



THE UNIVERSITY *of* EDINBURGH
informatics

Machine Learning Practical (MLP)

Initialization, Normalization, Pretraining

Changjian Li

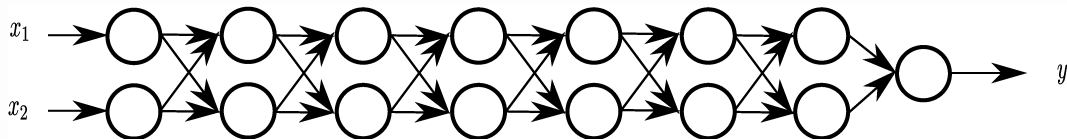
Weight Initialization

How do we initialize
neural network weights?

Random weight initialization

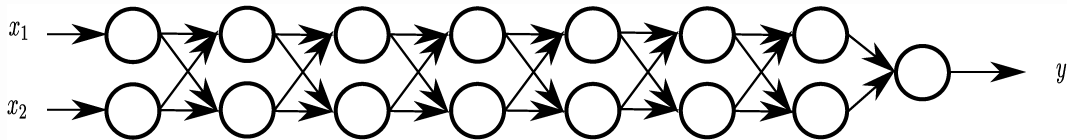
- Randomness helps to break “symmetry” between different units
- Initialise weights to small random numbers r , sampling weights independently from a Gaussian or from a uniform distribution
 - control the initialization by setting the mean (typically to 0) and variance of the weight distribution
- Biases may be initialised to 0

Vanishing/exploding gradients



- $\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{h}^{(l-1)}$, $\mathbf{h}^{(l)} = a(\mathbf{z}^{(l)})$ where $\mathbf{h}^0 = \mathbf{x}$
- $y = \mathbf{h}^{(L)}$ where L is the network depth

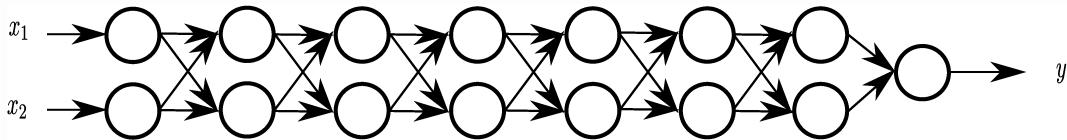
Vanishing/exploding gradients



- $z^{(l)} = W^{(l)} h^{(l-1)}$, $h^{(l)} = a(z^{(l)})$ where $h^0 = x$
- $y = h^{(L)}$ where L is the network depth
- Let's assume a is identity mapping,

$$y = W^{(L)} W^{(L-1)} \dots W^{(2)} W^{(1)} x$$

Exploding gradient problem

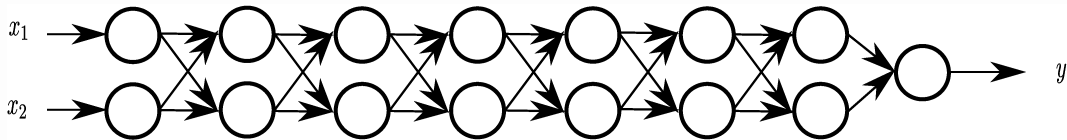


- Let's assume a is identity mapping,

$$y = \mathbf{W}^{(L)} \mathbf{W}^{(L-1)} \dots \mathbf{W}^{(2)} \mathbf{W}^{(1)} x$$

- $\mathbf{W}^{(l)} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \rightarrow y = \mathbf{W}^{(L)} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{L-1} x$ (Exploding gradients)

Vanishing gradient problem

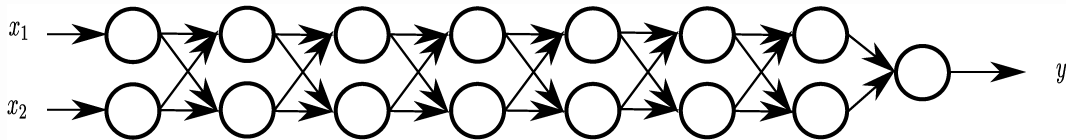


- Let's assume a is identity mapping,

$$y = W^{(L)} W^{(L-1)} \dots W^{(2)} W^{(1)} x$$

- $W^{(l)} = \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix} \rightarrow y = W^{(L)} \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix}^{L-1} x$ (Vanishing gradients)

