UNIVERSITY OF MINNESOTA
Driven to Discover℠

EE 5561: Image Processing and Applications

Lecture 19

Mehmet Akçakaya

Recap of Last Lecture

- Convolutional neural networks
  - Why convolutions instead of fully-connected layers?
    - Sparse weights
    - Parameter sharing
    - Translation equivariance
    - Works with inputs of different sizes
  - Variations on convolutions & visualizing convolutional layers
  - Other CNN components
    - Pooling layers
    - Normalization layers
    - Activation layers (from earlier)

Regularization in Neural Networks

- We talked about loss functions in training

$$\mathcal{L}(f_{\boldsymbol{\theta}}(X), Y) \triangleq J(\boldsymbol{\theta}; X, Y)$$

loss function

network parameters

input

output

  - We minimized $J(\boldsymbol{\theta}; X, Y)$ with respect to **θ**, when training the neural network
  - We might do additional changes to $J(\boldsymbol{\theta}; X, Y)$ to avoid overfitting or limit the capacity of the model
    - This is the concept of regularization (which we have spent quite a bit of time on)

- Idea: Instead minimize

$$J(\boldsymbol{\theta}; X, Y) + \alpha \mathcal{R}(\boldsymbol{\theta})$$

still defined over the database

positive hyperparameter

regularization term

Norm Regularizers

- What we know best!
- Generally: l_p norm regularizers, such as
  - l_1 norm: $$||\boldsymbol{\theta}||_1 = \sum_k |\theta_k|$$
  - l_2 norm: $$||\boldsymbol{\theta}||_2 = \left(\sum_k |\theta_k|^2\right)^{\frac{1}{2}}$$
  - l_∞ norm: $$||\boldsymbol{\theta}||_\infty = \max_k |\theta_k|$$
  - l_p norm: $$||\boldsymbol{\theta}||_p = \left(\sum_k |\theta_k|^p\right)^{\frac{1}{p}} \qquad p \geq 1$$

Norm Regularizers

- We know from before that l_2 norm is the computationally easiest to work with
  - Ridge regression (if MSE loss and linear model), a.k.a. Tikhonov regularization
  - For training, we now minimize

$$J(\boldsymbol{\theta}; X, Y) + \frac{\alpha}{2}||\boldsymbol{\theta}||_2^2 \qquad = J(\boldsymbol{\theta}; X, Y) + \frac{\alpha}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$$

  - Note its gradient is given as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X, Y) + \alpha\boldsymbol{\theta}$$

  - Thus the gradient update would be

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \epsilon\big(\alpha\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X, Y)\big) \\ &= \underset{\substack{\uparrow \\ \text{weight decay}}}{(1 - \epsilon\alpha)}\boldsymbol{\theta} - \epsilon\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X, Y) \end{aligned}$$

Norm Regularizers

- We also know that *l_1* norm regularizer is important
  - Related to sparsity → feature selection
  - Robust to outliers
  - For training, we now minimize

$$J(\boldsymbol{\theta}; X, Y) + \alpha ||\boldsymbol{\theta}||_1$$

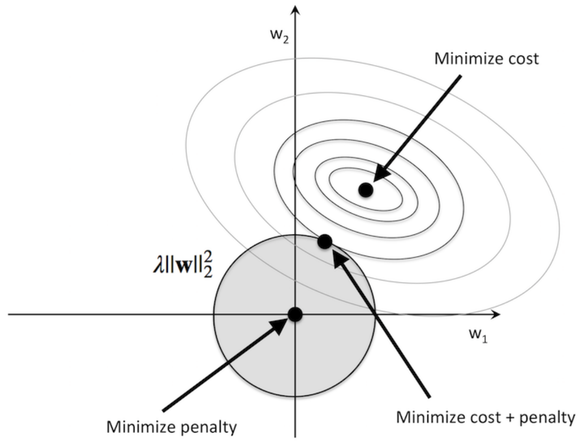
  - Note its gradient is given as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X, Y) + \alpha \cdot \mathrm{sign}(\boldsymbol{\theta})$$

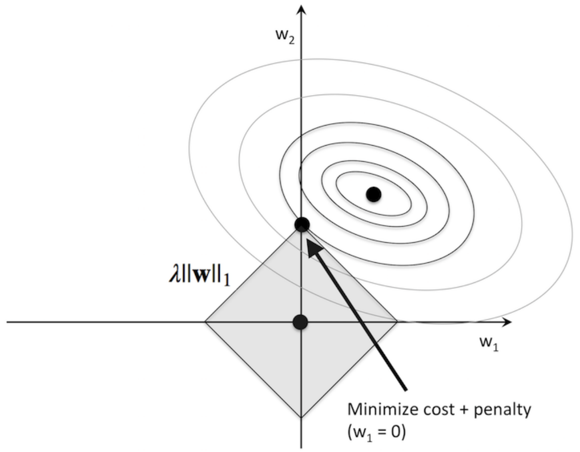
  - Computationally more difficult to work with

Norm Regularizers

- What we know from basic statistical image processing applies here too
  - l_2 norm regularization ~ Gaussian prior on weights



