**Shift invariance**: property that describes a system's unchanging response when the input is shifted;

useful in CNNs (e.g. obj recog.) vs **shift equivariance** (e.g. obj detection, segmentation)

**Assumptions**: translation invariance (ashift in input should simply lead to shift in the hidden representation) + locality

Pooling layer ensures ***approximate* translation invariance**

**Receptive field** of a layer k

For real-valued functions, of a continuous or discrete variable, conv = cross-correlation of f(x) & g(−x), or f(−x) & g(x)

**Convolutions & CNNs**

Output shape (width or height): **floor((W – K + 2P) / S) + 1**

Output shape (depth/num of channels): num of filters applied

Need to flip kernel horiz/vert.

Convolution:

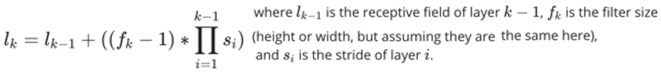
#params of 1 filter: filter H x filter W x filter C + 1 (bias)

Conv = only equivariant to translation; not equivariant to

warp, flipping… hence data aug for better model general. too

CNN approx. invariant if adding pooling layers.

A math equations on a white background

Description automatically generatedA group of squares with numbers

Description automatically generated

**CNN architecture**

- convolutions implemented through weight sharing, interpreted as weights of a filter function

- more output channels => NN can learn more complex + high-level features. Padding to preserve input shape.

- final FC layers: can be comp. + memory expensive (bottleneck w/ increasing num of classes, e.g. 1000 for ImageNet)

- 1x1 conv acts like an MLP per pixel, which aggregates across channels of input; introduces complexity + nonlinearity

- **AlexNet** for ImageNet: deeper + bigger/wider LeNet. Add dropout, sigmoid => ReLU, maxpool, heavy data aug, model ensembling (model averaging across multiple well-performing CNN models to achieve SOTA results) vs LeNet for MNIST

- **VGG**: group layers into blocks (num of blocks varies). Blocks can be easily parameterized, creating a more organized, modular arch. Simplifies design process + easier to fine-tune model for specific learning tasks.

- More layers of *narrow* convolutions outperforms using fewer *wide* ones: lots of simple fns > few complex fns

- **Inception**: deep + parallel paths with blocks to capture different types of features more effectively. Combines benefits of various convolutions & pooling operations while optimising computational cost

- **Batchnorm** normalises features within each minibatch, stabilising training process & speeding up convergence. Regulatisation by noise injection + no dropout needed (both control NN capacity) + ideal minibatch 64-256). B = batch. Test: fix the gamma & beta learned in training + instead batch statistics, use running average for mean and variance

- **ResNet**: get increasingly powerful AND nested functions (might not be convex) with more layers. “Taylor expansion” style parametrization. Input to act fn becomes f(x) ***+ x*** (x sometimes \* with 1x1 conv to change dim). Allows for deeper NNs w/ fn classes more likely to be nested. Better grad flow (solve vanishing). ResNet module: multiple ResNet blocks.

- **DenseNet** uses higher-order Taylor series expansion; feature maps reuse; may need to reduce res (transition layer)

- **curse of dim**: As the number of features or dimensions grows, the amount of data we need to generalise accurately grows exponentially

- **pooling**: permutation-invariant aggregation + downsampling; reduces res; hierarchical features; max-pooling breaks shift equivariance. **Separable filter**: filter can be written as conv of 2/more simple filters: e.g. 2D filter from 1D filters; and det(filter)=0.

A diagram of a diagram

Description automatically generated with medium confidenceA close-up of a number

Description automatically generated

For correlation: it’s g(x + tau)

A blue square with black numbers and a white background

Description automatically generated

A math equation with black text

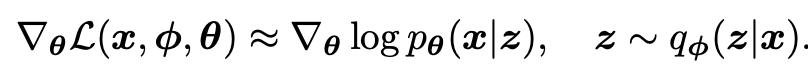
Description automatically generated with medium confidence

A math equation with black text

Description automatically generated with medium confidence



Tuto answers

A black and white math equation

Description automatically generatedA black and white logo

Description automatically generatedA close-up of a number

Description automatically generatedA close-up of a logo

Description automatically generatedA math equations and symbols

Description automatically generated with medium confidenceA black and white math equation

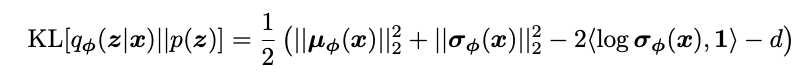
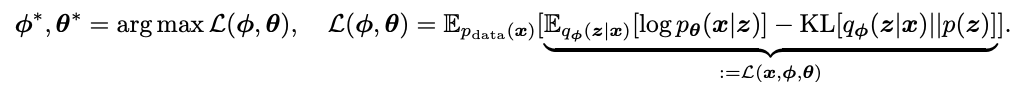
Description automatically generatedA mathematical equation with black letters