Vanishing gradient problem



- Let's assume a is identity mapping,

$$y = W^{(L)} W^{(L-1)} \dots W^{(2)} W^{(1)} x$$

- $W^{(l)} = \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix} \rightarrow y = W^{(L)} \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix}^{L-1} x$ (Vanishing gradients)

- Recall $\underbrace{\frac{\partial E}{\partial W^{(l)}}}_{\text{grads_wrt_params}} = \underbrace{\frac{\partial E}{\partial h^{(L)}}}_{\text{error.grad}} \cdot \underbrace{\frac{\partial h^{(L)}}{\partial h^{(L-1)}} \dots \frac{\partial h^{(l+1)}}{\partial h^{(l)}}}_{W^{(L)} W^{(L-1)} \dots W^{(l+1)}} \cdot \underbrace{\frac{\partial h^{(l)}}{\partial W^{(l)}}}_{h^{(l-1)}}$

Setting $\text{Var}(w)$

Consider a linear unit:

$$y = \sum_{i=1}^{n_{in}} w_i x_i$$

if w and x are zero-mean and iid (independent and identically distributed), then

$$\text{Var}(y) = \text{Var}\left(\sum_{i=1}^{n_{in}} w_i x_i\right) = n_{in} \text{Var}(x) \text{Var}(w)$$

So, if we want variance of inputs x and outputs y to be the same, set $\text{Var}(w_i) = 1/n_{in}$.

“GlorotInit” (“Xavier initialization”)

We would like to set the variance of each layer to be $1/n_{in}$, thus

- Uniform distribution: $w_i \sim U(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}})$. Hint $x \sim U(a, b)$, then $\text{Var}(x) = (b - a)^2/12$.
- However we need to take the backprop into account, hence we would also like $\text{Var}(w_i) = 1/n_{out}$
- As a compromise set the variance to be $\text{Var}(w_i) = 2/(n_{in} + n_{out})$
- This corresponds to Glorot and Bengio’s normalised initialization

$$w_i \sim U\left(-\sqrt{6/(n_{in} + n_{out})}, \sqrt{6/(n_{in} + n_{out})}\right)$$

(Glorot & Bengio, 2010)

“GlorotInit” (“Xavier initialization”)

We would like to set the variance of each layer to be $1/n_{in}$, thus

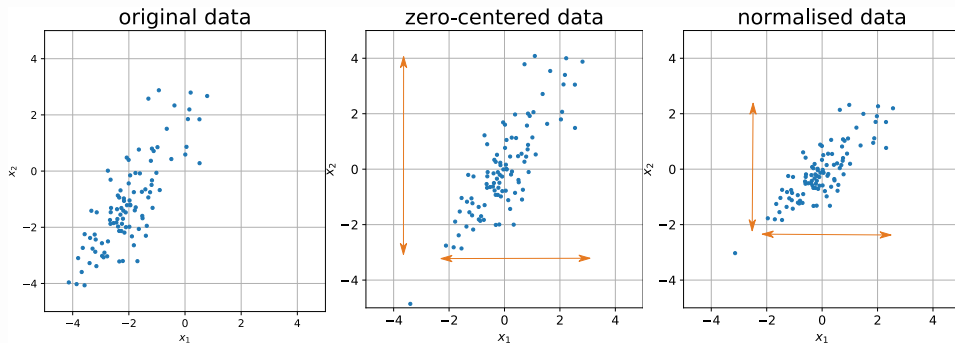
- Uniform distribution: $w_i \sim U(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}})$. Hint $x \sim U(a, b)$, then $\text{Var}(x) = (b - a)^2/12$.
- However we need to take the backprop into account, hence we would also like $\text{Var}(w_i) = 1/n_{out}$
- As a compromise set the variance to be $\text{Var}(w_i) = 2/(n_{in} + n_{out})$
- This corresponds to Glorot and Bengio’s normalised initialization

$$w_i \sim U\left(-\sqrt{6/(n_{in} + n_{out})}, \sqrt{6/(n_{in} + n_{out})}\right)$$

- For Normal distribution: $w_i \sim \mathcal{N}(0, \sqrt{1/n_{in}})$ and $w_i \sim \mathcal{N}\left(0, \sqrt{2/(n_{in} + n_{out})}\right)$.

