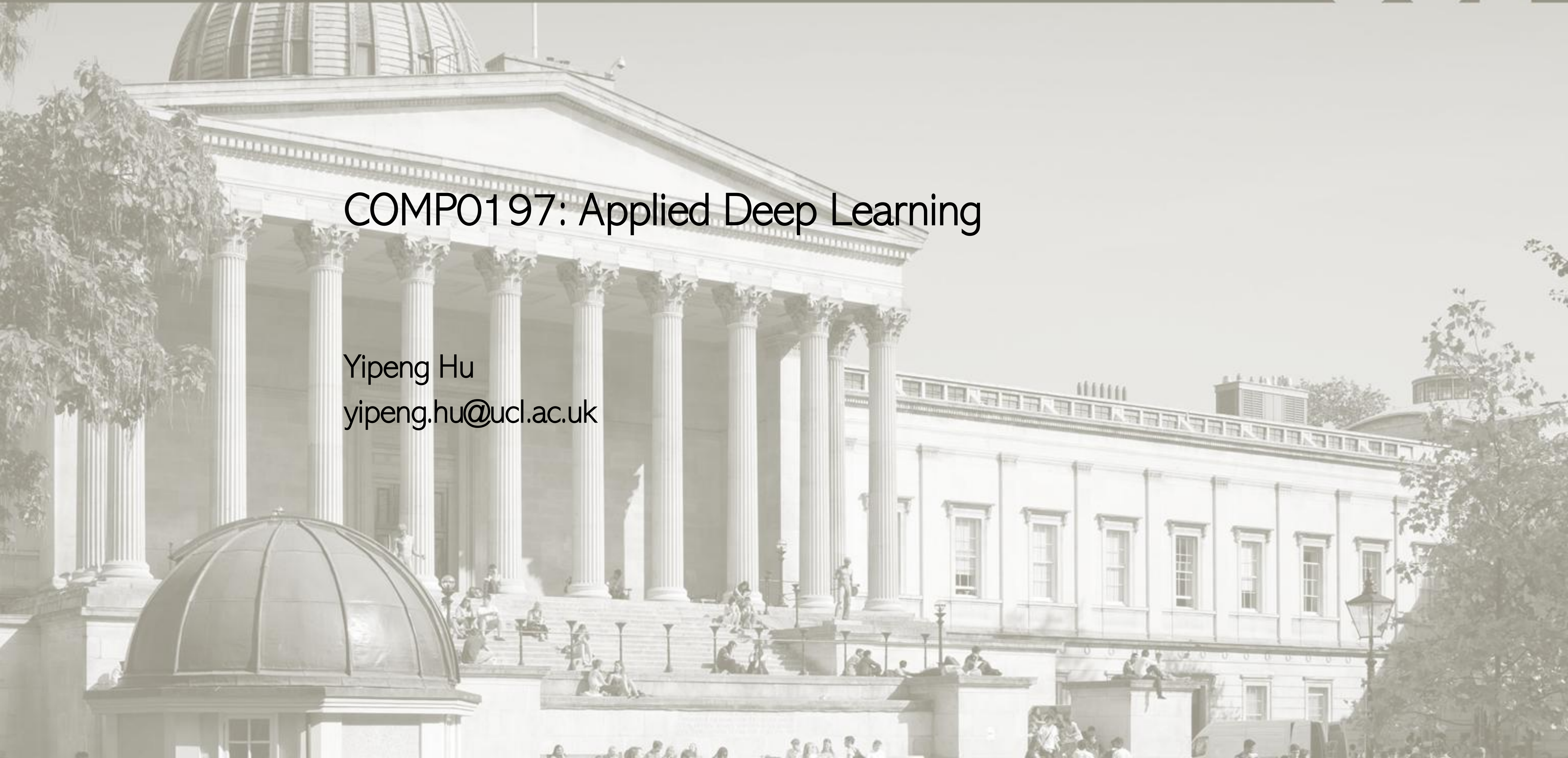
Department of Computer Science

UCL



COMP0197: Applied Deep Learning

Yipeng Hu
yipeng.hu@ucl.ac.uk

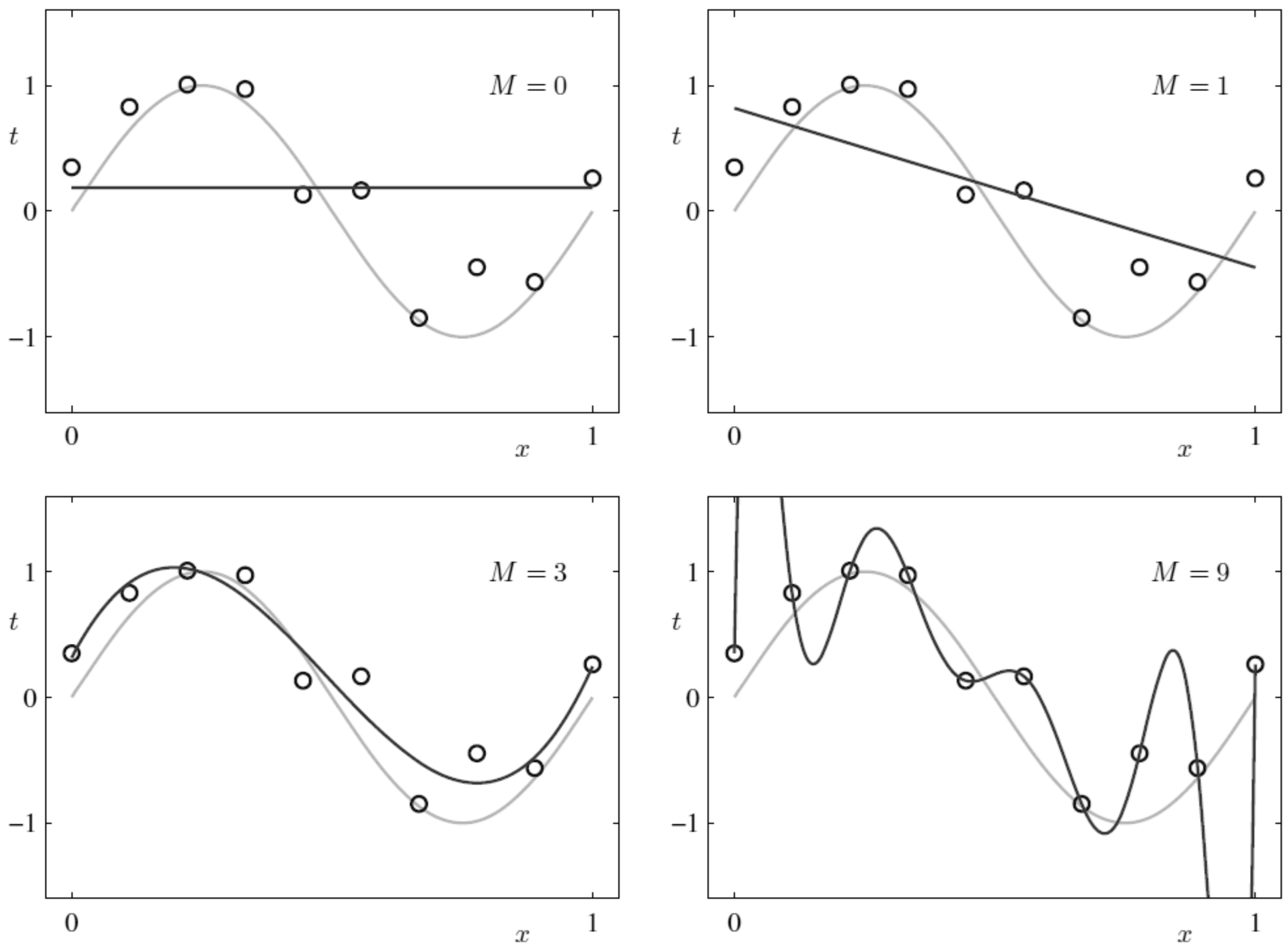
Regularisation

Model complexity, capacity, expressibility, generalisability

Example:

$$y = f(x;\theta) = w_3x^3 + w_2x^2 + w_1x + w_0$$

Overfitting and underfitting



Regularisation

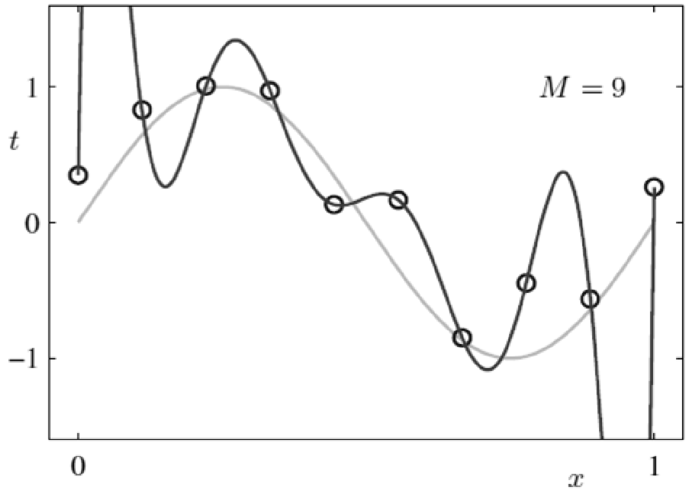
Linear polynomial: $y = f(x;\mathbf{w}) = \sum_{m=0}^{M} w_m x^m$

Mean-square-error (MSE) as loss: $\ell_\theta = \frac{1}{N}\sum_{n=1}^{N}(y_n - t_n)^2$

L^2-Norm (weight-decay): $\|\mathbf{w}\|^2 = \sum_{m=0}^{M} w_m^2 = w_0^2 + w_1^2 + w_2^2 + \cdots$

Hyperparameter: λ

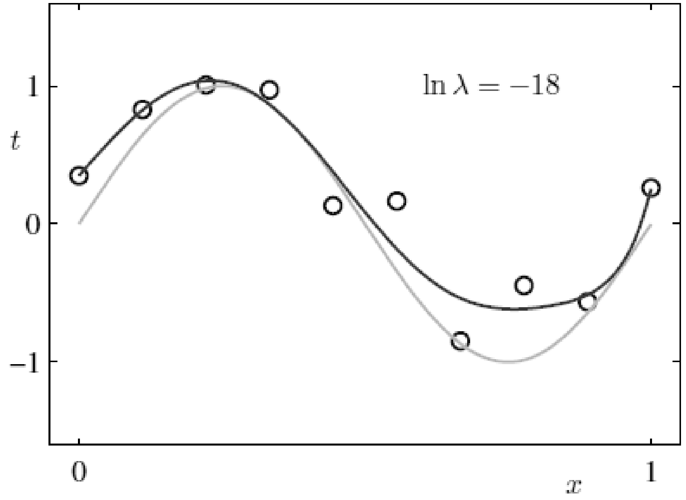
Regularised loss: $\tilde{\ell}_\theta = \ell_\theta + \frac{\lambda}{2}\|\mathbf{w}\|^2$