  - l_1 norm regularization ~ sparsity-promoting prior on weights



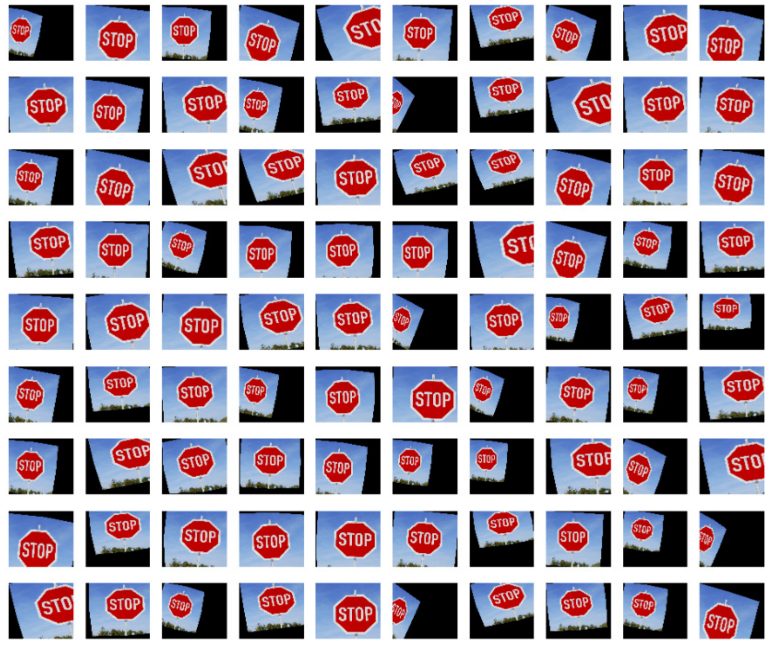
Regularization in Neural Networks

- Norm regularizers are theoretically easy to understand
  - Well-defined behavior on how it changes the optimization
  - But usually computationally hard to work with
    - Except l_2 norm, which leads to a very simple regularizer (weight decay)

- Thus, there are multiple non-norm regularizers used in neural networks

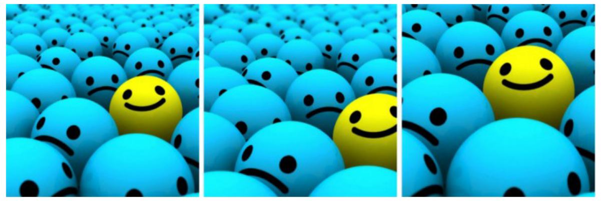
Data Augmentation

- Recall that our end goal is to have the best generalizability possible for our trained network
- Best way to improve generalizability: Get more data
- If we can't get more "real" data, we can "fake" the data
- Called data augmentation

