- Accumulate (sum of squared gradients) $r \leftarrow r + \hat{g} \odot \hat{g}$
- Update parameters $w \leftarrow w - \frac{\eta}{\delta + \sqrt{r}} \odot \hat{g}$

(δ: small constant to avoid dividing by zero; $\odot$: element-wise multiplication; Δw)

Parameters with large part. derivatives → ↓↓ η
Parameters with small gradients → ↑↑ η
} more stable convergence

RMSProp (Root Mean Square Propagation)

PROBLEM Adagrad can excessively decrease η for parameters with large variation in gradients during convergence → SLOW CONVERGENCE

SOLUTION: RMSProp maintains an exponentially decreasing average of past gradients, also called **RUNNING AVERAGE**.

- $\hat{g} \leftarrow \nabla_w L\left(f(x^{(i)}, w), y^{(i)}\right)$
- Accumulate $r \leftarrow \rho r + (1-\rho)\, \hat{g} \odot \hat{g}$ — ρ: Decay Rate, r: Running Average
- Update parameters $w \leftarrow w - \Delta w$

Avoid excessive decrease in η, as past gradient r count less.

ADAM (ADAptive Moments)

Combines concepts from RMSProp and Momentum. Introduce **BIAS CORRECTION TERMS** for first and second moments. → crucial because these terms are initialized to zero and require to adapt to the data.

ADAM automatically adapts η for each parameter based on momentum and past momentum estimation, helping to improving the effectiveness of optimization during learning process

Normalization

Problem COVARIATE SHIFT

Situation where most of the input: data falls into non-linear regions of the neural network's activation function. This can significantly slow down the learning process as the network may require multiple iterations to adapt its weights to the new data distribution.

Solution BATCH NORMALIZATION

Reconfigure parameters of a NN, can be applied to input or hidden layers.

IDEA: standardization of the inputs of a layer.

$$H' = \frac{H - \mu}{\sigma}$$

H: outputs of layer; μ: mean; σ: standard deviation; H': output "transformed"

Standardization can reduce representation power of features, squeezing them into a narrow range.

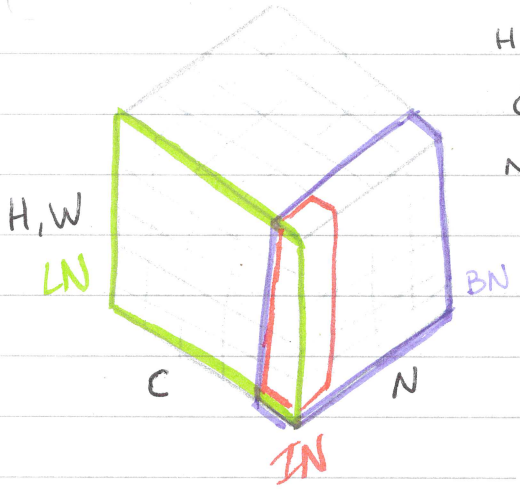
↳ To mitigate this problem we introduce **SCALING FACTOR (γ)** and **BIAS (β)** → Allow network to learn a linear transformation of the normalized output, thus allowing greater flexibility in data representation

$$\text{Output} = \gamma H' + \beta$$

TRAIN: back prop is performed through normalized activations and γ and β are learned

TEST: running averages of μ and σ collected during train and used to evaluate

Batch, Instance and Layer Normalization



H, W height, width
C channels
N batch size
(N, C, H, W)

LN: directly estimates normalization stat. from the summed inputs to the neurons within a hidden layer

IN: Allow to remove instance-specific contrast information from the content image. Simplifies generation

4. REGULARIZATION

Our goal is to avoid overfitting*. The challenge is that the algorithm must perform well on new, previously unseen inputs. The ability to perform well on that inputs is called GENERALIZATION.

*making gap between train and test small.

To control overfitting we can reduce model CAPACITY (ability to fit a wide variety of functions). We control capacity choosing the hypotesis space, the set of functions that the learning algorithm is allowed to select as being the solution.

Another solution can be average many different models.

Without Regularization we can reduce generalization error but not training error.

Regularization Strategies

Parameter Norm Penalties

Add a penalty to the loss function during training affects the size of weights.

$$\tilde{L}(w) = L(w) + \lambda \Omega(w)$$

→ Hyperparameter that control importance of regularization.

L2: add a term proportional to the square norm of model weights. $\Omega(W) = \frac{1}{2} \sum_i \|W_i\|^2$

Leads weights to tend to smaller values.

L1: add a term proportional to the absolute norm of model weights. $\Omega(W) = \sum_i |w_i|$

Favors sparsity of weights. Many weights tend to become zero. It makes the features most relevant more obvious.

Overlap between gradient error and the gradient of L1 or L2 is the point where there is the right balance between error reduction and model complexity.

Data Augmentation

Strategy that aims to improve model generalization by increasing the amount and variety of training data. Particularly effective in classification task: classifier must be invariant to a wide range of transformations of the input data. The goal is to create a diversity of samples that cover as much of the input data space as possible, thus helping the model learn a more robust and general representation of discriminative features.

Injecting Noise

to INPUT: make learned function smoother

to WEIGHTS: encourage network to be more robust to random variations in train. data

TO LABEL: discourage overconfident model behaviour

Early Stopping

Prevent model from becoming too specialized on training data.
Stop train when performance on the validation set starts to deteriorate, even if the performance on the training set continues to improve.
Training steps is another hyperparameter to be optimized.

Multi-Task Learning

Model is trained to do more than one task at same time.
Several tasks share the same set of input data and some intermediate layers of NN, but may have separate final layers for each task.
Parameters can be task-specific or generic.

Parameters Sharing

Enforce that certain parameters within the network are exactly the same. This reduce model complexity and share relevant information between different parts of the network.

Model Ensembles

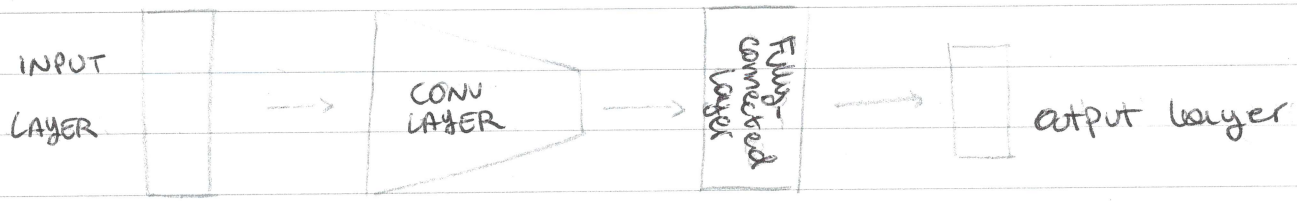
Train several models separately and then have all the models vote for the output. (e.g. Bagging) This is useful because different models produce differents error on test set.

DropOut

Randomly turn off some neurons during forward pass. Prevents hidden neurons from fitting too closely to each other and forces them to focus on extracting more useful features.

5. CONVOLUTIONAL NEURAL NETWORK

CNN handles with structured data.



CNN are a specialised version of feedforward NNs. Are designed with some specific features that include local connectivity and weight sharing, which allow to capture spatial characteristics of images without having to manage an excessive number of parameters.

CONVOLUTIONS

CNN replace general matrix multiplication with convolution in at least one of their layers.