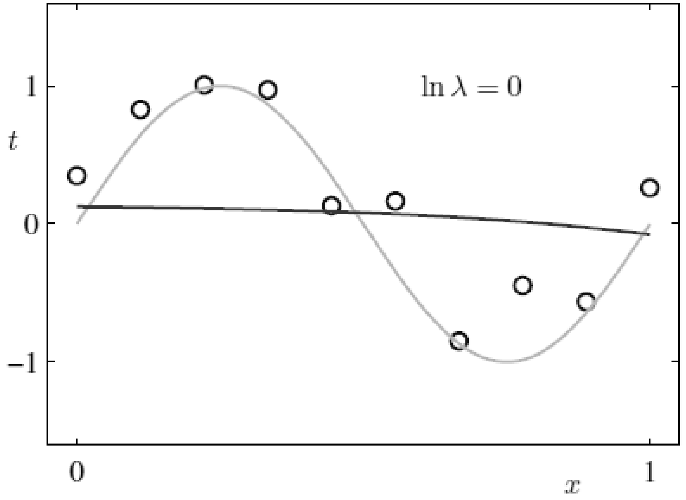


Solution

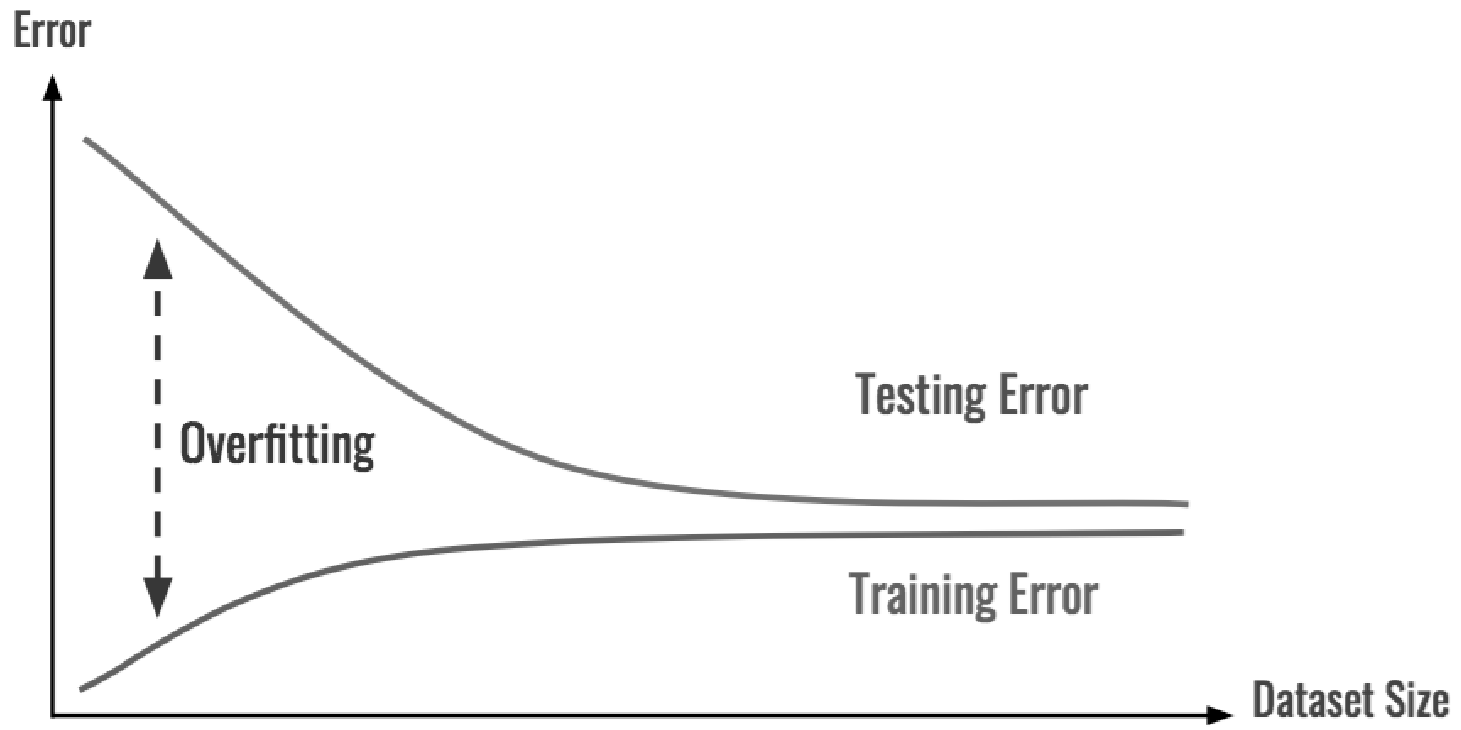
Least-square solution

Gradient-descent





Generalisability vs. dataset size



- Purpose of regularisation
- Approaches to regularising deep neural networks

Regularisation

- Training strategies

  Early stopping, curriculum learning, data resampling.

- Data augmentation

  Random data transformation, affinity and diversity

- Invariance and normalisation

  Spatial transformer networks, batch normalisation, reparameterization

- Parameter constraints

  Parameter norms, sparse representation, parameter sharing, multi-task, semi-supervised