Description automatically generatedA black text on a white background

Description automatically generatedA math equations and symbols

Description automatically generated with medium confidenceA black text on a white background

Description automatically generatedA black and white math equation

Description automatically generatedA group of mathematical equations

Description automatically generated

**Reparametrization trick: directly sampling *z* from *q* and passing samples through the model to compute gradients is not differentiable wrt parameters ɸ of dist. from which *z* is drawn 🡪 hard to backpropagate**

*Gradient wrt KL term is tractable, MC still required for first term*

*Analytic form:*

**VAE optimization objective:**

**Variational auto-encoder approach defines *q* distribution as a neural network:**



**Variational Inference:**

Latent variable model: , often constructed as

Expand the above objective:

**MLE:**

given a dataset , we want to fit to a generative model with parameter theta:

Probabilistic graph models (joint distribution): used to describe dependency structure

**VAE**

The definition of divergence is weaker than distance: does not need to satisfy symmetry or triangle inequality

We can work with PDFs:

G𝜽 () defined as a neural network transform parametrized by weights theta. We optimize MLE objective wrt 𝜽, which involves computing integral 🡪 intractable as it computes non-linear transformation G𝜽(z) for every single configuration of z within Gaussian p(z)

**Monte Carlo estimation: is still intractable, so we replace expectation with MC approximation:**

A diagram of a mathematical equation

Description automatically generated

We can ignore the constant terms wrt theta and instead work with the *maximum likelihood* object:

**KL Divergence: minimize this in order fit a distribution to a target one, asymmetric:**

A math symbols and signs

**Conditional VAE: generate data conditioned on additional information (class labels, viewing angle). *y* is the additional info**

A black text on a white background

Description automatically generated

Solution: use the variational lower bound as a tractable approximation to marginal log-likelihood



If ***x*** is cont., then G(z,y) is a neural network. We maximize a variational lower bound:



A black text with a white background

Description automatically generated

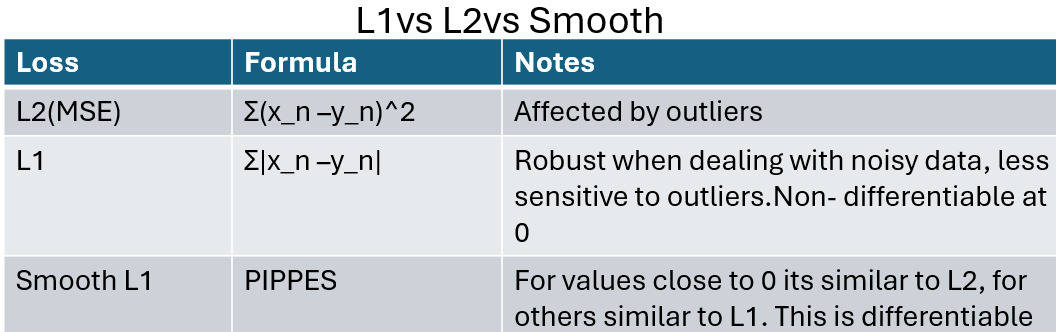
or minimize KL divergence:

**GAN.** Constructs a binary classification task to assist the learning of generative model dist. to fit data dist.

Logsigmoid: output is always negative

A close-up of a logo

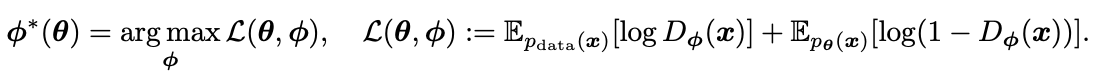
Description automatically generatedA graph of a function

Description automatically generated

A graph with a line drawn on it

Description automatically generated

Fit a binary classifier (discriminator) by maximizing the MLE objective:



Generator fools discriminator by **minimizing** log prob. of making the **right** decisions

A black text on a white background

Description automatically generatedA black text on a white background

Description automatically generatedA table with equations and formulas

Description automatically generated

Two-player game objective for GAN is:

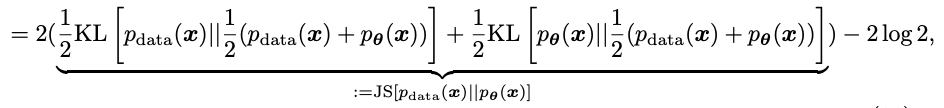


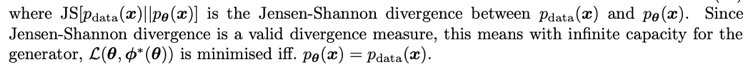
MC approximation: the evaluation of the obj. can directly define sampling process of , which also defines the distribution in an implicit way:

A graph with a line drawn on it

Description automatically generated

Jensen-Shannon divergence minimization: For a fixed generator, setting the gradient of GAN objective = 0 results in:





**Losses. L1/2/CE**: “reduce” to scalar using mean or sum.

**Smooth L1**:

**NLLL**:

**Derivatives** (g is the fn)

**Sigmoid:**

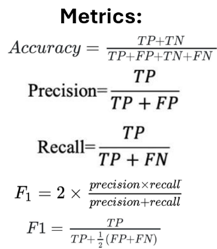
**Tanh:**

**ReLU:** 1 if z > 0, 0 if z <= 0

**Softmax (w/ CE loss):**

A math equations with numbers and symbols

Description automatically generated with medium confidenceUne image contenant texte, Police, écriture manuscrite, capture d’écran

Description générée automatiquement

*Alternative ”non-saturated” objective given fixed discriminator:* Saturation problem (near-perfect classf. at start of training 🡪 vanishing gradient); so, **max**. log prob. of making **wrong** predictions.

A black text on a white background

Description automatically generatedUne image contenant texte, Police, écriture manuscrite, capture d’écran

Description générée automatiquement

**Conditional GAN**

Distribution form of p is defined implicitly by the sampling process:

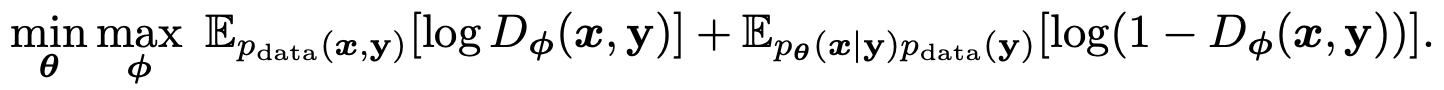
A math equations and formulas

Description automatically generated with medium confidenceA math equations and formulas

Description automatically generated with medium confidence

Learning is done by optimizing an adversarial objective (similar to GAN, min max L(***x,y***) )

A math equations and formulas

Description automatically generated with medium confidence

**CE loss:** combines logsoftmax(output) & NLLL. Use for classif. Average across obsv. within minibatch



**Binary CE loss:** CE loss for only 2 classes. Inputs as probas or logits

A math equations and formulas

Description automatically generated with medium confidenceA group of black symbols

Description automatically generated

**(above) KL div. loss:** dist. btw proba distributions

**GRUs**

- improves simple RNN with gates like LSTM

- vs. LSTM: GRU removes the input/output gates and the cell state, but maintains the forgetting mechanism in some form

**RNNs**

- can model dependencies within a sequence of arbitrary length

**LSTMs**

- Weights/biases shared across cells; sigmoid and tanh activation fns; final output/prediction = short-term memory outputted from last cell

- different paths for long-term & short-term memories to avoid gradient vanishing/expl problem from RNNs

- memory cell states and gates to control error flows

A diagram of a network

Description automatically generated

Simple RNN computes this mapping at each t:

A group of math equations

Description automatically generatedA math symbols with numbers

Description automatically generated with medium confidence

A diagram of a network

Description automatically generated

A math equations and formulas

Description automatically generated with medium confidenceA close-up of a computer code

Description automatically generated

BPTT =>

Can truncate:

A black text on a white background

Description automatically generated

**Seq2seq models**

A close-up of a text

Description automatically generatedA math equations with numbers

Description automatically generated with medium confidenceA mathematical equation with numbers and symbols

Description automatically generated

h0, c0 set to 0 or learnable. If 0, then we have the elems in c\_t and h\_t

A diagram of a solution

Description automatically generated with medium confidence

A close-up of a text

Description automatically generated

A diagram of a computer code

Description automatically generated with medium confidenceA black and red symbol

Description automatically generated

- weights/biases = shared across cells, where each cell = single-layer NN with 1 input and 1 output at t. no matter #cells/timesteps, no increase in #learnable params! BUT **issue w/ gradients**: vanishing or exploding => diff to train.

Some **tricks**: grad clipping; good W\_h init; unitary/orthog. W\_h

A black text with a plus and a white background

Description automatically generated

A close-up of a symbol

Description automatically generatedA black symbols with arrows

Description automatically generated with medium confidence

Hyperparam lambda; clip ***g*** if

ShapeA diagram of a multi-layered diagram

Description automatically generated with medium confidenceA diagram of a process

Description automatically generated

**Attention**

Single-head attention: *scaled dot product attention*

Multi-head attention: multiple alignment processes by projecting inputs into diff. subspaces, then performing dot product attention in subspace. Outputs are concatenated and projected

**Transformers**

*Position encoding: attention is equivariant to row permutations bc non-linearity a(.) is applied row-wise. Ordering info needs to be added to attention inputs, can either be learned or computed, learned*

is good if we know max value of index

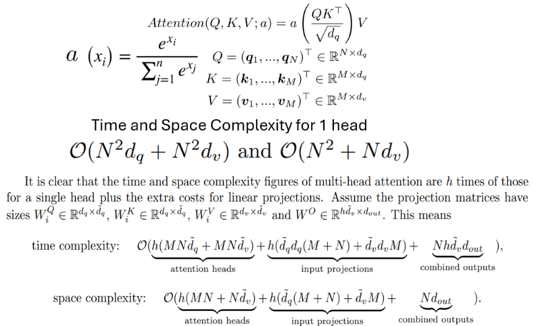
*Sinusoid embedding: use multiple periodic functions to embed*

input index, frequency of sin/cos wave is det. by *i*.