Convolution: take a matrix as input, produce a scalar ~~product~~ by calculating scalar product between pixel values and the kernel parameters. Kernel moves and the output is also a matrix

Convolutional Layer set of filters, each covering a small spatial portion of the input, known as the receptive fields. Each filter is convolved along the dimensions of the input data, producing a multidimensional feature map. FILTERS ARE LEARNED DURING TRAIN, that are activated in response to specific features at specific locations.

$$S(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, i-n)$$

I: Input, K: Kernel

There are 2 important parameters to control size of output and influence the size and shape of features extracted by the CNN:

- **PADDING:** addition of values (e.g. zero) in order to have input_size = out_size
- **STRIDE:** number of pixels in which the ~~filter~~ kernel is moved along the input image during convolution

$$\text{OUT\_SIZE} = \frac{(N-K)+2P}{S} + 1$$

N: Input, K: kernel, P: padding, S: stride

INSIDE A CNN

- **INPUT:** raw Image
- **CONV-LAYER:** fundamentals to extract features. Filters are applied to the image to detect SPECIFIC PATTERNS. Size of filter is learned.
- **NON LINEARITY:** non-linear activation function (e.g. ReLU) is applied. Allow the network to learn and represent complex relationships between images features.
- **SPATIAL POOLING:** Reduce the size of feature map, reducing number of params.
  ↳ control overfitting
- **NORMALIZATION LAYER:** (OPTIONAL) ensure data remain in a certain range to facilitate learning process.
- **FEATERE MAPS:** information extracted from input image. Each feature map corresponds to a specific feature detected by the network and can be interpreted as a semplified representation of the image, in terms of patterns and structures.

CNN ARCHITECTURES FOR CLASSIFICATION

(Vedi elenco e descrizioni da pag 43 a 50)

REGULARIZATION TECHNIQUES FOR CNN

CNNs are susceptible to overfitting. To mitigate this, we can adopt regularization techniques:

- **CUTOUT:** cut a random rectangular area of image, filling the region with the average pixel value of the entire image.
- **DROPBLOCK:** entire regions of neurons within a conv layer are deactivated. This prevents that the network focus (too much) on specific input features.
- **CUTMIX:** a random rectangular area of an image is replaced with a corresponding area from another image.

Object Detection

This is another task that can be achieved with CNNs. It consists in classify the object in the image AND identify their exact location through bounding boxes.

↳ Do simultaneously classification and regression task.

IMAGE ⟶ CNN ⟶ Bounding Boxes with labels for each object

(1) Selective search for Object Recognition (2013)

Generate proposed Region of Interest (ROI). } Preliminary step before classification
Assign a likelihood score for each ROI of containing an object }
Group similar pixels together to form regions.

(2) R-CNN Region-Based CNN (2014)

Is the first CNN-based approaches for object ~~rego~~ recognition.
Use proposed regions to identify objects.
Use CNN to extract features and classify objects within the regions.
Use Selective Search to explore the entire space of bounding boxes.
Train and Test are very slow.

(3) Fast R-CNN (2015)

- RoI Pooling: introduce a ROI pooling layer that allows the best use of the information extracted from the convolutional network for each proposed region. → Reduce number of regions as input
- Convolution of the forward pass of a CNN : convolution is performed only once for the entire image.

(4) Faster R-CNN (2015)

- Use of a separate network for proposal generation. Instead of using Selective Search, a separate network RPN (Region Proposal Network) is used.
- End-to-end architecture : allow the entire end-to-end model to be trained in a single step, improve efficiency and ease of training.

(5) YOLO You only Look Once (2016)

- Treat object detection problem as a direct regression problem, in which the entire image is divided into a grid and for each grid cell the bounding box and class probability of the objects whitin that cell are directly predicted.

- Use of a single conv. network
- split image into gid → for each grid cell predicts a set of bounding boxes.
  ↳ consider several regions of the image simultaneously.
- Bounding box selection: for each bounding box, the YOLO grid Produces a class probability and offset values for the bounding box.

Instance Segmentation

Assign a label to each pixel of an image, distinguishing individual objects within the scene. It requires a detailed understanding of the structure and shape of the objects themeselves.

Mask R-CNN (2017)

Extension of Faster R-CNN that adds an additional branch to the neural network for instance segmentation.

- use ROI Align, ensure more precise alignment of feature maps during the instance segmentation phase.
  - Pixel level segmentation: a label is assigned to each pixel
- FCN added: adds a fully convolutional network on top of selected features to produce accurate and detailed segmentation mask.

CLIP

CLIP Contrastive Language-Image Pretraining

Combines 2 main neural networks:

Image Encoder: uses a Vision Transformer based network or deep convolut. network to convert images into embedding vectors

Text Encoder: employs a Transformer-type neural network, similar to that used in language models such as GPT, to convert text into embedding vectors.

The main goal of CLIP is to create a shared embedding space in which images and text descriptions are semantically aligned.
During training, the model uses a **contrastive learning** technique to optimize the similarity between image embeddings and corresponding text descriptions, while minimizing the similarity between images and unrelated text. This approach allows CLIP to make inferences and classifications based on text descriptions, without the need to train task-specific models.

CLIP was trained on a large and diverse dataset containing approximately 400 million image-text pairs taken from the Internet. Although this dataset is sizable, even larger dataset, such as LAION, containing billions of images, are now also being used.

CLIP aims to maximize the similarity between pairs of images and texts, corresponding to the **diagonal of similarity matrix**. At the same time, it tries to minimize the similarity (increase dissimilarity) between all other pairs, that is, between unrelated images and texts.

CLIP's capabilities extend to numerous applications, called **downstream tasks**, such as image classification and search: it can be used to search for images based on text description and vice versa. A notable aspect of CLIP is its **zero-shot learning** capability, which allows it to generalize to

new categories without the need for additional training specific to those categories.

LIMITATIONS:

(1) LIMITED CAPABILITY: CLIP is limited to connecting image and text embedding space

(2) USE RESTRICTED TO SPECIFIC TASK: although it exels at classification and search, is less suitable for more complex tasks such as text generation or visual question answering.

(3) LANGUAGE GENERATION CAPABILITY ABSENCE: Lacks the ability to generate language.

HOW TO IMPROVE THE MODEL?

(1) Data Scalability
(2) Model Design (for both image and text encoder)
(3) Objective Functions (contrastive learning)

vedi Riassunti

APPUNTI SLIDE

The contrastive loss works by pulling together embeddings of correctly paired images and texts, while pushing apart incorrect pairings. Key design choices include training both encoders from scratch, and using symmetric cross-entropy loss with a learnable temperature parameter that scales logits for better training dynamics.

In terms of evaluation, CLIP's performance was benchmarked on over 30 datasets, revealing impressive zero-shot accuracy. Model performance is sensitive to prompt (prompt engineering). Custom prompt templates tailored to specific domains further boost performance.
This led to prompt ensembling, where multiple prompts are averaged to stabilize predictions, resulting in even better performance.
↳ LIMITATION: prediction depends on prompt wording

Overcome prompt-dependency limitation

CoOp (conTexT Optimization) replace fixed text prompts to learnable vectors. Adapts promts to tasks by learning them from data.

↳ PROBLEM: overfits to base classes and fails in novels.

CoCoOp (conditional CoOp) makes prompts dynamic - conditioned on each image through a similar auxiliary network (Meta-Net). This lets the model adapt to input-specific cues, enhancing generalization and robustness to domain shifts, though at the cost of increased training time.

6. RECURRENT NEURAL NETWORK

HANDLING VARIABLES-LENGTH SEQUENCES

RNN are flexible tools that can perform many different tasks:

1) ONE to MANY
2) MANY to ONE
3) MANY to MANY

Many to One: Sentiment Analysis

Model have to classify a product as positive, neutral, negative, based on reviews. Input are in various sizes. Each word is transformed into a hidden representation that reflects the context and propagates through subsequent words, as these words are not independent from each other.

Predictions are tipically made about the last hidden state, which should include the context of previous words.

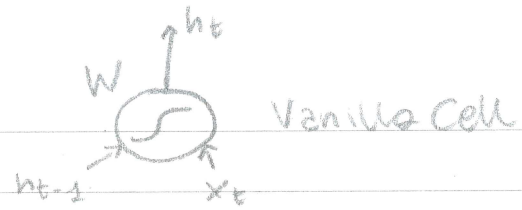
One to Many: Image Captioning

Fixed input (image -> matrix) but variable length outputs (sentence that describe the image). The latent representation of the image obtained from a CNN is combined with the latent representations of the words to model the dependencies between them.

Many to Many: Machine Traslation

The task is to translate a sentence from a language to another.