- Unsupervised learning

  Autoencoder, generative adversarial network

- Randomness

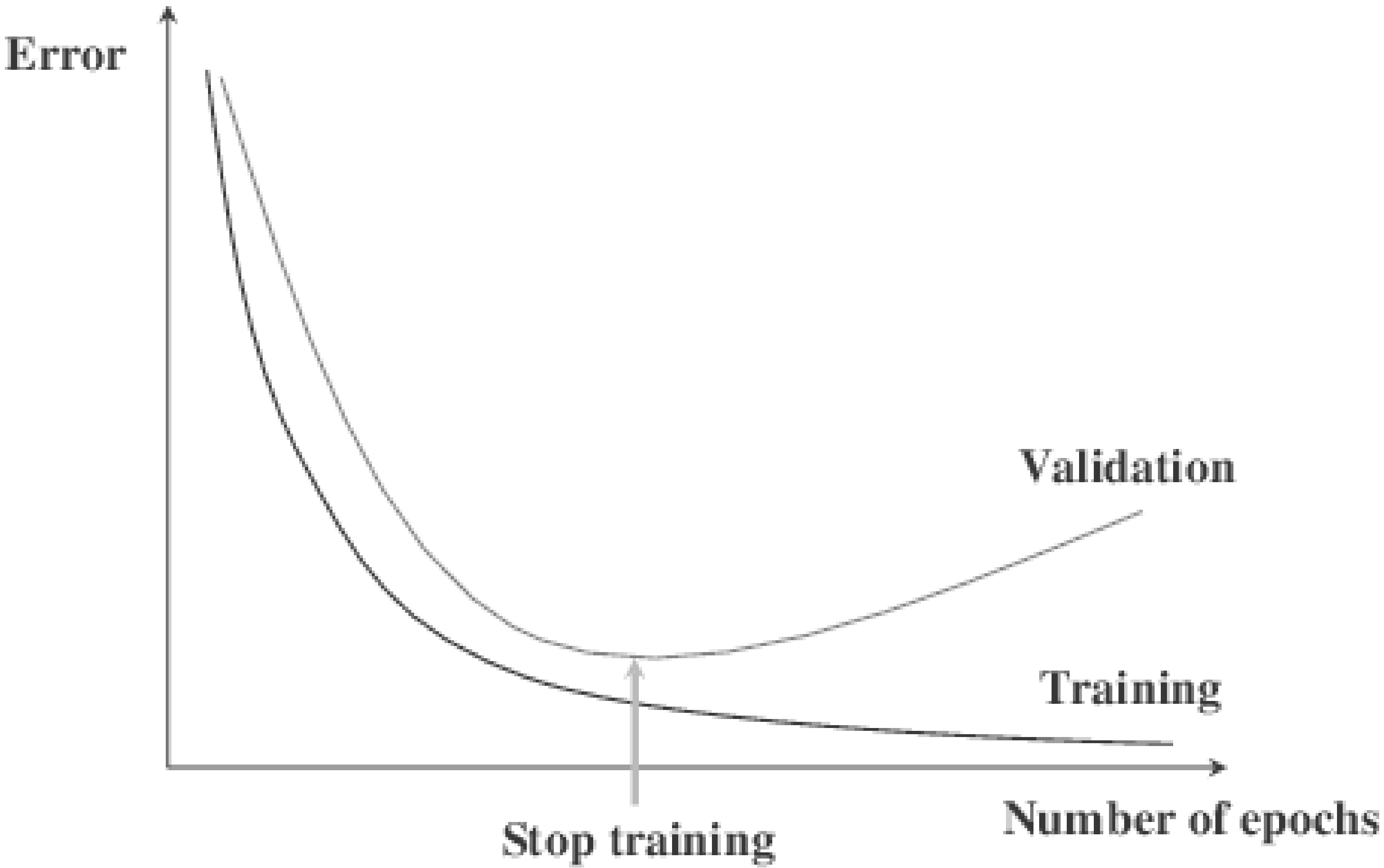
  Noise, dropout

- Model combining

  Model averaging, ensemble, bagging/bootstrap aggregating, boosting

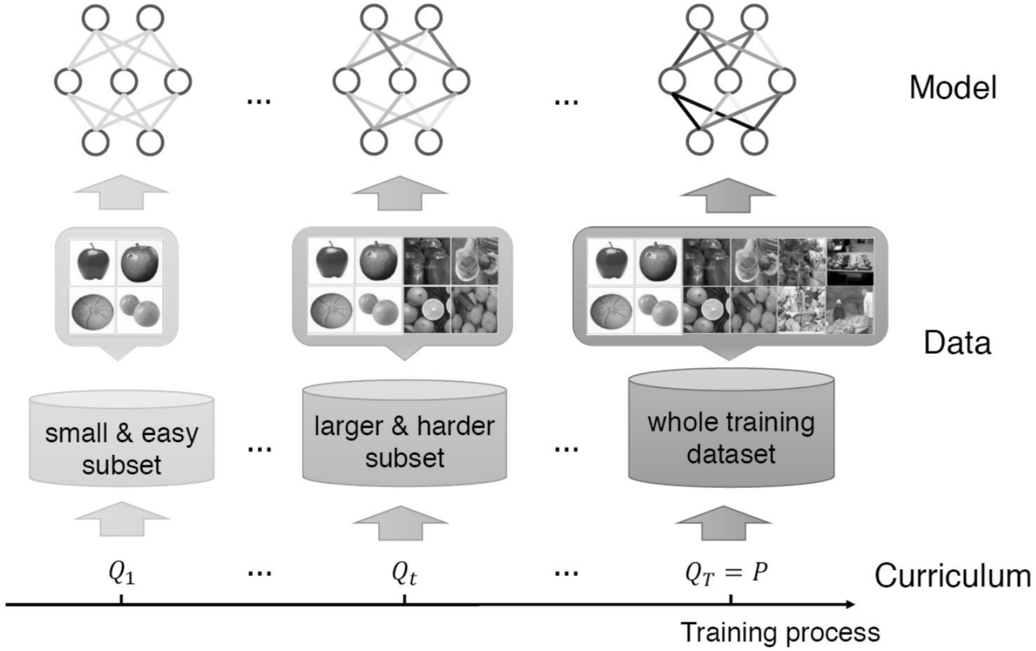
Regularisation | Training Strategies

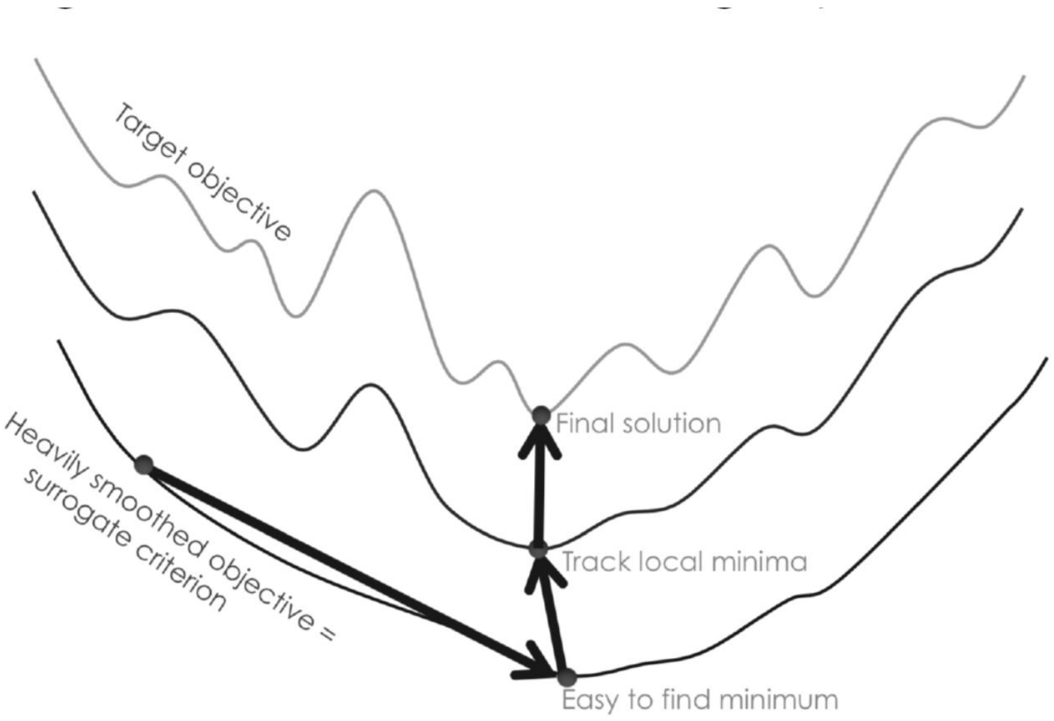
– Early stopping



– Curriculum learning

1. Sorting example difficulty (e.g. task complexity, measured by number of classes)
2. Pacing the curriculum learning (e.g. data-resampling)





Wang, X., Chen, Y. and Zhu, W., 2021. A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

– Data resampling

- Stochastic (minibatch) gradient descent, sampling without replacement
- Data resampling, e.g. up/down-sampling for imbalanced classes, similar to weighted loss $\ell_{\mathbf{w}}(\mathbf{y}_n, \mathbf{t}_n; \omega_n) = \frac{1}{N}\sum_{n=1}^{N} \omega_n d(\mathbf{y}_n, \mathbf{t}_n)$

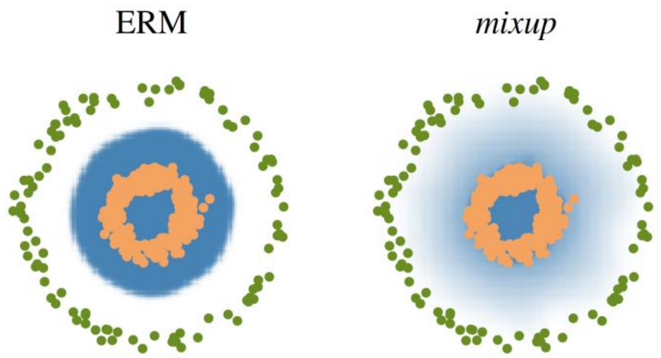
- Empirical risk minimisation
  - Data (generating) distribution: $\mathbb{E}_{(\boldsymbol{x},y)\sim p_{data}}[\ell(f(\boldsymbol{x},\boldsymbol{\theta}),y)]$
  - Training data distribution: $\mathbb{E}_{(\boldsymbol{x},y)\sim \hat{p}_{data}}[\ell(f(\boldsymbol{x},\boldsymbol{\theta}),y)]$
  - Empirical risk, e.g. minibatch: $\mathbb{E}_{(\boldsymbol{x},y)\sim \hat{p}_{data}(\boldsymbol{x},y)}[\ell(f(\boldsymbol{x},\boldsymbol{\theta}),y)] = \frac{1}{M}\sum_{m=1}^{M} \ell(f(\boldsymbol{x}^m,\boldsymbol{\theta}),y^m)$

    i.e. $\hat{p}_{data}(\boldsymbol{x},y) = \frac{1}{M}\sum_{m=1}^{M} \delta(\boldsymbol{x} = \boldsymbol{x}^m, y = y^m)$
  - Approximate data distribution using vicinity distribution: : $\hat{p}_v(\boldsymbol{x},y) = \frac{1}{M}\sum_{m=1}^{M} v(\tilde{\boldsymbol{x}},\tilde{y}|\boldsymbol{x}^m,y^m)$

    e.g. $v(\tilde{\boldsymbol{x}},\tilde{y}|\boldsymbol{x}^m,y^m) = \mathcal{N}(\tilde{\boldsymbol{x}} - \boldsymbol{x}^m)\delta(y = y^m)$

- Consider what the “target” test data distribution is

- "mixup"



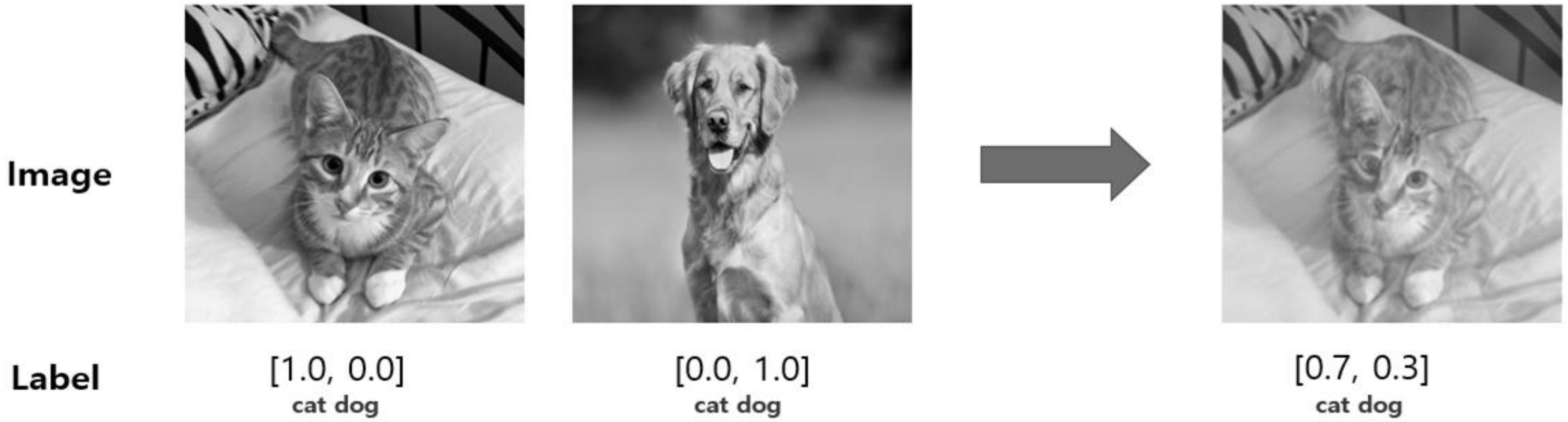
$$\hat{p}_{data}(\boldsymbol{x}, y) = \frac{1}{M}\sum_{m=1}^{M} \delta(\boldsymbol{x} = \boldsymbol{x}^m, y = y^m)$$

$\hat{p}_v(\boldsymbol{x}, y) = \frac{1}{M}\sum_{m=1}^{M} v(\tilde{\boldsymbol{x}}, \tilde{y} | \boldsymbol{x}^m, y^m) = \frac{1}{M}\sum_{m=1}^{M} \mathcal{N}(\tilde{\boldsymbol{x}} - \boldsymbol{x}^m)\delta(y = y^m)$ - *adding Gaussian noise to data*

$\hat{p}_\mu(\boldsymbol{x}, y) = \frac{1}{M}\sum_{m=1}^{M} \mu(\tilde{\boldsymbol{x}}, \tilde{y} | \boldsymbol{x}^m, y^m)$,

where $\mu(\tilde{\boldsymbol{x}}, \tilde{y} | \boldsymbol{x}^m, y^m) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}[\delta(\tilde{\boldsymbol{x}} = \lambda \boldsymbol{x}^m + (1-\lambda)\boldsymbol{x}^n, \tilde{y} = \lambda y^m + (1-\lambda) y^n)]$ - *minimising empirical vicinal risk, i.e. mixup*

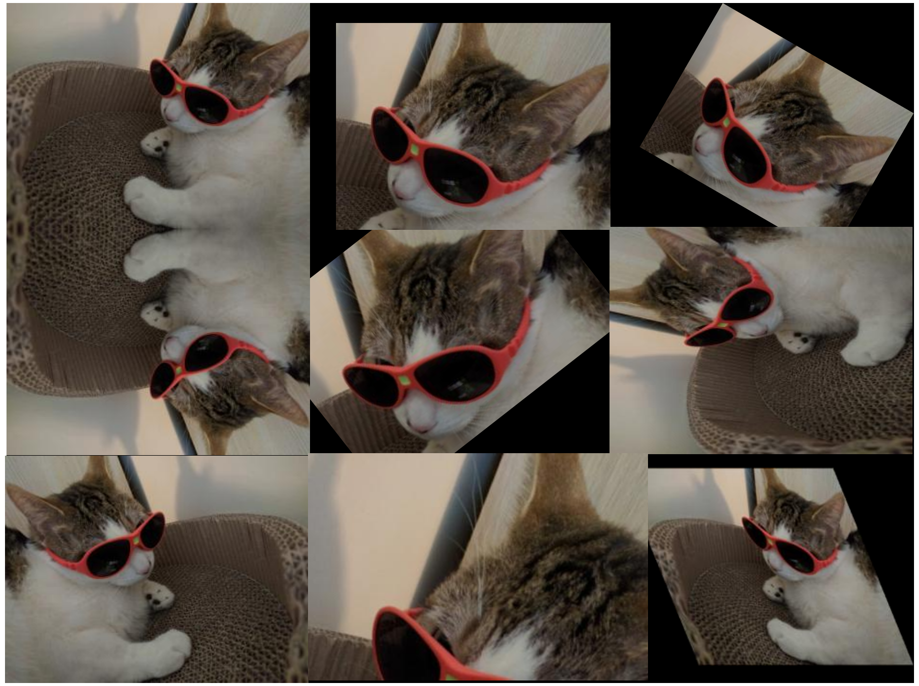
$$\tilde{\boldsymbol{x}} = \lambda \boldsymbol{x}^m + (1-\lambda)\boldsymbol{x}^n$$
$$\tilde{y} = \lambda y^m + (1-\lambda) y^n$$