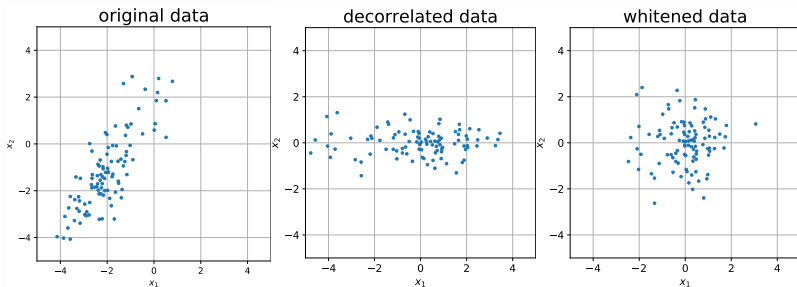
Normalization

Data Preprocessing – Normalization



- $X \in \mathbb{R}^{d \times N}$: training set, x_i : i -th sample (column)
- **Mean subtraction:** $\hat{x}_i = x_i - \text{mean}(X)$
- **Normalization:** $\tilde{x}_i = \hat{x}_i / \text{std}(X)$
- We also need to normalise test set with the **same** mean and std

Data Preprocessing – PCA and Whitening

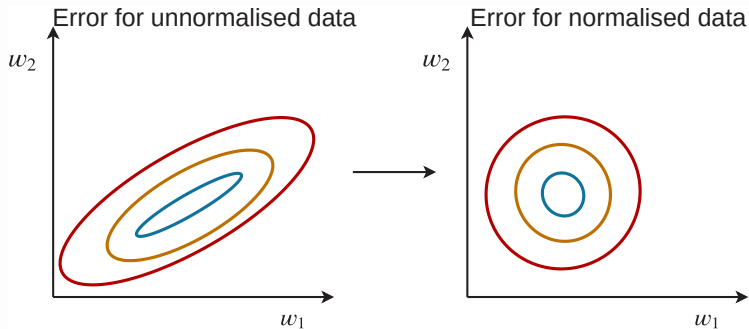


PCA (Principal Components Analysis)

- Decorrelate the data by projecting onto the principal components.
- Also possible to reduce dimensionality by only projecting onto the top P principal components.

Whitening: PCA + Scale each dimension

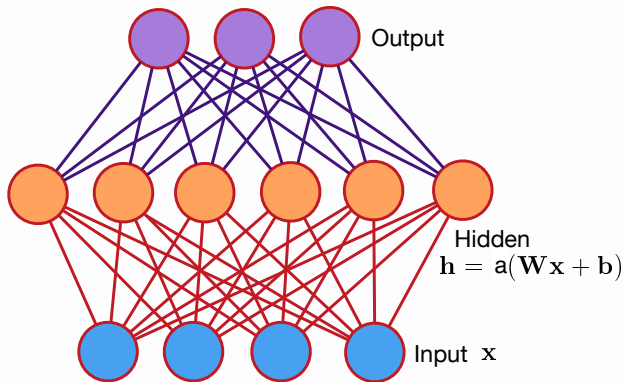
Why is whitening useful?



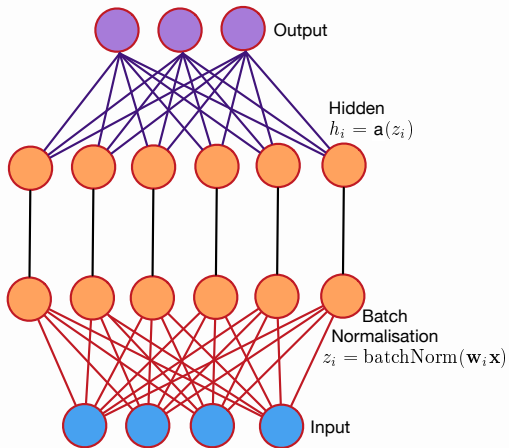
- Network can learn better from unbiased inputs

Can initialization prevent
vanishing/exploding gradient problem?