Each word in the input sentence is mapped to a hidden representation that takes into account the context of the preceding words. The neural network uses these representations to generate the translated sentence, word by word, maintaining sequential dependencies between words. This allow the model to produce a sentence with the same meaning of the input.



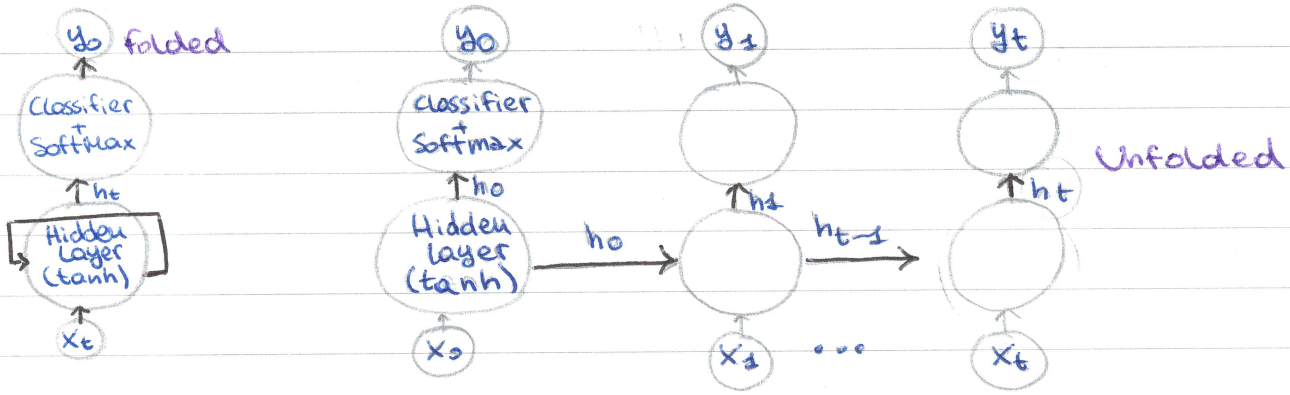
VANILLA RNN

RNN are, in essence, Neural Network with cycles.

Vanilla RNN has a recurrent design that performs the same parametric function for each input (x_t, each marked with a timestep) while the output (h_t the hidden representation marked with the corresponding timestep) depends on the previous computation (h_{t-1}). After producing the output, this is used to generate the hidden representation of the next RNN cell (h_{t+1}).

2 types of repr. → FOLDED synthetic

↘ UNFOLDED provides a more explicit description of the computations



STRUCTURE of a RNN CELL: $h_t = f_W(x_t, h_{t-1}) = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} = \tanh(W_x x_t + W_h h_{t-1})$

The function tanh provides nonlinearity to the network.

During training, W weights are learned through the optimization process:

$$y_t = \text{SOFTMAX}(W_y h_t + b_y)$$

In RNN weights are shared among different timesteps: the weights W_x, W_h, W_y are shared between different timesteps iterations, allowing the model to capture sequential dependencies within the data.

The PROBLEM of vanishing gradient is more severe in RNN than in feedforward NN. Also backpropagation through time can be computationally expensive for a large number of timesteps.

SOLUTION: TRUNCATED BPTT (Backpropagation Through Time)

IDEA: instead of running the entire sequence through the network, we divide it into smaller blocks of fixed length. Each block is treated as a separate training unit. → TRAIN IS MORE EFFICIENT.

After presenting a block to the network, backprop. is performed only for a limited number of backward time steps. This means that the network does not have to keep track of information too far back in time.

To control how far back in time to go during backprop. we use 2 parameters K_1 and K_2. K_1 determines lenght of the blocks. K_2 defines number of time steps over which to perform backprop. within each block.

Algorithm:

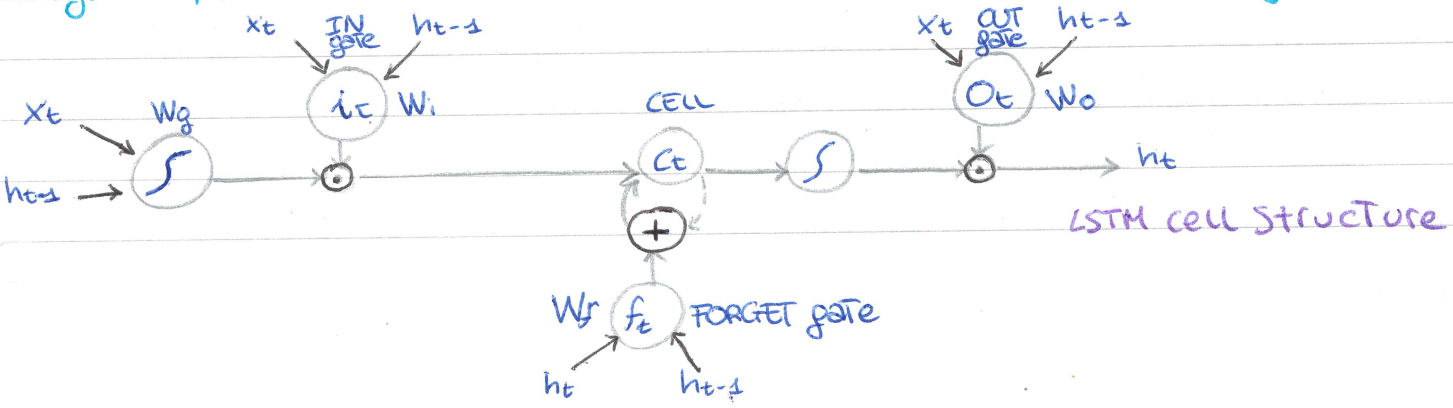
1. Sequence of K_1 timesteps of IN e OUT pairs
2. Unroll the network then calculate and accumulate errors across K_2 Timesteps
3. Roll-up the network and update weights.
4. Repeat

Adjusting K_1 and K_2 allow to balance training speed and network's ability to capture long-range time dependencies

LONG-SHORT TERM MEMORY

Classical RNN suffers ~~from~~ from an inefficient design in its individual cell, as efforts to improve learning have not produced significant improvements in the vanishing gradient problem. To solve it, LSTM was introduced.

LSTM cell produce hidden state h_t and also adds another component called Memory Cell (C_t), which is responsible for mantaining and deleting information, based on the input context. This means that some of the previous informations must be remembered, some must be forgotten, and some new information must be added to memory.



There are 3 different parametric components, called GATE

INPUT gate: allow input to alterate the state of the memory ~~bt~~ cell or block it. Its output is computed by the sigmoid function: $i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$ i_t is used as a trade-off between the information passing through the current input and the information encoded by the previous cell. (C_{t-1}).

FORGET gate: can modulate the self-recurrent connection of the memory cell, allowing the cell to remember or forget its previous state as needed. It's output f_t is computed by the sigmoid function applied to the input x_t and the previous dependency h_{t-1}, and its associated with its own parameters W_f: $f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$

MEMORY CELL CALCULATION: combination of previous 2 gates: $C_t = f_t \odot C_{t-1} + i_t \odot g_t$

↳ $f_t \odot C_{t-1}$: forget gate decides how much of the cell's previous state C_{t-1} should be forgotten or retrained for the next state C_t.

↳ $i_t \odot g_t$: how much of new info g_t should be added to the state of the cell C_t

OUTPUT gate: allow the state of the memory cell (C_t) to have an effect on other neurons, thus affecting the output of the LSTM unit (h_t). First, a sigmoid layer decides which parts of the cell state (C_t) the model is going to produce as output (associated with its parameters W_o): $o_t = \sigma\left(W_o \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_o\right)$

CALCULATE OUTPUT: a tanh layer is used on the state of the memory cell to shrink values between -1 and 1, which are then multiplied by the output of the ~~a~~t gate: $h_t = O_t \odot \tanh(C$

NOTE: non linear functions BUT calculations to update the value of the cell (C_t) are linear, so the gradient flow from C_t to C_{t-1} involves only BACKPROPAGATION THROUGH ADDITION AND ELEMENT-BY-ELEMENT MULTIPLICATION.