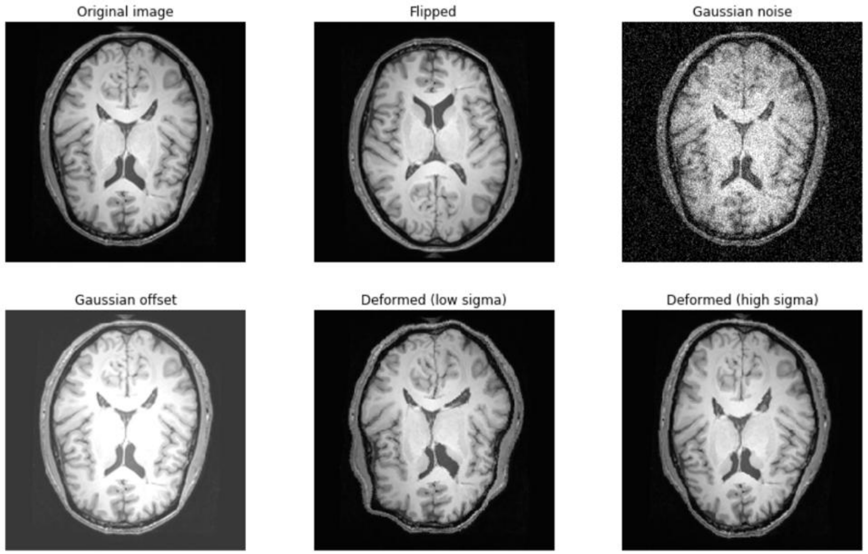


- Data augmentation

Regularisation | Data Augmentation

- **Random data transformation**
- Colour/intensity/contrast space
- Imaging-specific parameters
  - e.g. photometric transformation (luminance, illuminance, flux, intensity…),
  - bias field for MR,
  - perspective transformation
- Spatial transformation
  - Geometric: flipping, cropping, rotation
  - Affine
  - Nonlinear deformation
- (Unsupervised) generative models*





– Affinity and diversity

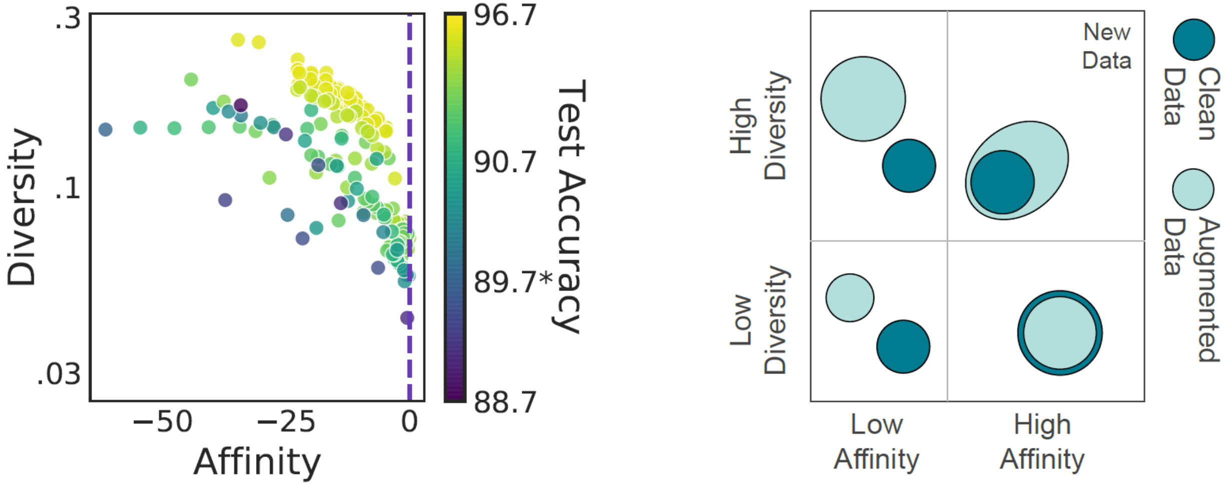
- Affinity quantifies how much an augmentation shifts the training data distribution from that learned by a model.
  e.g. difference in validation accuracies due to augmentation

$$\mathcal{T}[a; m; D_{val}] = \mathcal{A}(m, D'_{val}) - \mathcal{A}(m, D_{val})\,.$$

- Diversity quantifies the complexity of the augmented data with respect to the model and learning procedure.
  e.g. expected training loss due to augmentation

$$\mathcal{D}[a; m; D_{train}] := \mathbb{E}_{D'_{train}}\left[L_{train}\right]\,.$$

– Other definitions*



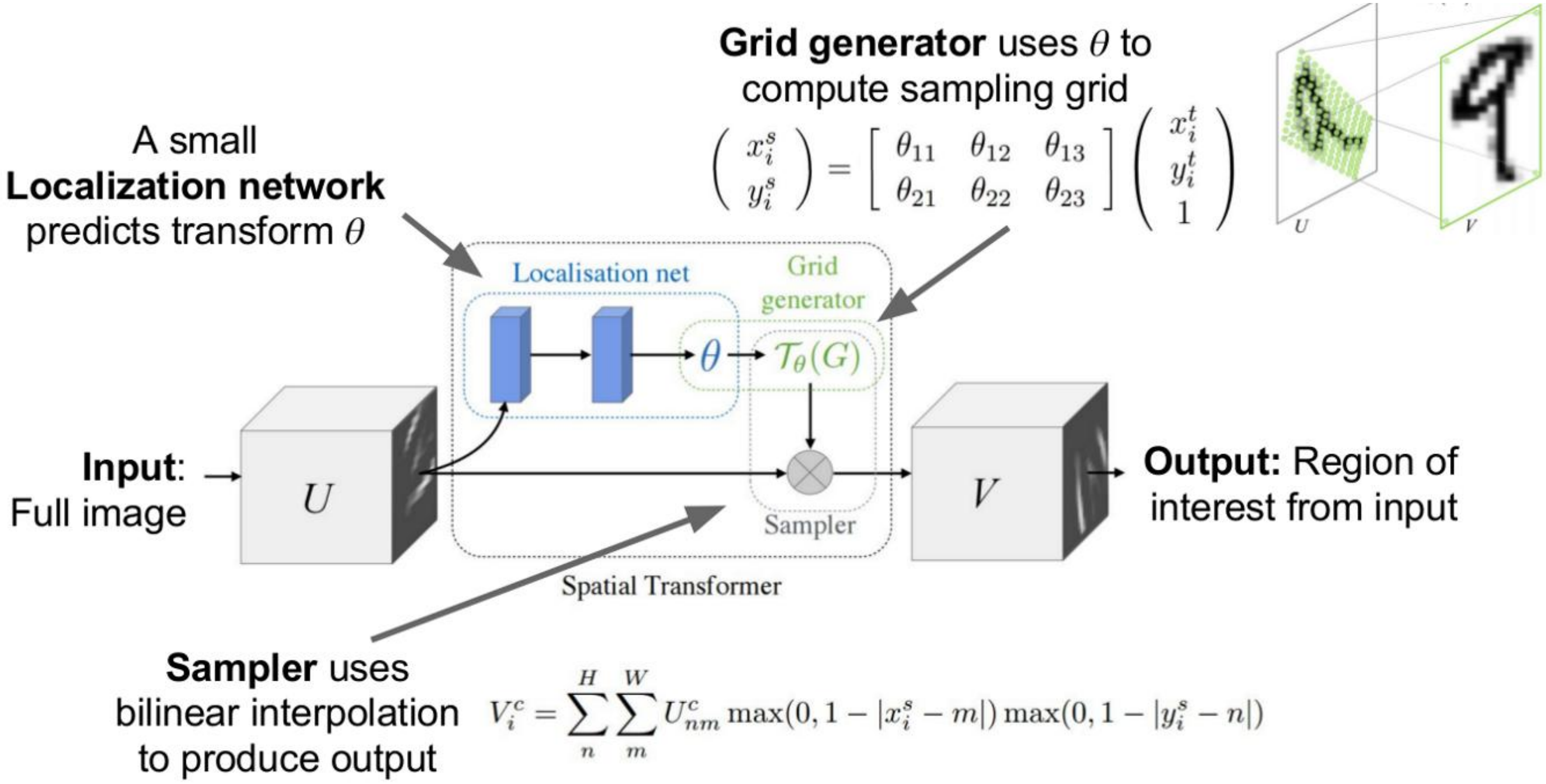
(a) Affinity vs Diversity (b) Model's View of Data

Gontijo-Lopes, R., Smullin, S.J., Cubuk, E.D. and Dyer, E., 2020. Affinity and diversity: Quantifying mechanisms of data augmentation. *arXiv preprint arXiv:2002.08973.*

Regularisation | Invariance and Normalisation

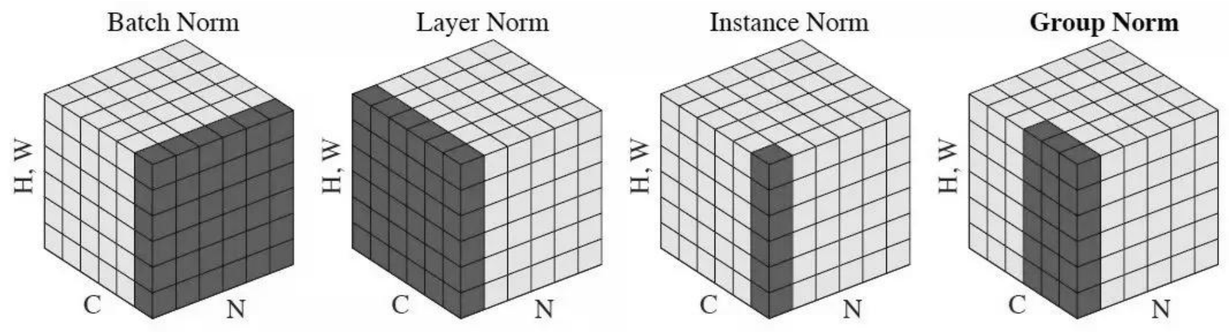
– Spatial transformer networks

“Random transformation in training a model = encouraging the model invariant to the transformation”



– Batch normalisation

- Normalise features to a standard Normal distribution within mini-batch
- Learnable parameters for linearly transforming maintaining expressibility
  - Makes bias redundant in previous network layers
  - Use population statistics during inference
- Benefits
  - Introduce both random additive and multiplicative noise during training
  - Reduce inter-layer dependency



- Batch/layer/instance/group normalization
- BN for CNN and LN for RNN?
- A form of re-parameterisation of layer activations $\hat{\mathbf{x}} = \dfrac{\mathbf{x} - \mu}{\sigma}$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- Weight normalisation

$$\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}$$

- Input normalisation
- Label normalisation – data scaling
- Permutation invariance
- Translation invariance

…

Regularisation | Parameter Constraints

– Parameter norms

- Weight decay by penalising L2- and L1-norms

$$\ln p(\mathbf{w}|\mathbf{t}) = -\frac{\beta}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)\}^2 - \frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \text{const.}$$

- Difference between L2- and L1-norms

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i| \qquad \nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \alpha\text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{X},\boldsymbol{y};\boldsymbol{w})$$

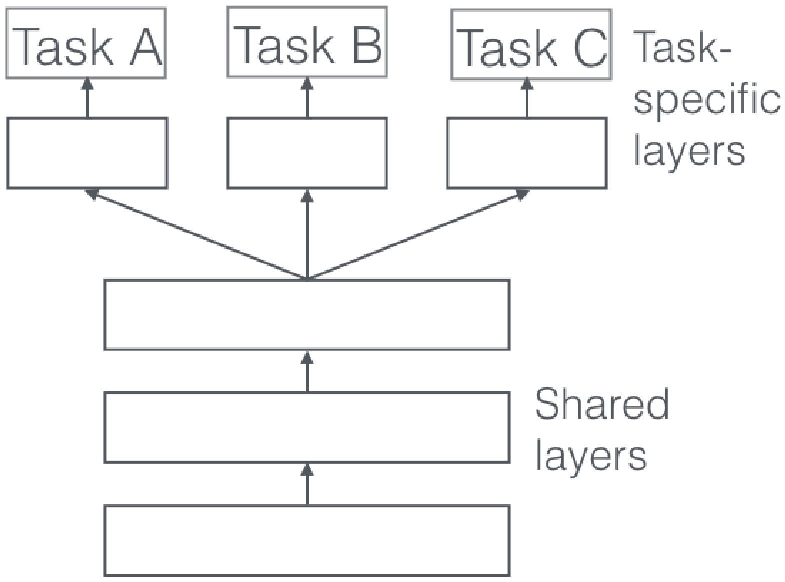
- L2 norm gradient scales with w, therefore less likely to become zero, i.e. sparsity.