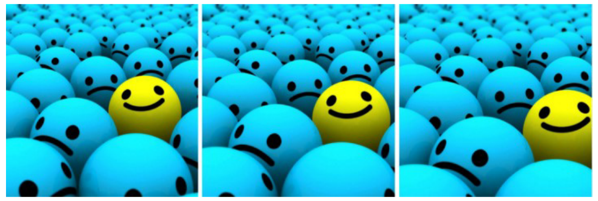


Data Augmentation

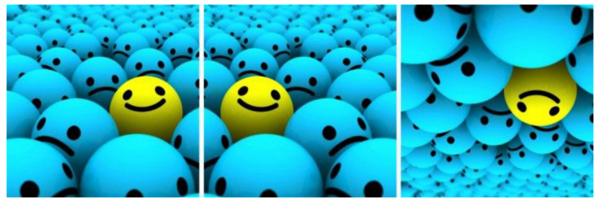
Crop



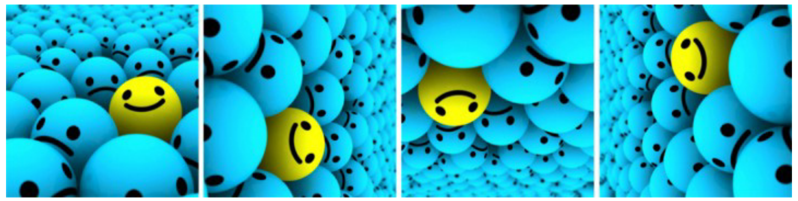
Scale



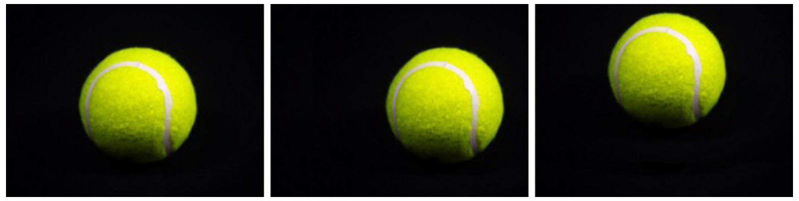
Flip



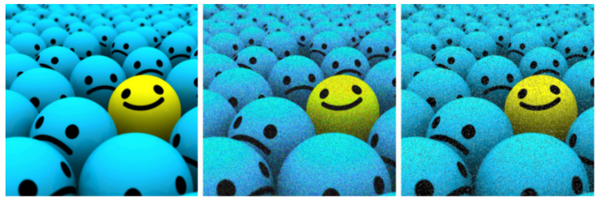
Rotate



Translate



Add noise

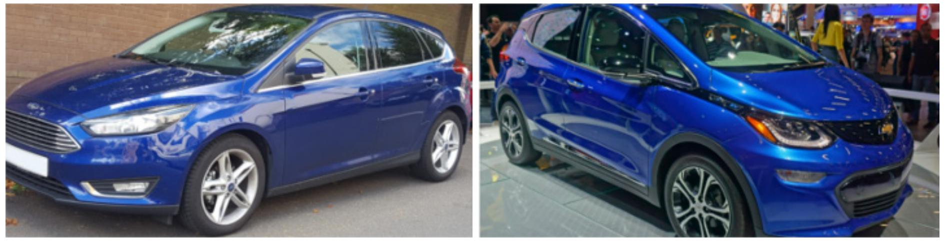


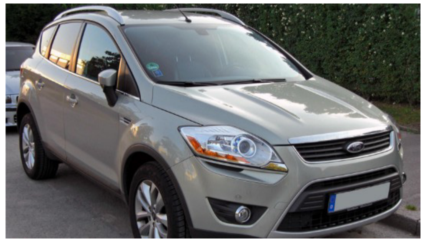
Why these?

- In real life, databases of images are taken in a limited set of conditions
- In applications, we want to do well in a variety of conditions (different orientation, scaling, brightness, etc)

Data Augmentation

- Augmentation may help even if we have lots of data?
  - May increase the amount of relevant data
  - Consider the hypothetical scenario:
    - Database of images with brand A (Ford) and brand B (Chevrolet)
    - All brand A cars face left, all brand B cars face right
    - Train network with a huge database
    - At test time



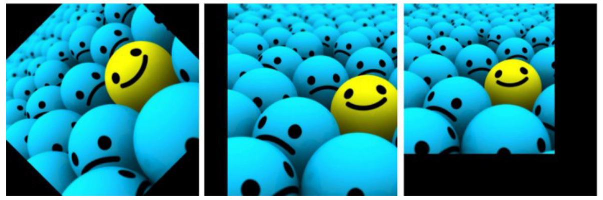


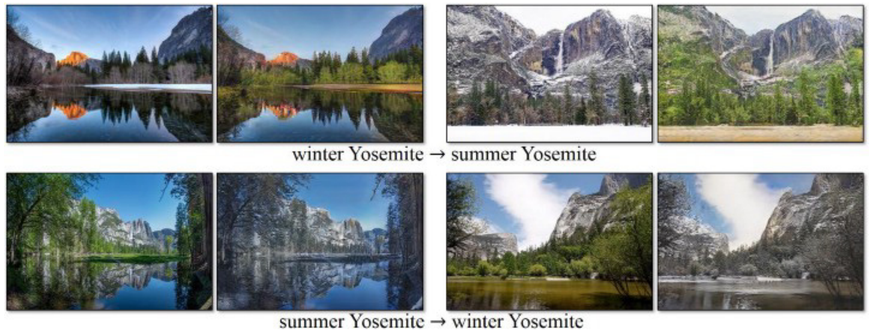
    - Network will pick brand B (right facing), even though it is brand A
    - Because the neural network learns features that distinguish two clasesses
    - The most relevant feature here is right-left facing
    - Augmentation would help here!

https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

Data Augmentation

- Practical points:
  - Some spatial transformation (e.g. rotation/translation) may lead to areas out of the field-of-view (or black regions)
  - Need to decide what to do with these
    - Idea ~boundary processing (e.g. reflect, symmetric, wrap, or do nothing)
- More advanced augmentation methods exist
  - Add noise to intermediate layers or output ("label smoothing")
  - Generative models (e.g. convert summer background to winter background) [Later]

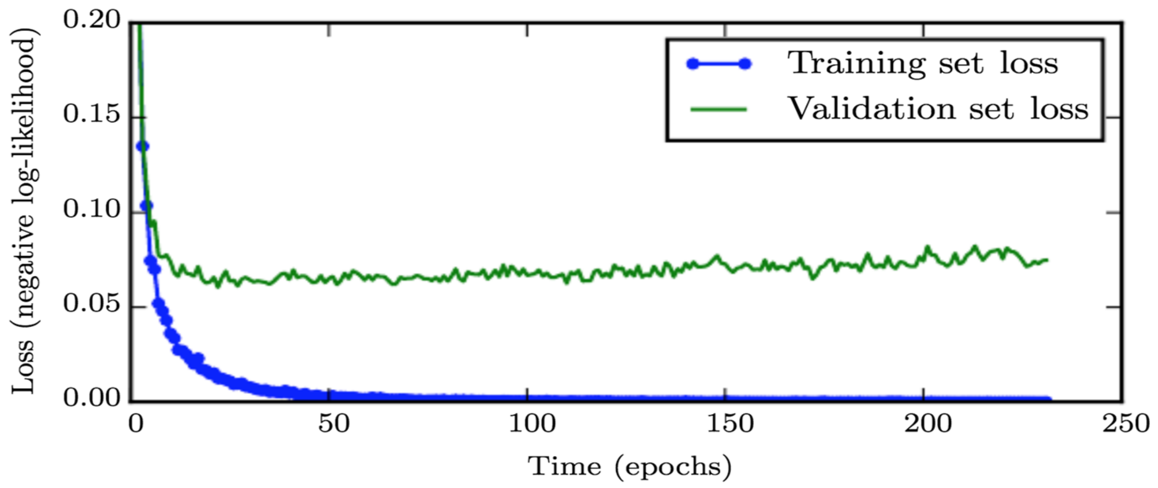



winter Yosemite → summer Yosemite
summer Yosemite → winter Yosemite

https://junyanz.github.io/CycleGAN/

Early Stopping

- Stop the training when validation set loss goes up



Early Stopping

- Why regularization?
  - We are effectively restricting the optimization to a "small" volume of the parameter space near the initial parameter value
  - Taylor series expansion around the optimal solution ***θ****

$$J(\boldsymbol{\theta}; X, Y) = J(\boldsymbol{\theta}^*; X, Y) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^T \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$$

$$\text{Note} \qquad \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X, Y)|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} = 0 \qquad \text{by optimality of } \boldsymbol{\theta}^*$$