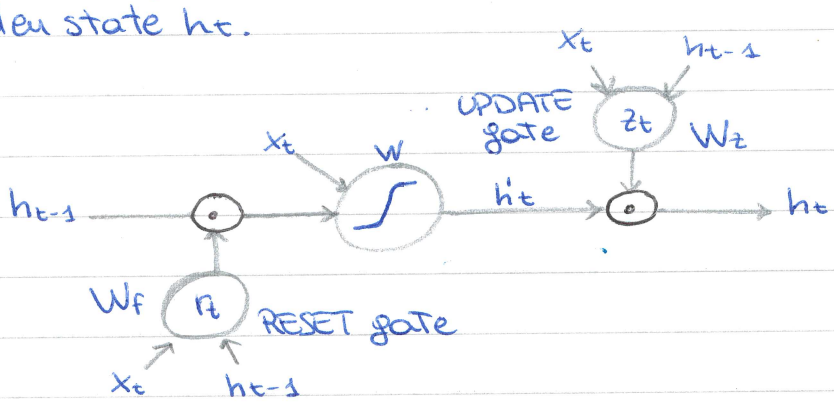
SUMMARIZING:

$$\begin{pmatrix} g_t \\ i_t \\ f_t \\ O_t \end{pmatrix} = \begin{pmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} \begin{pmatrix} W_g \\ W_i \\ W_f \\ W_o \end{pmatrix} \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$
$$h_t = O_t \odot \tanh(C_t)$$

GATED RECURRENT UNIT (GRU)

Introduced to adress the complexity of LSTM. It semplify LSTM by removing one of its gates. Despite it, the model is sufficiently resilient to the vanishing gradient problem. The information flow is stored only in the hidden state h_t.



GRU cell structure

RESET Gate: used to decide how much of the passed information to forget. The output r_t is computed by the sigmoid function applied to the input x_t and the previous hidden state h_{t-1}, and its associated with its own parameters W_r:

$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_t\right)$$

Difference with LSTM: the value of the input modulation is no longer based ~~on~~ solely on the complete information from the previous hidden state h_{t-1}, but instead on a "restricted stream" determined by r_t: $h'_t = \tanh W\left(\begin{pmatrix} x_t \\ r_t \odot h_{t-1} \end{pmatrix}\right)$

Update Gate: helps the model determine how much of the past information needs to be transmitted to the future. Output z_t is computed by the sigmoid function applied to the input x_t and the previous hidden state h_{t-1}, and its associated with its own parameters W_z:

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_z\right)$$

Final representation of hidden state is calculated by point product:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t$$

7. SEQUENCE MODELS

RNN had difficulty to handlying long sequences. Attention models was introduced to handle this problem.

CAPTIONING MODELS

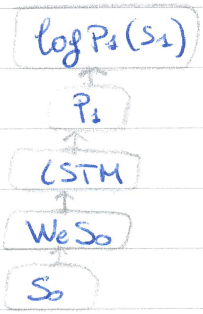
From a sequence of pixels to a sequence of words

Process of generate a textual description from an image.

For text we use RNN, for images we use CNN.

Show and Tell the procedure begins with feature extraction of the image using GoogleNet, which produces a feature vector: I.
I is inserted into LSTM sequence at time t=-1, only once.
Starting at t=0, a sequence of words vectors is inserted.
Each word vector is represented as a one-hot vector S_t with dims = size of vocabulary

At each timestep, word vector S_t is transformed through an embedding layer $We S_t$, and provided to LSTM. Input is processed along the previous hidden state to produce a probability of the next word. The word with highest probability, denoted as $\log P_1(S_1)$ is chosen as next word in the sequence.



The QUALITY of generated caption depends strongly on the performance of CNN.

For inference time, the model uses one of these methods:

Sampling 1st word is extracted according to a prob. P_1, then the corresponding embedding is provided as input and the next word is extracted with P_2.
Repeat until the special end-of-sentence token is extracted.

Beam Search iteratively consider the k best sentences up to time t as candidates to generate sentences of length t+1, keeping only the k best of them. This approach allows more world combinations to be explored, improving quality of generated sentences.

BLEU SCORE (Bilingual Evaluation Understudy)

Metric used to evaluate performance of texts generated in NLP models.

Text generated: CANDIDATE
Correct text: REFERENCES
} measure similarity

BLEU $\in [0, 1]$, is the proportion of matching n-grams with references found in the candidate compared to the total number of n-grams in the candidate.

Captioning models with Attention

Show, Attend and Tell

Captioning with attention represents a significant innovation, especially for image and text processing. The key IDEA is the integration of two key components: CNN for image feature extraction and a RNN with an attention mechanism to generate captions.

↳ MODEL FOCUS ATTENTION ON SPECIFIC PARTS OF THE IMAGE WHILE GENERATING THE CORRESPONDING DESCRIPTION

FEATURE EXTRACTOR: CNN used to extract features from image → mantain spatial info

RNN with Attention: extracted feature maps are passed to each cell of RNN during caption generation → network focus attention on specific parts of the image as it generates each caption word.

↳ Each RNN cell produce → PROBABILITY OF PREDICTED WORD $P(y_t)$
↳ ATTENTION MAP INDICATING RELEVANT PARTS OF IMAGE α_t

Each LSTM receive 3 MAIN INPUTS:

- Previous latent space h_{t-1}
- Predicted word in the current sequence (y_{t-1})
- Context vector z_t → dot product between feature map and previous attention mask

SOFT ATTENTION context vector z_t is computed as a weighted combination of image features, where the weights $\alpha_{i,t}$ represents the weighting difference coefficients $z_t = \sum_i \alpha_{i,t} d_i$.

↳ Neural Network dinamically focus on specific parts of image

HARD ATTENTION attention map gives score 1 to highest weight among all zero to the remaining.

Difference in performance with soft att. is minimal → H.A. often neglected, S.A. can be trained with opt. algorithms such SGD.

MACHINE TRASLATION

SEQ 2 SEQ

In 2016 Google improve Google Translate using Neural Machine Traslation (NMT) instead of Statistical Machine Traslation (SMT).
The distinguish factor was the implementation of end-to-end learning.
It means that the machine traslator handles the entire traslation process without the need for separate components or intermediate steps.
↳ Introduction of encoder-decoder approach to handle input of sequences of variable lengths.

ENCODER LSTM is responsible for mapping the variable length input phrase into a fixed-dimensional vector (CONTEXT VECTOR). The latter serves as a connection between the encoder and the decoder.

DECODER LSTM that maps the fixed vector into a variable length output sequence

RNN ENC-DEC

Was introduced in a paper where context vector is used for word decoding.
The model follow the principle of transforming variable-lenght IN sequence into a variable lenght out sequence via a fixed-lenght vector, CONTEXT VECTOR (=seq2se
Enc e Dec are trained jointly → learn latent representation that depends on context vector. SignificantImprovement → use GRU

PROBLEM: compress all info (expecially in long sentences) in a fixed-lenght context vector. → Quality of output depends on sequence lenght

RNN WITH ATTENTION

In this architecture, encoder is a bidirectional LSTM. Every word is represented by a vector that is the concatenation of the hidden states of the LSTMs that read the sequence both left-to-right and right-to-left.
Decoder generates output sequence one step at a time using latent representation of encoder and the dynamic context provided by attention mask.
Context change with each decoding step → Decoder pay attention to different parts of input sequence.

The attention mechanism works as follow:

1. Calculation of latent Representations: concatenating outputs of bidirectional Encoder ← LSTM units $h_j = [\overrightarrow{h_j} ; \overleftarrow{h_j}]$

2. Model Alignment when compute output at Time t, an alignment score e_{tj} is computed between previous state of decoder s_{t-1} and EACH latent representation h_j of input sequence $e_{tj} = a(s_{t-1}, h_j)$

3. Weights Attention: softmax function is applied to the alignment scores to obtain attention weights α_{tj}: $\alpha_{tj} = \text{softmax}(e_{tj})$

4. Vector Context: weighted sum of the latent representations, using attention weights

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$$

↳ combination of all relevant inputs for current output.