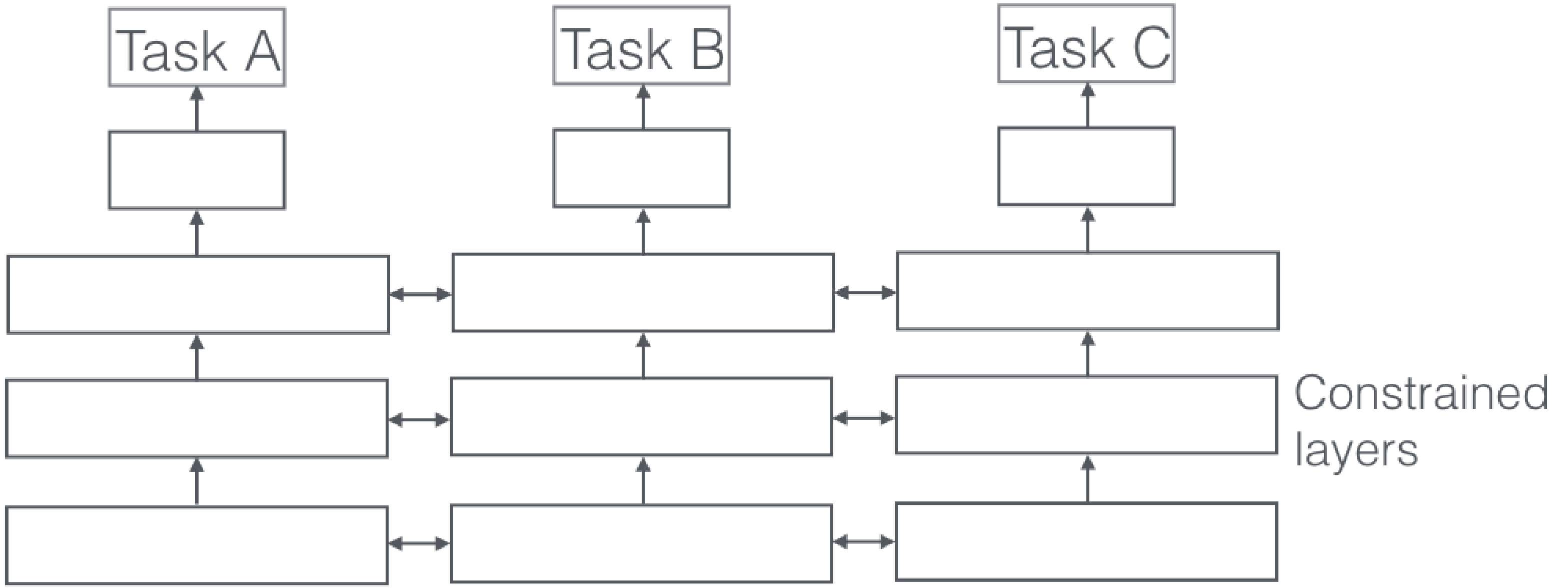
- **Parameter sharing**
- CNN
- RNN
- Tiring

$$\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = \|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\|_2^2$$

- Multi-task learning*
- Semi-supervised learning*.

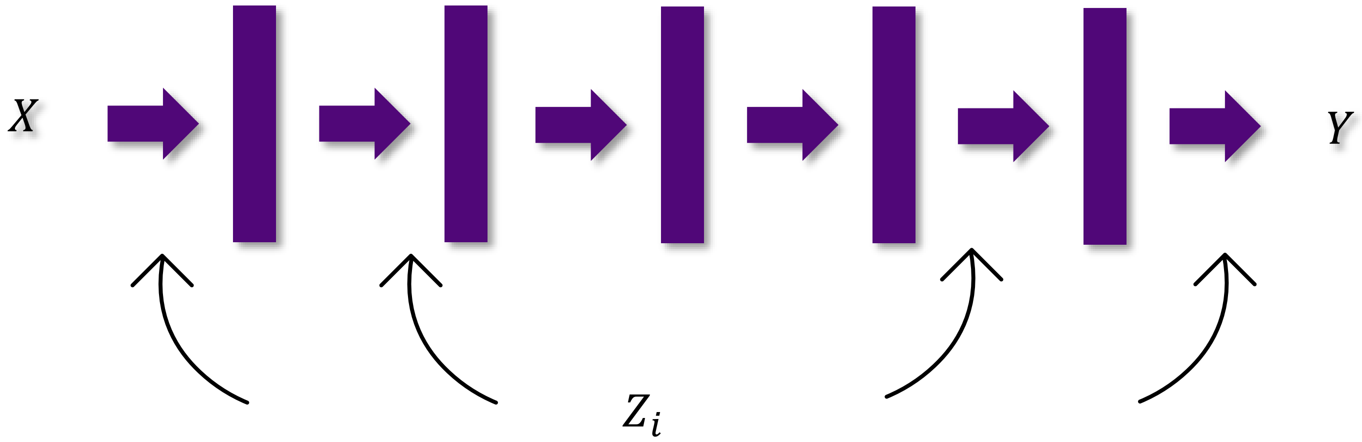
– Multi-task learning





– Multi-task learning

- Generalised forumaltion
  - Z_i: One-hot task index / indicator / descriptor
  - $f(Y|X) \to f(Y|X, Z_i)$

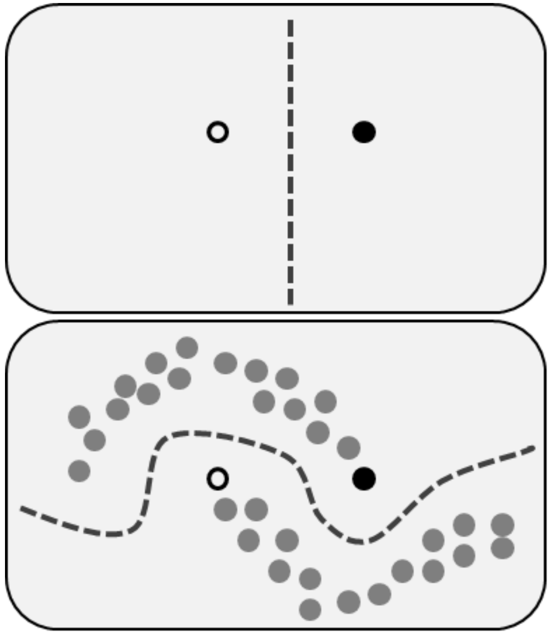


- Implementation
  - Shared and task specific parameters (negative transfer)
  - Concatenation/summation
  - Multiplication conditioning (gating/multi-head)

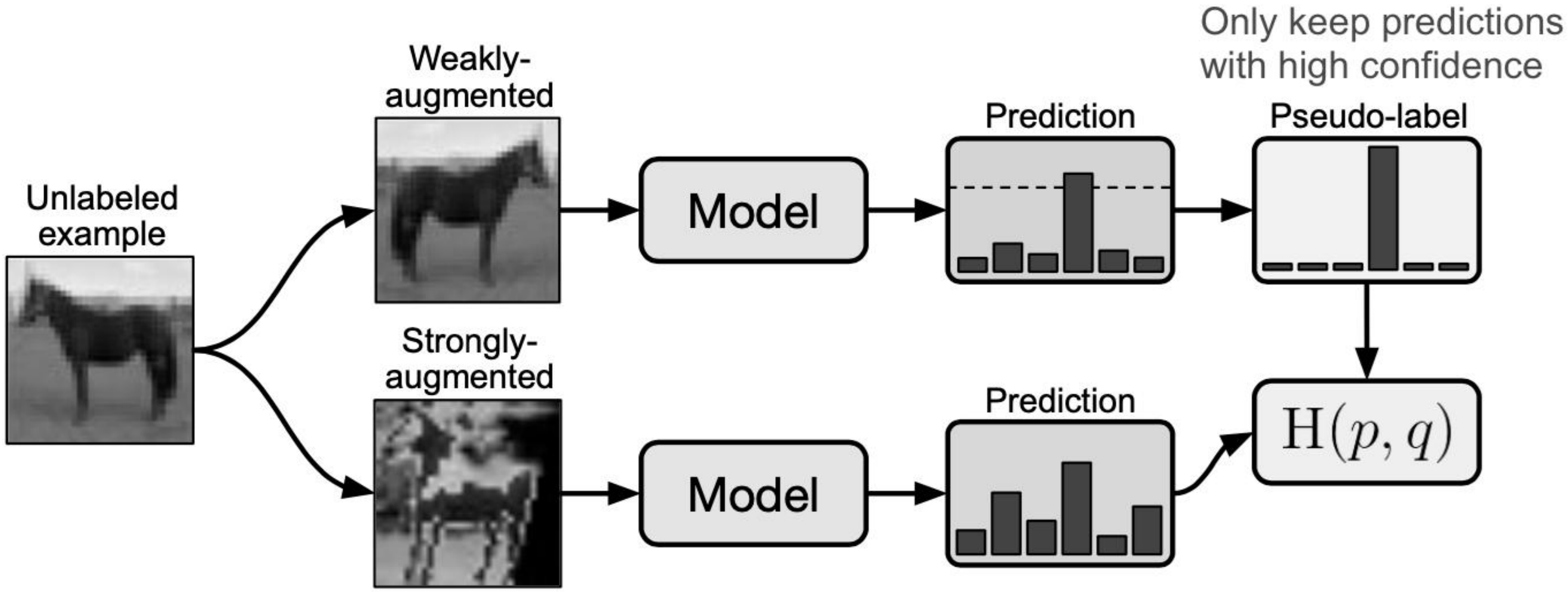
Regularisation | Unsupervised Learning and Generative Modelling

