IFT 4031/7031,
Machine Learning for Signal Processing
Week 7: Intro to Deep Learning

Cem Subakan





UNIVERSITÉ LAVAL



Mila

Admin

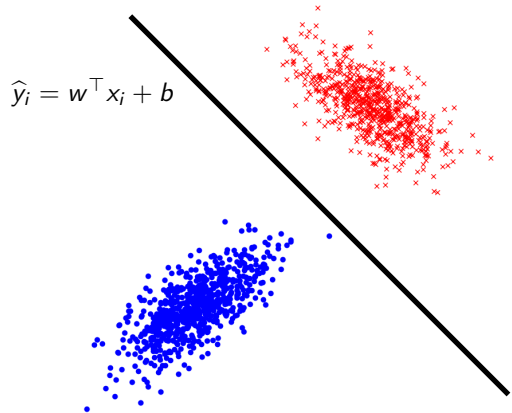
- Homework 2 will be out soon.
  - Le devoir 2 sera bientôt disponible.
- We will assign a TA for each team. You are expected to work closely with them.
  - On va assigner un auxiliaire d'enseignement à chaque équipe. Vous devez travailler étroitement avec lui/elle.
- VALERIA is available now. How is it going?
  - Comment ca va avec VALERIA?

Today

- Neural Networks / Réseaux de Neurones
  - We will talk more about why / On parler davantage sur le ‘pourquoi’.
- Deep Learning
  - We will talk about how / On va parler sur ‘comment’.

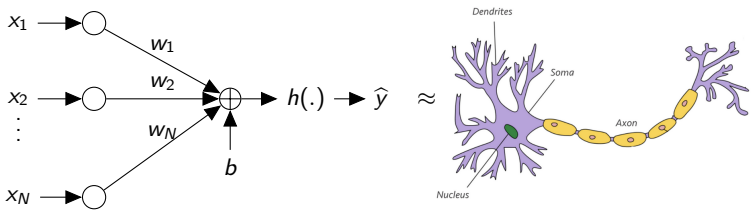
Linear Classifier Again

- Linear classifier / Classificateur Linéaire



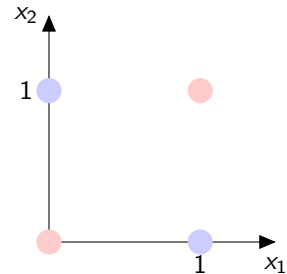
Why 'Neural' Network

- $\widehat{y} = w^\top x + b$



The XOR problem

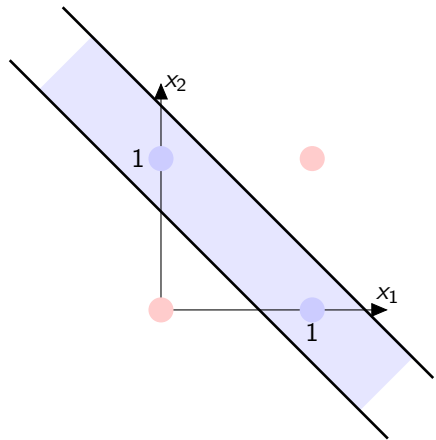
x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0



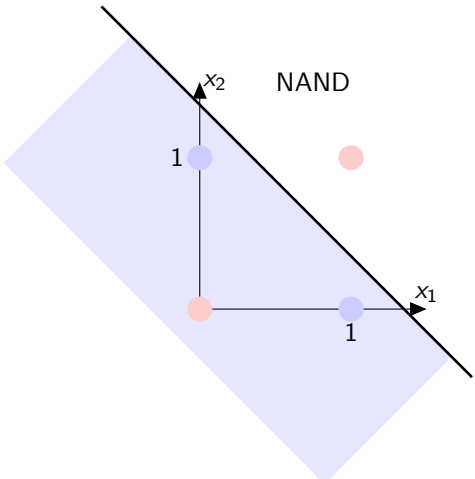
- Can we linearly separate this? / Peut-on séparer ça linéairement?
  - **No!** / **Non!**

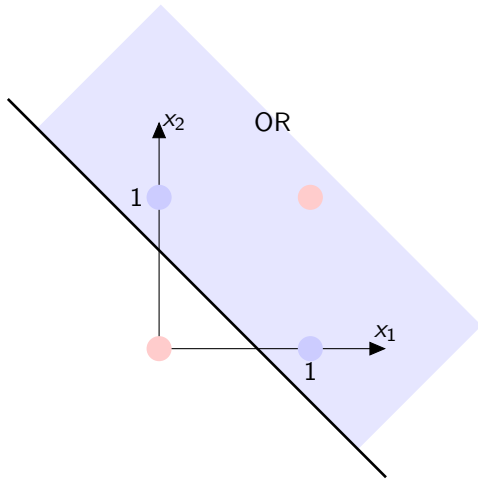
An ideal solution

How do we do this though? /
Comment peut-on faire ça?



Combine!

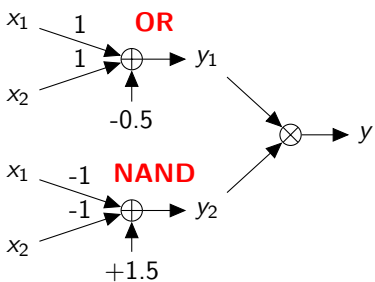




Two parallel perceptrons

x_1	x_2	y_1
0	0	-0.5
1	0	0.5
0	1	0.5
1	1	1.5

x_1	x_2	y_2
0	0	1.5
1	0	0.5
0	1	0.5
1	1	-0.5

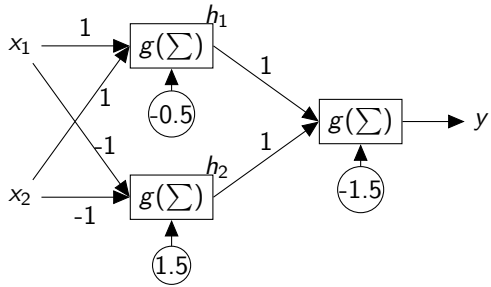


x_1	x_2	y
0	0	-0.75
1	0	0.25
0	1	0.25
1	1	-0.75

Or, everything with a single network

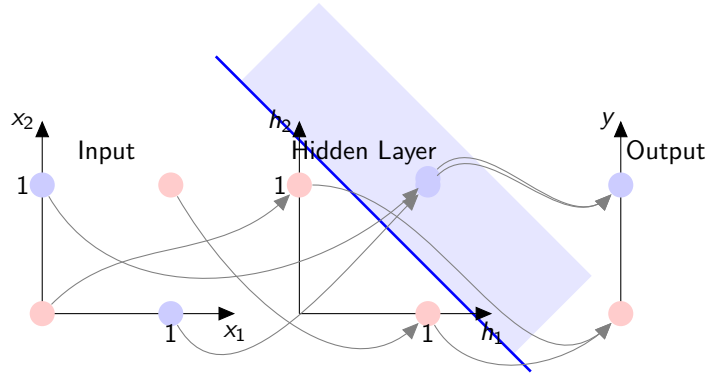
x_1	x_2	h_1
0	0	$u(-0.5) = 0$
1	0	$u(0.5) = 1$
0	1	$u(0.5) = 1$
1	1	$u(1.5) = 1$

x_1	x_2	h_2
0	0	$u(1.5) = 1$
1	0	$u(0.5) = 1$
0	1	$u(0.5) = 1$
1	1	$u(-0.5) = 0$



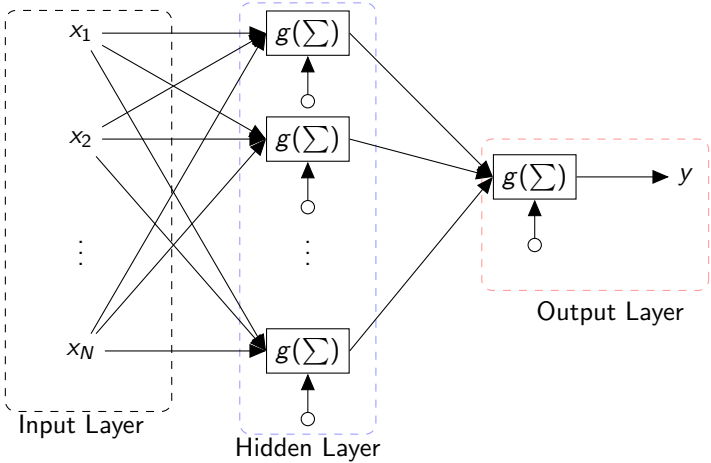
x_1	x_2	h_1	h_2	y
0	0	0	1	$u(-0.5) = 0$
1	0	1	1	$u(0.5) = 1$
0	1	1	1	$u(0.5) = 1$
1	1	1	0	$u(-0.5) = 0$

What is each layer doing?