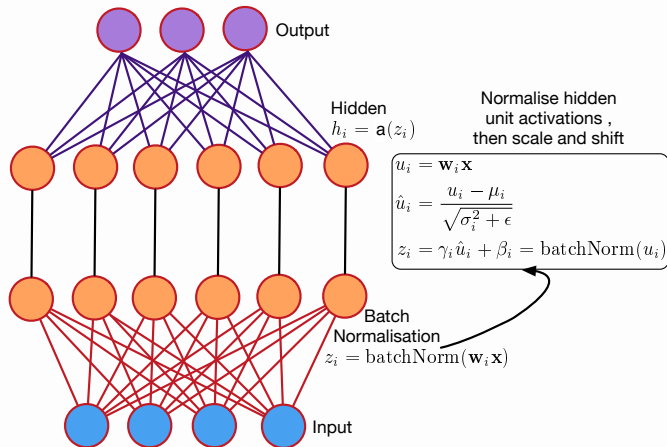
Batch Normalization



Batch Normalization



Batch Normalization

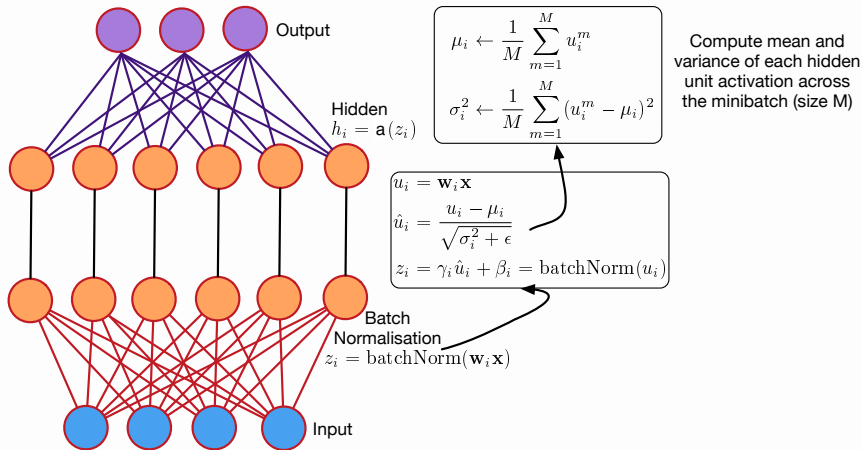


Normalise hidden
unit activations ,
then scale and shift

$$u_i = \mathbf{w}_i \mathbf{x}$$
$$\hat{u}_i = \frac{u_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$
$$z_i = \gamma_i \hat{u}_i + \beta_i = \text{batchNorm}(u_i)$$

Batch
Normalisation
 $z_i = \text{batchNorm}(\mathbf{w}_i \mathbf{x})$

Batch Normalization



Batch Normalization

- **Train-time**

- Use minibatch statistics to normalise activations of each layer
- γ and β are learned parameters which scale and shift the normalised activations
- `batchNorm` depends on the current training minibatch (to compute mean and variance)

- **Runtime** – use the moving average of the mean and variance computed during training

Batch Normalization – gradients (for reference)

$$\frac{\partial E}{\partial \hat{u}_i^m} = \frac{\partial E^m}{\partial z_i^m} \cdot \gamma_i$$

$$\frac{\partial E}{\partial \sigma_i^2} = \sum_m \frac{\partial E^m}{\partial \hat{u}_i^m} \cdot (u_i^m - \mu_i) \cdot \frac{-1}{2} (\sigma_i^2 + \epsilon)^{-3/2}$$

$$\frac{\partial E}{\partial \mu_i} = \left(\sum_m \frac{\partial E^m}{\partial \hat{u}_i^m} \cdot \frac{-1}{\sqrt{\sigma_i^2 + \epsilon}} \right) + \frac{\partial E}{\partial \sigma_i^2} \cdot \frac{1}{M} \sum_m -2(u_i - \mu_i)$$

$$\frac{\partial E}{\partial u_i^m} = \frac{\partial E^m}{\partial \hat{u}_i^m} \cdot \frac{1}{\sqrt{\sigma_i^2 + \epsilon}} + \frac{\partial E}{\partial \sigma_i^2} \cdot \frac{2(u_i - \mu_i)}{M} + \frac{\partial E}{\partial \mu_i} \cdot \frac{1}{M}$$

$$\frac{\partial E}{\partial \gamma_i} = \sum_m \frac{\partial E^m}{\partial z_i^m} \cdot \hat{u}_i^m$$

$$\frac{\partial E}{\partial \beta_i} = \sum_m \frac{\partial E^m}{\partial z_i^m}$$