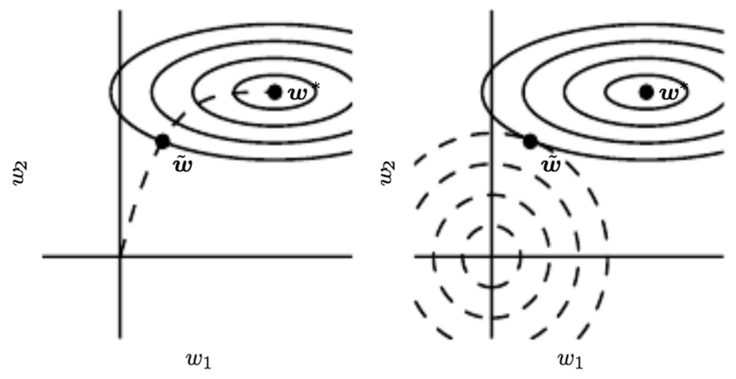
$$\text{Thus} \qquad \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X, Y) = \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$$

  - At iteration *t* we have the gradient update

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \epsilon \mathbf{H}(\boldsymbol{\theta}^{(t-1)} - \boldsymbol{\theta}^*)$$

$$\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* = (\mathbf{I} - \epsilon \mathbf{H})(\boldsymbol{\theta}^{(t-1)} - \boldsymbol{\theta}^*) = (\mathbf{I} - \epsilon \mathbf{H})^t (\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*)$$

  - If loss is MSE, one can show this is similar to l_2 regularization with $t \approx \frac{1}{\epsilon \alpha}$

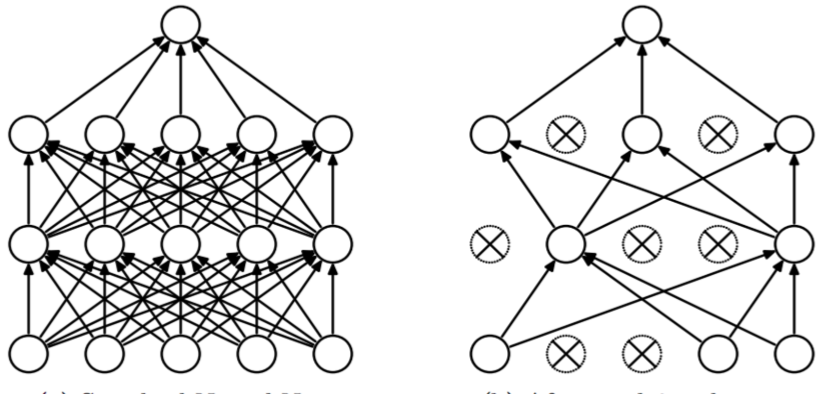


Dropout Regularization

- One of the most important regularization methods for neural networks
- Co-adaptation: When all weights are learned together, some of the connections have more predictive capability than others
  - Strong weights become stronger, weak weights become weaker
  - So only a fraction of nodes are actually trained
  - Problem: Expanding the network size does not help → accuracy remains limited
- Idea: Randomly drop nodes from computation graph throughout training

Dropout Regularization

- Idea: Randomly drop nodes from computation graph throughout training
  - Output turned to zero and not participating in back-propagation
  - ~ Training multiple networks with different compositions & "averaging" their results

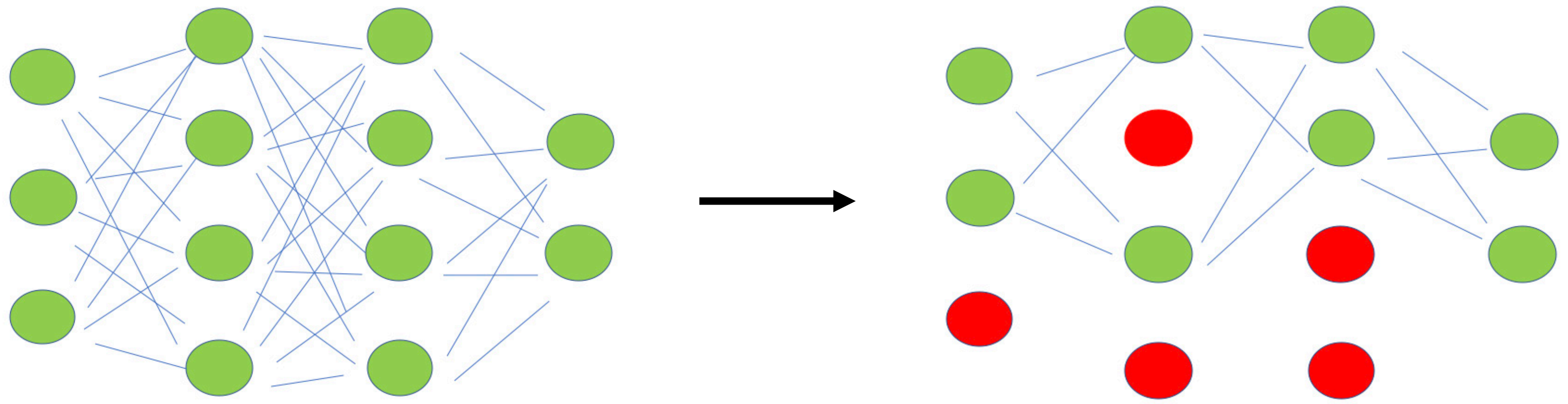


(a) Standard Neural Net (b) After applying dropout.