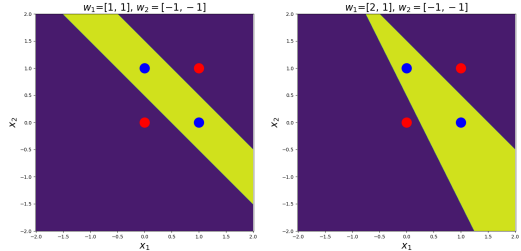
- First layer makes things linearly separable.
  - Premiere couche rend les choses sépérable
- Output layer finishes the job.
  - La couche de sortie finit le job.

Two layer feed-forward neural network



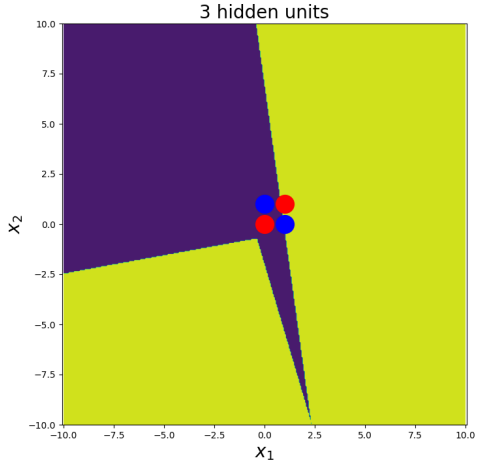
The hidden layer and decision boundary

- The decision boundary implied by network we have talked about
  - La borne de décision impliqué par le réseau dont on a parlé



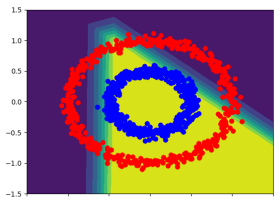
- Each hidden unit defines a line; the network combines these lines into the decision boundary.
  - Chaque unité cachée définit une droite; le réseau les combine pour former la frontière de décision.
- What if we have more than one hidden unit?
  - Et si on a plus qu'une seule unité caché?

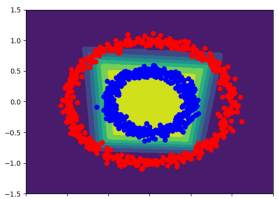
Three hidden units

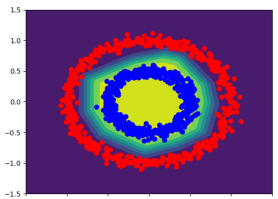


Classifying Circles

- With 2, 3 and 10 Hidden Units / Avec 2, 3 et 10 Unités Cachées







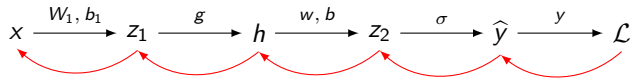
How do we learn the weights? The loss

- The networks on the circles were trained. How? / Les réseaux pour les cercles ont été entrainés. Comment?
- Two layer network: $h = g(W_1 x + b_1)$, $\widehat{y} = \sigma(w^\top h + b)$ / Réseau à deux couches
  - The output layer is logistic regression on the learned features h! / La couche de sortie est une régression logistique sur les caractéristiques apprises h!
- So we use the same loss as last week (cross-entropy) / Donc la même fonction de coût que la semaine passée:

$$\mathcal{L}(\theta) = \sum_n \left(-y_n \log \widehat{y}_n - (1 - y_n) \log(1 - \widehat{y}_n)\right), \qquad \theta = \{W_1, b_1, w, b\}$$

  - Multiclass: softmax output and $\mathcal{L} = -\sum_n \log \widehat{y}_{n,c_n}$ / Multiclasse: sortie softmax.
- Minimize with gradient descent: we need $\nabla_\theta \mathcal{L}$ for every weight. / On a besoin du gradient pour chaque poids.

Backpropagation



Forward: store $z_1, h, z_2, \widehat{y}$. Backward: one more chain rule factor at each step.
Passe avant: on garde $z_1, h, z_2, \widehat{y}$. Passe arrière: un facteur de plus à chaque pas.

$$\frac{\partial \mathcal{L}}{\partial z_2} = \frac{\partial \mathcal{L}}{\partial \widehat{y}} \frac{\partial \widehat{y}}{\partial z_2} = \widehat{y} - y \quad \text{(as in logistic regression)}$$

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial \widehat{y}} \frac{\partial \widehat{y}}{\partial z_2} \frac{\partial z_2}{\partial h} = (\widehat{y} - y)\, w$$

$$\frac{\partial \mathcal{L}}{\partial z_1} = \frac{\partial \mathcal{L}}{\partial \widehat{y}} \frac{\partial \widehat{y}}{\partial z_2} \frac{\partial z_2}{\partial h} \frac{\partial h}{\partial z_1} = (\widehat{y} - y)\, w \odot g'(z_1)$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \widehat{y}} \frac{\partial \widehat{y}}{\partial z_2} \frac{\partial z_2}{\partial h} \frac{\partial h}{\partial z_1} \frac{\partial z_1}{\partial W_1} = \big((\widehat{y} - y)\, w \odot g'(z_1)\big)\, x^\top$$

Each line reuses the previous one. / Chaque ligne réutilise la précédente.

Backpropagation for deep networks

- Layer l: $z_l = W_l h_{l-1} + b_l$, $h_l = g(z_l)$, with $h_0 = x$. / Couche l.
- Define $\delta_l = \partial \mathcal{L} / \partial z_l$. Backprop is a single recursion, from the last layer to the first: / La rétropropagation est une seule récursion, de la dernière couche à la première:

$$\delta_l = \left(W_{l+1}^\top \delta_{l+1}\right) \odot g'(z_l), \qquad \frac{\partial \mathcal{L}}{\partial W_l} = \delta_l \, h_{l-1}^\top, \qquad \frac{\partial \mathcal{L}}{\partial b_l} = \delta_l$$

- The backward pass costs about as much as the forward pass. / La passe arrière coûte à peu près autant que la passe avant.
- We never write this by hand: automatic differentiation does it (`loss.backward()` in PyTorch). / On ne l'écrit jamais à la main: la différentiation automatique le fait.

The training loop: mini-batch SGD

```
model = MLP(n_hidden=10)
opt = torch.optim.SGD(model.parameters(), lr=0.01)

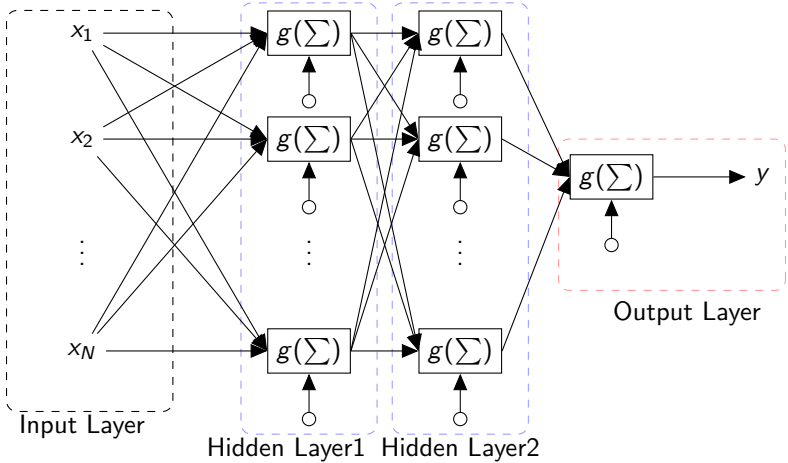
for epoch in range(n_epochs):
    for x, y in loader:           # mini-batches
        y_hat = model(x)          # forward pass
        loss = F.binary_cross_entropy(y_hat, y)
        opt.zero_grad()
        loss.backward()           # backprop
        opt.step()                # gradient step
```

- Each step uses a mini-batch of B examples, not the whole dataset. / Chaque pas utilise un mini-batch de B exemples.
- The mini-batch gradient is a noisy but cheap estimate of the full gradient. / Une estimation bruitée mais peu coûteuse du gradient.
- One epoch = one pass over the training data. / Une époque = un passage sur les données.