5. Output Generation: c_t with previous state of decoder is used to generate next state of decoder s_t and output word y_t

LOCAL ATTENTION

Model focuses only on a subset of the input sequence, rather than the entire sequence, to compute context vector c_t. This subset is determined by an aligned position indication p_t, which the model learns during training based on the previous decoder state s_{t-1}.

↳ Instead of using a global view of all input tokens, local attention utilizes a position-based approach → Model adapt dinamically to input.

1. Aligned Position $p_t = L_s \cdot \sigma(v_p^T \tanh(W_p s_{t-1}))$
   ↳ lenght input ↳ learnable parameters

2. Context Vector $c_t = \sum_j \alpha_{tj} h_j$

3. Decoder State Update $\tilde{s}_t = \tanh(W_c [c_t ; s_t])$
   ↳ current state of RNN cell

4. Output Probability $P(y_t | y_{<t}, x) = \text{softmax}(W_s \tilde{s}_t)$
   ↳ output words generated up to time $t-1$

GNMT (Google Neural Machine Traslation)

Encoder-Decoder system with attention mechanisms. Attention model leverages latent representations learned by the encoder LSTM and those associated with the target words in encoder. Decoder's output is processed through a softmax layer.

Encoder: several layers of LSTMs → start bidirectional lower layers LSTMs
↳ stack unidirectional LSTMs

Model divided in 8 parts → each on a different GPU (more efficient)

Decoder: only the output of lower layer is used to derive attention context.

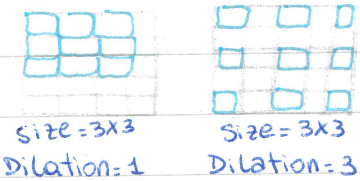
8. TRANSFORMERS

FROM RNN TO TRANSFORMERS

Since an image can be viewed as a two-dimensional input patched sequence, it becomes feasible to apply CNNs to sequence modeling tasks.
CNNs deals with sequence modeling with a bottom-up approach, extracting features hierarchically from local regions of the input.

↳ ADVANTAGES: - process of each element in the sequence occurs uniformly across the entire input. → homogeneity of model's operations.
- enhanced scalability, particularly when accelerated by GPU computing.
- Leveraging CNNs for sequence modeling, can benefit from their efficiency in capturing local patterns, and their ability to exploit PARALLEL COMPUTATION.

Dilation: spacing between elements of a filter as it moves across input sequence.



→ Allow the network to capture info. over larger spans of the input sequence

↳ Extends the receptive field of the convolutional filter, enabling it to incorporate more context from the input data while mantaining the computational efficiency of the original kernel size.

TEMPORAL CONVOLUTIONAL NETWORK

Are distinguished for 2 main features:

(1) sequence length preservation: can take a sequence of any lenght and map the input to an output sequence of the same length.

(2) causality in convolutions: convolutions are causal, ensuring that there is no loss off information from the future to the past.

Achieve (1) => 1D fully convolutional network (FCN), where each hidden layer has same length of input

Mantain (2) => employ dilated causal convolution → 1D convolution where the output at time t is computed using specific elements of the input positions of the previous level, determined by dilation rate.
↳ each output depends only on specific past inputs

REGULARIZATION => ReLU and dropout layers. Weight normalization block that accelerates convergence without introducing dependencies between examples of same batch.

"CONVOLUTIONAL SEQ2SEQ LEARNING"

In this work Google's seq2seq learning model is adopted and CNN-based approach is introduced. The task is to translate a sentence from German to English. Researchers replaced bidirectional LSTM in the encoder with a Temporal CNN, which produces latent representations used for attention computation. In addition, the decoder was also transformed to a CNN-based architecture.

↳ IMPROVES TRAINING EFFICIENCY

"ATTENTION IS ALL YOU NEED"

This work introduce **transformers**, that are based on a **self-attention** mechanism. The tranformer architecture is composed by:

ENCODER Processes entire input sequence and outputs an ecoded sequence of same length

DECODER generates output sequence one word at time. It is conditioned by encoded sequence and previous output. This ensure to take in account The context.

Self-Attention

Mechanism that captures context of other words relevant to the one we are currently processing. → Model focuses on the words important to understand meaning of the sentence.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Differentiable ↳ use backprop during train

- Query (Q): current word
- Key (K): potentially relevant words
- Value (V): content of relevant words

} obtained by multiplying word embeddings (x) by 3 weight matrices (W^Q, W^K, W^V) learned during training.

(QK^T): score → how similar 2 words are

d_k: size of the key

Multiply each value (V) by the softmax score, keeping the values of important words. Then sum-up these weighted values to get self-attention output for that position (z).

Multi-Head Attention

Improves model's ability to focus on different positions in the sentence and provide the attention level with more "subspaces of representation". Input vectors are divided into several separate "heads". Each head operates on linear projections (Q, K, V). After calculating attention for each head, results are concatenated and multiplied by an additional weight matrix, to produce final attention level output.

Positional Encoding

Before passing the embeddings as input, it's important to add information regarding the position of each word.

Combining embeddings and positional encoding we obtain "time-signal embeddings".

EMBEDDING VECTOR → meaning of word
↘ relative position in the input seq.

One technique can be the use of sinusoidal functions, which allow different positions in embedding vector to be represented.

ARCHITECTURE

Pag 82 notes x immagine

Encoder → Multi-Head Attention
↘ Feed-Forward Network → Processes output vectors of multi-head Attention

Decoder → Multi-Head Attention → Allow to focus on different parts of Encoder outputs
→ Masked Multi-Head Attention each position can only attend to previous pos.
↘ Feed-Forward Network

Encoder and Decoder { RESIDUAL CONNECTIONS, LAYER NORMALIZATIONS } provide grater stability during train and facilitate flow of gradients

Pre-training language models on large-scale datasets, followed by fine-tuning for specific tasks.

↳ Revolution for natural language processing.

LOOK AT THE SLIDES, I THINK IT'S MISSING SOME TOPICS :)

VISION TRANSFORMER (VIT)

In CNN a limitation is the concept of receptive field, which is related to the size of the convolutional kernel. This limits CNN to capture complex, long-range relationships between images.

ViT apply the concept of self-attention to images. This allow to learn relationships between pixels in fixed areas of image. In this case, self-attention is used to model self-similarity within images.

To passes the input image to the decoder, it is divided into patches. It is added a class token used to collect global informations about the image. At the end, the MLP head only looks at the data from the class token of the last layer and no other information.

Transformers don't generalize well when trained on insufficient amount of data.

SWIN TRANSFORMER

Can serve as a general skeleton for dense recognition tasks.
This model builds hierarchical feature maps by joining image patches in the deepest layers. By using dynamic partitioning, it improves the accuracy of localized representations.

9. GENERATIVE MODELS

So far we have explored Supervised Learning, now it's time to introduce Unsupervised Learning. Generative models are Probabilistic generative models.

Given a distribution of training data $P_{data}(x)$, the model learns distribution of generated samples $P_{model}(x)$ ⇒ GOAL $P_{model}(x)$ closest as possible to $P_{data}(x)$.