Allows network to learn very flexible position encoding

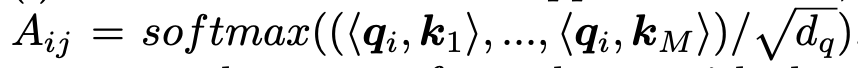
function

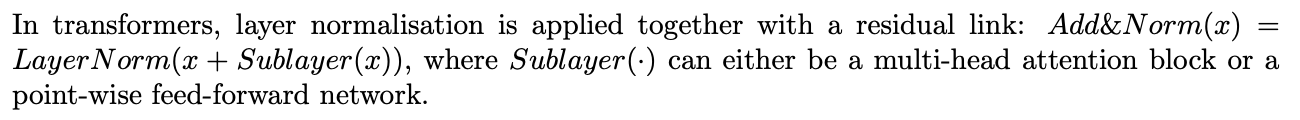
Point wise feed forward network: used as ff layer, multi-head attention (after *Add&Norm*) returns a matrix of size N x d\_out, which can be processed “point-wise”, treating rows in the attention outputs as “datapoint” inputs for next layer



soft attention: a(.) is the softmax function,

hard attention: one-hot vector for each row with equal to 1





**Margin Ranking/Ranking/Contrastive Losses**

A close-up of a black text

Description automatically generatedA close-up of a black text

Description automatically generatedUne image contenant texte, capture d’écran, Police

Description générée automatiquementUne image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementA white paper with black text

Description automatically generatedA screenshot of a math book

Description automatically generated

**FLOPs**

**Efficient training: grad accumul., checkpointing**

**Triplet Margin Loss**

A white background with black text

Description automatically generatedBlack text on a white background

Description automatically generatedBlack text on a white background

Description automatically generated

A white background with black text

Description automatically generated

**Cosine Embedding Loss**





**Finetuning**

A black and white symbol with white text

Description automatically generatedA black and white symbol with white text

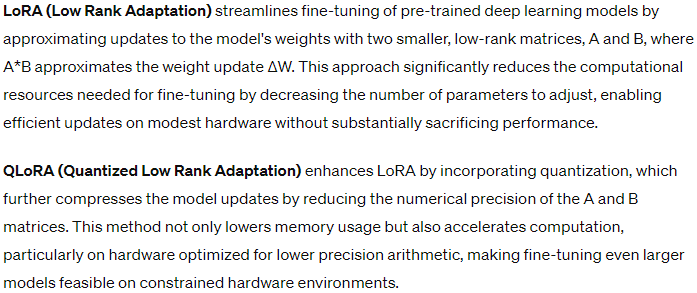
Description automatically generatedA black and white symbol with white text

Description automatically generatedA close-up of a text

Description automatically generatedA diagram of a cell state

Description automatically generatedA diagram of a cell state

Description automatically generatedA diagram of a cat

Description automatically generatedA math equations and symbols

Description automatically generated with medium confidence

**GRU**

**LSTM**

**A red circle with a white s in it

Description automatically generated**A diagram of a fraction

Description automatically generatedA screenshot of a computer program

Description automatically generated

**Conv**

* Don’t assume that in for a single kernel, the same kernel is applied to each channel of the input image
* Max pool and average pool are applied PER channel (not aggregating across channels of input image): only operate on width and height of input
* Mean and var learned gamma beta: to give new dist
* Valid padding = no padding at all
* Same padding = as much padding as needed to preserve the dims of input
* 0 padding needed for same padding: (K – 1) / 2
* Num of params for a conv layer with (more than 1) F filters of the same shape: F \* (width of filter x height of filter x input channels + 1) where 1 is bc we have 1 bias per filter
* Batchnorm output: same shape as input. Num learnable params: 2 x num of channels in input. It’s 2 bc you have gamma and beta
* Pooling layers = no learnable params

**Backprop / gradients**

* Add the expression for dJ/dy\_hat when using J = cross entropy loss with softmax being used:

1/m (y\_hat – y)

* If largest singular value of a weight matrix > 1, then gradient explosion happens

**DAG**

* Conditional = joint / priors
  + Hence for q(z1,z2 | x1, x2): the priors of x1 and x2 cancel out and you don’t write them as factors