Going Deeper

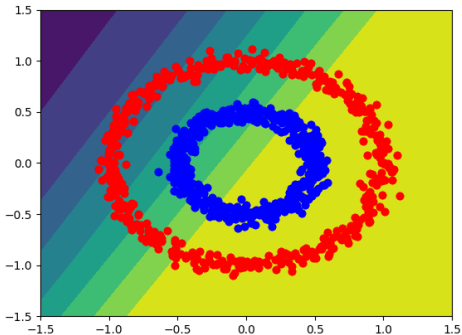
- As we talked about, the output layer learns a linear separator.
  - Comme on en a parlé, la couche de sortie apprend un sépateur linéaire.
- If the output layer is getting not-separable data, we can go deeper!
  - Si la couche de sortie prend du data non-séperable, on peut utiliser un réseau plus profond.

Three layer feed-forward neural network



So we can just go arbitrarily deep?

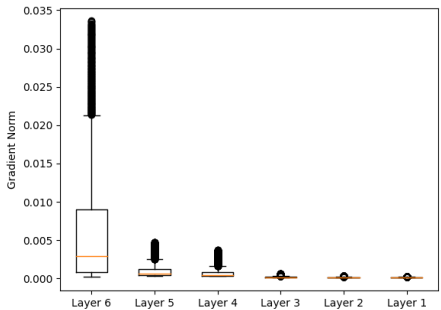
- No! Here's what you learn with 7 layers! / Non! voici ce qu'on apprend avec 7 couches:



- We basically can not learn anything beyond a linear classifier! Why?
  - On ne peut rien apprendre au-delà d'un classificateur linéaire. Pourquoi?

Vanishing gradients

- Here's the gradient distribution with respect to layers / La distribution de gradient par rapport aux couches



- We basically do not update the lower layers! / On ne mets pas les premières couches à jour!

Why do gradients vanish?

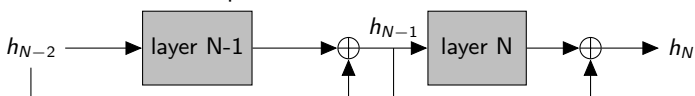
- Unroll the backprop recursion / Déroulons la récursion:

$$\delta_1 = \operatorname{diag}\big(g'(z_1)\big)\, W_2^\top \operatorname{diag}\big(g'(z_2)\big)\, W_3^\top \cdots \operatorname{diag}\big(g'(z_{L-1})\big)\, W_L^\top\, \delta_L$$

- A product of $L-1$ factors: if each one shrinks the gradient, the gradient vanishes exponentially with depth. / Un produit de $L-1$ facteurs: si chacun réduit le gradient, il disparaît exponentiellement avec la profondeur.
- Sigmoid: $\sigma'(z) = \sigma(z)(1-\sigma(z)) \leq 1/4$. With 7 layers: $(1/4)^6 \approx 2 \times 10^{-4}$. / Sigmoïde.
- Leaky ReLU (our 7 layer example): $g'(z) = 0.1$ for $z < 0$, so it also shrinks the gradient. / Leaky ReLU (notre exemple): réduit aussi le gradient.

Skip connections

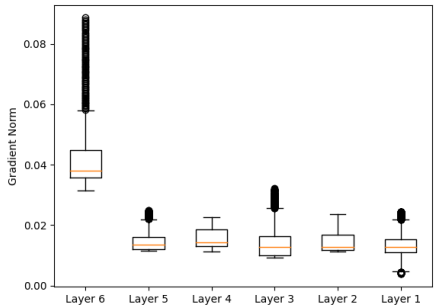
- What if we introduce skip connections?
  - On va introduire des connections qui saute des couches.



- These are known as residual layers also. / On appele ça des 'residual layers' aussi.

Mitigating vanishing gradients

- Here's the gradient distribution with respect to layers with skip connections / La distribution de gradient par rapport aux couches avec des connections qui sautent



- Much better! / Vraiment meilleure!

Why skip connections help

- Residual layer / Couche résiduelle: $h_l = h_{l-1} + f_l(h_{l-1})$
- Its Jacobian / Sa jacobienne:

$$\frac{\partial h_l}{\partial h_{l-1}} = I + \frac{\partial f_l(h_{l-1})}{\partial h_{l-1}}$$

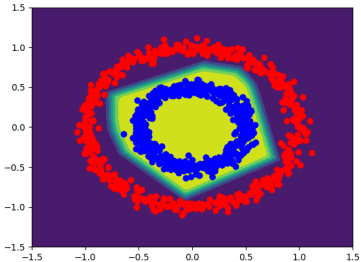
- Backprop through L residual layers / À travers L couches résiduelles:

$$\frac{\partial \mathcal{L}}{\partial h_0} = \left(I + \frac{\partial f_1}{\partial h_0}\right)^\top \left(I + \frac{\partial f_2}{\partial h_1}\right)^\top \cdots \left(I + \frac{\partial f_L}{\partial h_{L-1}}\right)^\top \frac{\partial \mathcal{L}}{\partial h_L}$$

  - Expanding the product, one term is just $\partial \mathcal{L}/\partial h_L$: the gradient reaches the first layers directly. / Un des termes est simplement $\partial \mathcal{L}/\partial h_L$: le gradient arrive directement aux premières couches.

Residual layers in action

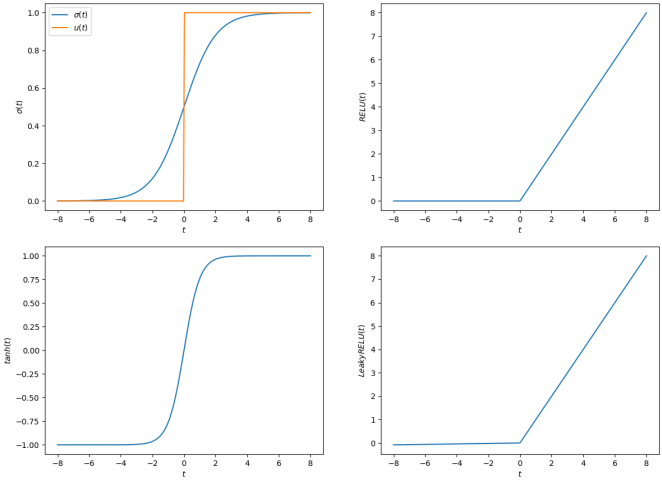
- We use a 7 layer residual feed-forward network / On emploie un réseau residual à 7 couches



- Exact same network as before, but I introduced the skip connections. 6 layers with 3 hidden units, and an output layer with leaky relu activation.
  - Le meme réseau que avant, mais j'ai introduit des connections qui saute. 6 couches avec 3 unités cachés chaqu'un, et on utilise des leaky relu non-linéarité.

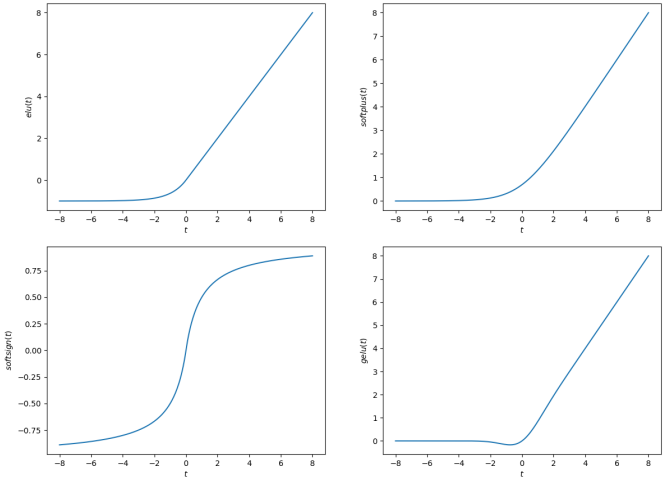
Nonlinearity Functions

- Which one to choose? / Lequel doit-on choisir?

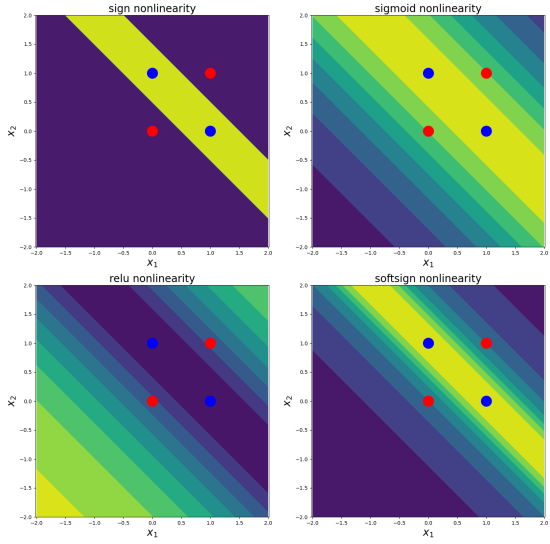


Activation Functions

- Which one to choose? / Lequel doit-on choisir?