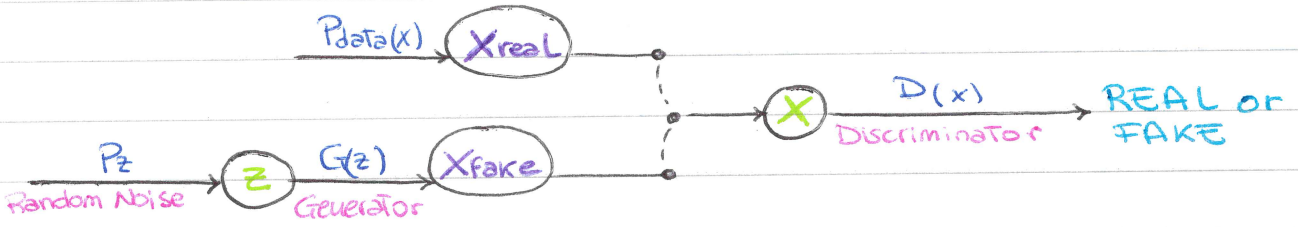
Explicit Density Estimation
explicitly define and solve for $P_{model}(x)$

Implicit Density Estimation
Learn model that can sample from $P_{model}(x)$ without explicitly defining it

GENERATIVE ADVERSARIAL NETWORKS

GANs exploit an adversarial approach with two neural networks: Generator and Discriminator

Generator learns to transform RANDOM NOISE into COMPLEX SAMPLES that resemble those from a TARGET DISTRIBUTION. This process is supervised by discriminator, which simultaneously learns to distinguish between REAL DATA SAMPLES and THOSE GENERATED by the Generator.



Generator try to produce samples indistinguishable from real ← ADVERSARIAL
Discriminator try to differentiate between real or fake ← Both improves iteratively

Objective Function:

Min-Max optimization problem for jointly train G and D:

$$\min_{\theta_g} \max_{\theta_d} \left[E_{x \sim P_{data}} \log D_{\theta_d}(x) + E_{z \sim P_z} \log \left(1 - D_{\theta_d}(G_{\theta_g}(z))\right) \right]$$

$D_{\theta_d}(x)$: Discriminator Output for Real Data
$D_{\theta_d}(G_{\theta_g}(z))$: Discriminator Output for Generated Fake Data $G(z)$

- Discriminator aims to MAXIMIZE the objective such that $D(x)$ is close to 1 (real) and $D(G(z))$ close to 0 (fake).
- Generator aims to MINIMIZE objective such that $D(G(z))$ is close to 1.

Training GANs:

Stochastic Gradient Descent alternating between

gradient ascent → discriminator: $\max_{\theta_d} [E_{x \sim p_{data}} \log(D_{\theta_d}(x)) + E_{z \sim p_z} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$

gradient descent → generator: $\min_{\theta_g} [E_{z \sim p_z} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$

Function to be optimized is complex → TRAIN IS UNSTABLE

Plotting gradient of G is quite flat where the model should learn more.

↳ Turn it into MAXIMIZATION PROBLEM → NOT minimize prob. D is correct, BUT maximize prob. D is wrong.

↳ New training procedure have 2 ascending gradients:

$\max_{\theta_d}$ is the same, $\max_{\theta_g} [E_{z \sim p_z} \log(D_{\theta_d}(G_{\theta_g}(z)))]$

Infereuce:

~~→ In questi modelli (probabilistic generative models) la fase di inferenza è quella dove viene generato un nuovo output (SAMPLING)~~

Discriminator is just an auxiliary object to match the distribution learned from the generator with the distribution of our data. ~~→ Il ruolo del discriminatore era solo quello di "guidare" il generatore durante il~~

If this version of GAN is trained with a dataset unconstrained, it performs poorly. Problems of GANs:

- Overfitting: there are not indicators that indicates overfitting
- Evaluation: no standardized and reliable metrics to evaluate GAN
- Training Stability: GANs are unstable
- Parameter oscillations and divergence during train
- Mode Collapse generator, instead of capturing all modes of the target distribution, ends up capturing only one sub-mode (i.e. it learns only a certain area). So at the end Generator isn't able to capture diversity of target.

Deep Convolutional GAN (DCGAN)

DIFFERENCES FROM CLASSICAL CNN OF DISCRIMINATOR

- NOT use Pooling, but only convolutions with stride
- Leaky ReLU is used as activaction function.
- Only 1 fully connected layer is presented before softmax.
- Batch normalization after most levels
  - ↳ ! introduce correlation between samples within the minibatch.

Arithmetic of GAN

Generator learns to map points in the latent space to specific images, and this mapping will be different each time model is trained.

To perform output, generator performs vector arithmetic with faces

EXAMPLE: Smiling Woman − Neutral Woman + Neutral Man = Smiling Man

EVALUATION OF GANS

We are not only interested in the quality of generated images, but also in the overall distribution of the GAN-generated samples. However, there's no a direct way to calculate the probability of high-dimensional ~~space~~ samples or to compare distributions. A paper proposed Inception Score and Fréchet Inception Distance (FID).

Evaluation problem → Ability of GAN to generate RECOGNIZABLE OBJECTS
↘ VARIETY OF OBJECTS generated

↳ Both are associated with probability → computed using classifier InceptionV3 (output to compute Inception score)

Inception Score: maximized if the entropy of the distribution of the labels generated is minimized (recognizable) and predictions of classification are uniformly distributed over all possible labels (variety).

FID: real and fake are passed through a classification network and for a chosen layer the activations are calculated. Metric that takes embeddings of the layer and treat them as multivariate normal distributions. Mean and covariance are calculated for both the generated samples and the actual data are compared using Multivariate Normal Fréchet Distance.

DIFFERENT GAN LOSSES

LSGAN: cross entropy replace with least Square because less sensible to outliers.

$$L_D^{LSGAN} = E[(D(x)-1)^2] + E[D(G(z))^2] \qquad L_G^{LSGAN} = E[(D(G(z))-1)^2]$$

↳ Better quality generated images and more robust generators to mode collapse.

Wasserstein GAN: Wasserstein distance aligns 2 distributions with the concept of optimal transport. → Aligning the 2 distributions with this distance providing better gradients and more stable training. Implementing with DCGAN allow to correlate samples quality with this distance.

GANS TRAINING TRICKS (to contrast mode collaps)

Feature matching: generator try to focuses on matching the statistics of features from an intermediate layer of discriminator. Minimizing difference between these statistics, generator avoid overfitting

MiniBatch Discrimination: helps generator produce more diverse outputs by allowing discriminator to examine relationships between samples in a mini-batch. Enables D to identify if G is generating similar images, forcing G to variety.

Virtual Batch Normalization each example is normalized based on statistics collected on a reference batch of examples chosen once and fixed at the beginning of training, and on itself.

GANS ZOO

Conditional GAN: Links an input with its corresponding label, using supervised informations. Labels are an extension of latent space.

Image-To-Image translation discriminative network evaluates how realistic and consistent the generated image looks with the input. Train require couples.

Text-to-Image Synthesis : G receive text and noise and produce an image. D distinguishes between real images with matching descriptions and generated.

Cycle GAN: translating images between 2 different domains, overcoming the need for matching pairs of images. Add one element : TRANSLATOR, to convert images from one domain to another.

Progressive GAN: Progressive growth of G and D networks. Enable high-quality images.

Style GAN: G can capture and manipulate style attributes of images.

Drag-GAN: allow interactive editing of images via simple Drag-and-Drop method

AUTOENCODERS AEs

Are unsupervised models that learn to represent input data in a space of reduced size and then reconstruct it. This process occurs in 2 main steps:

ENCODER: INPUT → ENCODER → latent representation of lower dimension (z)
↳ contains only the most relevant features

DECODER: z → DECODER → reconstructed sample x' (same dims as input)

} Symmetric Networks

x' COMPARED WITH x via a RECONSTRUCTION LOSS FUNCTION (Norm L2)

↳ GOAL of train: optimize enc. e dec. weights to minimize loss.

Afer train, decoder can be discarded and the encoder can be used to initialize a supervised model for predicting labels from the z feature space.

AUTOENCODERS ARE VERY FLEXIBLE FOR DIFFERENT TYPES OF DATA

VARIATIONAL AUTOENCODERS VAEs

VAEs use a direct approach to estimate the likelihood function of the data.

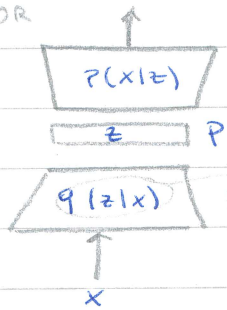
Likelihood: $P_\theta(x) = \int P_\theta(z)\, P_\theta(x|z)\, dz$ (PRIOR, DECODER) z: latent representation

we cannot compute $P_\theta(x)$ directly because it is mediated by z, which is a latent representation learned from training samples and not from data itself.