  - Note this is not "ensemble learning", which trains multiple models and averages those → too expensive in practice
  - Dropout drops random neurons with *every* pass through the network
    - Essentially sees a "new network" at every pass
    - Weights are still shared between these "networks" (unlike ensemble learning)

Dropout Regularization

- Idea: Randomly drop nodes from computation graph throughout training
  - Set a dropout rate, $p<1$, for each layer (probability that a unit is dropped)
  - This is a random variable, Bernoulli(p)



Dropout Regularization

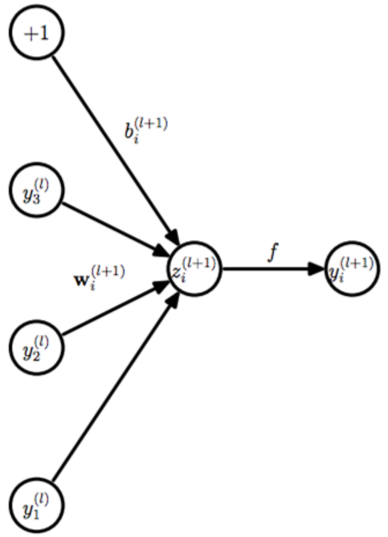
− Probabilistic Formulation

Normal Feedforward Net

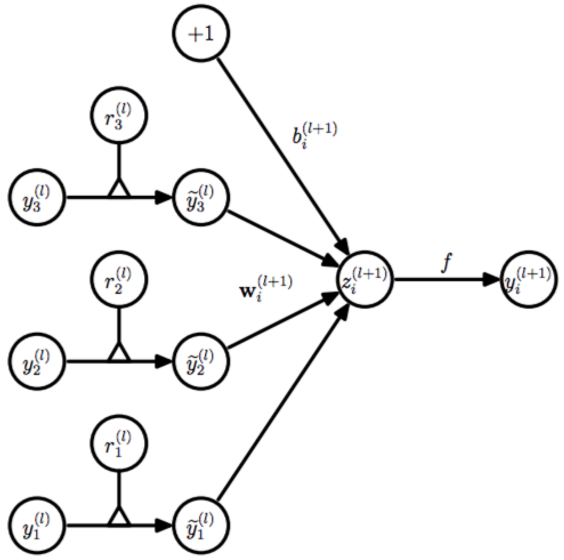
$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)} \\ y_i^{(l+1)} &= f(z_i^{(l+1)}), \end{aligned}$$

With Dropout

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)} \\ y_i^{(l+1)} &= f(z_i^{(l+1)}). \end{aligned}$$