Effect on the XOR problem



Which one to choose?

- The effect is mainly on optimization.
  - L'effet est principalement sur l'optimization.
- The saturating non-linearities can make it so that you get stuck during optimization!
  - Les non-linéarités qui saturisent fait en sorte qu'on est bouché pendant l'optimization!
- Overall there is no definitive answer, except some cases. (e.g. inputting negative numbers to log)

Do you think we can do the same thing here?

- Do you think dot products would work here? / Pensez-vous que le produit scalaire fonctionnerait ici?

Query

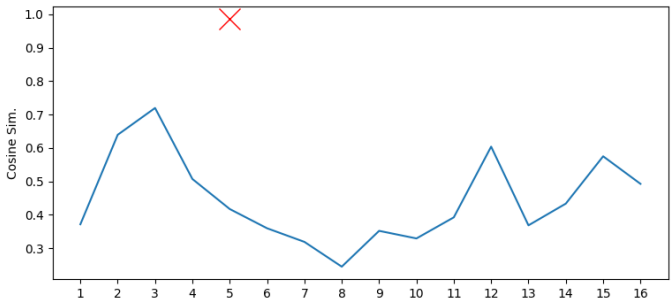


Classes



No!

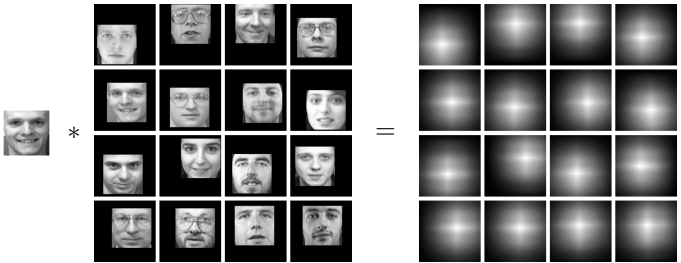
- Inner products / Produits scalaires

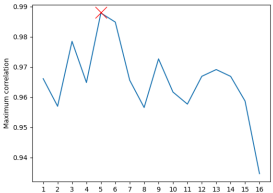


- No clear signal, what can we do? / Pas de signal claire, qu'est-ce qu'on peut faire?

Convolutions! (well technically not)

- Correlations!





Correlation operation

- The operation / L'opération:

$$y(t) = x * h = \sum_{t'} x(t + t')h(t')$$

- Note that this is technically 'correlation' in the signal processing, not convolution. But ML people call it convolution.
  - Notez qu'en traitement de signal, on appelle cette opération 'corrélation'. Mais les gens dans ML appelle ça la convolution.

More than just translation

- Now let's say we want to have a model to detect the same guy in images like this. / Disons qu'on veut détecter la meme personne dans des images variés comme les suivantes.



- Note that we need to be able to do correlations with multiple filters! (to account for rotations, scaling, and more)
  - Notez qu'on a besoin d'effectuer des correlations avec plusieurs filtres afin de prendre en compte les rotations, de plusiers échelles, et plus.

Convolutional Layers to the rescue

- Augment the operation / Augmentons l'opération

$$y(t,c) = x * h(c) = \sum_{t'} x(t+t')h(t',c)$$

- The 2D version:

$$y(i,j,c) = x * h(c) = \sum_{i',j'} x(i+i',j+j')h(i',j',c)$$

- The Multichannel version

$$y(t,c_2) = \sum_{c_1} x(c_1) * h(c_1,c_2) = \sum_{t',c_1} x(t+t',c_1)h(t',c_1,c_2)$$

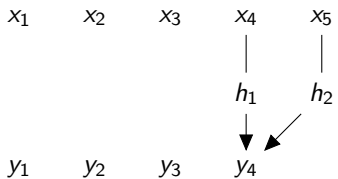
- Convolutions can be strided also! Les convolutions peut-etre 'strided' aussi.

Why convolutional layers?

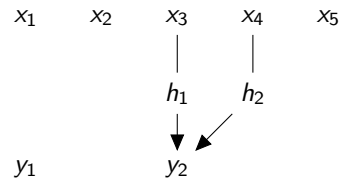
- Locality: each output looks at a small window of the input. / Localité: chaque sortie regarde une petite fenêtre de l'entrée.
- Weight sharing: the same filter is applied at every position. / Partage de poids: le même filtre à chaque position.
- Translation equivariance: shift the input, and the output shifts the same way. / Équivariance par translation: si on décale l'entrée, la sortie est décalée pareil.
- Number of weights for 1 second of 16 kHz audio / Nombre de poids pour 1 seconde d'audio à 16 kHz:
  - Dense layer, 16000 inputs $\rightarrow$ 16000 outputs: $16000^2 \approx 256$M weights / Couche dense
  - Conv layer, 64 filters of 400 taps (25 ms): $64 \times 400 = 25.6$k weights, and it outputs 64 channels at every position. / Couche convolutionnelle

Strided convolutions

- Stride is the hop size we saw earlier for STFT / Stride est le 'hop size' qu'on a vu avant
- Stride = 1



- Stride = 2



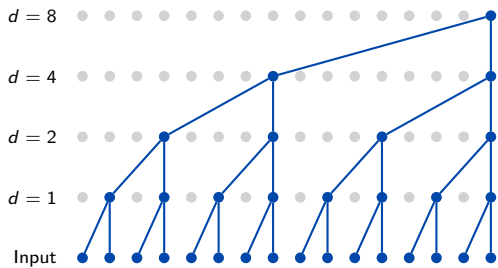
- This enable to subsample and convolve at the same time. / On sous-echantillone et convolue simultanement.

Dilated convolutions

- Dilation d: the filter skips $d-1$ samples between taps. / Le filtre saute $d-1$ échantillons.

$$y(t) = \sum_{t'} x(t - d\,t')\,h(t')$$

- Doubling d at each layer: the receptive field grows exponentially with depth. / Le champ récepteur grandit exponentiellement avec la profondeur.
- Receptive field / Champ récepteur: $1+\sum_l (k-1)\,d_l = 1+1+2+4+8 = 16$ here $(k=2)$. Used in WaveNet. / Utilisé dans WaveNet.



Batch Norm

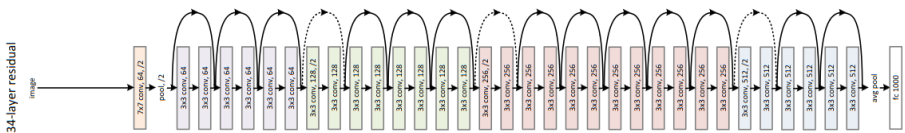
- The operation / L'opération:

$$BN(x) = \frac{x - \mathbb{E}[x]}{\sqrt{\text{var}[x] + \epsilon}}\gamma + \beta$$

- $x, \gamma, \beta \in \mathbb{R}^L$. γ, β are learnt on training data / sont appris pendant l'entrainement. ϵ is a small fixed constant / est une petite constante fixe.
- During training, $\mathbb{E}[x], \text{var}[x]$ are computed over the mini-batch; at test time we use running averages from training. / Pendant l'entrainement, on les calcule sur le mini-batch; au test, on utilise des moyennes mobiles.

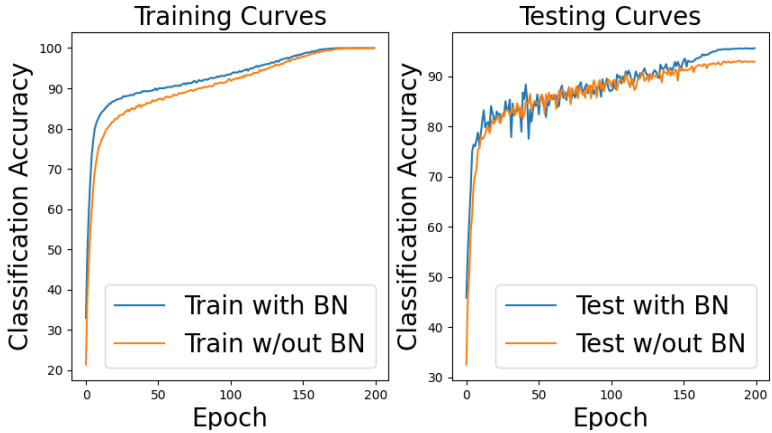
With and without batchnorm

- The architecture (Resnet18)



- We will test the performance with and without batchnorm on CIFAR10. / On va tester la performance avec et sans batchnorm sur CIFAR10.