– Semi-supervised learning

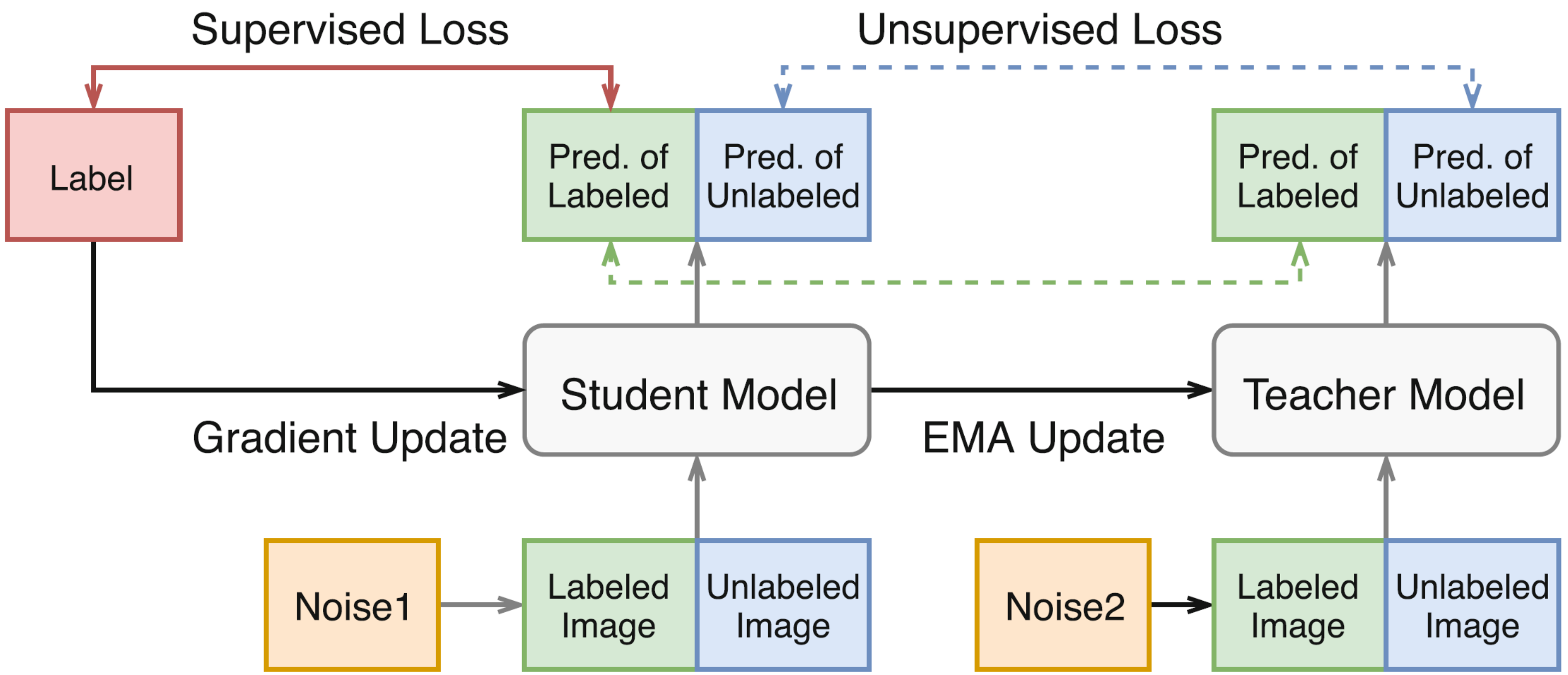
- Supervised learning: labelled data = data + labels, $f(Y|X)$
- Semi-supervised learning: labelled data $f(X,Y)$ + unlabelled data $f(X)$



- Pseudo labels and entropy minimization, consistency



- Semi-supervised learning
  - Student-Teacher



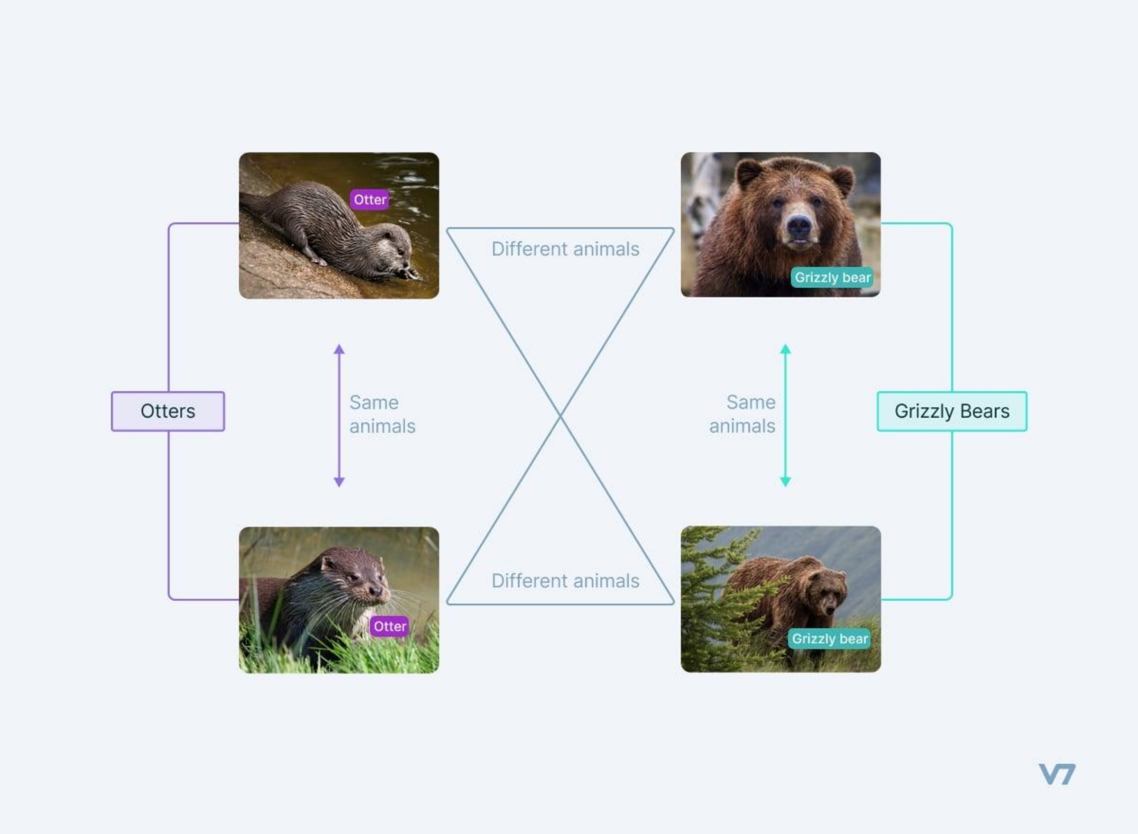
– Semi-supervised learning

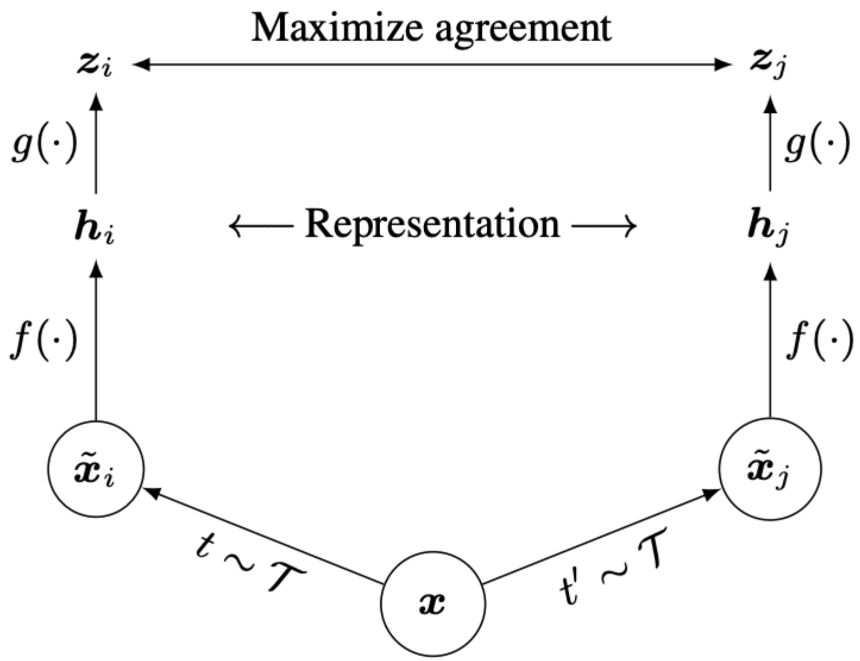
– Contrastive learning

Semi-supervised learning (for utilising unlabelled data)

Representation learning (for benefit other tasks)