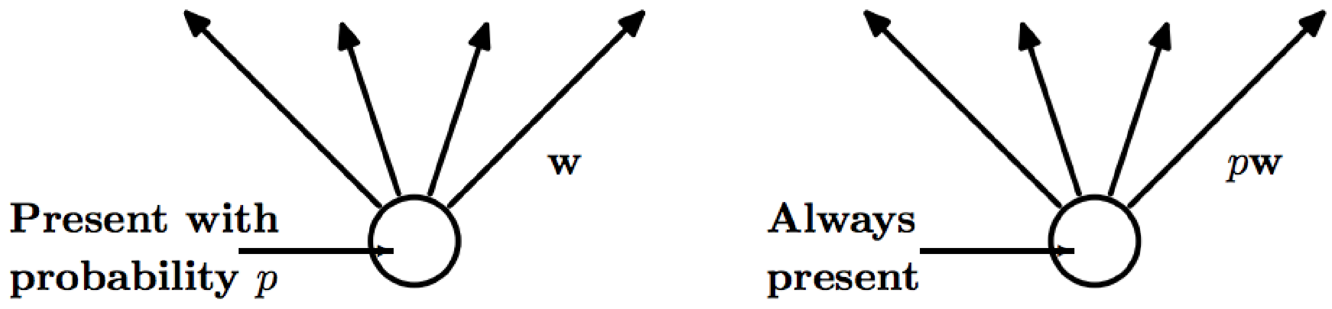
(a) Standard network



(b) Dropout network

Dropout Regularization

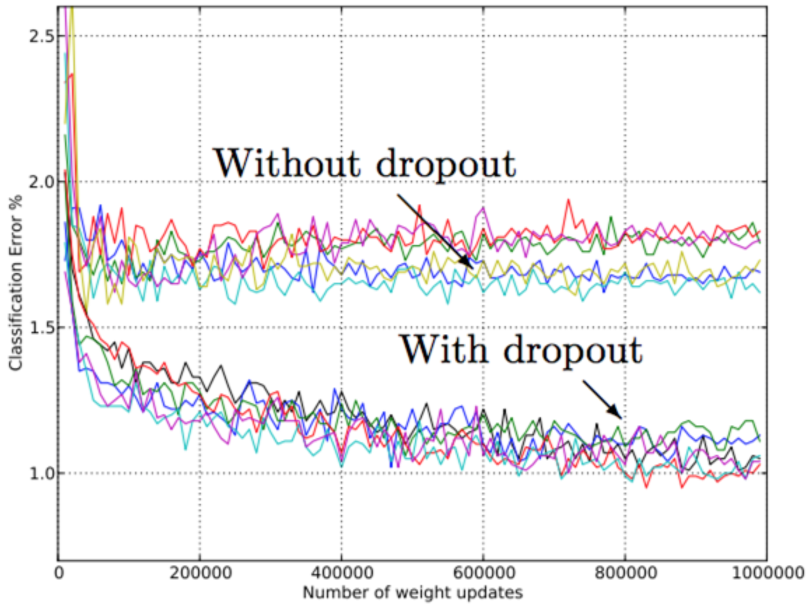
- At test time:
  - Want to use the full network
  - But now the neurons will receive more connections (& more activations), e.g.
    - If p = 0.5, then on average we will drop 2 out of 4 connections in a layer during training
    - But at testing, they will all be active → The response they produce will be large by ~2-fold
  - There is a correction added to compensate for this effect
    - Scale the weights by p at test time



(a) At training time (b) At test time

Dropout Regularization

- On MNIST



- Variants exist
  - e.g. replace Bernoulli RV with Gaussian RV $N(1, \sigma^2)$
  - Advantage: Expected value is 1 → So no change during inference time

Types of Learning Problems

- A lot of techniques named “XX learning”
  - Here we will try to cover some of these relevant for our tasks
  - We will split them into two: Learning tasks & learning methods
    - Learning tasks: Supervised learning, unsupervised learning, semi-supervised learning, self-supervised learning, …
    - Learning methods: Active learning, transfer learning, federated learning, …
  - We have been looking at supervised learning so far
    - Matched input & label data, train using these matched information
  - We will first look at “learning methods” in the supervised setting
  - Then we will revisit learning tasks

Learning Techniques

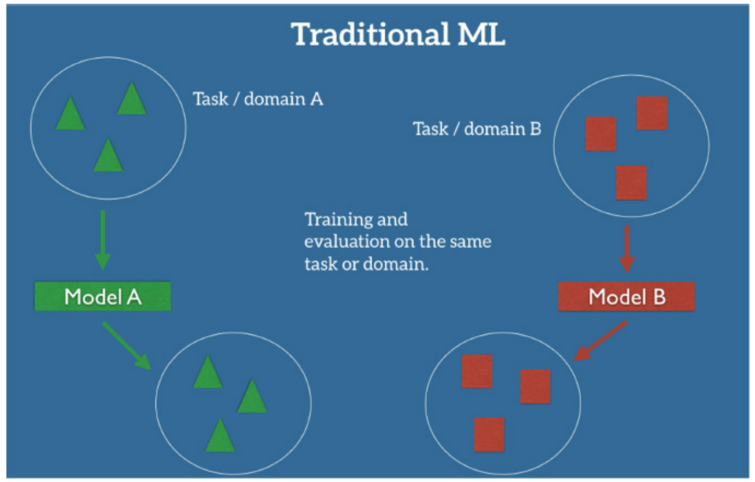
- These are the methodologies with which you learn a neural network for an application
- Active learning
  - Strategy to identify which specific examples/labels in our training data can best improve model performance
  - Simple example: Oracle (human-in-the-loop) identifies important examples
  - More advanced techniques automate this (e.g. via uncertainty quantification)
- Ensemble learning
  - Train ≥2 models on some data, then combine predictions
  - Idea: Ensemble performance > individual mode

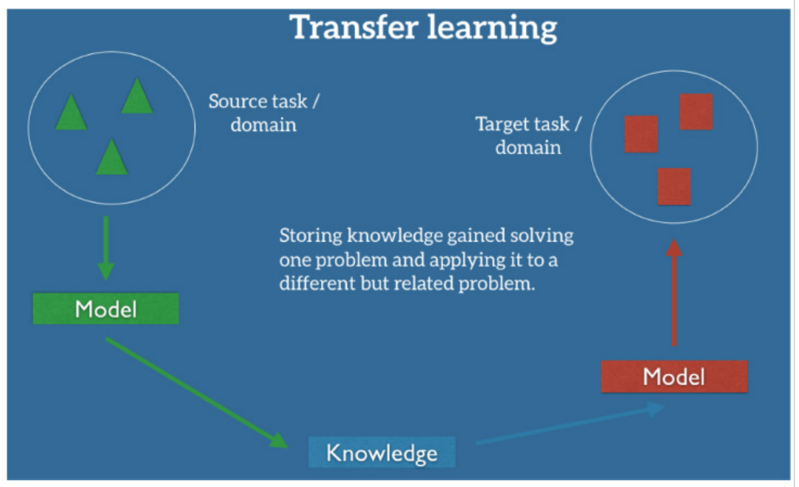
Learning Techniques

- Federated learning
  - Train on multiple decentralized devices
  - Each device has its own training data
  - Only model weights are shared and aggregated centrally
  - Important in areas where data sharing is difficult (e.g. medical information)
  - Variation: Decentralized learning, where no central aggregation is done
- Online learning
  - Update model as more observations come in time
  - Idea: Underlying data distribution changes over time & compensate for it