With and without batchnorm



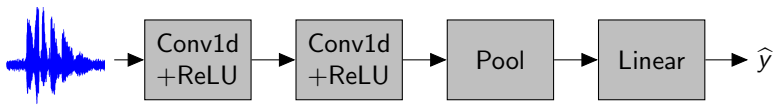
It's slightly better with BN! / C'est un peu mieux avec BN!

Convolutional Nets for Classifying Audio

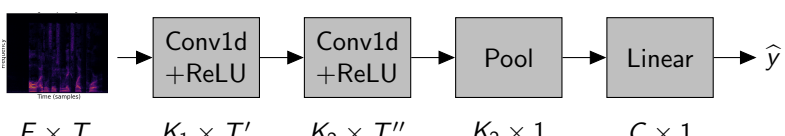
- How can we use Convolutional Nets for Classifying Audio? / Comment peut-on utiliser des réseaux convolutionnels pour classifier audio?
- Two options: Time domain / Time-Frequency Domain
  - Deux options / Domaine Temporelle / Domaine Fréq-Temporelle
- Let's explore both! / Éxplorons les deux!

Audio Classification Pipeline

- Time domain classification pipeline / Le pipeline de classification dans le domaine de temps

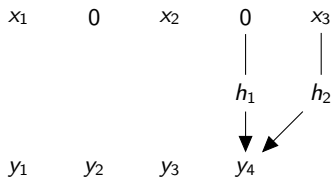


- Time-Frequency domain classification pipeline / Le pipeline de classification dans le domain frequence-et-temps



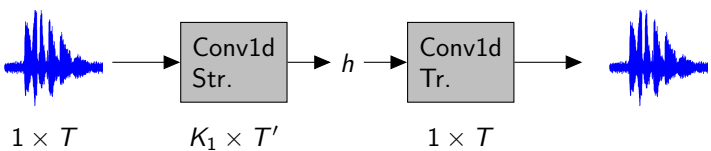
Transposed Convolution

- An upsampling operation / Une opération de suréchantillonnage.
- Very similar to the upsampling in DSP: Insert zeros, then filter (convolve with stride=1):
  - Comme le DSP classique on insére des zéros, puis filtrer avec un stride=1.

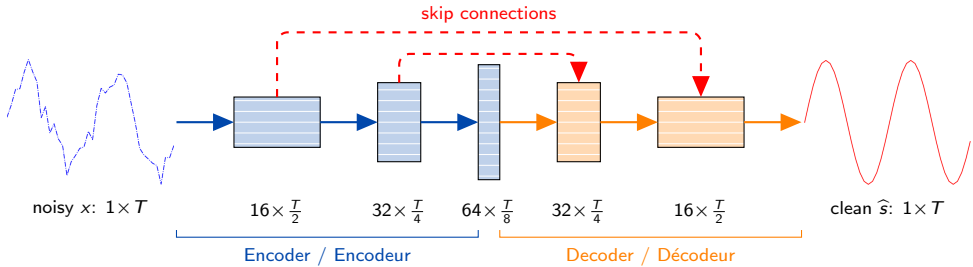


Transposed Convolution Application: Autoencoder

- We can subsample with strided conv layers, and then upsample with transposed-conv layers. / On peut sousechantillonner avec des couches strided-conv, et puis augmenter la résolution avec des couches transposed-conv.



Application: denoising with a convolutional encoder-decoder



- Encoder: Conv1d with stride 2 halves the length. Decoder: ConvTranspose1d doubles it.
  - Encodeur: Conv1d divise la longueur par 2. Décodeur: ConvTranspose1d la double.
- Skip connections (as in U-Net) carry fine details from the encoder to the decoder.
  - Les connexions résiduelles (comme U-Net) passent les détails fins de l'encodeur au décodeur.
- Train on (noisy, clean) pairs: $\mathcal{L} = \|\widehat{s} - s\|_1$. Used in SEGAN (2017), Demucs (2020).
  - Entraînement sur des paires (bruité, propre).

The denoising encoder-decoder in PyTorch

```
kw = dict(kernel_size=4, stride=2, padding=1)

class DenoiseNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.e1 = nn.Conv1d(1, 16, **kw)
        self.e2 = nn.Conv1d(16, 32, **kw)
        self.e3 = nn.Conv1d(32, 64, **kw)
        self.d3 = nn.ConvTranspose1d(64, 32, **kw)
        self.d2 = nn.ConvTranspose1d(32, 16, **kw)
        self.d1 = nn.ConvTranspose1d(16, 1, **kw)

    def forward(self, x):                # (B, 1, T)
        h1 = F.relu(self.e1(x))          # (B, 16, T/2)
        h2 = F.relu(self.e2(h1))         # (B, 32, T/4)
        h3 = F.relu(self.e3(h2))         # (B, 64, T/8)
        g2 = F.relu(self.d3(h3)) + h2    # (B, 32, T/4)
        g1 = F.relu(self.d2(g2)) + h1    # (B, 16, T/2)
        return self.d1(g1)               # (B, 1, T)

loss = F.l1_loss(model(x_noisy), s)
```

- Output length with k=4, s=2, p=1: / Longueur de sortie:
  - Conv1d: $\lfloor (T + 2p - k)/s \rfloor + 1 = T/2$
  - ConvTranspose1d: $(T - 1)s - 2p + k = 2T$
- T must be divisible by 8, so the skip shapes match. / T divisible par 8.
- No ReLU at the end: the signal can be negative. / Pas de ReLU à la fin: le signal peut être négatif.

Convolution Groups

- Grouping decides how to map M input dimensions to N output dimensions using K filters. / Les groupes décident comment mapper les M dimensions d'input à N dimensions d'output en utilisant K filtres.

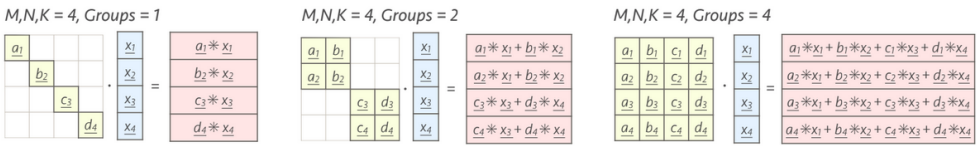
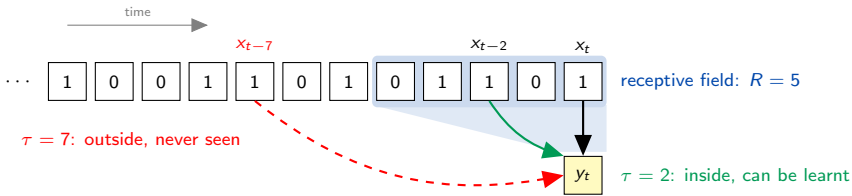


image taken from uiuc mlsp class

- In PyTorch, the default `groups=1` mixes all input channels; `groups` = number of input channels keeps each channel separate (depthwise). The figure counts channels per group.
  - Dans PyTorch, la valeur par défaut `groups=1` mélange tous les canaux; `groups` = nombre de canaux d'entrée les traite séparément (depthwise). La figure compte les canaux par groupe.

Using Convolutional Nets as TDNN

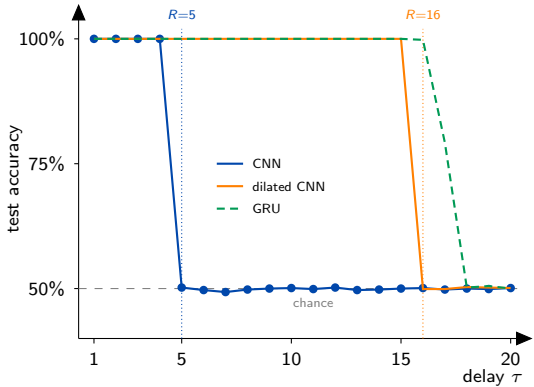
- Let's say we work with bit sequences. / Disons qu'on travaille avec des séquences binaires.
- XOR again: $y_t = XOR(x_t, x_{t-\tau})$. CNNs will have finite attention span. / Les CNNs regardent une fenêtre fixe.



- If $\tau \geq R$, CNNs will fail, no matter what! The output does not depend on what they see: 50% at best.
  - Si $\tau \geq R$, les CNNs vont échouer: la sortie ne dépend pas de ce qu'ils voient. 50% au mieux.