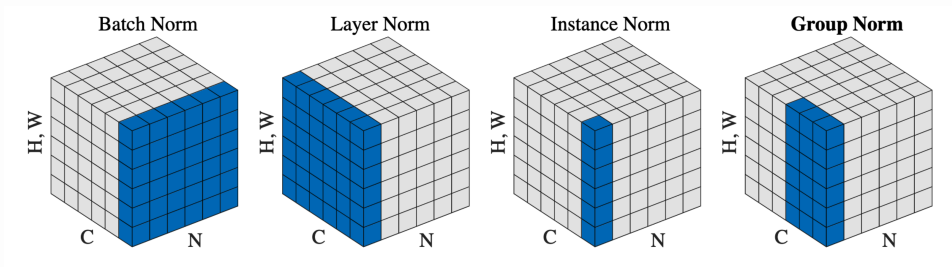
Pros/cons of Batch Normalization

- 😊 Makes training many-layered networks easier
 - Allows higher learning rates
 - Weight initialization less crucial
- 😊 Can act like a regulariser – maybe reduces need for techniques like dropout
- 😊 Can be applied to convolutional networks
- 😊 In practice (image processing) – achieves similar accuracy with many fewer training cycles
- 😊 Very widely used, and very useful for many-layered networks (e.g. visual object recognition)

Pros/cons of Batch Normalization

- 😊 Makes training many-layered networks easier
 - Allows higher learning rates
 - Weight initialization less crucial
- 😊 Can act like a regulariser – maybe reduces need for techniques like dropout
- 😊 Can be applied to convolutional networks
- 😊 In practice (image processing) – achieves similar accuracy with many fewer training cycles
- 😊 Very widely used, and very useful for many-layered networks (e.g. visual object recognition)
- 😞 Performs poor for small mini-batch sizes
- 😞 Cannot be applied to RNNs

Other normalization layers



(Wu and He, 2018)

- Layer norm (Ba et al. 2016)
- Instance norm (Ulyanov et al. 2016)
- Group norm (Wu and He, 2018)

Pretraining

Pretraining – Past

Past work address the issue of training deep models by *stacked* pretraining

- Train the first hidden layer
- Add a new hidden layer, and train only the parameters relating to the new hidden layer. Repeat.
- The use the pretrained weights to initialise the network – *fine-tune* the complete network using gradient descent

Pretraining – Present

- Now careful weight initialization and normalization strategies allow successful training of deep neural networks

Pretraining – Present

- Now careful weight initialization and normalization strategies allow successful training of deep neural networks
- But pretraining is still very useful for transfer learning, especially when we have a small training set. *Transfer learning* is the process of transferring knowledge from one (source) task to another (target).
 - Supervised pretraining: Pretrain on a large labelled source dataset,
 - Unsupervised pretraining: Pretrain on a large unlabelled source dataset,

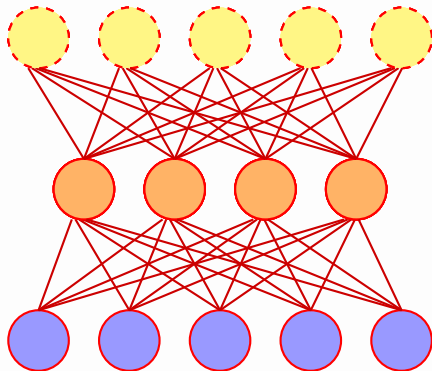
then initialise your model weights with the pretrained ones and finetune them on your target task.

Supervised pretraining

Supervised training of deep learning models requires large labelled datasets!

Assume that we have

- a small *target* labelled training set for our task,
- a large *source* labelled training set for another related task,

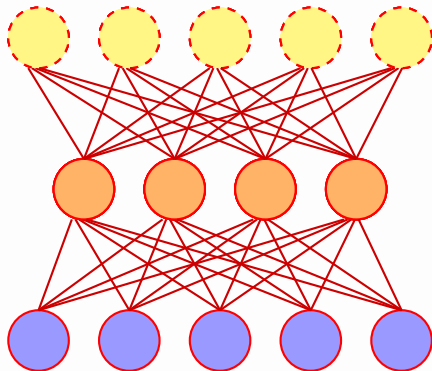


Supervised pretraining

Supervised training of deep learning models requires large labelled datasets!