e.g. SimCLR



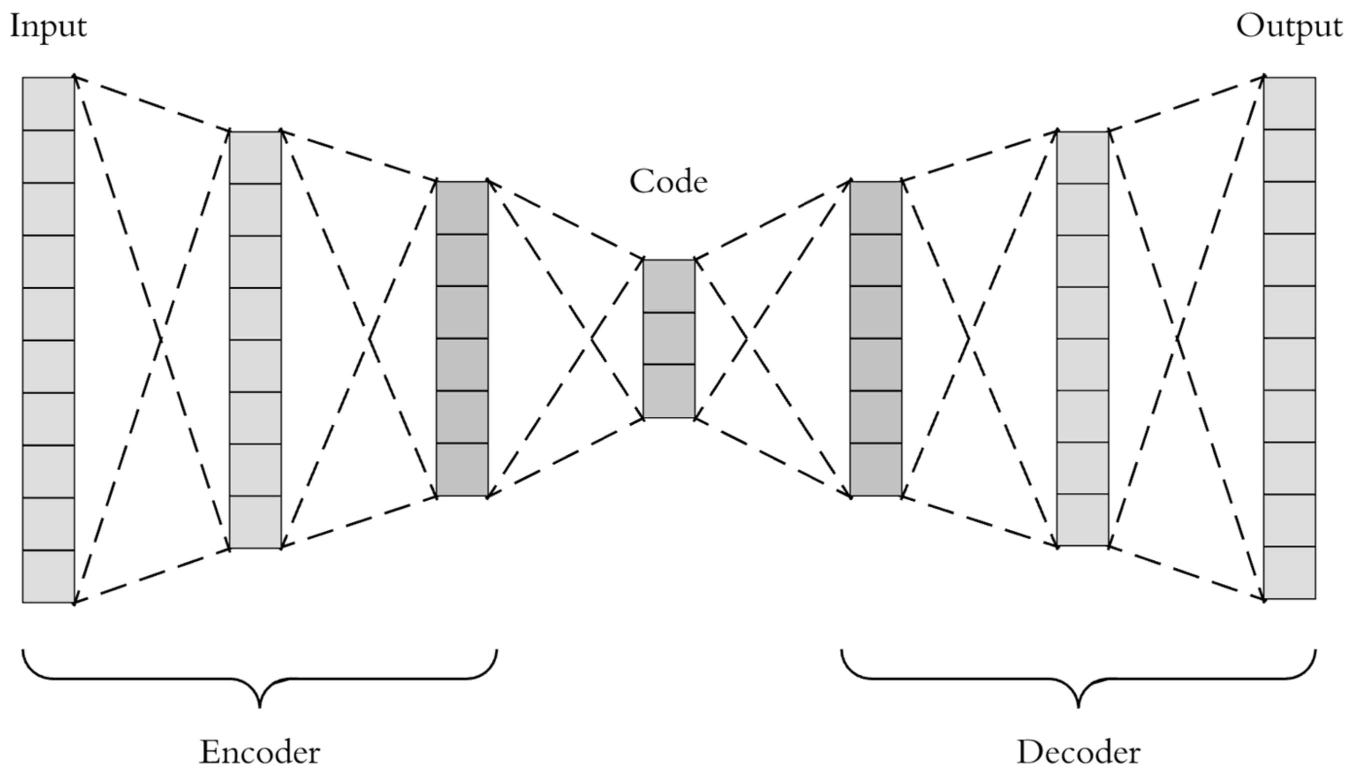


Autoencoder

Autoencoder architecture: encoder - (low-dimension) ”code” - decoder

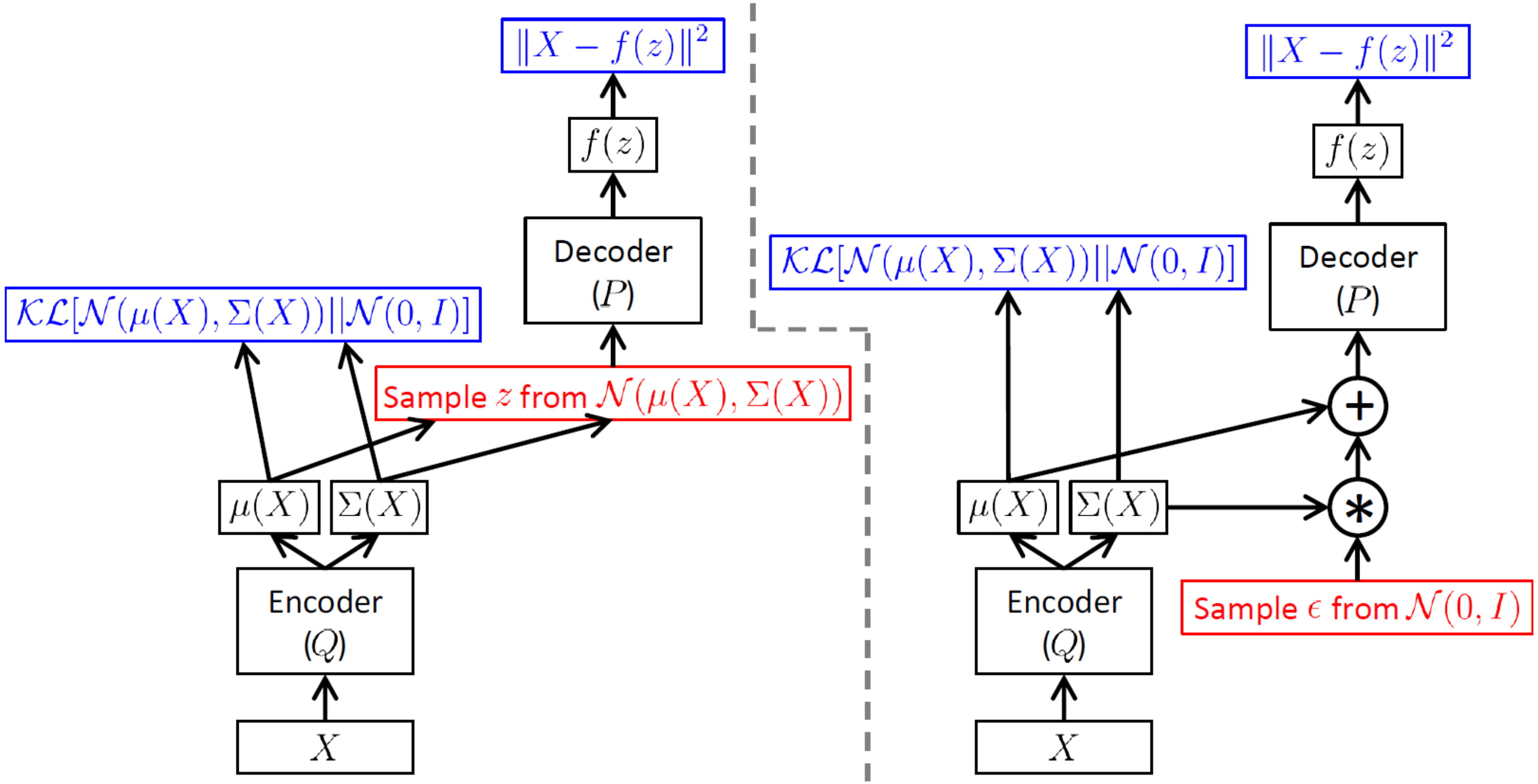
Training loss: self-reconstructing difference, e.g. CE, MSE

Latent “code space”