XOR with a delay: an experiment

- One model per delay τ, tested on new sequences. / Un modèle par délai τ.
- CNN (4 layers, kernel 2): $R = 5$. Chance from $\tau = 5$. / Hasard dès $\tau = 5$.
- Dilated CNN (1, 2, 4, 8), same number of parameters: $R = 16$. Chance from $\tau = 16$. / Mêmes paramètres: hasard dès $\tau = 16$.
- GRU: no fixed window, but it must learn to keep τ bits in memory. Fails from $\tau \approx 17$: long memories are hard to learn (next slides). / Pas de fenêtre fixe, mais une longue mémoire est difficile à apprendre.



Recurrent Neural Nets

- The vanilla RNN Idea:

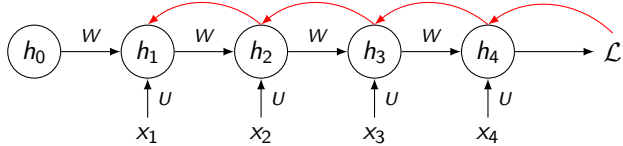
$$h_t = g(Wh_{t-1} + Ux_t)$$

- $h_t \in \mathbb{R}^K$ defines the 'state'. / définit l'état. $W \in \mathbb{R}^{K \times K}$.
- $x_t \in \mathbb{R}^L$ is in the input / est l'entrée. $U \in \mathbb{R}^{K \times L}$.
- We define the temporal dependencies with a recursion, so we have infinite attention span (in theory) / En théorie on peut modéliser des dépendes à l'infinie.
- There will be a vanishing gradient problem. / On va avoir un probleme de gradients qui se disparaient.

$$\begin{aligned} h_t =& g(Wg(Wh_{t-2} + Ux_{t-1}) + Ux_t) \\ =& g(Wg(Wg(Wh_{t-3} + Ux_{t-2}) + Ux_{t-1}) + Ux_t) \\ =& g(Wg(Wg(W(\dots g(Wh_0 + Ux_1)\dots) + Ux_{t-2}) + Ux_{t-1}) + Ux_t) \end{aligned}$$

- This will definitely suffer from that for long term dependencies. / Ça va définitivement avoir des problèmes pour des dépendences longue termes.

Backpropagation through time



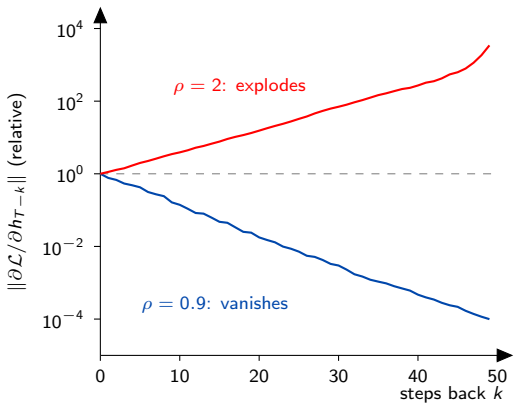
$h_t = g(z_t)$, $z_t = Wh_{t-1} + Ux_t$. Loss on the last state. Write $D_t = \mathrm{diag}(g'(z_t))$.
Perte sur le dernier état. On note $D_t = \mathrm{diag}(g'(z_t))$.

$$\partial\mathcal{L}/\partial h_3 = W^\top D_4 \, \partial\mathcal{L}/\partial h_4$$
$$\partial\mathcal{L}/\partial h_2 = W^\top D_3 \, W^\top D_4 \, \partial\mathcal{L}/\partial h_4$$
$$\partial\mathcal{L}/\partial h_1 = W^\top D_2 \, W^\top D_3 \, W^\top D_4 \, \partial\mathcal{L}/\partial h_4$$
$$\partial\mathcal{L}/\partial W = \textstyle\sum_{t=1}^{4} D_t \left(W^\top D_{t+1} \cdots W^\top D_4 \, \partial\mathcal{L}/\partial h_4\right) h_{t-1}^\top$$

k steps back: k factors $W^\top D_t$, with the same W every time. / k pas en arrière: k facteurs, toujours la même W.
A sum over t, but each term contains $\partial\mathcal{L}/\partial h_t$ (red): a product of $4-t$ factors. The terms from far back vanish or explode. / Chaque terme contient un produit de $4-t$ facteurs.

Vanishing or exploding gradients

- Scalar case: $h_t = w\, h_{t-1}$ gives $\frac{\partial h_T}{\partial h_{T-k}} = w^k$. $0.9^{50} \approx 0.005$, $1.1^{50} \approx 117$. / Cas scalaire.
- Matrix case: $(W^\top)^k$ grows or shrinks with the spectral radius ρ of W (largest $|\lambda|$). $g' \leq 1$ shrinks it more. / Dépend du rayon spectral ρ de W.
- Measured on a tanh RNN, 64 units. / Mesuré sur un RNN tanh.
- Exploding: clip the gradient norm. Vanishing: change the architecture. / Explosion: on coupe la norme du gradient. Disparition: on change l'architecture.



RNNs with gates

- Minimal gated unit:

$$
\begin{aligned}
f_t &= \sigma(W_f h_{t-1} + U_f x_t) \\
h_t &= f_t \odot h_{t-1} + (1 - f_t) \odot \tanh(W_h h_{t-1} + U_h x_t)
\end{aligned}
$$

- This allows better gradient flow. / Ça permet mieux débit des gradients.

GRU / LSTM

- GRU:

$$
\begin{aligned}
f_t &= \sigma(W_f h_{t-1} + U_f x_t) \\
r_t &= \sigma(W_r h_{t-1} + U_r x_t) \\
h_t &= f_t \odot h_{t-1} + (1 - f_t) \odot \tanh(W_h(r_t \odot h_{t-1}) + U_h x_t)
\end{aligned}
$$

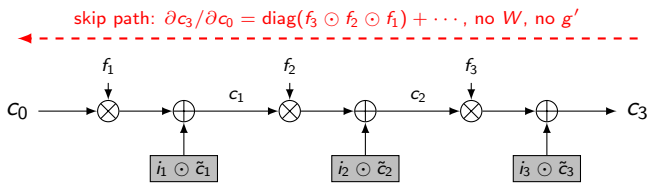
- LSTM: (More gates!)

$$
\begin{aligned}
f_t &= \sigma(W_f h_{t-1} + U_f x_t) \\
i_t &= \sigma(W_i h_{t-1} + U_i x_t) \\
o_t &= \sigma(W_o h_{t-1} + U_o x_t) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_c h_{t-1} + U_c x_t) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Gates are skip connections through time

ResNet (depth):	$h_l = h_{l-1} + F_l(h_{l-1})$	$\partial h_l / \partial h_{l-1} = I + \cdots$
GRU (time):	$h_t = f_t \odot h_{t-1} + (1 - f_t) \odot \tilde{h}_t$	$\partial h_t / \partial h_{t-1} = \mathrm{diag}(f_t) + \cdots$
LSTM (time):	$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$	$\partial c_t / \partial c_{t-1} = \mathrm{diag}(f_t) + \cdots$

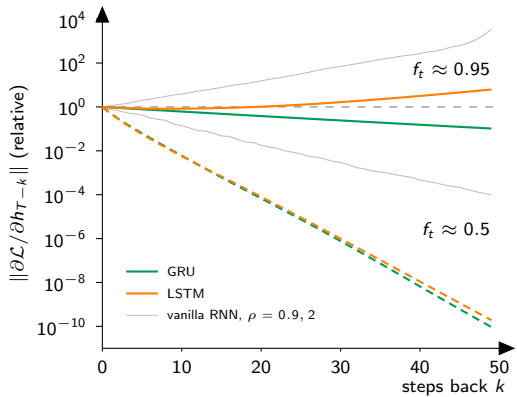
$\tilde{h}_t, \tilde{c}_t$: the tanh terms of the previous slide. The $\cdots$ are the paths through W. / Les $\cdots$ passent par W.



- Along the skip path the gradient is only multiplied by the gates: if $f_t \approx 1$, it passes unchanged. / Le gradient n'est multiplié que par les portes.
- Like ResNet, but the network learns how much to skip (LSTM: "constant error carousel", Hochreiter & Schmidhuber, 1997). / Comme ResNet, mais le réseau apprend combien sauter.

Gradients in GRU and LSTM

- Same measurement, 64 units. LSTM: gradient w.r.t. the cell state c_t. / Même mesure.
- Default init, gate bias 0: $f_t \approx 0.5$. The gradient still vanishes! / Biais 0: le gradient disparaît quand même!
- Gate bias 3: $f_t \approx 0.95$. The skip path is open: the gradient survives 50 steps. / Biais 3: le chemin direct est ouvert.
- In practice: initialize the forget-gate bias to 1 or more (Jozefowicz et al., 2015). The gates are learnt, so the network can keep $f_t \approx 1$ where it needs memory. / En pratique: biais de la porte d'oubli ≥ 1.