Assume that we have

- a small *target* labelled training set for our task,
 - a large *source* labelled training set for another related task,
1. Train our model on the source training set *from scratch*,

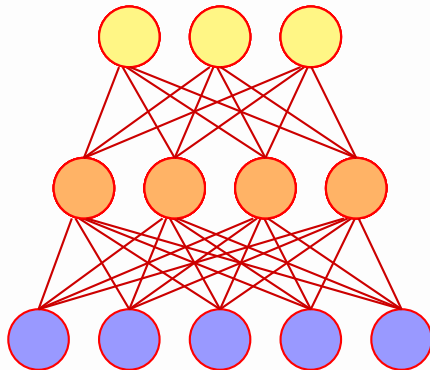


Supervised pretraining

Supervised training of deep learning models requires large labelled datasets!

Assume that we have

- a small *target* labelled training set for our task,
 - a large *source* labelled training set for another related task,
1. Train our model on the source training set *from scratch*,
 2. Replace the output layer for the target task and randomly initialise this layer's weights,

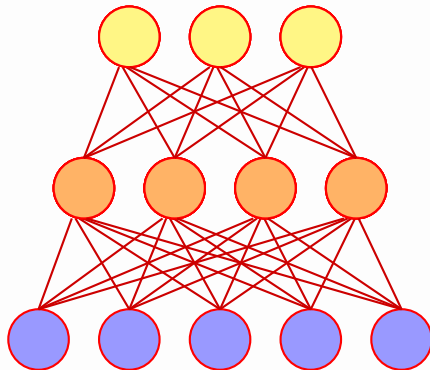


Supervised pretraining

Supervised training of deep learning models requires large labelled datasets!

Assume that we have

- a small *target* labelled training set for our task,
 - a large *source* labelled training set for another related task,
1. Train our model on the source training set *from scratch*,
 2. Replace the output layer for the target task and randomly initialise this layer's weights,
 3. Finetune (typically re-training with a low learning rate) the model on the target training set.



Unsupervised pretraining

Assume that we have

- a small *target* labelled training set for our task,
- a large *source* unlabelled training set with similar data distribution,

We follow the same pipeline as before:

1. Train our model on the source training set *from scratch*,
2. Replace its output layer for the target task and randomly initialise this layer's weights,
3. Finetune (re-train with lower learning rate) our model on the target training set.

Unsupervised pretraining

Assume that we have

- a small *target* labelled training set for our task,
- a large *source* unlabelled training set with similar data distribution,

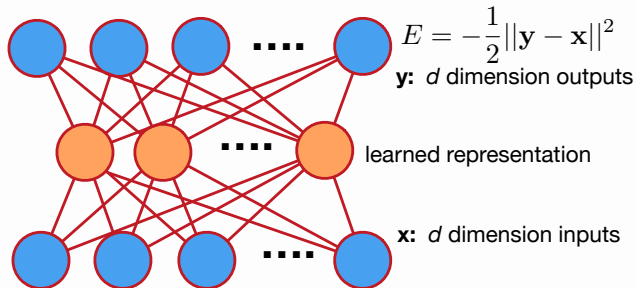
We follow the same pipeline as before:

1. Train our model on the source training set *from scratch*,
2. Replace its output layer for the target task and randomly initialise this layer's weights,
3. Finetune (re-train with lower learning rate) our model on the target training set.

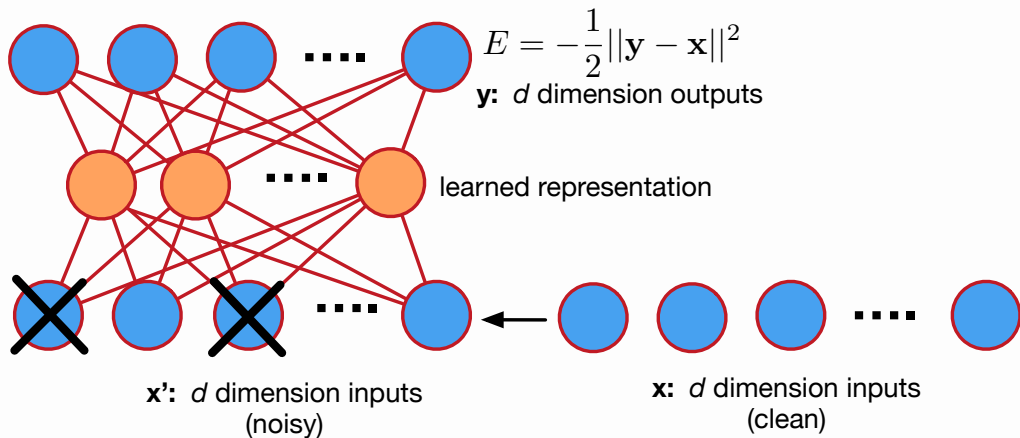
How can we train our model on a training set without any labels?