SOLUTION: in addition to decoder network modeling, define additional encoder network that approximates $P_\theta(z|x)$ (POSTERIOR).

↳ Allow to derive a LOWER BOUND on the data likelihood that is tractable and we can optimize.

Encoder and decoder are probabilistic.



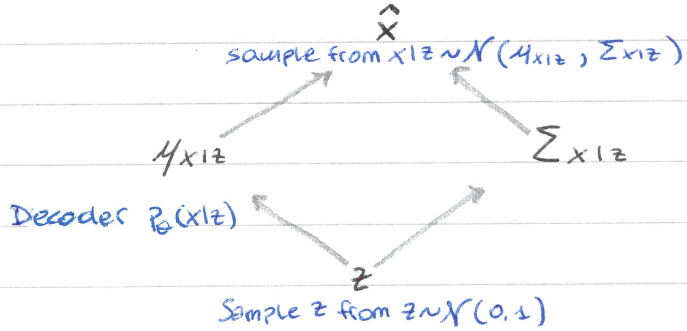
MAXIMIZING LIKELIHOOD LOWER BOUND: $E_z[\log P_\theta(x^{(i)}|z)] - D_{KL}(q_\phi(z|x^{(i)}) \| P_\theta(z))$

Reconstruction Term — Regularization Term

- Applied to decoder to train it to generate meaningful results
- forces the latent space to fit a Gaussian Distribution

} Allow latent space to be continuous and well-structured

GENERATING DATA
↓
use decoder and sample z from prior



VQ-VAEs

In VAEs prior and posterior are assumed to be normally distributed with diagonal variance.

In VQ-VAES are used discrete latent variables. Prior and posterior are categorical, and samples drawn from these distributions index an embedding table.

Encoders model a categorical distribution, from which we sample To obtain integral values. These values are used to index an embedding dictionary, and the indexed values are then passed to the decoder.

SUMMARY VAEs

- Allow generating data
- Derive and optimize a lower bound
- PRINCIPLED APPROACH TO GENERATIVE MODELS
- ALLOW INFERENCE OF $q(z|x)$, can be useful for other task.
- Lower quality images wrt GANs.

GAN vs VAE

GAN Provides mechanism to generate samples less distinguishable from real data.

VAE Learn a probability distribution over training data that can be used to draw new sam

GENERATIVE MODELS PROPERTIES

High Quality Sampling: sample not distinguishable from real data ✓ ✗ → mode collapse

Coverage: Samples should represent entire training distribution. ✗ ? → Dipende dalla Prior

Well-behaved latent space: Every latent z should correspond to a x, and a small change ✓ ✓ in z should effect x.

Interpetable latent space: manipulating each dimension of z should correspond to ? ? changing an interpetable property of the data.

Efficient likelihood computation: If the model is probabilistic, we would like to be n/a ✗ able to calculate the probability of new examples efficiently and accurately.

GAN VAE

10. DIFFUSION MODELS

Are generative models invented at the same time as GANs, but They gained popularity only in 2020. This delay is due to several factors: computational complexity, need for more powerful hardware and more.

Training can be divided in 2 steps:

1. Forward Diffusion Process ($q(x_t|x_{t-1})$): contents of an input image are "destroyed" by adding noise over time t until the data distribution becomes a Gaussian
2. Reverse Diffusion Process ($p_\theta(x_{t-1}|x_t)$): remove noise from the image in order to reconstruct the original image from a Gaussian distribution of noise.

FORWARD PASS

To approximate a Gaussian distribution, the added noise is increased slightly at each step (t). To obtain a better approximation, the direct diffusion procedure require MANY ITERATIONS (T). → Trade off: +STEPS → ↑approximation → + resources

All intermediate steps are Gaussianly distributed:

$$q(x_t|x_{t-1}) = \mathcal{N}(\underbrace{\sqrt{1-\beta}\, x_{t-1}}_{\text{mean}}, \underbrace{\beta_t I}_{\text{variance}})$$

Determina quanto rumore viene aggiunto

β_t represents the (small) increase in the noise variance at each time interval t.

Defining $\alpha_t = (1-\beta_t) \rightarrow q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{\alpha_t}\, x_{t-1}, (1-\alpha_t)I)$

Instead of designing an algorithm that compute this iteratively, we can use a closed-form formula to directly sample a noisy image at a specific time interval t using the REPARAMETRIZATION TRICK (Also in VAEs):

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon_0^*$$

$\bar{\alpha}_t = \bar{\Pi}^T \alpha_t$ product of all alphas in the chain. $x_t \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)I)$

ε_0^* random variable of Gaussian noise $\varepsilon \sim \mathcal{N}(0,1)$.

The final formula depends only on the input image x_0.

↳ It allow us to directly sample x_t at any time interval.

REVERSE PASS

We cannot simply use $P(x_{t-1}|x_t)$ because it is intractable. We need to train a Neural Network $p_\theta(x_{t-1}|x_t)$ to approximate the ~~inverse~~ posterior distribution.

The approximation follows a normal distribution (inverse of a Gaussian process is still a Gaussian distribution), and NN learns only the parameters μ_θ of the inverse Gaussian process, since variance is fixed, it is shared among the

intermediate inverse steps: $P_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t,t), \Sigma_t)$

↳ Instead of optimizing the loss function, we can optimize the Variational Lower Bound. Eventually, the network can be trained with ELBO (evidence lower bound)

$$\mathcal{L} = \underbrace{\mathbb{E}_{q(x_1|x_0)}[\log P_\theta(x_0|x_1)]}_{\text{Boundary Condition}} - \underbrace{\sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)}[D_{KL}[q(x_{t-1}|x_t,x_0)\,||\,P_\theta(x_{t-1}|x_t)]]}_{\text{KL Divergence at every other step}}$$

- Maximizing $\mathcal{L}$ is equivalent to maximizing the likelihood of data we want to generate.
- $\mathbb{E}$ stands for expected value: we're taking an average over all possible first noisy steps.
- $\log P_\theta(x_0|x_1)$ encourage the model to be good at taking a slightly noisy image and reconstructing the original.
- $\sum_{t=2}^{T}$ we are summing up a cost for each step in the reverse diffusion process
- $\mathbb{E}\, q(x_t|x_0)$ averaging over the distributions of noisy images at each step.
- $D_{KL}[\dots]$ measure how different 2 distributions are.

The inverse process, which removes the added noise, is described by the following closed-form formula: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(x_t,t)\right) + \sigma_t z$

$\varepsilon_\theta(x_t,t)$ it's the noise prediction made by the NN. This formula tell us how to take a noisy piece of data and make it a little less noisy. We do it by

1) Estimate noise that was added (using NN)
2) Subtracting the ~~est~~imated noise
3) Adding a bit of new random noise.

} Repeat many times

GENERATIVE TRILEMMA

GENERATIVE MODELS
- GAN → Fast Sampling + High Quality
- VAE → Fast Sampling + Diversity
- Diffusion Models → High Quality + Diversity

Recently, diffusion models have made progresses in fast sampling, try to balance all 3 properties

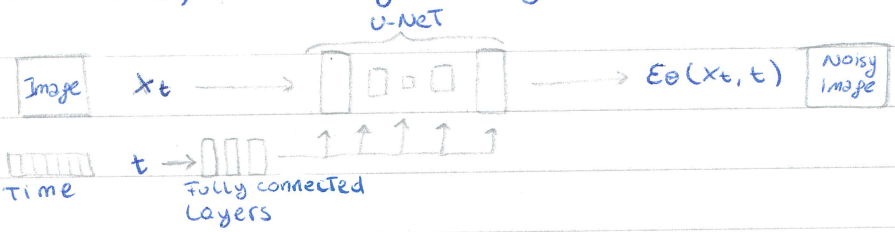
STOCHASTIC EQUATION and DENOISING SCORE MATCHING

Stochastic Equation: In forward diffusion process, noise is added to the original image at each step. This process can be described by a stochastic equation that separates the effect of noise into 2 components : AVERAGE CHANGE and RANDOM VARIATION. In reverse process, they guide the model in removing noise.