So CNN or RNN?

- It really depends, usually they give similar results. / Ça depend. En générale les résultats sont similaires.
- CNNs are typically easier to parallelize. / Les CNNs sont typiquement plus facile à paralleliser.

Self-attention

- Self-attention is the basis of the transformer architecture which obtains state-of-the art results in several domains such as NLP, computer vision, speech recognition. / Self-attention est base de l'architecture transformer qui obtient SOTA dans plusieurs domains.

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

$$Q = XW^Q,\ K = XW^K,\ V = XW^V$$

v_2		0	0	1	0	v_0
$\frac{v_2}{2} + \frac{v_3}{2}$		0	0	0.5	0.5	v_1
v_1	=	0	1	0	0	v_2
v_0		1	0	0	0	v_3

Self-attention does not know the order

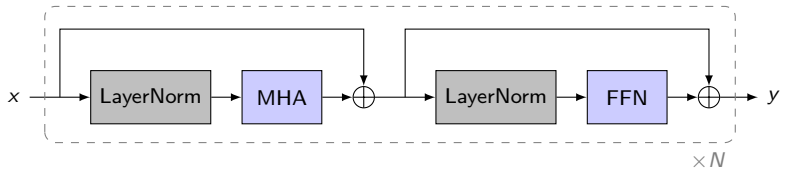
- Permute the rows of X (the time steps): the output rows are permuted the same way. Self-attention is permutation equivariant. / Si on permute les lignes de X, les sorties sont permutées de la même façon.
- Fix: add a positional encoding p_t to each time step / Solution: ajouter un encodage de position:

$$\tilde{x}_t = x_t + p_t, \qquad p_{t,2i} = \sin\left(\frac{t}{10000^{2i/d}}\right), \quad p_{t,2i+1} = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

- Multi-head attention: H attentions in parallel, each with its own W_h^Q, W_h^K, W_h^V, then concatenate / H attentions en parallèle, puis concaténer:

$$\text{MHA}(X) = [\text{head}_1, \ldots, \text{head}_H]\, W^O, \qquad \text{head}_h = \text{Attention}(XW_h^Q, XW_h^K, XW_h^V)$$

The transformer block



- MHA: multi-head self-attention. FFN: two linear layers with a ReLU, applied to each time step. / FFN: deux couches linéaires avec ReLU, à chaque pas de temps.
- Skip connections and normalization, like for deep MLPs! LayerNorm normalizes each time step over its features, not over the batch. / Connexions résiduelles et normalisation. LayerNorm normalise chaque pas de temps sur ses caractéristiques.
- Stack N blocks. (The original paper puts LayerNorm after the addition.) / On empile N blocs.
- SepFormer uses these blocks (next slides). / SepFormer utilise ces blocs.

How to apply Self-Attention on Source Separation?

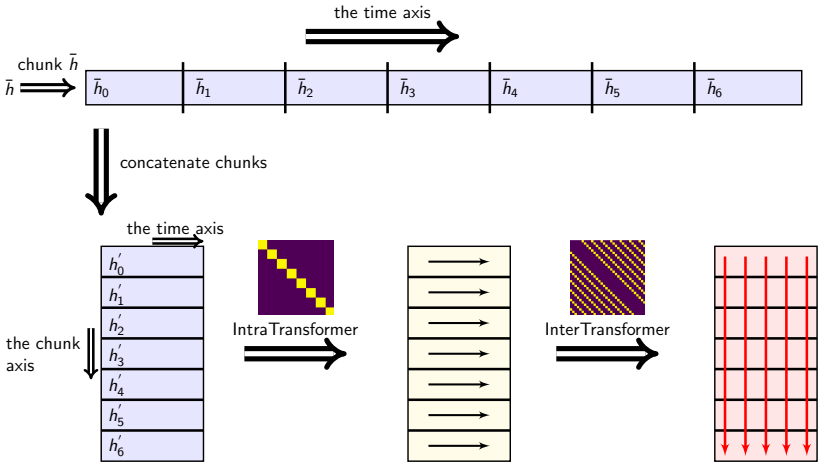
- It is not straightforward how to apply self-attention on long sequences. / C'est pas facile appliquer self-attention directement sur des séquences longues.
- Self-attention has a quadratic complexity with respect to inputs size. / Self-attention a une complexité quadratique par rapport à la taille de l'entrée.

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

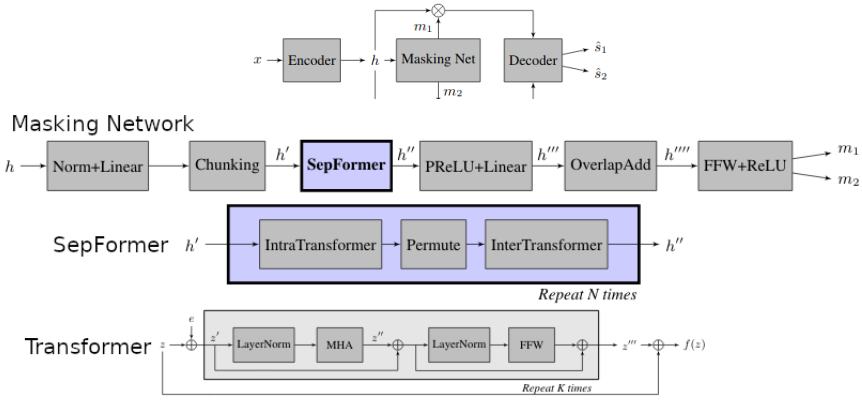
$$Q = XW^Q,\ K = XW^K,\ V = XW^V$$

- Can we find a way to efficiently model real world signals? / Peut-on trouver une façon pour modeliser des séquences longues?
- We propose **SepFormer**! / Nous avons proposé SepFormer qui a obtenu des résultats SOTA dans speech separation.

The Dual-Path Transformer Pipeline



The whole network

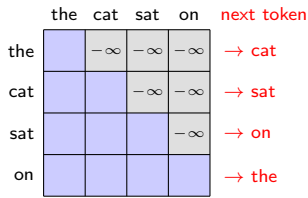


Large language models: causal self-attention

- An LLM: a stack of transformer blocks on text tokens. / Une pile de blocs transformer sur des tokens.
- Causal mask: token t only attends to tokens $\leq t$. / Masque causal.

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right) V, \qquad M_{ij} = -\infty \text{ if } j > i$$

- Training: predict the next token / Prédire le prochain token: $\mathcal{L} = -\sum_t \log p(x_{t+1} \mid x_{\leq t})$
- Generation: sample a token, append it, repeat. / Échantillonner, ajouter, répéter.
- GPT-3: 96 blocks, 175B parameters. Same recipe on audio tokens: AudioLM. / Même recette pour l'audio.



A note on optimization

- Neural Network Optimization is highly non-convex. / Optimization des réseaux de neurons sont hautement non-convex.
- A convex optimization problem

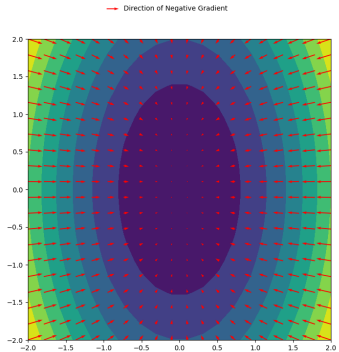
$$\begin{aligned} &\min_x f(x) \\ &s.t. g_i(x) \leq 0,\ i \in 1, \ldots, m \\ &\quad h_j(x) = 0\ j \in 1, \ldots, p \end{aligned}$$

  - $f(x)$ needs to be convex
  - $g_i(x)$ needs to be convex
  - $h_j(x)$ needs to be affine.
- In NNs $f(x)$ is highly non-convex!
- Finding globally optimal solution is not guaranteed, harder optimization problem.