Unsupervised learning: Autoencoders

- An autoencoder is a neural network trained to map its input into a distributed representation from which the input can be reconstructed
- Example: single hidden layer network, with an output the same dimension as the input, trained to reproduce the input using squared error cost function



Denoising Autoencoder



Unsupervised learning: Denoising Autoencoders

- Basic idea: Map a corrupted input to a clean output
- Forces the learned representation to be stable and robust to noise and variations in the input
- To perform the denoising task well requires a representation which models the important structure in the input
- The aim is to learn a representation that is robust to noise, not to perform the denoising mapping as well as possible
- Noise in the input:
 - Random **Gaussian** noise added to each input vector
 - **Masking** – randomly setting some components of the input vector to 0
 - **“Salt & Pepper”** – randomly setting some components of the input vector to 0 and others to 1

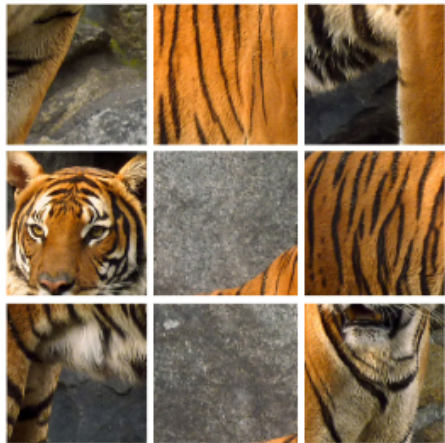
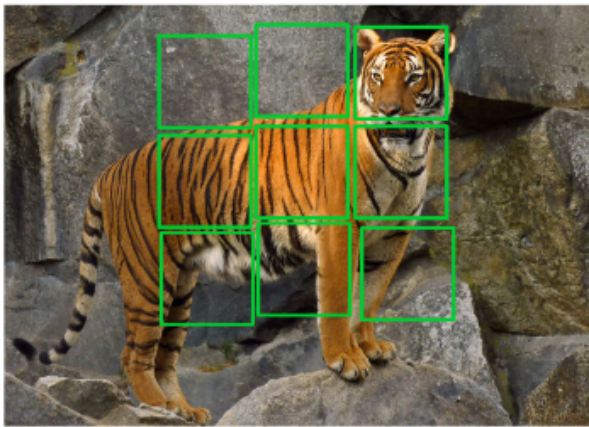
Self-supervised learning

Self-supervised learning is an unsupervised method but can also be seen as a supervised learning strategy for a *pretext task* where annotations (labels) can be obtained for free.

Some examples are:

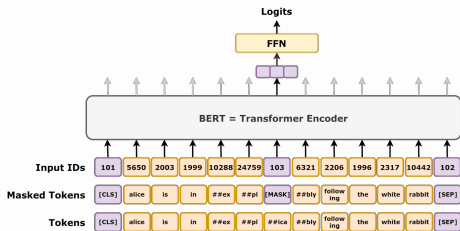
- CV - Dividing images into small patches to form a jigsaw puzzle, estimating the right combination
- CV - Learning representations that are invariant to geometric transformations
- NLP - Masking out words, estimating them from its context
- NLP - Estimating whether one sentence comes after another one in a document

Self-supervised learning: Solving jigsaw puzzles

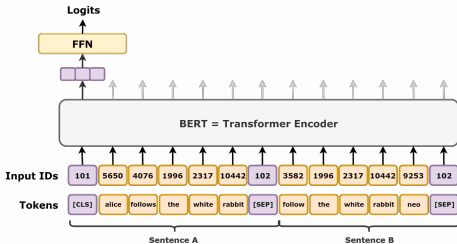


(Noroozi & Favaro, 2016)

Masked Language Model



masked language modeling



next sentence prediction

- Predict the masked token given its context
- Predict if two spans of text appeared sequentially in the training corpus

(Devlin et al. 2018)

Summary

- Vanishing/exploding gradient problem
- Weight initialization, data and batch normalization
- Pretraining and transfer learning
- Reading: Goodfellow et al, 8.4, 8.7.1, 14.1-2



<https://forms.cloud.microsoft/e/WdeYNv4A5c>