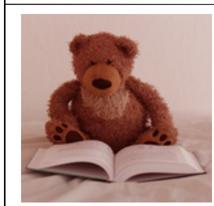
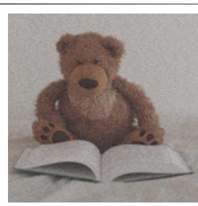
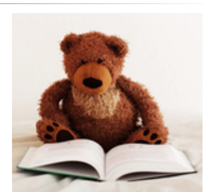
Data processing

r **Data augmentation** – Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

Original	Flip	Rotation	Randomcrop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB isslightlychanged -Capturesnoise thatcanoccur withlightexposure	-Additionofnoise -Moretolerancetoignored qualityvariationof inputs	-Partsofimage -Mimicspotential lossofpartsofimage	-Luminositychanges -Controlsdifference inexpositiondue totimeofday

r **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch {xi }. Bynotingμ2B,σ B the mean and variance of that we want to correct to the batch, it is done as follows:

← –
√xi−μBxiy+βσ2B+ε

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Training a neural network

r **Epoch** – In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

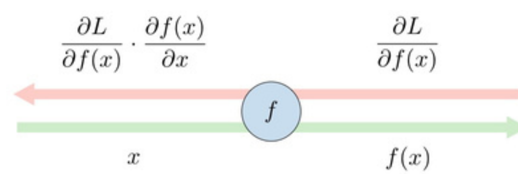
r **Mini-batch gradient descent** – During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

r **Loss function** – In order to quantify how a given model performs, the loss function L is usually used to evaluate to what extent the actual outputs y are correctly predicted by the model outputs z.

r **Cross-entropy loss** – In the context of binary classification in neural networks, the cross-entropy loss L(z,y) is commonly used

[andisdefinedasfollows:]L(z,y)=−ylog(z)+(1−y)log(1−z)

r **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight w is computed using the chain rule.



Using this method, each weight is updated with the rule:

∂ L(z ,y)
w

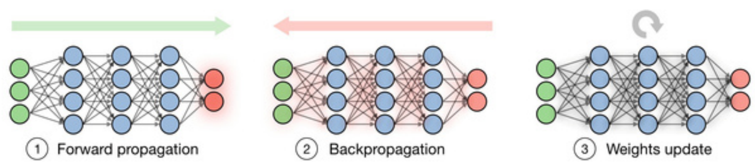
←−w−α∂w

r **Updating weights** – In a neural network, weights are updated as follows:

• Step 1: Take a batch of training data and perform forward propagation to compute the loss.

• Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.

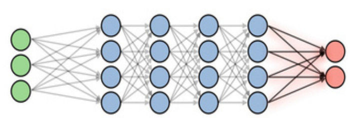
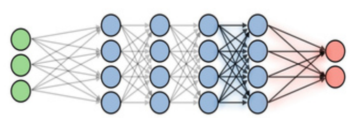
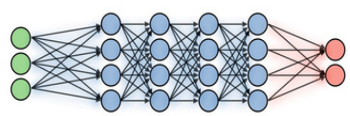
• Step 3: Use the gradients to update the weights of the network.



Parameter tuning

r **Xavier initialization** – Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.

r **Transfer learning** – Training a deep learning model requires a lot of data and more importantly a lot of time. It is often useful to take advantage of pre-trained weights on huge datasets that took days/weeks to train, and leverage it towards our use case. Depending on how much data we have at hand, here are the different ways to leverage this:

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

r **Learning rate** – The learning rate, often noted α or sometimes η, indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

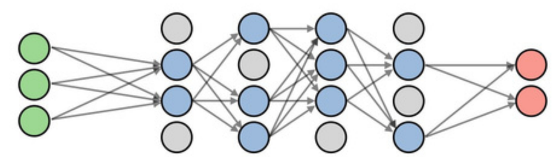
r **Adaptive learning rates** – Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful. They are summed up in the table below:

Method	Explanation	Update of w	Update of b
Momentum	- Dampens oscillations - Improvement to SGD - 2 parameters to tune	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	- Root Mean Square propagation - Speeds up learning algorithm by controlling oscillations	$w \longleftarrow w - \alpha\frac{dw}{\sqrt{s_{dw}}}$	$b \longleftarrow b - \alpha\frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - Most popular method - 4 parameters to tune	$w \longleftarrow w - \alpha\frac{v_{dw}}{\sqrt{s_{dw}}+\epsilon}$	$b \longleftarrow b - \alpha\frac{v_{db}}{\sqrt{s_{db}}+\epsilon}$

Remark: other methods include Adadelta, Adagrad and SGD.

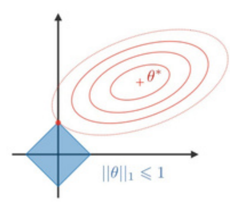
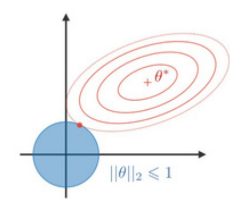
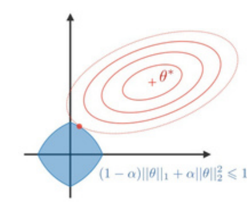
Regularization

r **Dropout** – Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability $p > 0$. It forces the model to avoid relying too much on particular sets of features.

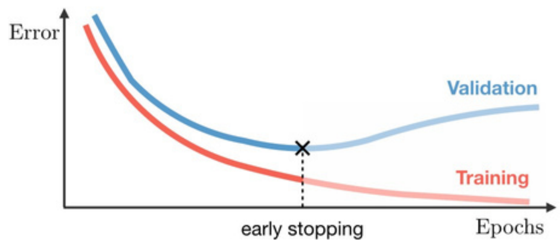


Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter $1 - p$.

r **Weight regularization** – In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights. The main ones are summed up in the table below:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$... + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda\Big[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\Big]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

r **Early stopping** – This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.



Good practices

r **Overfitting small batch** – When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

r **Gradient checking** – Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

	Numerical gradient	**Analytical gradient**
Formula	$\frac{df}{dx}(x) \approx \frac{f(x+h)-f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
Comments	- Expensive; loss has to be computed two times per dimension - Used to verify correctness of analytical implementation -Trade-off in choosing h not too small (numerical instability) nor too large (poor gradient approx.)	- 'Exact' result - Direct computation - Used in the final implementation

? ? ?