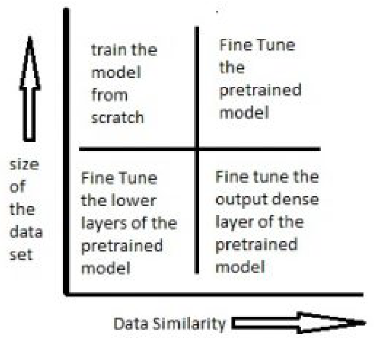
Learning Techniques

- Transfer learning
  - Learn in one setting then exploit this to improve generalization in another setting
    - From simpler to more complex tasks
    - From a source domain (with distribution A) to another (but similar) target domain (with distribution B)




Transfer learning
Source task / domain
Target task / domain
Storing knowledge gained solving one problem and applying it to a different but related problem.
Model
Model
Knowledge

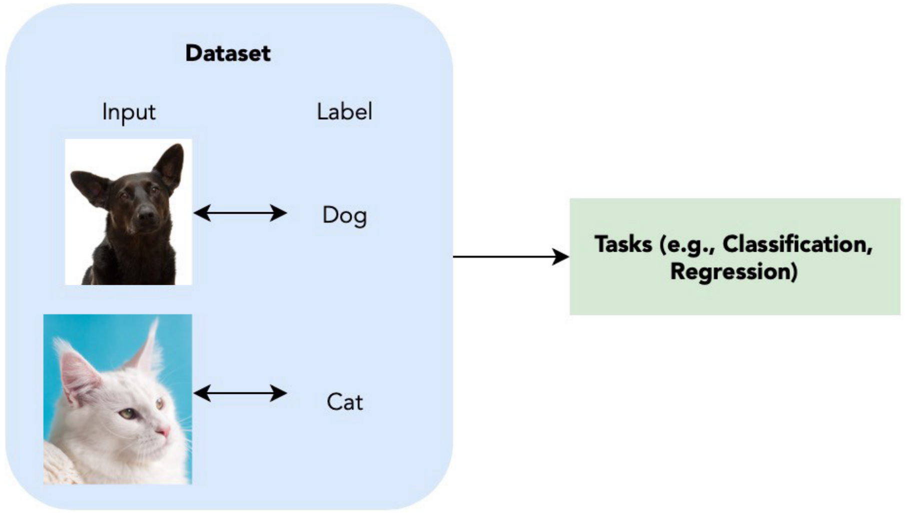
- Practically:


train the model from scratch
Fine Tune the pretrained model
Fine Tune the lower layers of the pretrained model
Fine tune the output dense layer of the pretrained model
size of the data set
Data Similarity

http://ruder.io/transfer-learning/index.html#whytransferlearningnow

Learning Tasks

- These are problems about how we learn an algorithm based on what kind of data is available
- Supervised learning
  - Learn a mapping between input and target/label

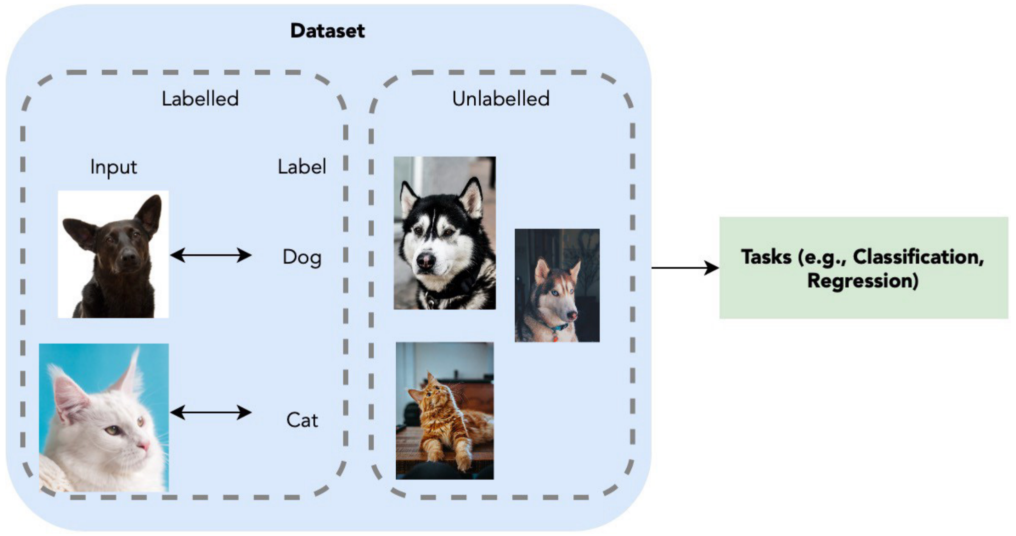


  - Issue: labels are difficult to get in real-life

https://towardsdatascience.com/supervised-semi-supervised-unsupervised-and-self-supervised-learning-7fa79aa9247c

