- **Losses and Classifiers**
  - $f(x; W) = Wx + b$ with three viewpoints
    * Geometric: weights W are hyper-planes cutting up data space.
    * Algebraic: optimization problem, try to minimize loss value.
    * Visual: each weight corresponds to a "template" (i.e. pattern).
  - Classifier loss functions : SVM (linear) vs Softmax (logistic)
    * SVM (hinge) loss : $\sum_{j \neq y_i} \max(0, s_i - s_{y_i} + 1)$, square to penalize large errors.
    * Softmax (cross entropy) loss : $-\log(\sum_i (e^{s_i} / \sum_j e^{s_j}))$, computes log-prob for each correct class.
    * Full : loss + regularized term (sum of squares of all weights, prevent overfitting)
  - Loss optimization : gradient descent $w' = w - \text{learningrate} \times \nabla_W L$, compute $\nabla_W L = \frac{\partial}{\partial w} L$
  - Numeric gradient : $\nabla_W L = \frac{L(W+\delta) - L(W)}{\delta}$ for small δ, easy but inaccurate.
  - Analytic gradient : derive ∇L as a function of W, uses computational graph data structure.
  - Stacking layers : apply non-linearity since 2 linear layers can be combined into one. Feature transform of non-linear layers can map non-linear space into linear ones, easier to predict from.
- **Normalizations, Initializations and Optimizers**
  - Normalizations : try to keep data on the same range.
    * Batch norm : store mean-std of dataset on training (moving average), norm across each channel.
    * Instance norm : Gaussian individual norm across channel and sample.
    * Layer norm : Gaussian norm across each sample.
  - Initialization : optimal weights to start with
    * Normal (Gaussian) initialization tends to zero for larger (deeper) layers.
    * Xavier initialization scales weight by std = $1/\sqrt{D_{in}}$, still bad with ReLU.
    * Kaiming initialization corrects ReLU by std = $\sqrt{2/D_{in}}$
  - Optimizers : Update the weights to lower loss values.
    * SGD : compute gradient descent on either full (expensive) or minibatch (approximate)
    * Problems with SGD : unstable from high condition number, prone to local minima
    * Momentum : $v' = v - \rho \nabla w$, $w' = w - \alpha v'$. Computes velocity from ∇w, ρ acts as friction.
    * Nesterov Momentum : compute gradent after applying velocity (look ahead)
    * AdaGrad : scale gradients with root-sum-of-squares of historical gradients, norm gradient across each dimention. Decays eventually.
    * RMSProp : "Leaky AdaGrad" stores historical sum with moving average.
    * ADAM : Momentum + RMSProp + Bias Correction (avoid explosion from initial guesses for the betas).
  - DL uses first degree optimization, since second degree approx (though more accurate) has extremely high complexity.
  - Learning Rate Decay : slow down training to find global minima (cosine vs linear).
  - Linear warmup : increase LR for the first few epochs before decaying (start with low LR to prevent explosion)
  - Dropout : neglect some features at training time, add back those features and scale accordingly on testing time.
- **Convolutional Neural Networks**
  - Parameters : feature size, kernel size, zero-padding, stride
  - LeNet5 : old architecture with 2 5×5 convolution layers and fully connected classifier.
  - AlexNet : first use of ReLU, heavy data augmentation, uses normalization (not common anymore). Most params are in FC layers.
  - VGG : deeper network, uses smaller kernel 3×3 (combine to increase receptive field), still retains FC classifiers.
  - ResNet : deeper models (100+ layers) should technically be as good as shallow ones but identity maps are difficult to optimize. Uses residual connection to emulate identity function. No more FC classifiers except very last layer.
- **Object Detection**
  - Modal (only visible part) vs Amodal (including hidden parts) object detection.
  - IoU : intersection over union of bounding box, good because it scales with box sizes.
  - Multi-task training : optimize bounding box and classification losses simultaneously.
  - Dealing with multiple objects - Two Stage Detector
    * Sliding window : literally predicts every region - super slow and memory intensive
    * Region proposal : use traditional CV to estimate 2000 regions that may contain an object.
    * R-CNN : re-scale each region to 224×224 and feed each one to a CNN + classfier + bbox regressor. Slow since CNN is used 2000 times.
    * Fast R-CNN : run the entire image through a CNN first.
    * Faster R-CNN : use another CNN to compute region proposal.
  - Dealing with multiple objects - One Stage Detector
    * General idea : divides image into grid cell with assumption each grid can contain a box, background is now a class.
    * RetinaNet : center boxes around grid, predict box with a class and scaling factor for predefined anchors (preset of aspect ratios), deal with class imbalance using focal loss (cross entropy, weighted on hard misclassified samples)
    * FCOS : bbox are predicted as distances from center, centerness loss to ensure boxes stay close to grid. "Anchor Free", more generalizable.