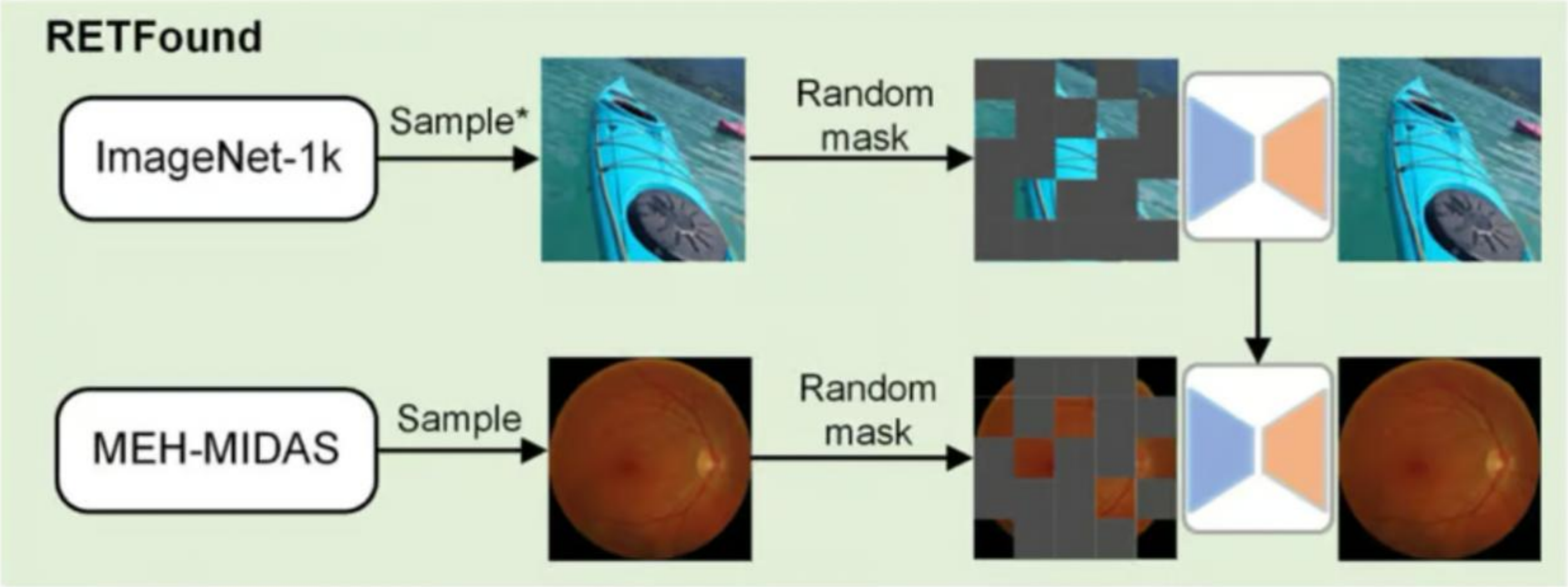


Variational autoencoder

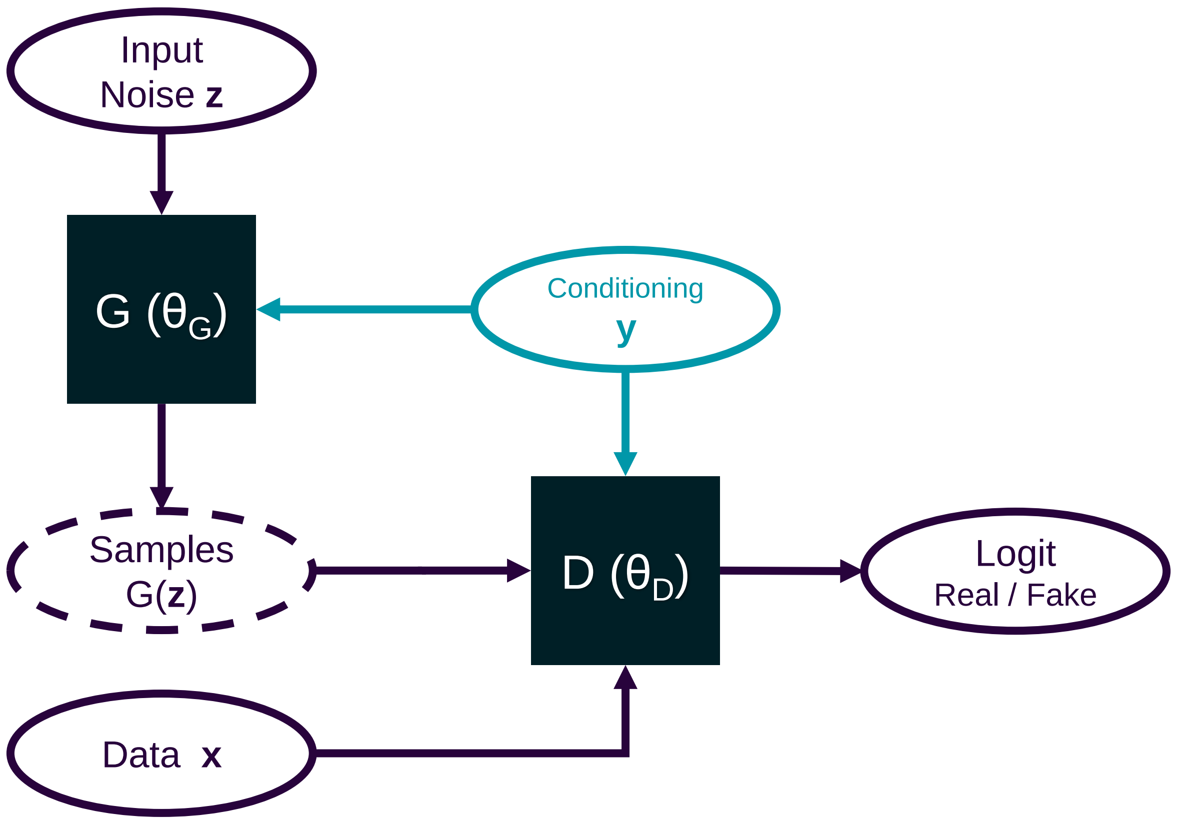
Reparameterisation of latent space



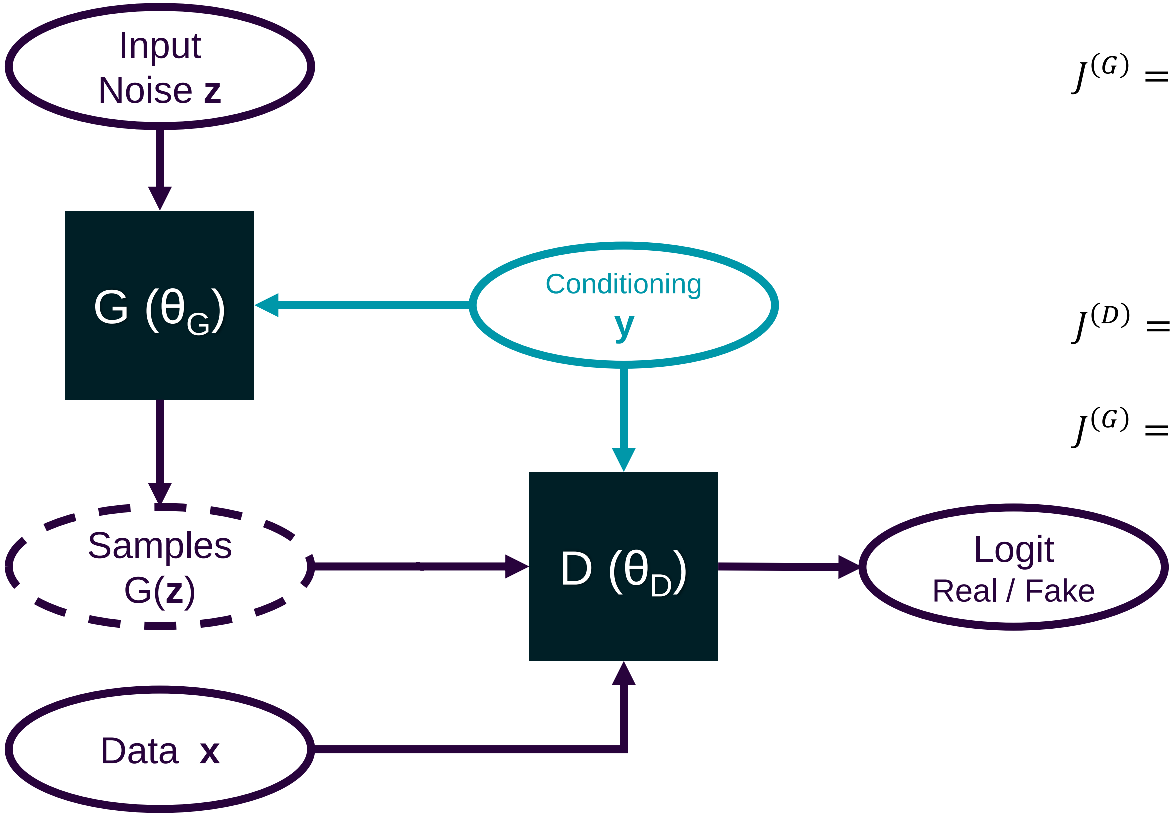
Self-supervised learning: Masked autoencoder



Generative adversarial networks



Generative adversarial networks



$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim P_{data}} \log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim N} \log\left(1 - D(G(\mathbf{z}))\right)$$

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim N} \log D\left(G(\mathbf{z})\right)$$

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_{data}} \log D(\mathbf{x},\mathbf{y}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim N,\mathbf{y}\sim P_{cond}} \log(1 - D(G(\mathbf{z},\mathbf{y}),\mathbf{y}))$$

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim N,\mathbf{y}\sim P_{cond}} \log D(G(\mathbf{z},\mathbf{y}),\mathbf{y})$$

Regularisation | Randomness

Noise

- Gaussian
- Uniform with predefined range
- Bernoulli distribution, i.e. dropout

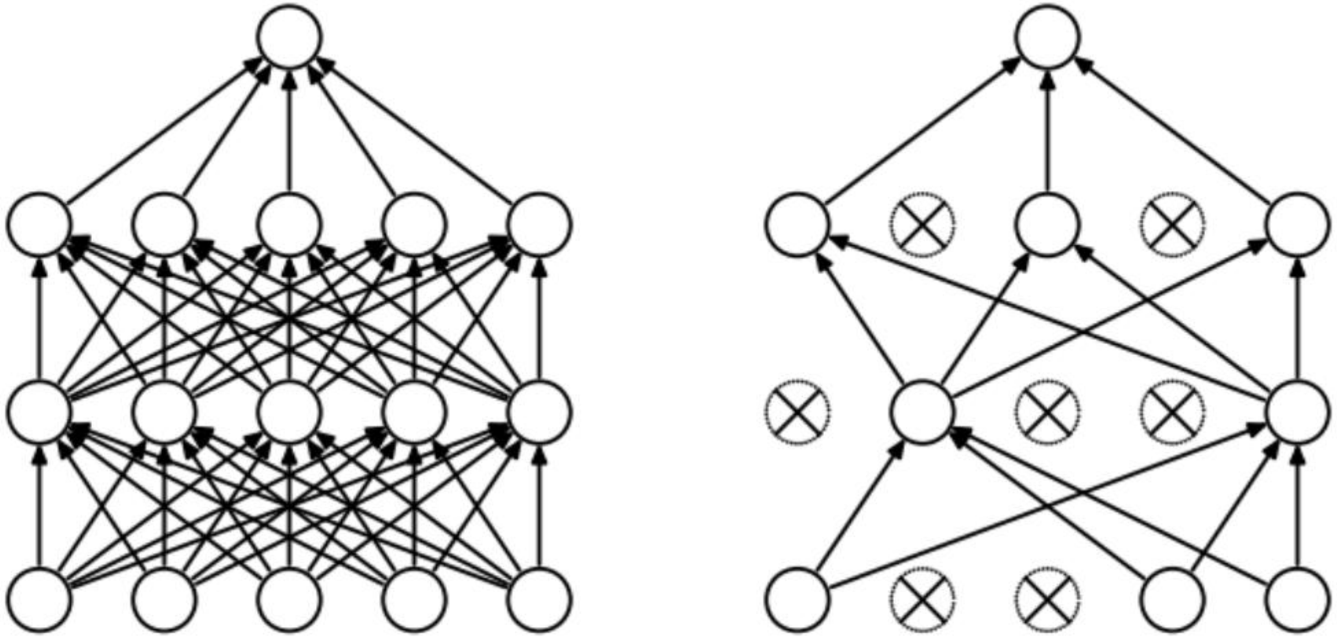
- Inputs, e.g. augmentation, invariant to noise
- Weights, e.g. RNN, at initialisation
- Output targets, e.g. label smoothing
- Edges / hidden units, e.g. dropout*

- Sampling from uncertainty
- Equivalent to penalty on weight norms

. . .

Dropout

- Drop as many as 80% of nodes
- Use all nodes at inference
- Dropout as model ensemble
- Dropout as random noise
- Dropout as Bayesian model sampling
- Uncertainty estimation using dropout
- Posterior estimation at inference
- With convolutional layers?



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

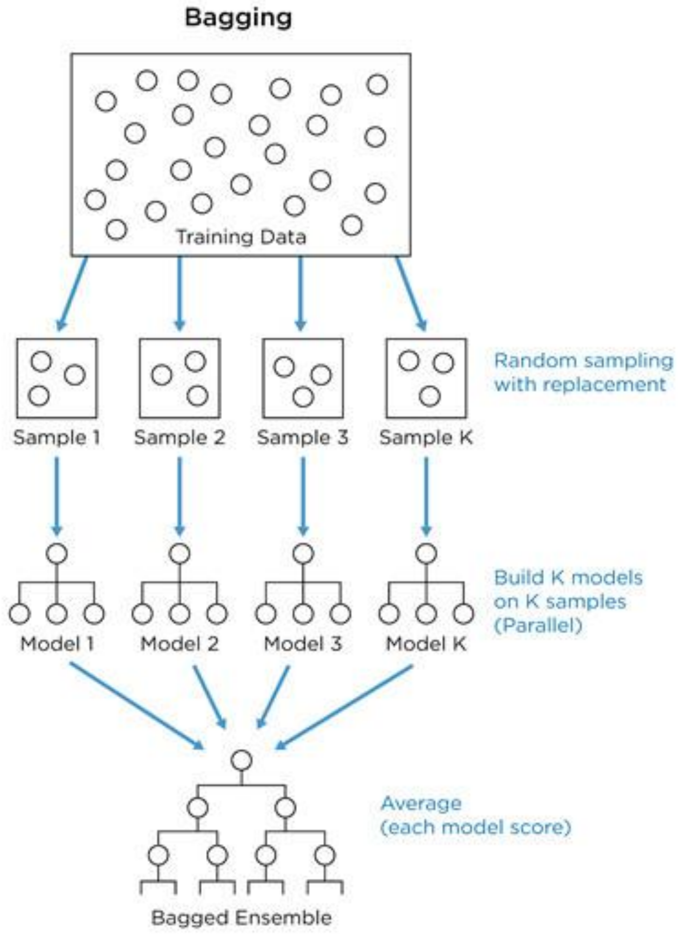
Regularisation | Model Combining

Bagging = bootstrap aggregating ∈ model averaging = ensemble

- Independent errors are multiplicative
- Fully correlated errors does not accumulate
- Optimising multiple low-bias models “cancels” variance, i.e. more samples to estimate

- Bagging: the same kind of model, trained multiple times with bootstrapped datasets
  - Re-trained on the same dataset
  - Challenge winners!

- Committee: multiple models
  - e.g. dropout
- Boosting: multiple models in sequence (convert weak learners to a strong learning)
  - e.g. Tree-based models*



- Training strategies

  Early stopping, curriculum learning, data resampling.

- Data augmentation

  Random data transformation, affinity and diversity

- Invariance and normalisation

  Spatial transformer networks, batch normalisation, reparameterization

- Parameter constraints

  Parameter norms, sparse representation, parameter sharing, multi-task, semi-supervised

- Unsupervised learning

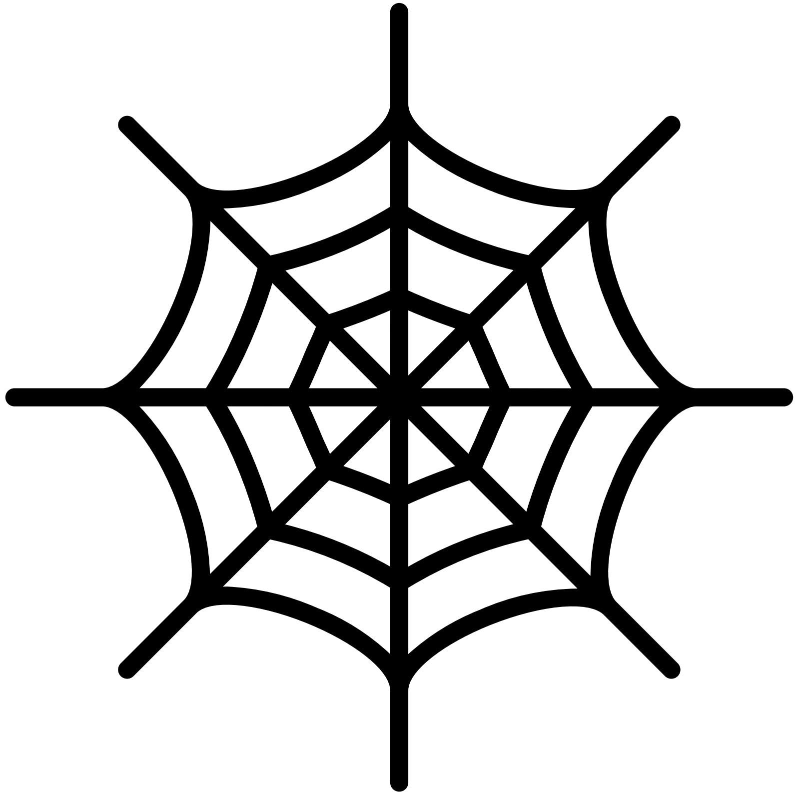
  Autoencoder, generative adversarial network

- Randomness

  Noise, dropout

- Model combining

  Model averaging, ensemble, bagging/bootstrap aggregating, boosting



Add two different regularization methods to the “image classification” tutorial and compare the results.