Learning Tasks

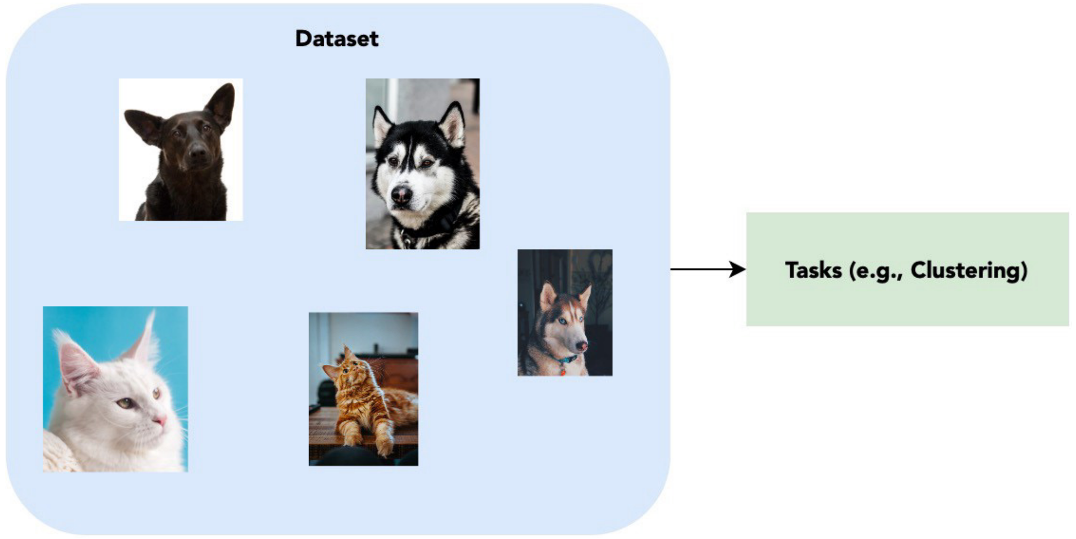
- Semi-supervised learning
  - Few labeled data + a large number of unlabeled examples



  - The unlabeled data provides info about underlying data distribution, e.g.
    - Discover patterns in unlabeled data
    - Use supervised methods to label them (pseudo-labels), and assign a confidence marker to guide training

Learning Tasks

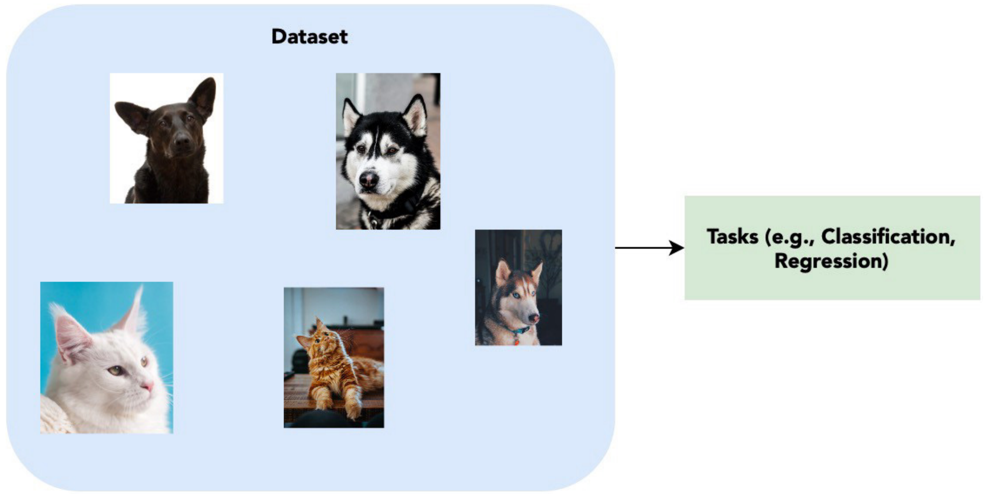
- Unsupervised learning
  - No labeled data or a-priori information



  - The network learns to identify patterns in data
  - Tasks are a little different, e.g. clustering, recommendation systems

Learning Tasks

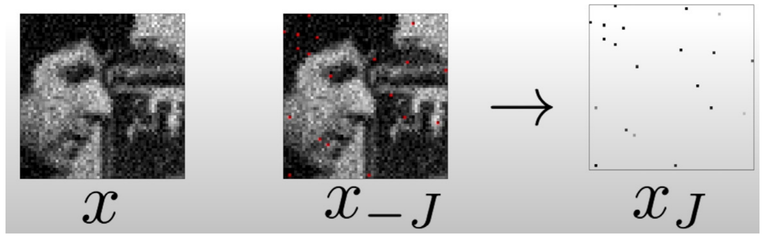
- Self-supervised learning
  - Circumvents problem of external labels → Still solves similar tasks to supervised learning



  - Input data is used to guide the learning
  - Technically unsupervised, but the loss function is framed as supervised
  - Main idea: Create supervisory labels from input
    - Learn to predict some selected parts of the input from other parts of the input

Learning Tasks

- Self-supervised learning
  - For instance, in regression tasks (denoising, reconstruction):
    - Mask out part of the data & try to predict it



    - Same idea in natural language processing (mask out words in a sentence and predict them)
  - Other ideas: Learn a different “pretext” task, fine-tune in “downstream” task (~transfer learning)
    - Here the pretext task is generated from the input data (e.g. inpainting, colorization)
    - The model is expected to learn the structure of data while learning to solve this task
    - Then apply it to downstream task
    - e.g. Contrastive learning: Do data augmentation to generate “positive” (same) and “negative” (different) input pairs → Learn to amplify features of positive pairs while hindering those of negatives

Recap

- Regularization in neural networks
  - Norm-based regularizers
  - Data augmentation
  - Early stopping
  - Dropout
- Learning Problems
  - Learning methods: Active, ensemble, federated, online, transfer learning
  - Learning tasks: Supervised, semi-supervised, unsupervised, self-supervised learning