Denoising Score Matching : aims to minimize difference between noise added and noise estimated by NN. Allow model to refine its ability to interpolate between latent spaces.

ARCHITECTURE

Temporal information is embedded via Fourier features (sinusoidal functions). For images, U-Net architecture is used with the addition of attention mechanisms, commonly employed to preserve and reconstruct complex spatial details.



TEXT CONDITIONING

How to guide diffusion models to generate text-based images?

Text conditioning uses textual information to influence image generation.

- **Textual Embeddings:** textual descriptors are converted into numerical vectors These embeddings are integrated into the diffusion model, guiding the generation of the image.
- **Training:** model is trained on text and image pairs, minimizing difference between generated image and target image based on Text.

STABLE DIFFUSION

Latent Space: compressed latent space obtained through an encoder. Process is more efficient A decoder will reconstruct image from latent space.

Denoising U-Net: In reverse use U-Net for denoising. Equipped with attention to capture more and complex image details.

Conditioning Space: include additional conditions (text, semantic information, ...) to guide image generation.

11. LARGE MULTIMODAL MODELS

Multimodal architectures are computational models designed to process and integrate data from different modalities or input types.

All methods To Train encoder (to learn a representation of our data - images + text -) share the same goal: build a robust model capable of learning robust representation of data. There are 4 main methods:

Supervised learning: train on a large amount of manually annotated data.

Image-Only (Non-) Contrastive Learning: allows encoder to be trained on image only, without the need for labels.

Masked Image Modeling some parts of the image are "masked" and the model's task is to predict the missing parts. Force encoder to understand visual context.

Contrastive Language-Image Pre-Training: train encoder that can understand both language and images (e.g. CLIP).

Image-Only (Non-) Contrastive Learning

Dependency on negative samples. The use of negative samples can be replaced with asymmetric architectures and clustering. DINO is based on clustering. It* use a teacher-student framework with cross-entropy loss. The student network extracts the parameters of interest, and the output goes through a softmax to determine whether an image belongs to a certain cluster. Teacher network is updated using (EMA) of parameters learned from student network.

student and Teacher have same architecture but a different set of weights

Gradient propagates only in student network.

Masked Image Modeling

BEiT: BERT Pre-training of Image transformers. Before pre-train, learn an "image tokenizer" where an image is tokenized into discrete visual tokens.

Contrastive Language-Image pre-training

Learning image representations from web-scale noisy text supervision.

For train is a simple contrastive learning, large-scale pre-tain.

Downstream: zero-shot image classification and image-text retrieval.

Image model design Improvement for image encoder is FLIP, which try to scale CLIP training via randomly masking out image patches with a high masking ratio.

This allow encoder to process only ~~non-masked~~ non-masked patches.

K-LITE introduced a mechanism to extend text knowledge via external knowledge in order To provide more detailed informations. For example Wikipedia can be used. During train, the model acquires the ability to read and understand a specific knowledge source. During evaluation, the knowledge provides an additional source of information to improve model inference. This method improve performance on the basis of the dataset used.

Model design Improvement: Multimodalities. Adding more modes (audio, video, etc.). ImageBind linking all modalities (7) into a common space. The objective is to create a single joint embedding space for all modalities, utilizing images as the binding factor. Employs pairs of modalities (I, M) where I is the image. It obtain two different modalities (M_1, M_2) aligned even though it is trained using ontly the pairs (I, M_1) and (I, M_2).

To improve interpretability, was introduced STAIR that creates a sparse, semantic representation of imagest and text. For each image or text, STAIR constructs an embedding space in which each dimension of the vector corresponds to a vocabulary word.

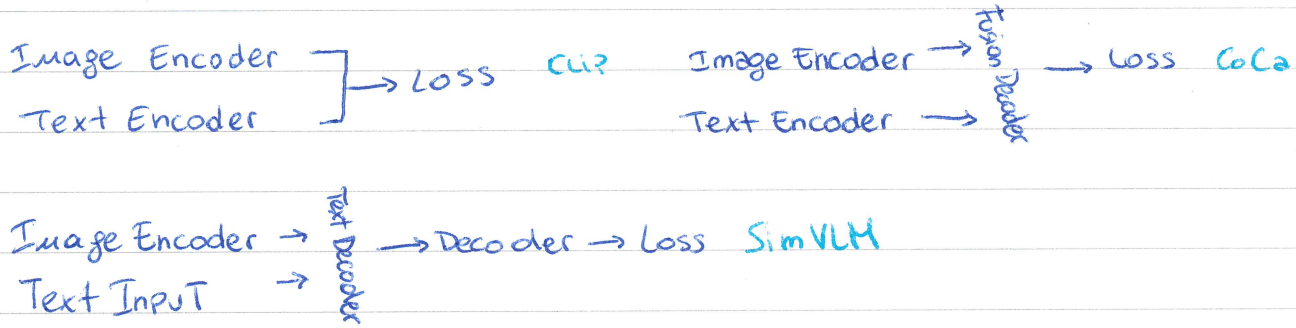
Each token in the dictionary is associated with a non-negative scalar value, indicating the token's importance to the image or text in question. STAIR uses a unit called TOKEN PROJECTION HEAD to project representations into its sparse embedding space. We can observe the relevance of each word in embedding space.

Objective Function Improvement:

- FLIP: uses 2 encoders. Visual encoder based on ViT takes an image divided into patces. A special token [cls] is added. Patches and [cls] are linearly projected and then processed by the model. After that, the text is processed by a Transformer-only decoder (Text encoder). This handles the token sequences generated by the words in the Text. The objective function of FLIP is fine-grained alignment: TOKEN WISE SIMILARITY (sim. between each token in text and each ~~in~~ patch in the image), POOLING and AGGREGATION (max pooling to aggregate similarities).

CoCa : enhances clip by adding a generative branch allowing model to captioning images. This lead to the creation of a basic encoder-decoder image-text pretrained model, jointly trained using a contrastive loss and a captioning loss. Unified single-encoder, dual-encoder, encoder-decder paradigms: one image-text foundation model with the capabilities of all three approaches. Multimodal decoder cross-attending to image encoder outputs to learn multimodal representations.
Model not only creates a global representation of the text, but also predicts word by word the text associated with the image.

SimVLM : employs a transformer encoder-decoder model. The idea is to have the model generating text sequentially, using the preceding tokens and visual context to predict each subsequent word. It doesn't compete with clip in terms of performance.



LLMS AS UNIVERSAL INTERFACE

These models need to handle a multimodal prompt. The idea is to train them on vision-conditioned language-generation tasks.

Frozen LM Prefix

This model consists of 3 main components: visual encoder, language embedder and a language model for text generation. The weights of last 2 components doesn't change during train or inference. The resulting textual and visual representations are concatenated and then sent to the language model ~~encoder~~ decoder (LLM), which generates the textual output in an autoregressive manner.

Next, fine-tuning of the image encoder is performed to improve quality of its representations. Its outputs are used as soft prompts for the LLM, which means that they guide the behaviour of language model without change its weights.

Flamingo

Represents an advance over previous frozen models. It links powerful pre-trained visual encoders and pre-trained language models using innovative architectural components.

Flamingo's visual encoders, based on a Normalizer-Free ResNet architecture, are trained similarly to CLIP (for visual scene). For textual encoder, use BERT, and then it's discarded after the visual encoder is trained because it's not used in final model.

Flamingo's main innovations include the Perceiver Resampler and Gated XATTN-DENSE layers, which are trained from scratch.

The model is trained with sequences of heterogeneous data (image and text). Perceiver Resampler provides meaningful representations of fixed size to address the challenge of handling varying sizes inputs.

Flamingo uses Chinchilla as its language model.