  - Dealing with scale - Feature Pyramid

    * Many stages of CNN at each resolution (higher stage = bigger box, lower res).
    * Add up features of higher res with upscaled lower res image.
    * Connected final features to an object detector at each res.
  - NMS : iteratively pick a box and eliminating boxes with high IoU. Repeat until no box is left.
  - mAP : sorts predicted score, for each score plot a point on the P-R curve, AP = AUC of this curve, mAP = average AP across classes.
  - Semantic Segmentation : FCN (CNN+up/down-sampling+TransposedCNN) takes in local+global features, is end-to-end, and scale-able by size.
  - Instance Segmentation : Mask-R-CNN predicts a semantic mask out of each predicted bounding box.
- **Recurrent Neural Networks and Transformers**
  - Key concept : hidden state $x_t = f(h_{t-1}, x)$ where $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$ and output $y_t = W_{hy}h_t + b_y$.
  - Seq2Seq : Many to one (encodes input seq into one vector) + One to many (decodes output seq into many vectors). Start from $\langle$START$\rangle$ token and end with an $\langle$END$\rangle$ token.
  - CV use cases in image captioning (use CNN's encoded vector as another trainable term for hidden state).
  - Seq2Seq with RNN and attention : RNN generates one context vector that gets bottle-necked and reused.
    * Takes in Q, K, V and outputs a mapped version of Q with context from V.
    * K has the same length as V but is in the same dim as Q.
    * Compute atten weight $W = \text{softmax}(QK^T/\sqrt{d_k})$
    * Output is in the form WV and will have the same length as Q with dimension of V.
    * Here S_{t+1} is from $Q = S_t$, $K = V = (h_1, h_2, ..., h_T)$. Allows for related words to pop-up in translation (or image regions).
  - Vision Transformers : 3 ideas
    * Attention in between CNN : Still a CNN, marginal improvements
    * Replace CNN with local attention : tricky and still marginal improvements
    * Pixel-wise Transformer : super memory consuming
    * ViT : split image into patches, treat encoded vectors from each patch as sequence, add a learnable classification token at the end.
- **Generative Models**
  - Explicit/Probabilistic models : directly compute $p(x)$ over data x
    * Autoregressive : predicts pixels using data from previous pixels (like an RNN), is extremely slow.
    * VAEs : normal AE but latent vector z is sampled from μ and σ, allows for interpolation. Generate blurry images since L_2 is bad.
    * DDPM : add noises to images until converges to $N(0, 1)$, U-Net takes in timestep and noisy image to predict added noises.
  - Implicit models : compute $p(x|y)$ over data x and condition y (either true vs fake or classes)
    * GANs : trains a minimax optimization between generator and discriminator. Can mode collapse if G and D are outbalanced.
  - Stylization : pattern generation uses gram matrix (correlation matrix between each channels from a pretrained image classifier), captures sets of features (thus a texture).
  - Style transfer takes in content and style images. Minimizes gram matrix loss with style and feature loss with content. A dedicated CNN can be trained for a style to improve on speed.
  - Explainable AI Stuff: try to show what's going on in a CNN
    * Nearest neighbor or dim reduction (i.e. PCA or t-SNE) on output vectors.
    * Masks part of input image to show impact on prediction. Saliency via backprop / Guided backprop computes gradient of prediction in respect to pixel.
    * Gradient Ascent : Start from zero image and generates an ideal image of each class. Penalizes L_2 norm of generated images.
    * Adversarial Attacks : generate fake samples to fool the network - easy
    * Adversarial Defance : make your model smart enough to not be fooled - hard
    * DeepDream : amply certain features on already existing images, generates traces of that feature.
- **Self-supervised learning**
  - Sparse AE : reconstruct input from a bottleneck latent vector.
  - Denoising AE : removed added noise into clean images.
  - Context prediction : predict relative location of two patches from an image. Find neighboring patches with KNN.
  - Context encoder : paint missing pixels from a mask. Uses an AE and randomly masks parts of an image.
  - Colorization : model must learn underlying image structure to know which color to fill.
  - Deep Clustering : generated pseudo label with random model and K-means, re-train model with cluster assignments.
  - RotNet : predict image rotations using rotation as pretext
  - ExemplarCNN : predict where an augmented image is from, given a pool of input images.
  - Contrastive learning : minimize similarity across classes, minimize similarity within the same class.
  - Masked AE : reconstruct with most patches are missing with ViT before using a Seq2Seq decoder (another ViT) to generate missing parts.
  - Multi-modal SSL : takes in other inputs (i.e. text caption) i.e. CLIP - matching text and images.