Gradient descent

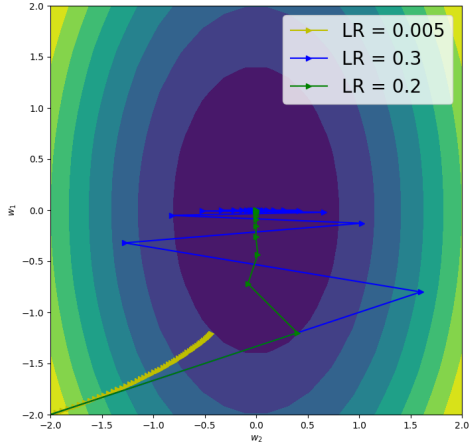
- The basic updates

$$\underbrace{w_t}_{\text{parameters}} = w_{t-1} - \underbrace{\eta}_{\text{learning rate}} \underbrace{\nabla \mathcal{L}(w)}_{\text{the gradient}}$$



The effect of learning rate

- $\mathcal{L}(w) = 3w_1^2 + w_2^2$

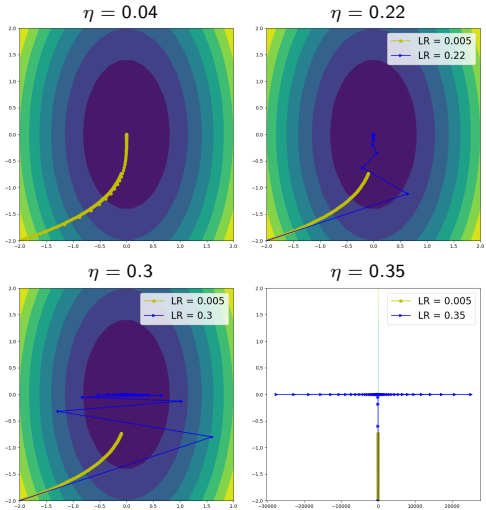


Learning rate sweep

- $\mathcal{L}(w) = 3w_1^2 + w_2^2$, Hessian $= \mathrm{diag}(6, 2)$
- Gradient step on w_1 / Pas de gradient sur w_1:

$$w_1 \leftarrow w_1 - 6\eta\, w_1 = (1 - 6\eta)\, w_1$$

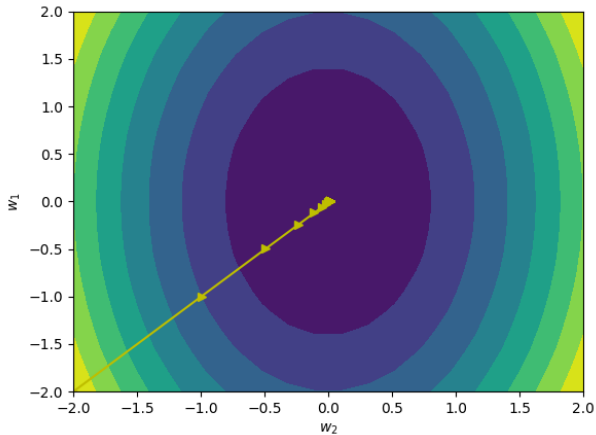
- Converges iff $|1 - 6\eta| < 1$, i.e. $\eta < 2/\lambda_{\max} = 1/3$ / Converge si et seulement si $\eta < 1/3$
  - $\eta > 1/6$: oscillates / oscille
  - $\eta > 1/3$: diverges / diverge



Momentum

- Momentum, low pass filters gradients

$$m_t = \eta \nabla \mathcal{L}(w) + \mu m_{t-1}$$
$$w_{t+1} = w_t - m_t$$



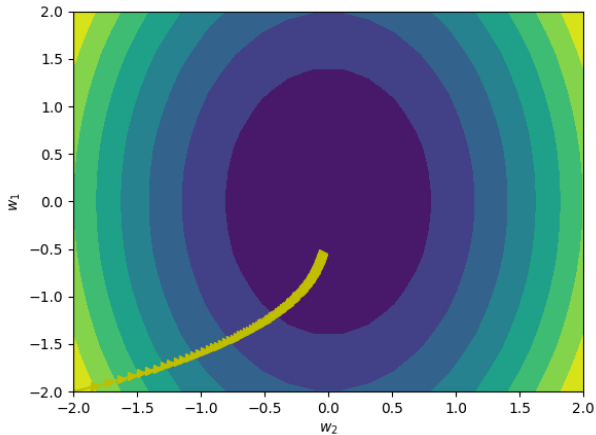
Per Parameter Updates

- Adagrad: The idea is to penalize large updates / L'idée est de pénaliser les mises à jours grands.

$$G_{t+1} = G_t + \nabla \mathcal{L}(w_t)^2$$

$$w_{t+1} = w_t - \eta \frac{\nabla \mathcal{L}(w_t)}{\sqrt{G_{t+1}} + \epsilon}$$

- Slows down too much! / Ça relantit trop!



More embellished versions

RMSProp

$$g_{t+1} = \gamma g_t + (1-\gamma)\nabla\mathcal{L}(w)^2$$
$$w_{t+1} = w_t - \eta\frac{\nabla\mathcal{L}(w)}{\sqrt{g_{t+1}}}$$

Note the per parameter update, large updates are penalized. / Notez qu'on des mises à jours differents pour chaque parametres afin de pénaliser les grands changements.

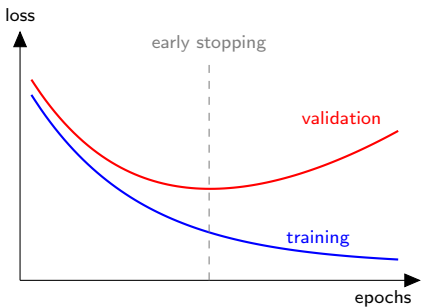
Adam

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla\mathcal{L}(w)$$
$$g_{t+1} = \beta_2 g_t + (1-\beta_2)\nabla\mathcal{L}(w)^2$$
$$w_{t+1} = w_t - \eta\frac{m_{t+1}}{\sqrt{g_{t+1}}}$$

Same as RMSProp but uses momentum smoothing. / La meme chose que RMS Prop mais on utilise momentum smoothing.

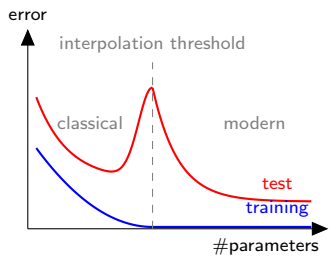
Overfitting in neural networks

- Neural networks often have more parameters than training examples. / Souvent plus de paramètres que d'exemples d'entrainement.
- They can memorize the training set: the training loss keeps going down, the validation loss goes up. / Ils peuvent mémoriser l'ensemble d'entrainement.
- Always keep a validation set and monitor it during training. / Gardez toujours un ensemble de validation.
- Early stopping: keep the weights with the best validation loss. / Arrêt précoce: garder les poids avec le meilleur coût de validation.



Double descent

- Classical view: test error is U-shaped in model size. / Vue classique: l'erreur de test a une forme en U.
- Deep nets: past the interpolation threshold (#params $\approx$ #examples), test error goes down again. / Après le seuil d'interpolation, l'erreur redescend.
- At the threshold, essentially one model fits the data exactly, noise included. Bigger models have many solutions, and SGD finds a simple one. / Plus grand, SGD trouve une solution simple.
- Also over epochs: early stopping is not always optimal. / Aussi au fil des époques.
  - Belkin et al. 2019, Nakkiran et al. 2020



Regularization

- Weight decay: penalize large weights, $\mathcal{L}(\theta) + \lambda \|\theta\|^2$. / Pénaliser les grands poids.
- Dropout: during training, set each hidden unit to zero with probability p (and scale the others by $1/(1-p)$); no dropout at test time. / Pendant l'entrainement, mettre chaque unité cachée à zéro avec probabilité p; pas de dropout au test.
  - The network cannot rely on any single unit. / Le réseau ne peut pas dépendre d'une seule unité.
- Data augmentation: new training examples from transformations that keep the label. / Augmentation de données: transformations qui gardent l'étiquette.
  - Audio: add noise, time shift, speed perturbation, SpecAugment (mask bands of time and frequency in the spectrogram). / Audio: bruit, décalage, vitesse, SpecAugment.
- Early stopping and batch norm also act as regularizers. / L'arrêt précoce et batch norm régularisent aussi.

Recap

- What changed since the 90s? / Qu'est-ce qui a changé depuis les 90s?
- Cynical answer: Not much, we just have GPUs now and some bells and whistles. / Réponse cynique: Pas beaucoup.
- More thoughtful and calm answer: We know how to make things work now, many tricks. (plus compute) / On connait plus des astuces maintenant qui fait les choses fonctionner.
- Today we talked about how to make MLPs work, modern conv nets, RNNs, Self-attention. / On a parlé des MLPs, convnets, RNNs, self-attention.

Reading material

- Convolution arithmetics: `https://github.com/vdumoulin/conv_arithmetic`
- Wavenet: `https://arxiv.org/pdf/1609.03499.pdf`
- Transformer: `https://arxiv.org/abs/1706.03762`
- Resnet: `https://arxiv.org/pdf/1512.03385.pdf`
- Deep double descent: `https://arxiv.org/abs/1912.02292`

Next week

- Clustering.